UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Physics and Technology

Exploring the Behavior of Open-Source Diffusion Model Inpainting Algorithms

Vebjørn Halvorsen

FYS-3941 Master's thesis in applied physics and mathematics 30 SP
25 January 2023

This thesis document was typeset using the *UiT Thesis LaTeX Template*.
© 2023 – `http://github.com/egraff/uit-thesis`

“A computer would deserve to be called intelligent if it could deceive a human
into believing that it was human.”
–Alan Turing

Abstract

The present study aimed to examine the performance of an open-source diffusion model inpainting algorithm under varying conditions of inpainting strength and mask radius. However, the results obtained were unexpected and raise significant concerns. Our findings indicate that the algorithm not only modifies the pixels within the designated mask, as intended, but also alters pixels outside of the mask, even those that are unrelated to the inpainted subject. This unexpected behavior has potentially significant implications, particularly in the context of utilizing this algorithm for fine-detailed medical imaging, where the consequences of inaccurate inpainting could be severe.

Utilizing heatmaps and the calculation of mean squared error (MSE), we observed that areas of the image characterized by consistent pixel color, such as the sky or water, tend to undergo minimal alteration during the inpainting process. However, areas of the image that are more varied in color and texture, such as mountain ridges and grass, tend to experience higher, yet relatively low levels of alteration. The heatmaps further reveal that the edge of the inpainting mask is a particularly sensitive area for pixel alteration.

The second experiment conducted in this study, which involved varying the radius of the inpainting mask while keeping the strength constant, showed that as the mask radius increases, the MSE may increase or decrease in a trend-like manner.

This study provides valuable insights into the behavior of the inpainting algorithm, and highlights areas that may require further research. It is important to investigate the relationship between inpainting strength and mask radius in more detail, as well as identify the specific characteristics of images that contribute to their lower MSE. Additionally, the unexpected results of this study regarding the alteration of pixels outside the masked area require further investigation and consideration in the field of utilizing diffusion models for inpainting.

Acknowledgements

I would like to express my sincerest gratitude to my supervisor, Dr. Benjamin Ricaud. Thank you, Benjamin, for sharing your wealth of knowledge and experience in both machine learning and computer vision. Your guidance and direction made this thesis project both fascinating and informative. My deepest appreciation also goes out to my siblings, parents, and grandparents for their unwavering support throughout my academic journey. Lastly, I would like to thank my partner Milla for her constant belief in me and my project.

Contents

List of Figures

1 Introduction

As visual beings, humans can easily recognize items, conceive new settings, and visually comprehend the world around us. When trying to make a computer interpret images, the task's true complexity becomes obvious. Images are just a large collection of numbers that computers organize spatially on a pixel grid, with each color represented by a tuple of three integers in the RGB color model. The main problem in the subject of computer vision is making it possible for computational models to comprehend and complete a variety of tasks using picture input. Recently, diffusion models [1, 2] have been able to generate visual results which are quite astonishing and image augmentation and generation procedures might have been permanently altered by this new technology (Imagen [3], Dalle2 [4] and Midjoruney [5] are some examples). Even while the technology is in theory open source, a concern is that many of these ready-made models have the disadvantage of being behind a paywall for the user.

Fortunetely, in August 2022 the technology was released to the public by a collaboration from Stability AI [6, 7], CompVis LMU [8] and Runway [9]. They released the weights and the code with no paywall and made everybody with a computer powerfull enough able to create visually outstanding image generations. The inpainting method, which seems to have real-world applications beyond just producing beautiful photos, will be tested as part of this study to determine the boundaries of this open source diffusion model.

(a) Prompt: *"full body portrait of a small child, cyberpunk, in the background a gigantic gundam in a blurred city scene"* **(b)** Prompt: *"aerial view of a giant fish tank shaped like a tower in the middle of new york city, 8k octane render, photorealistic"* **(c)** Prompt: *"the entire universe contained inside a glass jar, super realistic, hyper detailed, dramatic lighting, 4k"*

Figure 1.1: Some example images from Midjourneys community showcase generated images. Each image is create with the prompt written in the captions. Source [5].

1.1 Image generation with deep generative models

The task of image generation using deep generative models (DGMs) is to synthesize unseen images that accurately reflect the underlying probability distribution of a given dataset. This endeavour is highly complex and requires a combination of advanced mathematical and statistical concepts, in addition to the implementation of multiple machine learning techniques.

DGMs are the ideal choice for this task, as they are specifically designed to learn the intricate patterns and structures within a set of images and use this information to generate new images. This is achieved by learning a distribution over the space of all possible images, allowing for sampling from this distribution in order to produce novel images.

DGMs have several advantages for image generation, including their capacity to capture high-level features and abstractions in the data. This allows for the production of images which are not only realistic in appearance, but also have meaningful and logical content. Additionally, deep generative models can be trained on large datasets, thereby providing them with the necessary complexity and diversity to generate realistic images of the real world.

In conclusion, image generation with DGMs is an engaging and difficult area of research that could have a major impact on a variety of applications, such as computer vision, graphics, and artificial intelligence.

/2 Theory

2.1 Generative models

Unsupervised learning is a powerful method for understanding any kind of data distribution, and generative models have been particularly effective in recent years. These models aim to learn the underlying distribution of the training data, allowing them to generate new data points with some variations. This is useful because it is not always possible to fully understand the distribution of our data, either intuitively or explicitly. To accomplish this, we can harness the learning capabilities of neural networks to develop a function that roughly matches the model distribution to the real distribution.

In generative models, we use the training data to estimate the prior probability, $P(Y)$, and the likelihood probability, $P(X|Y)$. By applying Bayes' Theorem, we can then calculate the posterior probability, $P(Y|X)$. Here, X represents the observed data and Y represents the latent variables or hidden states. The goal of generative models is to estimate the probability of the observed data given the latent variables, $P(X|Y)$, and the probability of the latent variables, $P(Y)$. By using Bayes' Theorem to compute the probability of the latent variables given the observed data, $P(Y|X)$, we can gain insights into the underlying relationships between these variables. This computation forms the foundation of Bayesian generative models.

In the past, Variational Autoencoders (VAE) and Generative Adversarial Networks (GAN) were two of the most commonly used and effective techniques.

GANs try to find an equilibrium between the generator and the discriminator, while VAEs try to maximize the lower bound of the data log-likelihood. It is important to note that while generative models have achieved great success in various tasks, they are not a panacea and have their own limitations. For example, they may struggle to capture highly structured or multi-modal data distributions, and can be difficult to train. Despite these limitations, generative models continue to be an active area of research, with new techniques and approaches being developed all the time.

Recently, the field of diffusion models has seen a spike in popularity, with increasing numbers of studies devoted to this area of research. They were first introduced by Sohl-Dickstein et al. [1] and have undergone significant improvements and refinements in subsequent papers. For example, in the study conducted by Dhariwal and Nichol [10], diffusion models were shown to outperform the then-current state-of-the-art generative models, effectively rendering GANs a thing of the past. A more in-depth examination of diffusion models can be found in Section 2.2.

Another important aspect of generative models is evaluating their performance. This can be a challenging task, as it is often difficult to quantify how closely the model distribution matches the real data distribution. One popular method for evaluating generative models is the "Inception Score" (IS) [11], which measures the quality and diversity of the generated samples. The IS is calculated by training an image classification model on the generated samples and real data, and then measuring the classification accuracy. A higher IS indicates that the generated samples are of high quality and diverse. However, the IS has been criticized for being sensitive to the choice of image classification model and not necessarily indicative of the realism of the generated samples.

Other metrics for evaluating generative models include the "Fréchet Inception Distance" (FID) [12], which measures the distance between the feature distributions of the generated samples and real data, and the "Kernel Maximum Mean Discrepancy" (MMD) [13], which measures the distance between the distributions using a kernel function. Like the IS, these metrics also have their own limitations and trade-offs, and it is important to consider the specific task and evaluation criteria when choosing a metric.

In conclusion, generative models have proven to be a powerful tool for understanding and generating data distributions. These models have been successful in various tasks, but also have their own limitations. The field of diffusion models is a relatively new area of research that has shown promising results in outperforming other generative models.

2.2 Diffusion models

The concept of diffusion models was first introduced in the context of deep learning by Sohl-Dickstein et al. [1]. At a high level, diffusion models operate by intentionally corrupting the training data through the addition of noise, and subsequently learning to reverse this process through denoising techniques. This allows diffusion models to generate coherent images from noise.

As generative models, diffusion models can be utilized to produce data similar to that which they have been trained on. This is achieved by adding Gaussian noise to the training data incrementally and learning to reverse this process through a denoising technique. Once trained, the diffusion model can be used to generate data by applying the learned denoising technique to randomly sampled noise. Images are one type of data for which the effectiveness of diffusion models can easily be visualized, and the results produced by current models are noteworthy. It is worth noting that the original diffusion model proposed by Sohl-Dickstein et al. [1] does not incorporate a latent space, a feature that was introduced in later papers and models. The mathematical foundations presented in this section 2.2 are derived from Sohl-Dickstein et al. [1], Ho et al. [14], Nichol and Dhariwal [15], with explanations provided in Karagiannakos et al. [16] and Luo [17].

Furthermore, by conditioning the image generation process, diffusion models can be utilized in conjunction with text-to-image guidance to generate an almost unlimited number of images from text alone. The image generation process can be guided by inputs from embeddings such as CLIP [18], providing robust text-to-image capabilities. A more in-depth, theoretical and mathematical explanation of diffusion models in general will be provided in the following sections of this theory section.

2.2.1 Forward process

A diffusion process is a statistical method that utilizes the properties of a Gaussian distribution and a Markov chain to gradually introduce noise to a given data distribution. This process can be used to approximate the posterior distribution of a set of latent variables, denoted as $X_1, ..., X_t$, given an initial variable X_0. The approximate posterior distribution is represented as $q(X_1 : T \mid X_0)$, where the latent variables have the same dimensionality as the initial variable.

The use of a Gaussian distribution allows for the incorporation of uncertainty in the process, while the Markov chain ensures that the evolution of the latent variables is dependent only on their current state and not on their past states.

This results in a temporal consistency in the latent variables, allowing for the modeling of temporal dependencies in the data.

The forward process of a diffusion process is illustrated in Figure 2.1. This figure shows the progression of the latent variables through time, starting from the initial variable X_0 and ending at the final variable X_T. The incorporation of noise at each time step through the use of the Gaussian distribution and the Markov chain results in the approximation of the posterior distribution of the latent variables.

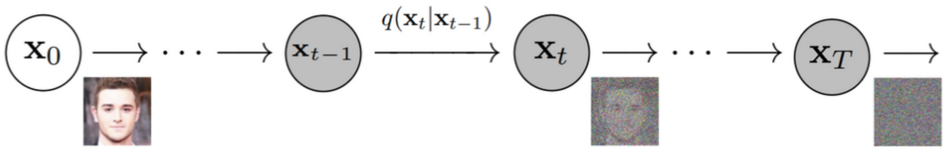


Figure 2.1: Illustration of the transition from clear to noisy image. Modified from Ho et al. [14].

The diffusion process depicted in Figure 2.1 can be represented mathematically using the framework of Markov chains. Specifically, the process can be expressed as in Equation 2.1, which represents a sequence of sampling steps from a Gaussian distribution $\mathcal{N}$. The mean of the distribution at time step t is given by $\mu_t = \sqrt{1-\beta_t}\mathrm{x}_{t-1}$, while the covariance matrix is represented by $\Sigma = \beta_t \mathrm{I}$. Here, β_t is a noise scheduler that can be varied based on the time step t, and I represents the identity matrix.

$$q\left(\mathrm{x}_{1:T} \mid \mathrm{x}_0\right) := \prod_{t=1}^{T} q\left(\mathrm{x}_t \mid \mathrm{x}_{t-1}\right) := \prod_{t=1}^{T} \mathcal{N}\left(\mathrm{x}_t; \sqrt{1-\beta_t}\mathrm{x}_{t-1}, \beta_t \mathrm{I}\right) \tag{2.1}$$

One issue with this representation is that computing the distribution at a particular time step, such as step 450 out of 500, requires recalculating the distribution for the previous 449 steps. To address this issue, the reparameterization trick can be used to rewrite the expression in a recursive form. This involves defining $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}t = \prod s = 0^t \alpha_s$, with $\epsilon_0, \ldots, \epsilon_{t-2}, \epsilon_{t-1} \sim \mathcal{N}(0, \mathrm{I})$. Using this reparameterization, it can be shown that:

$$
\begin{aligned}
\mathrm{q}\left(\mathrm{x}_t \mid x_{t-1}\right) &= \mathcal{N}\left(x_t, \sqrt{1-\beta_t} x_{t-1}, \beta_t I\right) \\
&= \sqrt{1-\beta_t} x_{t-1} + \sqrt{\beta_t} \epsilon \\
&= \sqrt{\alpha_t} x_{t-1} + \sqrt{1-\alpha_t} \epsilon \\
&= \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \sqrt{1-\alpha_t \alpha_{t-1}} \epsilon \\
&= \sqrt{\alpha_t \alpha_{t-1} \alpha_{t-2}} x_{t-3} + \sqrt{1-\alpha_t \alpha_{t-1} \alpha_{t-2}} \epsilon \\
&= \sqrt{\alpha_t \alpha_{t-1} \cdots \alpha_1 \alpha_0} x_0 + \sqrt{1-\alpha_t \alpha_{t-1} \cdots \alpha_1 \alpha_0} \epsilon \\
&= \sqrt{\alpha_t} x_0 + \sqrt{1-\bar{\alpha}_t} \epsilon
\end{aligned}
\tag{2.2}
$$

By utilizing the trick in equation 2.2, it is possible to generate a sample at any time step t using only the input sample x_0 and without the need to iterate through all previous time steps. This is achieved through the following equation:

$$
\mathrm{x}_t \sim q\left(\mathrm{x}_t \mid \mathrm{x}_0\right) = \mathcal{N}\left(\mathrm{x}_t ; \sqrt{\bar{\alpha}_t} \mathrm{x}_0, \left(1-\bar{\alpha}_t\right) \mathrm{I}\right) \tag{2.3}
$$

Note that β_t is a hyperparameter, which means that it is a value that is chosen prior to training any algorithms and is not learned through the training process. As a result, it is possible to precompute a_t and $\bar{\alpha}_t$ for all time steps t. This property will be useful later on when calculating the loss L_t.

2.2.2 Different variance schedules

There are various options for selecting the schedule of the variance parameter β_t over the T time steps. This schedule can be fixed to a constant value, or it can be varied using a variety of mathematical functions such as linear, quadratic, or cosine. In the original work on denoising diffusion models by Sohl-Dickstein et al. [1], which was later improved upon by Ho et al. [14], a linear schedule was used, with $\beta_1 = 10^{-4}$ and $\beta_T = 0.02$. However, it has been demonstrated by Nichol and Dhariwal [15] that using a cosine schedule can yield even better results.

2.2.3 Backward process

So far, we have demonstrated the process of adding noise to images at different time steps as a means of preparing for denoising. However, the key question remains: how does the denoising actually work? As T approaches infinity, the

Figure 2.2: Latent samples from linear and cosine schedules at linearly spaced t values between 0 and T are shown in the top and bottom, respectively. The latter quarter of the linear schedule's latents are virtually entirely noise, but the cosine schedule gradually introduces noise. Source: [15]

noised xt becomes increasingly similar to an isotropic Gaussian distribution (when $\Sigma = \sigma^2\mathrm{I}$). By generating noise from this distribution, $\mathcal{N}(0, \mathrm{I})$, and learning the distribution $q(\mathrm{x}t - 1 \mid \mathrm{x}_t)$, it is possible to generate the "previous" noise. If this process is repeated for a sufficient number of time steps, we can eventually generate a sample from the original distribution, $q(\mathrm{x}0)$. However, the challenge lies in learning the distribution $q(\mathrm{x}t - 1 \mid \mathrm{x}_t)$, which is unknown in practice.

Estimating the statistical properties of $q(\mathrm{x}_{t-1} \mid \mathrm{x}t)$ is intractable (there is no existing algorithm capable of finding this distribution). Instead, we must rely on approximations using the data distribution itself. This is where deep learning and machine learning excel, as we can use a neural network, $p\theta$, to approximate this distribution by parameterizing the mean and variance (as shown in Figure 2.3). By selecting a small enough value for β_t, we can use this neural network to estimate these parameters.

$$p_\theta\left(\mathrm{x}_{t-1} \mid \mathrm{x}_t\right) = \mathcal{N}\left(\mathrm{x}_{t-1}; \boldsymbol{\mu}_\theta\left(\mathrm{x}_t, t\right), \Sigma_\theta\left(\mathrm{x}_t, t\right)\right) \tag{2.4}$$

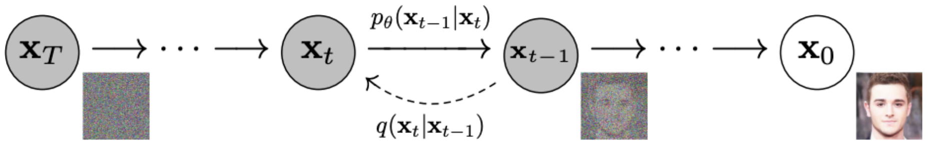

$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$

Figure 2.3: Illustration transferring between different time steps T between X_T and X_0 in a Markov chain. Source Ho et al. [14].

As previously mentioned, by repeatedly applying the reverse formula at each time step, it is possible to recover the original (estimated) data distribution from x_T. This process can be thought of as "unwinding" the noise that was added at each time step, eventually arriving at the original distribution.

$$p_\theta\left(\mathrm{x}_{0:T}\right)=p_\theta\left(\mathrm{x}_T\right)\prod_{t=1}^{T}p_\theta\left(\mathrm{x}_{t-1}\mid\mathrm{x}_t\right) \tag{2.5}$$

By incorporating the time step t as a parameter for the model to learn, we can estimate the Gaussian parameters for each time step. This allows us to determine the mean, $\mu_\theta(\mathrm{x}t,t)$, and covariance matrix, $\Sigma\theta(\mathrm{x}_t,t)$, for each time step t. This additional information can be leveraged to improve the accuracy of the model and enhance its ability to capture the underlying distribution of the data.

2.2.4 Training a diffusion model

It can be observed that the combination of distributions p and q is similar to what is present in the learning process of a Variational Autoencoder (VAE). This suggests that we can train our model by optimizing the negative log-likelihood of our training data, which is given by the Loss function: $-\log\left(p_\theta\left(x_0\right)\right)$. However, this probability is difficult to compute as it depends on all previous time steps $x_0,x_1,x_2,x_3,\ldots,x_{T-3},x_{T-2},x_{T-1},x_T$. As a solution, we can instead compute the Variational Lower Bound (ELBO) [19], which is commonly used in VAEs. The underlying principles of the ELBO are simple to understand. Imagine we have a function $f(x)$ that we cannot calculate, but have another function $g(x)$ that we can compute (figure 2.4) and which is always less than $f(x)$. Then, if we maximize $g(x)$, we can be sure that $f(x)$ will also increase. In our case, $-\log\left(p_\theta\left(x_0\right)\right)$ is the $f(x)$.

Deriving the equation for the ELBO involves a high degree of complicated math, which is not presented here (the full derivation can be found in Luo [17] from equation 34 to 58). However, at the end of these calculations, we arrive at an equation for the ELBO.

$$\begin{aligned}\log p(\mathrm{x})\geq&\mathbb{E}_{q(x_1\mid x_0)}\left[\log_\theta\left(\mathrm{x}_0\mid\mathrm{x}_1\right)\right]-\\&D_{KL}\left(q\left(\mathrm{x}_T\mid\mathrm{x}_0\right)\|p\left(\mathrm{x}_T\right)\right)-\\&\sum_{t=2}^{T}\mathbb{E}_{q(\mathrm{x}_t\mid\mathrm{x}_0)}\left[D_{KL}\left(q\left(\mathrm{x}_{t-1}\mid\mathrm{x}_t,\mathrm{x}_0\right)\|p_\theta\left(\mathrm{x}_{t-1}\mid\mathrm{x}_t\right)\right)\right]\\&=L_0-L_T-\sum_{t=2}^{T}L_{t-1}\end{aligned} \tag{2.6}$$

Equation 2.6 from Karagiannakos et al. [16] presents a revised version of equa-

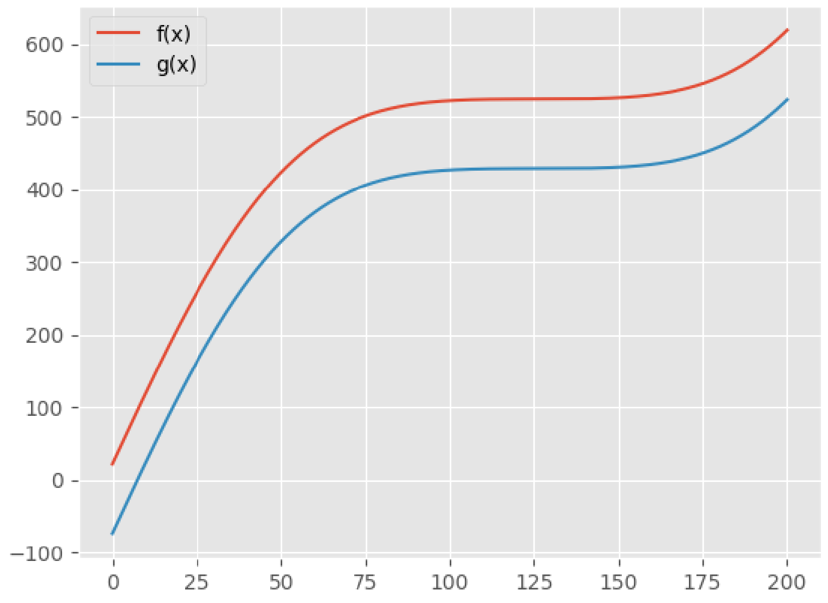


Figure 2.4: Plot of two arbitrary functions illustrating how $f(x) \geq g(x)$ for intuition behind ELBO.

tion (5) in Ho et al. [14]. This rewritten equation is used in the learning process of a Variational Autoencoder (VAE). The ELBO, or evidence lower bound, is an important concept in VAE training because it allows us to approximate the log-likelihood of the data, which is the function we want to maximize in order to improve the performance of our model. By optimizing the ELBO, we can train our VAE to reconstruct the input data accurately while also generalizing well to new data. The original equation (5) in Ho et al. [14] expresses the ELBO in a different form, but Karagiannakos et al. [16] showed that it can be rewritten as equation 2.6 without loss of generality. This rewritten form of the ELBO has several advantages, such as being more tractable for optimization and having a clearer interpretation in terms of the reconstruction loss and regularization term.

$$\mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathrm{x}_T \mid \mathrm{x}_0\right) \| p\left(\mathrm{x}_T\right)\right)}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathrm{x}_{t-1} \mid \mathrm{x}_t, \mathrm{x}_0\right) \| p_\theta\left(\mathrm{x}_{t-1} \mid \mathrm{x}_t\right)\right)}_{L_{t-1}} \underbrace{- \log p_\theta\left(\mathrm{x}_0 \mid \mathrm{x}_1\right)}_{L_0}]$$

In the following, we will analyze the terms in equation 2.6.

1. The first term $\mathbb{E}q\left(x_1 \mid x_0\right)\left[\log p\theta\left(\mathrm{x}_0 \mid \mathrm{x}_1\right)\right]$ can be understood as a reconstruction term, similar to the one in the evidence lower bound (ELBO) of a variational autoencoder.

2. The second term $D_{KL}\left(q\left(\mathrm{x}_T \mid \mathrm{x}_0\right) \| p\left(\mathrm{x}_T\right)\right)$ measures the similarity between x_t and a typical Gaussian. It should be noted that this term does not contain any trainable parameters, and therefore is not affected during training.

3. The third term $\sum_{t=2}^{T} L_{t-1}$, also known as L_t, captures the difference between the required denoising steps $p_\theta\left(\mathrm{x}_{t-1} \mid \mathrm{x}_t\right)$ and the approximations $q\left(\mathrm{x}_{t-1} \mid \mathrm{x}_t, \mathrm{x}_0\right)$.

To effectively denoise a given noisy image, it is necessary for the model to have a clear understanding of the desired outcome. This is accomplished by conditioning the reverse diffusion step, represented by $q(\mathrm{x}_{t-1} \mid \mathrm{x}_t, \mathrm{x}_0)$, on the input image x_0. Essentially, this means that we sample the previous step x_{t-1} given x_t and the known clean image x_0.

The goal of the optimization process, as stated in equation 2.6, is to maximize the likelihood of the denoising steps Lt. This is achieved through the use of the reparameterization trick on $q(x_t \mid x_{t-1})$. By defining $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=0}^{t} \alpha_s$, we are able to express the distribution in a form that allows for efficient optimization. We can apply this same technique to $q(\mathrm{x}_{t-1} \mid \mathrm{x}_t, \mathrm{x}_0)$ in order to achieve the desired result.

$$q\left(\mathrm{x}_{t-1} \mid \mathrm{x}_t, \mathrm{x}_0\right) = \mathcal{N}\left(\mathrm{x}_{t-1}; \tilde{\boldsymbol{\mu}}\left(\mathrm{x}_t, \mathrm{x}_0\right), \tilde{\beta} t \mathrm{I}\right)$$

where

$$\begin{aligned} \tilde{\beta}_t &= \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t \\ \tilde{\boldsymbol{\mu}}_t\left(\mathrm{x}_t, \mathrm{x}_0\right) &= \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathrm{x}_0 + \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\mathrm{x}_t \end{aligned} \tag{2.7}$$

It is now possible to render the ELBO fully tractable through the use of a mathematical calculation that proceeds towards a solution. By examining the last line in Equation 2.2, we can represent it as an expression for x_0:

$$\mathrm{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(\mathrm{x}_t - \sqrt{1 - \bar{\alpha}_t \boldsymbol{\epsilon}} \right) \Big) \quad \text{where} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I) \tag{2.8}$$

By substituting equation 2.8 into 2.7 we get a mean $\tilde{\boldsymbol{\mu}}_t$ that only depends on x_t (no longer including x_0):

$$\tilde{\boldsymbol{\mu}}_t \left(\mathrm{x}_t \right) = \frac{1}{\sqrt{\alpha_t}} \left(\mathrm{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon} \right) \Big) \tag{2.9}$$

Now we can use a neural network to approximate the the noise $\boldsymbol{\epsilon}$ and consequently the mean. Implementing the new notations yield:

$$\tilde{\boldsymbol{\mu}}_\theta \left(\mathrm{x}_t, t \right) = \frac{1}{\sqrt{\alpha_t}} \left(\mathrm{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta \left(\mathrm{x}_t, t \right) \right) \tag{2.10}$$

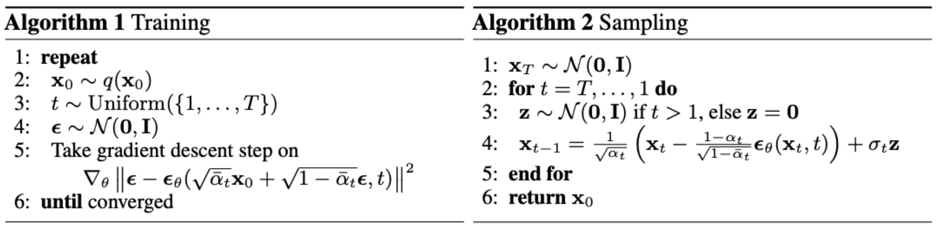
The loss function L_t (which is the term for denoising in the ELBO) can now be written as:

$$\begin{aligned} L_t &= \mathbb{E}_{\mathrm{x}_0, t, \epsilon} \left[\frac{1}{2 \left\| \Sigma_\theta \left(x_t, t \right) \right\|_2^2} \left\| \tilde{\boldsymbol{\mu}}_t - \boldsymbol{\mu}_\theta \left(\mathrm{x}_t, t \right) \right\|_2^2 \right] \\ &= \mathbb{E}_{\mathrm{x}_0, t, \epsilon} \left[\frac{\beta_t^2}{2 \alpha_t \left(1 - \bar{\alpha}_t \right) \left\| \Sigma_\theta \right\|_2^2} \left\| \boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta \left(\sqrt{\bar{a}_t} \mathrm{x}_0 + \sqrt{1 - \bar{a}_t} \boldsymbol{\epsilon}, t \right) \right\|^2 \right] \end{aligned} \tag{2.11}$$

The authors of Ho et al. [14] conducted experiments in their article that led them to discover that using a simpler loss function (Equation 14 in [14]) could actually improve the performance of the model compared to utilizing the complete loss function shown in Equation 2.11:

$$L_{\text{simple}} \left(\theta \right) := \mathbb{E}_{t, \mathrm{x}_0, \boldsymbol{\epsilon}} \left[\left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta \left(\sqrt{\bar{\alpha}_t} \mathrm{x}_0 + \sqrt{1 - \bar{\alpha}_t \boldsymbol{\epsilon}}, t \right) \right\|^2 \right] \tag{2.12}$$

It is worth noting that in Ho et al. [14], the authors chose to train the network solely on the mean while keeping the variance constant. This was improved upon by Nichol and Dhariwal [15], who allowed the network to also learn the covariance matrix Σ by modifying L_t^{simple}, resulting in better performance from the model.



Algorithm 1 Training

```
1: repeat
2:   $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3:   $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
4:   $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5:   Take gradient descent step on
         $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar\alpha_t}\mathbf{x}_0 + \sqrt{1-\bar\alpha_t}\boldsymbol{\epsilon}, t) \right\|^2$
6: until converged
```

Algorithm 2 Sampling

```
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: for $t = T, \dots, 1$ do
3:   $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4:   $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar\alpha_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: end for
6: return $\mathbf{x}_0$
```

Figure 2.5: Sudo code for the forward and backward process (sampling and training). Source: Ho et al. [14]

2.2.5 Architecture

Now that we have a mathematical understanding of how to train and improve the performance of our model using a loss function, the next step is to carefully consider the architecture of our neural network. In particular, we need to choose a network that is well-suited to the task of denoising images.

As a reminder, our goal is to predict the parameters of the noise in an image in order to remove it and produce a clear image. Since the input to our model is a noisy image and the output is a denoised image of the same size, we need to choose a neural network that is able to effectively learn this mapping.

One type of network that has been successful for image denoising is the U-net, introduced by Ronneberger et al. [20] in 2015. The U-net achieved state-of-the-art results for biomedical image segmentation at the time, and its architecture has proven effective for other image processing tasks as well.

The U-net is similar to an autoencoder, a type of network that learns to reconstruct its input data by compressing it through a bottleneck and then decompressing it. The U-net also has a bottleneck in the middle of the network, which helps it focus on the most important features in the input image. However, the U-net also includes additional features, such as skip connections, that allow it to more effectively preserve the spatial resolution of the input image during denoising.

In summary, the U-net is a neural network that is well-suited to the task of denoising images because of its ability to effectively learn the mapping from noisy inputs to denoised outputs and its ability to preserve the spatial resolution of the input image. While the U-net is just one example of a neural network that could be used for this task, it is a strong choice due to its proven effectiveness for image denoising and other image processing tasks.

As we can see from Figure 2.6, the U-net gets its name from its distinctive shape,

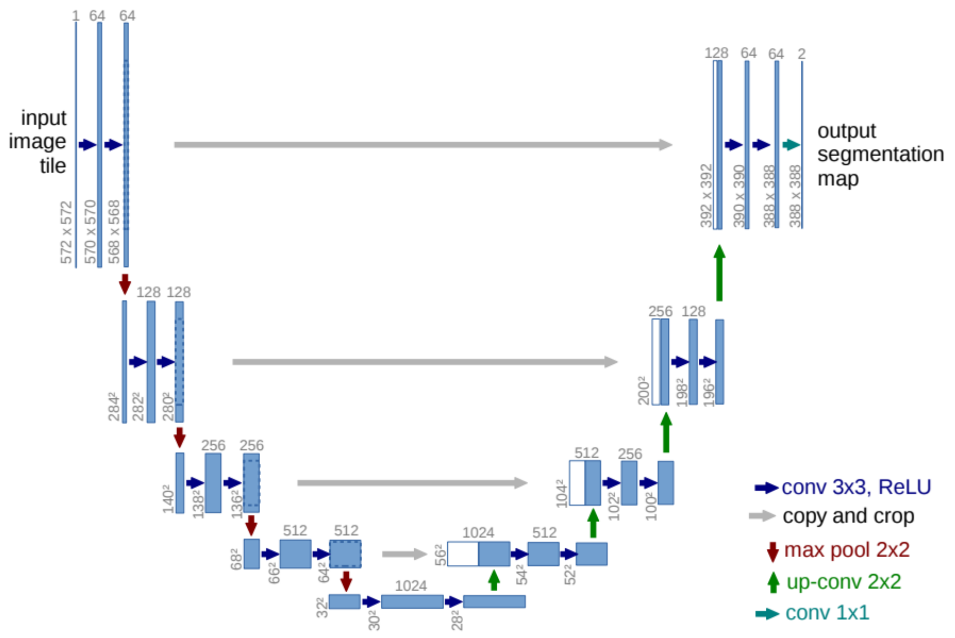


Figure 2.6: U-network design (example for 32x32 pixels in the lowest resolution). A multi-channel feature map correlates to each blue box. On the top of the box, there is a channel count indicator. At the lower left corner of the box, the x-y size is displayed. Copied feature maps are represented by white boxes. The various operations are shown by the arrows. Source: Ronneberger et al. [20]

which resembles the letter "U". The U-net takes an input image, typically with three color channels, as input. On the left half of the "U," there are normal 3×3 convolutions with the ReLu activation function and 2×2 maxpooling layers for down-sampling. On the right half of the "U," there are up-convolutions (also known as transposed convolutions, as discussed in Chapter 4 of Dumoulin and Visin [21]) for upsampling.

One key feature that sets the U-net apart is the use of skip connections, as illustrated by the gray lines crossing the "U." These skip connections help the U-net recover spatial information that may be lost during the down-sampling process and improve the gradient flow, which aids in the denoising process. This is achieved by concatenating the upsampled layers in the decoding part with the corresponding down-sampled layers in the encoding part, allowing the network to "retain" information before it is lost in the bottleneck.

During training, for each time step t in denoising equation 2.5, the U-net model processes every image in the batch and attempts to denoise them to the best of its ability, while also adjusting its weights during each training epoch. The original denoising diffusion probabilistic model implementation [14] includes Wide ResNet blocks, group normalization blocks, and self-attention blocks in the U-net. To represent the time step t, the authors use sinusoidal position embeddings, which allow the neural network to "know" at what specific time step (noise level) it is operating for each image in the batch, drawing inspiration from the Transformer [22]. For more information on the U-net and its various components, see Ronneberger et al. [20] and this helpful blog post from Huggingface [23].

To further illustrate the role of the U-net in the denoising process, let's consider the following example:

Imagine we have a noisy image and we want to use the U-net to denoise it. The image is fed into the left half of the U-net, where it undergoes convolution and down-sampling. This process helps the network identify and extract important features from the image, such as edges and shapes. However, as the image is down-sampled, some spatial information is lost.

The extracted features are then passed through the bottleneck of the U-net, which serves as a bottleneck of information. The bottleneck helps the network focus on the most important features and discard less relevant ones.

After passing through the bottleneck, the features are passed to the right half of the U-net, where they are upsampled using up-convolutions. At this point, the U-net makes use of skip connections to "recover" the spatial information that was lost during down-sampling. The skip connections do this by concatenating the upsampled layers with the corresponding down-sampled layers in the encoding part of the network.

Finally, the denoised image is produced by the U-net and can be compared to the original, noisy image using a loss function. This comparison is used to adjust the weights of the network and improve its performance during training.

Overall, the U-net is an effective neural network architecture for image denoising due to its ability to identify and extract important features from the input image, its use of a bottleneck to focus on the most relevant information, and its use of skip connections to recover lost spatial information.

2.3 Guided diffusion

Up until this point, our model has been able to denoise images based solely on the training images. However, there is another way to incorporate user input into the denoising process, known as guided diffusion.

Guided diffusion allows the user to provide guidance on the content of the generated images by incorporating embeddings of images or text into the diffusion process. These embeddings, which take the form of vectors, allow the model to learn from both the embeddings and the images. To apply a condition to our diffusion model p_θ, we add extra information y at each step of the t steps in the diffusion process:

$$p_\theta\left(\mathrm{x}_{0:T} \mid y\right) = p_\theta\left(\mathrm{x}_T\right) \prod_{t=1}^{T} p_\theta\left(\mathrm{x}_{t-1} \mid \mathrm{x}_t, y\right) \tag{2.13}$$

By applying the embedding at each time step (rather than just the first), we ensure that the model is constantly adjusted in the correct direction, even if the results drift. Using text embeddings as input is a good example of this type of conditional diffusion technique. As shown in Figure 2.7, this approach can produce highly creative results when using user-defined text prompts as input to a conditional diffusion model.

It's important to note that the choice of embedding method (image or text) and the specific implementation of the guided diffusion process will depend on the specific task and the resources available. In general, however, guided diffusion can be a powerful tool for allowing the user to exert more control over the output of the model and to produce more targeted, relevant results.

The objective of guided diffusion is to learn the gradient of the log posterior distribution of xt given y, denoted as $\nabla \log p\theta\left(\mathrm{x}_t \mid y\right)$. This can be achieved through the application of Bayes' rule:

$$\begin{aligned} \nabla_{\mathrm{x}_t} \log p_\theta\left(\mathrm{x}_t \mid y\right) &= \nabla_{\mathrm{x}_t} \log\left(\frac{p_\theta\left(y \mid \mathrm{x}_t\right) p_\theta\left(\mathrm{x}_t\right)}{p_\theta(y)}\right) \\ &= \nabla_{\mathrm{x}_t} \log p_\theta\left(\mathrm{x}_t\right) + \nabla_{\mathrm{x}_t} \log\left(p_\theta\left(y \mid \mathrm{x}_t\right)\right) \end{aligned} \tag{2.14}$$

Note that the term $p_\theta(y)$ in the denominator can be discarded, as it is a constant with respect to x_t. This is because we are only interested in the loss from the images, rather than in relation to the conditional input y, as indicated by the gradient operator $\nabla \mathrm{x}_t$.

To adjust the influence of the conditional input on the guidance, we introduce a scaling factor s:

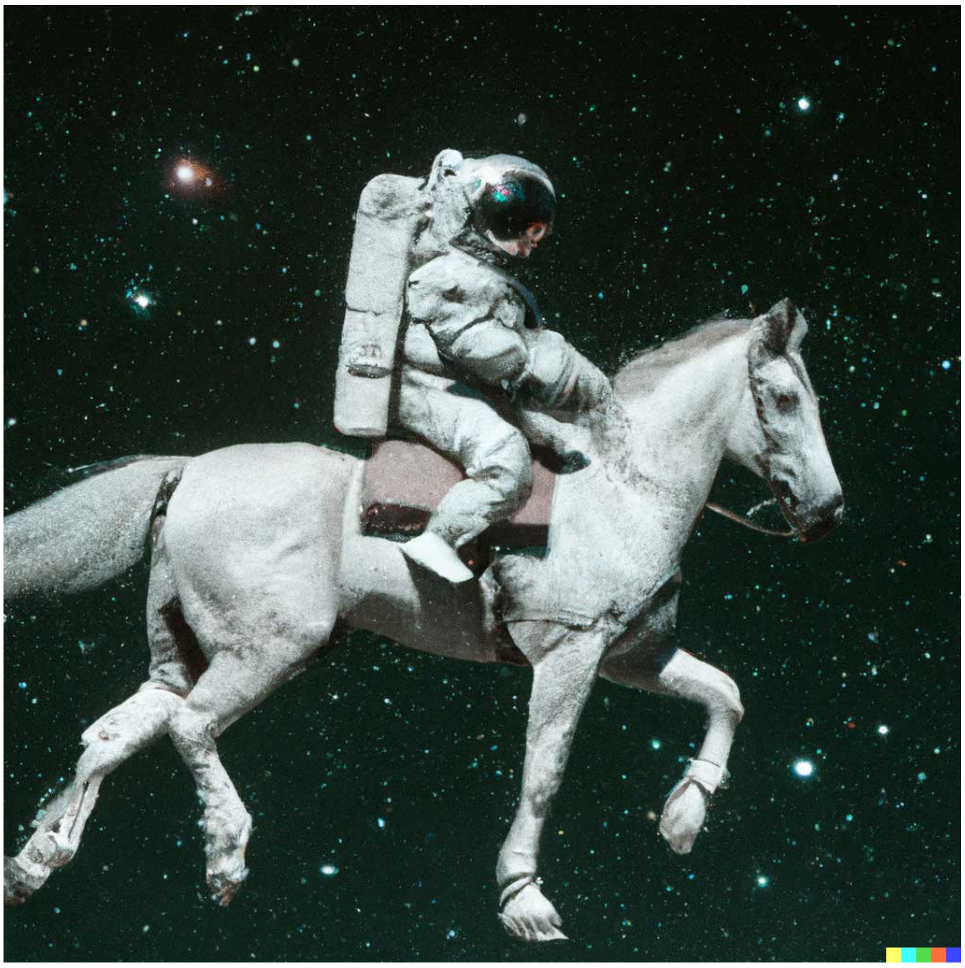


Figure 2.7: Showcase image created with Dalle 2 using the prompt: *An astronaut riding a horse in a photorealistic style*. Source: [4]

$$\nabla \log p_\theta\left(\mathrm{x}_t \mid y\right)=\nabla \log p_\theta\left(\mathrm{x}_t\right)+s \cdot \nabla \log \left(p_\theta\left(y \mid \mathrm{x}_t\right)\right) \tag{2.15}$$

The scaling factor s allows for fine-tuning the degree to which the guidance provided by the conditional input y affects the optimization of the log posterior distribution of x_t. This can be particularly useful in cases where the strength of the guidance signal may vary, or when it is desirable to adjust the balance between the guidance and the inherent structure of the data represented by $p\theta(\mathrm{x}_t)$.

The value of s can be determined through a variety of methods, such as empirical experimentation or theoretical analysis. In general, a larger value of s will result in a stronger influence of the guidance signal on the optimization process, while a smaller value will allow the inherent structure of the data to play a more prominent role.

It is important to note that the use of a scaling factor does not necessarily imply that the guidance signal is necessarily weaker or less reliable than the data. Rather, it allows for flexibility in balancing the influence of the two sources of information in the optimization process.

In the following subsections (2.3.1 and 2.3.2), we will describe two different methods for implementing guided diffusion.

2.3.1 Classifier guidance

A method to guide the diffusion process is to utilize a classifier, denoted as $f_\phi\left(y \mid \mathrm{x}_t, t\right)$, to direct the diffusion towards a desired class y during training, as proposed by Sohl-Dickstein et al [1]. and later implemented by Dhariwal and Nichol [15]. This can be achieved by training the classifier on the noisy image xt, and using the resulting gradients to guide the diffusion process. To do this, we may consider constructing a class-conditional diffusion model, with mean $\mu\theta(\mathrm{x}t \mid y)$ and variance $\Sigma_\theta(\mathrm{x}_t \mid y)$.

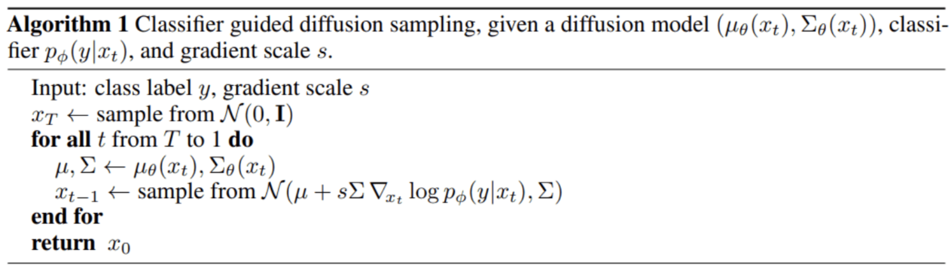
Since the probability distribution p_θ follows a normal distribution $\mathcal{N}(\mu_\theta, \Sigma_\theta)$, it can be shown from equation 2.15 that the mean of the distribution is perturbed by the gradients of $\log f_\phi(y \mid \mathrm{x}_t)$ for class y. This results in:

$$\hat{\mu}\left(\mathrm{x}_t \mid y\right) = \mu_\theta\left(\mathrm{x}_t \mid y\right) + s \cdot \Sigma_\theta\left(\mathrm{x}_t \mid y\right) \nabla_{\mathrm{x}_t} \log f_\phi\left(y \mid \mathrm{x}_t, t\right) \tag{2.16}$$

In the work of Nichol et al. [24], this concept was further developed by incorporating CLIP (Contrastive Language-Image Pre-training) embeddings [18] to guide the diffusion process. CLIP, as introduced by Saharia et al. [25], consists of an image encoder g and a text encoder h. These encoders produce the text embedding $h(c)$ and image embedding $g(\mathrm{x}_t)$, where c is a text caption. To perturb the gradients, the dot product of these embeddings can be taken:

$$\hat{\mu}\left(\mathrm{x}_t \mid c\right) = \mu\left(\mathrm{x}_t \mid c\right) + s \cdot \Sigma_\theta\left(\mathrm{x}_t \mid c\right) \nabla_{\mathrm{x}_t} g\left(\mathrm{x}_t\right) \cdot h(c) \tag{2.17}$$

This produced a way to correct and guide the diffusion process towards the user defined text input. See figure 2.8 for code example.


Algorithm 1 Classifier guided diffusion sampling, given a diffusion model $(\mu_\theta(x_t), \Sigma_\theta(x_t))$, classifier $p_\phi(y|x_t)$, and gradient scale s.

```
Input: class label $y$, gradient scale $s$
$x_T \leftarrow$ sample from $\mathcal{N}(0, \mathbf{I})$
for all $t$ from $T$ to 1 do
    $\mu, \Sigma \leftarrow \mu_\theta(x_t), \Sigma_\theta(x_t)$
    $x_{t-1} \leftarrow$ sample from $\mathcal{N}(\mu + s\Sigma \nabla_{x_t} \log p_\phi(y|x_t), \Sigma)$
end for
return $x_0$
```

Figure 2.8: Sudo code of algorithm for classifier guided diffusion sampling. Source: Dhariwal and Nichol [10]

2.3.2 Classifier-free guidance

Yet again we can use equation 2.15 to define a classifier-free guided diffusion model:

$$\nabla \log p\left(\mathrm{x}_t \mid y\right) = s \cdot \nabla \log\left(p\left(\mathrm{x}_t \mid y\right)\right) + (1 - s) \cdot \nabla \log p\left(\mathrm{x}_t\right) \tag{2.18}$$

The difference between classifier guidance and classifier free guidance is achieving guidance without the use of a second classifier model. This was first proposed by Ho and Salimans [26]. The authors combined the training of an unconditional model $\epsilon_\theta(x_t \mid y)$ with a conditional diffusion model $\epsilon_\theta(x_t \mid 0)$, and rather than training two separate classifiers, they really employ the same neural network. They randomly change the class y during training to 0, exposing the model to both the conditional and unconditional setup:

$$\begin{aligned} \hat{\boldsymbol{\epsilon}}_\theta\left(\mathrm{x}_t \mid y\right) &= s \cdot \boldsymbol{\epsilon}_\theta\left(\mathrm{x}_t \mid y\right) + (1 - s) \cdot \boldsymbol{\epsilon}_\theta\left(\mathrm{x}_t \mid 0\right) \\ &= \boldsymbol{\epsilon}_\theta\left(\mathrm{x}_t \mid 0\right) + s \cdot \left(\boldsymbol{\epsilon}_\theta\left(\mathrm{x}_t \mid y\right) - \boldsymbol{\epsilon}_\theta\left(\mathrm{x}_t \mid 0\right)\right) \end{aligned} \tag{2.19}$$

There are a couple of advantages using classifier-free over classifier diffusion:

- There only one model involved in the guided diffusion process
- When guiding on data that is challenging for a classifier to anticipate, it makes things simpler, for example with text embeddings.

Saharia et al. [25] proposal for Imagen [3] significantly depends on classifier-free guidance since they found that it is essential to producing samples with good image and text correlation.

2.4 Stable diffusion: Latent diffusion models

The application of diffusion models in image processing requires significant computational resources due to the direct manipulation of pixel data and the high-dimensional nature of RGB images. These models often involve repeated evaluations of functions and computation of gradients, which can be computationally demanding, requiring hundreds of GPU days in some cases [10]. Moreover, the inference process, which involves the creation of $50,000$ samples via repeated evaluations on noisy versions of the input space, can be time-consuming, taking approximately 5 days to complete on a single A100 GPU [10]. These challenges present two main problems: (1) the high computational requirements and associated carbon footprint of training such models, and (2) the time and memory demands of evaluating a trained model.

How do we go about resolving this problem?

By altering the models to operate on the latent space, it is possible to drastically reduce the number of parameters required in calculation. This can be accomplised by the use of autoencoders. These models where by the authors Rombach et al. [2] named LDMs or Latent diffusion models (also called stable diffusion).

To address the challenges presented by the computational demands and time-consuming nature of diffusion models in image processing, one potential solution is to modify these models to operate in the latent space through the use of autoencoders. These modified models, referred to as latent diffusion models (LDMs) or stable diffusion models by the authors Rombach et al. [2], can significantly reduce the number of parameters required in calculation, as illustrated in Figure 2.9. In this figure, before the use of the U-net structure, the images goes through an encoder $\mathcal{E}$ which brings them to the latent space, making it so that the number of parameters is reduced. When the U-net has done its work on the latent image and possible an additional embedding, the latents are brought back up to the pixel space with the use of the decoder $\mathcal{D}$. If we represent the simplified loss function for a typical diffusion model as equation 2.20 (from Rombach et al. [2]);

$$L_{DM} = \mathbb{E}_{x,\epsilon\sim\mathcal{N}(0,1),t}\left[||\epsilon - \epsilon_\theta\left(x_t, t\right)||_2^2\right] \tag{2.20}$$

the loss function for the latent diffusion model (LDM) can be represented as:

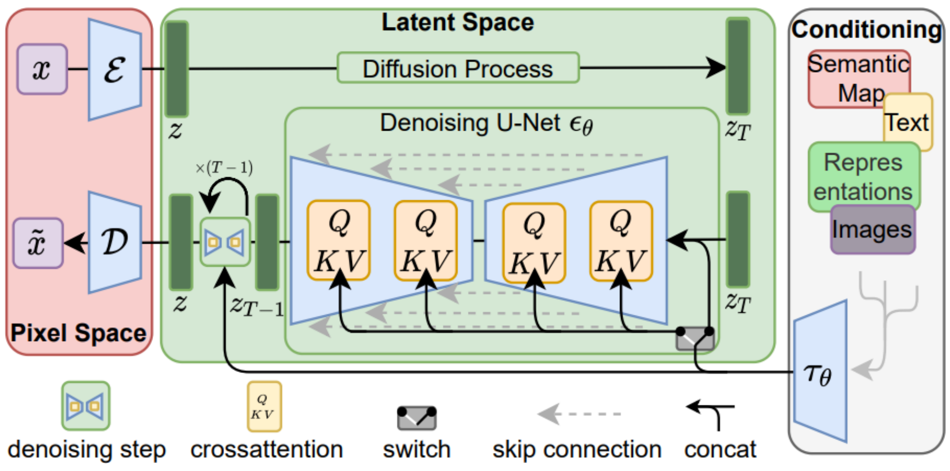


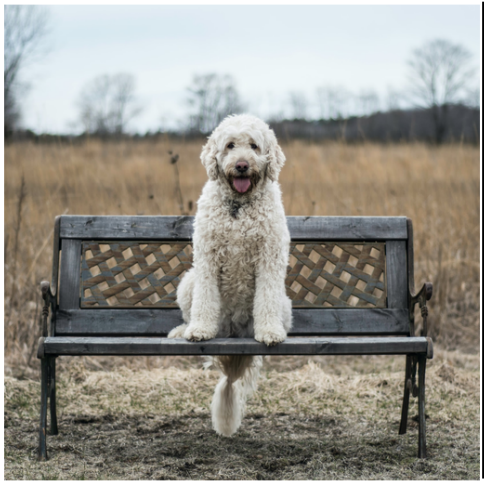
Figure 2.9: Illustration of the different machine learning components making up the LDM algorithm. Source: Rombach et al. [2]

$$L_{LDM} = \mathbb{E}_{\mathcal{E}(\mathrm{x}),t,\epsilon}\left[\|\epsilon - \epsilon_\theta\left(\mathrm{z}_t, t\right)\|^2\right] \tag{2.21}$$

As was previously mentioned, the incorporation of the use of latent space, thanks to Rombach et al. [2], was a significant step toward lowering the computational cost and making diffusion models efficient enough for the majority of people to use without the requirement of the most powerful GPU's. Latent diffusion models is also the models used throughout this master project.

2.4.1 Image inpainting

Image inpainting is a computer vision technique that involves filling in missing or damaged areas of an image with semantically and aesthetically plausible content. It has a wide range of applications, including object removal, image compositing, and photo restoration. The process of image inpainting can be difficult, particularly when large areas need to be filled, and requires strong generation skills. In recent times, image inpainting has also been incorporated into diffusion models, as the transition from generating an entire image to generating only a part of an image is not far off. However, changing the algorithms objective from image generation to inpainting demands new training cycles and new weights but the main fundamentals of diffusion models algorithm remains the same. The following example images are taken from an example of the inpainting algorithm using the diffusers library, which was published as a Google Colab page [27]. These images demonstrate the effectiveness of the inpainting algorithm in filling in "missing" parts of an image.


(a) Image of dog to be inpainted.

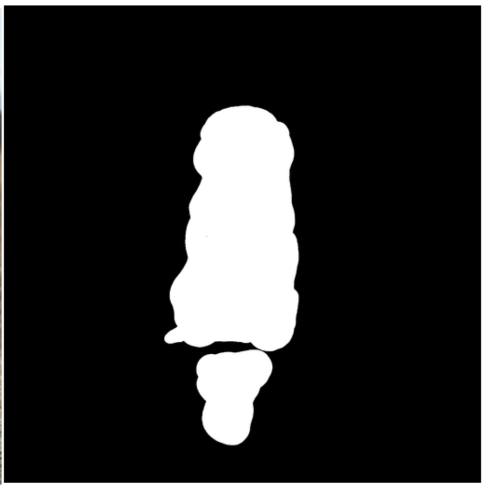
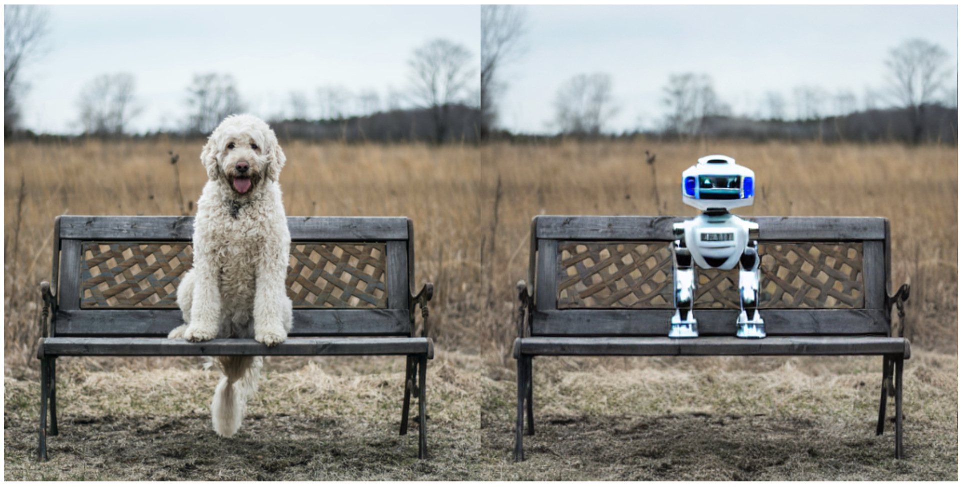

(b) The mask used in combination with the dog image

Figure 2.10: Example of image and mask ready to be inpainted. Source: [27]


(a) Example inpainting a dog to a robot

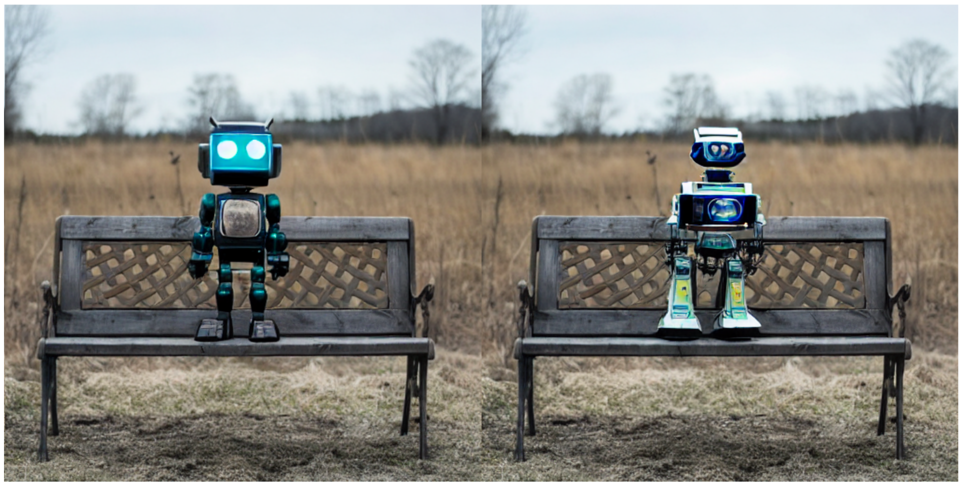

(b) A couple of more examples of the dog inpainted to a robot

Figure 2.11: Example of inpainted dog

In the field of image inpainting, there has been a continuous evolution of methods and enhancements over the years, beginning with the work of Sohl-Dickstein et al. [1] and continuing up to the present day papers like Rombach et al. [2]. In the initial paper on the topic, Sohl-Dickstein et al. [1] (section 2.5) proposed the use of multiple distributions and posterior computation to perform inpainting using a diffusion probabilistic model trained on specific images. Specifically, this approach involves sampling from the posterior distribution over the missing region of the image, given the rest of the image as context. However, this is just one of the ways in which diffusion models can be applied for image inpainting.

Subsequent papers, such as Lugmayr et al. [28], have introduced alternative methods for utilizing diffusion models for this purpose. One such approach involves intelligently combining known and unknown pixels to generate the missing pixels, given the context of the known pixels as illustrated in figure 2.12.

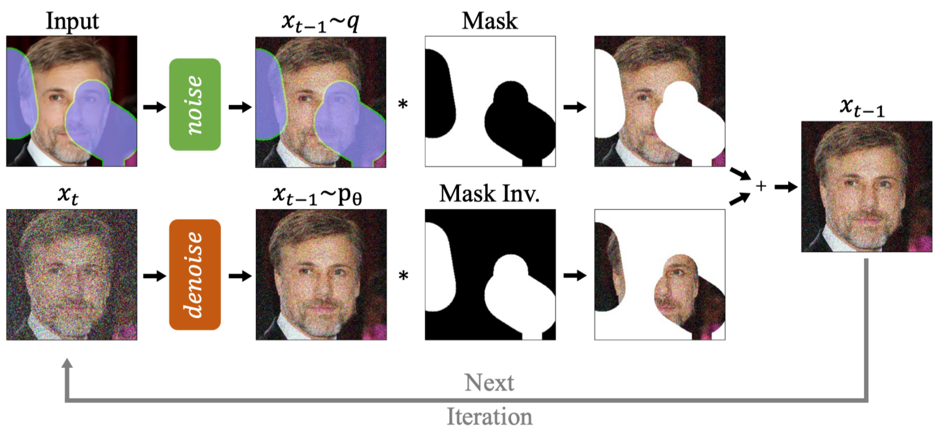


Figure 2.12: The RePaint method modifies the conventional denoising process by incorporating conditioning on the existing image content. Specifically, in each step of the method, known regions are sampled from the input image, while the inpainted regions are obtained from the output of a DDPM (deep generative models using diffusion probabilistic process). Source: Lugmayr et al. [28]

Diffusion models and inpainting algorithms have many applications, particularly in healthcare. One of the more interesting applications is the use of inpainting on medical data. A recent paper by Rouzrokh et al. [29] presented a proof-of-concept diffusion model that can be used for multitasking brain tumor inpainting on multi-sequential brain MRI studies. The model was developed to receive a 2D axial slice from different MRI sequences and inpaint a user-defined cropped area of the slice with realistic and controllable images of high-grade gliomas or tumor-less brain tissues. With this model, researchers can edit synthetic tumoral or tumor-less tissues on brain MRI slices, which is particularly valuable given the limited data on brain tumors. To achieve this, the authors trained a diffusion model that can transform an input image of an axial brain MRI slice with random noise into a synthetic but realistic image with user-specified attributes through 1000 steps of denoising, conditioned on the input regions of interest to the model. See figure 2.13

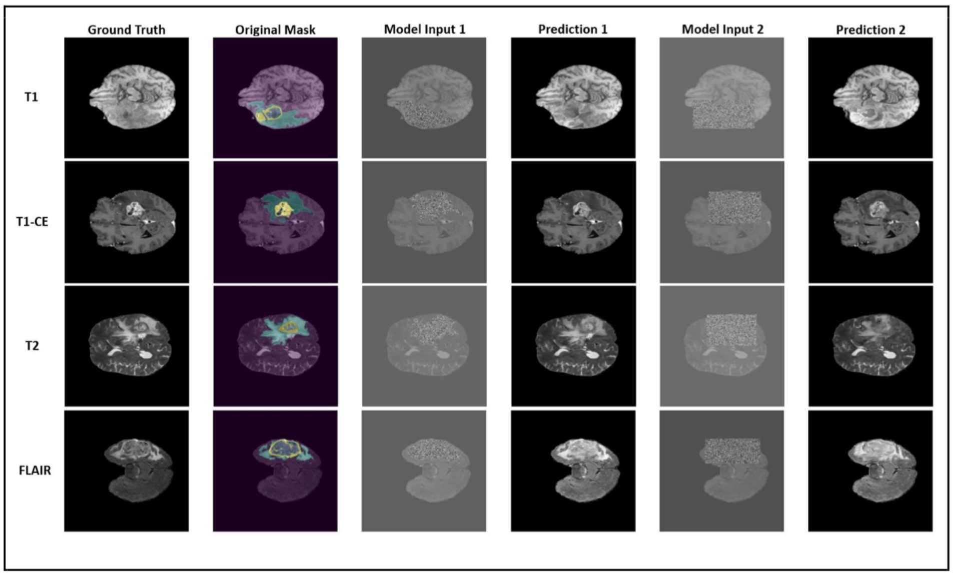


Figure 2.13: "*Generating tumoral lesions with predetermined regions of interest (ROIs) for necrotic tumor core, tumoral edema, and tumoral enhancement. In each instance, prediction 1 was done with a free-form input ROI and prediction 2 was done with a bounding box input ROI.*" Source: Rouzrokh et al. [29]

3 Method

Figure 3.1: Image created with the "StableDiffusionPipeline" from huggingfaces diffusers package [30]. The text prompt used: "*a home for all the critters of the forest, big tree, tall, lush, calm, book cover, ultra realistic, 4k, 8k*"

The initial goal of this project was to evaluate the potential of diffusion models for use in machine learning and deep learning applications, rather than to immediately apply them to a specific problem. To conduct experiments, it was necessary to implement a stable diffusion model on a local computer. After considering several options, the diffusers package [30] provided by HuggingFace was selected due to its widespread use and reputation within the machine learning community. HuggingFace is a community of machine learning developers and enthusiasts committed to advancing and making machine learning accessible to all. The diffusers package can be easily installed using the following command:

```
pip install --upgrade diffusers transformers accelerate
```

However, a significant obstacle arose when it became apparent that the GPU in my personal laptop, a MacBook Pro released in late 2019, was not compatible with CUDA, a parallel computing platform and programming model developed by NVIDIA for generic computing on GPUs. While it is possible to run diffusion models on a CPU, the process is much slower and can result in overheating of the chip over time. As a solution, the use of a GPU cluster provided by the UiT machine learning group was implemented. This cluster, which can be accessed via the Secure Socket Shell (SSH) network protocol in combination with vpn, allowed for the efficient execution of the diffusion models despite the lack of a compatible GPU on my personal laptop. It is worth noting that recent advances have resulted in the creation of implementations for Apple M1 chips, which perform well in this context.

3.0.1 Syncing between computer and cluster

When executing code on the cluster, it was necessary to transfer all the necessary files required for the specific code to be run, to the cluster's personal storage. This is where the Secure Shell (ssh) protocol came into play. The Secure Copy Protocol (scp) was utilized to transfer files with ssh. By simply executing the following command in the terminal:

```
scp /path/to/file username@a:/path/to/destination
```

This command would copy a file from the local system to the desired destination connected via ssh. However, in cases where it was necessary to copy files from the cluster to the local system, such as generated plots, "npy" or "pkl" files, or other files generated by the cluster, it was a simple matter of reversing the direction of the command. This is illustrated by the following command:

```
scp username@b:/path/to/file /path/to/destination
```

A problem that arose frequently was that in the use cases where the cluster generated a huge amount of files, it was necessary to copy each and every file individually, which proved to be a tedious task. In order to overcome this problem, an alternative solution was sought after. One such solution is the utilization of the fast and versatile file copying/syncing tool, rsync [31]. Rsync is a commonly used tool in Linux and OSX operating systems, it can be used to synchronize files between different systems by implementing an algorithm that minimizes the amount of data copied by only transferring the differences in data between the last time the sync command was executed and the current execution of the command.

This proved to be a tremendous relief as it was now incredibly easy to transfer entire folders of files between the cluster and the local system with one command line:

```
rsync OPTION SourceDirectory_or_filePath
 ↪  user@serverIP_or_name:Target
```

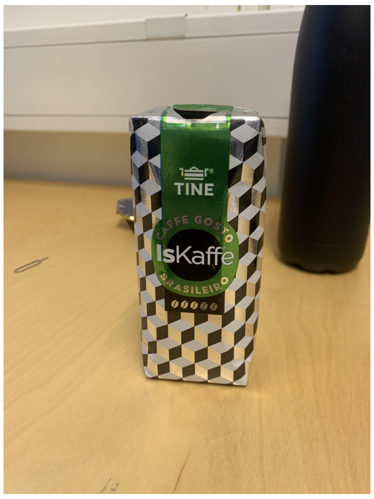
To further facilitate the transfer of files between the local system and the cluster, two ".sh" files were created, referred to as "sync_to_cluster" and "sync_from_cluster". These files allowed for even more rapid and effortless synchronization of all relevant files to or from the cluster and could easily be called upon from the terminal when necessary.

3.0.2 Utilizing Docker

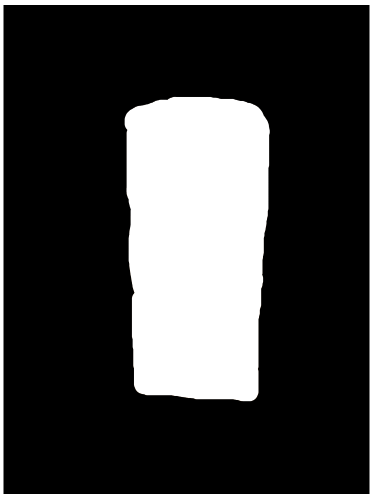
Docker [32] is an open-source platform that facilitates efficient and secure packaging and deployment of applications. This is achieved through the utilization of container-based virtualization technology, which isolates applications and their dependencies into self-contained units, making them easy to manage and deploy. This feature makes Docker an ideal tool for software development teams who aim to quickly and efficiently build and deploy applications. Additionally, Docker provides a high level of flexibility and scalability, allowing code to be tested and deployed on multiple platforms.

In the present context, the system employs the use of Docker images to run applications on the cluster. Specifically, in the context of running Stable Diffusion models on the cluster, various Python packages were required. To avoid the time-consuming process of having to install these packages each time the code was run, a custom Docker image was created. The Diffusers package offered by Hugging Face utilizes PyTorch as its base and PyTorch conveniently offers official Docker images that can be customized by the end user. Hence, a custom Docker image was created, and uploaded to Docker Hub, for the purpose of running this specific application on the cluster. This allowed for the cluster to download and utilize the image for this specific purpose.

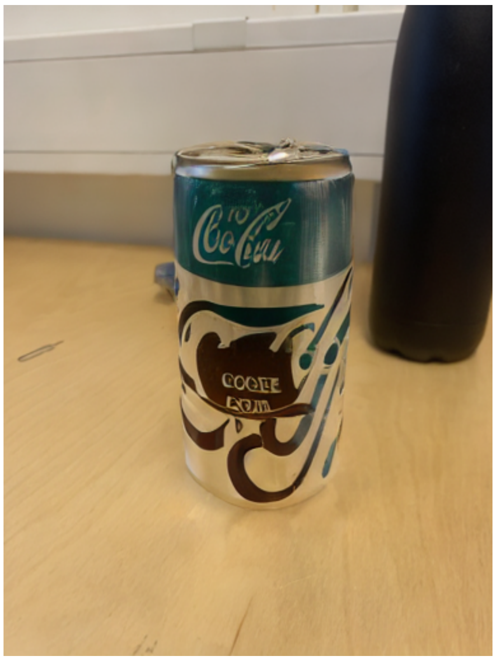
3.0.3 Inpainting



(a) Original image of a TINE IsKaffe Gosto Brasileiro



(b) Mask image given to the algorithm alongside the original image 3.2a

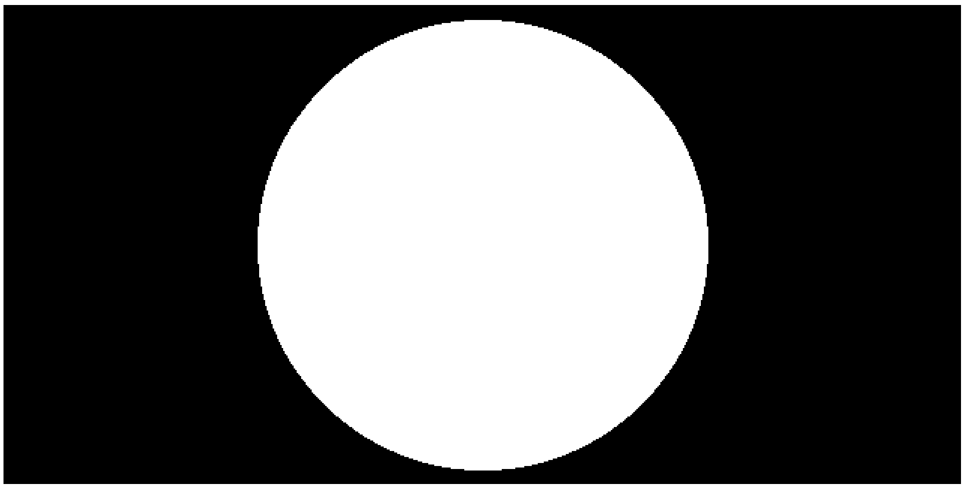


(c) Inpainting results

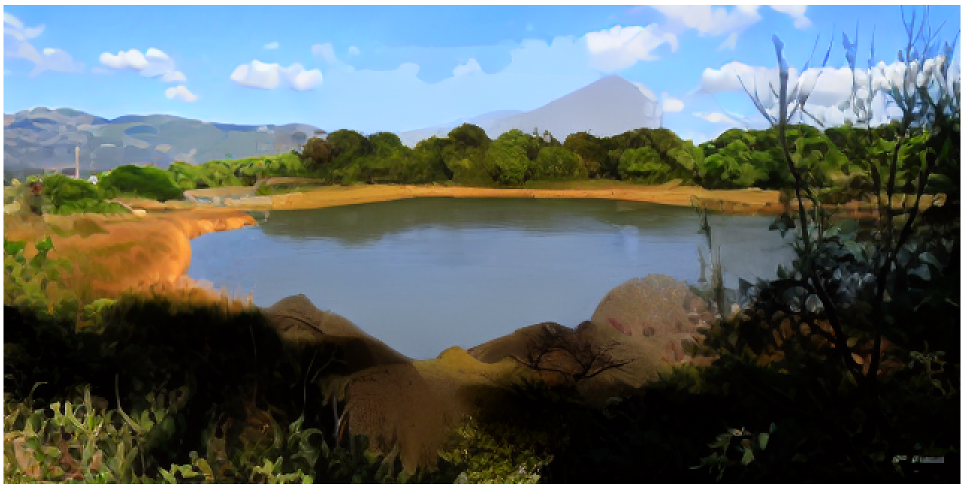
Figure 3.2: The first time while testing result yielded high quality, showcasing the capabilities of the technology. A strength of 0.6, guidance scale of 7, and text prompt "*cola can, red, standing on wooden table, 4k, 8k*" were used to restore the missing regions of the image. This result demonstrates the potential of this technology.

Upon using the inpainting method from the HuggingFace diffusers library, we observed that it contained a strength parameter which not only impacted the inpainting mask, where the diffusion model would perform inpainting, but also altered the pixels outside of the mask as can be seen in figure 3.3. This prompted us to investigate the effect of this parameter on the inpainting results. To do so, we conducted a series of experiments varying the strength parameter and analyzing the resulting inpainting performance. Our findings showed that adjusting the strength parameter had a significant influence on both the inpainting mask and the pixels outside of the mask, and that careful tuning of this parameter was necessary to achieve satisfactory results. These results were crucial in guiding the development of our inpainting methodology and ensuring the robustness of our approach.

(a) The original landscape images with nothing done to it. Source: [33]



(b) Mask image created with 94% of the smallest side as diameter of the circle.



(c) Inpainted mask area with strength parameter of 0.84 and no guidance scale (no text input from user)

Figure 3.3: An illustration of the inpainting process with the strength parameter set to 0.84 is presented. As demonstrated by a comparison of figures (a) and (c), it is evident that the pixels outside the image mask have undergone significant changes. This example represents one of the more pronounced instances of this phenomenon.

The higher the strength, the more intense the inpainting process will be as the number diffusion iterations increases. Conversely, the lower the strength, the less intense the inpainting process will be. This can be useful for fine-tuning the inpainting process to achieve the desired balance between image quality and inpainting intensity. Also, since inpainting requires a mask for the selected area which is to be inpainted, this could also be a varying factor when experimenting.

Despite the limited resources available on the theoretical and mathematical workings of the Hugging Face inpainting algorithm, I have made a conscious effort to gain a comprehensive understanding of the algorithm through studying the open-source code released on their GitHub page and referencing the papers by Ho et al. and Rombach et al. While inpainting is just a subset of the capabilities of the algorithm, it is important to recognize the significance of its application in the field of deep learning. Further research in this area, including in-depth explanations of the theoretical and mathematical workings of the algorithm, would greatly contribute to the evolution of diffusion models for inpainting.

Overall, the code provided by Hugging Face on their GitHub page is open for examination. However, due to the complexity and intricacies of the underlying algorithms and codebase, a thorough understanding of its inner workings may be challenging for those without advanced coding proficiency. Despite this, the inpainting pipeline utilized in this experiment can be analyzed and understood to a certain extent through examination of the provided code and documentation. Additionally, the results of this study provide valuable insights into the behavior of the inpainting algorithm, regardless of a complete understanding of its underlying codebase.

The following is an exact quote written as a comment in their inpainting pipeline [34] (line 502-506) about the strength parameter.

```
strength (‘float‘, *optional*, defaults to 0.8):
Conceptually, indicates how much to inpaint the masked
area. Must be between 0 and 1. When ‘strength‘ is 1, the
denoising process will be run on the masked area for the
full number of iterations specified in
‘num_inference_steps‘. ‘image‘ will be used as a reference
for the masked area, adding more noise to that region the
larger the ‘strength‘. If ‘strength‘ is 0, no inpainting
will occur.
```

The strength parameter is a continuous variable that ranges between 0 and 1 and plays a crucial role in the inpainting process. It governs the intensity of the inpainting applied to the masked regions of an image. Specifically, when the strength is set to 1, the algorithm will fill in the masked area as much as possible, using the denoising process for the full number of iterations specified in the other parameter $num\_inference\_steps$. The image is used as a reference for the masked area, thus the noise level in the masked area increases proportionally with the increase of strength. Conversely, a value of 0 for the strength parameter implies no inpainting is performed. This relationship between the strength parameter and the intensity of inpainting can be succinctly encapsulated mathematically as follows:

$$inpainting\_intensity = strength \times num\_inference\_steps$$

3.0.4 Experiments

First experiment: Varying strength with constant mask radius

The initial experiment was conducted on a dataset of 20 randomly selected landscape images sourced from Kaggle [33]. Each image underwent downsampling, specifically relative downsampling that adjusts the image size to be below a certain pixel threshold, while preserving the aspect ratio of the image.

This downsampling was necessary due to the limited amount of video random access memory (VRAM) available in the graphics processing unit (GPU) cluster provided. As access to command line commands that would reveal the exact hardware specifications was not possible, if an image with an excessive number of pixels was inputted, an out of memory error would be generated by PyTorch, indicating that approximately 10GB of VRAM was available.

As a result of experimentation, it was determined that through the utilization of various memory-saving options within the code, though at the expense of slower inference iterations, the maximum image size that could be processed without encountering an out-of-memory error was approximately $750 \times 750 = 562500$ pixels. However, it is worth noting that this value is not fixed and that the total number of pixels can be increased by adjusting the aspect ratio of the image such that the total number of pixels remains within close proximity to 562000.

For each image within the dataset of 20, a corresponding binary mask image was generated, where the masked regions were represented by white pixels and the non-masked regions by black pixels. This is a widely accepted standard for representing masks in the context of image inpainting, as depicted in Figure 3.2b and 3.3b. The masks for each image were created by utilizing approximately 80% of the length of the smallest side of the image, divided in half, as the radius for a circular white mask. This approach was chosen in order to maintain a consistent ratio of $\frac{\text{circle area}}{\text{image area}}$ across the 20 images, despite variations in image width and height, by ensuring that the circle never extends beyond the boundaries of the image.

For each image, a linearized inpainting strength between 0 and 1, consisting of 100 steps, was utilized to generate 100 inpainted images per original image. This resulted in a total of 2000 inpainted images, as depicted by the mask described above. This methodology allows for a comprehensive examination of the varying inpainting strength on the final result. The reason for maintaining a relatively large number of strength inputs per image was to investigate the potential correlation between the inpainting strength and the degree of change to the pixels outside the inpainting mask, which were subject to change. The task of generating a large number of inpainted images, as described in the previous statement, was quite time-consuming and labor-intensive for the computational hardware. Fortunately, the availability of a GPU cluster enabled a significant reduction in computation time, as each inpainting operation was completed in an average of 21 seconds. This resulted in a total inpainting time of close to 12 hours, with the time being dependent on the image size and the number of inference steps, where the number of inference steps increased proportionally with the strength parameter.

As a final step, we aimed to visualize the effectiveness of the inpainting algorithm on the pixels outside the masked area, by utilizing the Mean Squared Error (MSE) pixel-wise. In order to accurately calculate the change in pixels outside of the inpainting mask, it was necessary to ignore the inpainted area when comparing the pixels to the original image. To accomplish this, we replaced all inpainted pixels with white pixels in both the original image and the inpainted image, ensuring that the squared error within the inpainted area would not contribute to the overall error. This method has the potential to dilute the MSE due to the high number of zero errors introduced. To address this, a combination of replacing the inpainted pixels with white pixels and using Python code to identify and only apply the MSE to the relevant pixels was implemented. The results can be seen in section 4.2.

3.0.5 Second experiment: Varying radius for three different strengths

In the second experiment, we aimed to investigate the impact of varying the radius of the white circular mask on the degree of change in pixels outside the inpainted mask area. The same images as in experiment 1 3.0.4 was also utilized in this experiment. To accomplish this, 50 different mask sizes were utilized per image. The radius of the masks were linearly spaced between 20 percentage and up to 80 percentage of the smallest image side divided in half. This approach was implemented to prevent the mask from exceeding the boundaries of the image, as outlined in Section 3.0.4 of our methodology. Each of the 50 different mask sizes were then applied to three different strengths of 0.1, 0.5 and 0.9, resulting in a total of $20 \times 50 \times 3 = 3000$ inpainted images. This experimental design required a computational effort of approximately $\frac{3000}{2000} \times 12 = 18$ hours.

In order to visualize the results the same approach as in experiment 1 was used. Namely using MSE in order to compare the difference and making sure that the inpainted part not intervened with the MSE results by putting a white circle on the masked area on both the original and inpainted image. The results can be seen in this section 4.3.

3.0.6 Heatmap visualization

As we conducted our experiments and considered the reason for the inpainting algorithm's tendency to alter not only the pixels within the inpainted area, but also those surrounding it, we developed a hypothesis. The algorithm may introduce some additional noise to the entire image during certain inference steps in order to improve the fit of the inpainted area to the rest of the image.

Furthermore, if this hypothesis is true, we would expect to observe a higher level of change in the pixels near the border of the inpainting mask. To investigate this, we employed a heatmap of pixel change on the top three inpainted images displaying the highest degree of change. The results of this visual investigation can be found in the results section 4.2.3 and the code for Experiment 1, 2 and the heatmap visualisation can be seen at my **Github page** [35] in the file **inpainting_strength_testing.py**.

4 Results and discussion

The present study aimed to investigate the impact of inpainting strength and mask radius on the alteration of pixels outside the inpainting mask. A series of experiments were conducted in which images were inpainted with varying strengths and constant mask radiuses, as well as varying mask radiuses and constant strengths. The results of these experiments were analyzed using heatmap analysis to provide a visual representation of the alteration of pixels outside the inpainting mask. The results of these experiments and the subsequent heatmap analysis will be discussed in the following sections, providing insights into the behavior of the inpainting algorithm and the areas of the image that are most affected by the inpainting process.

4.1 Landscape images dataset

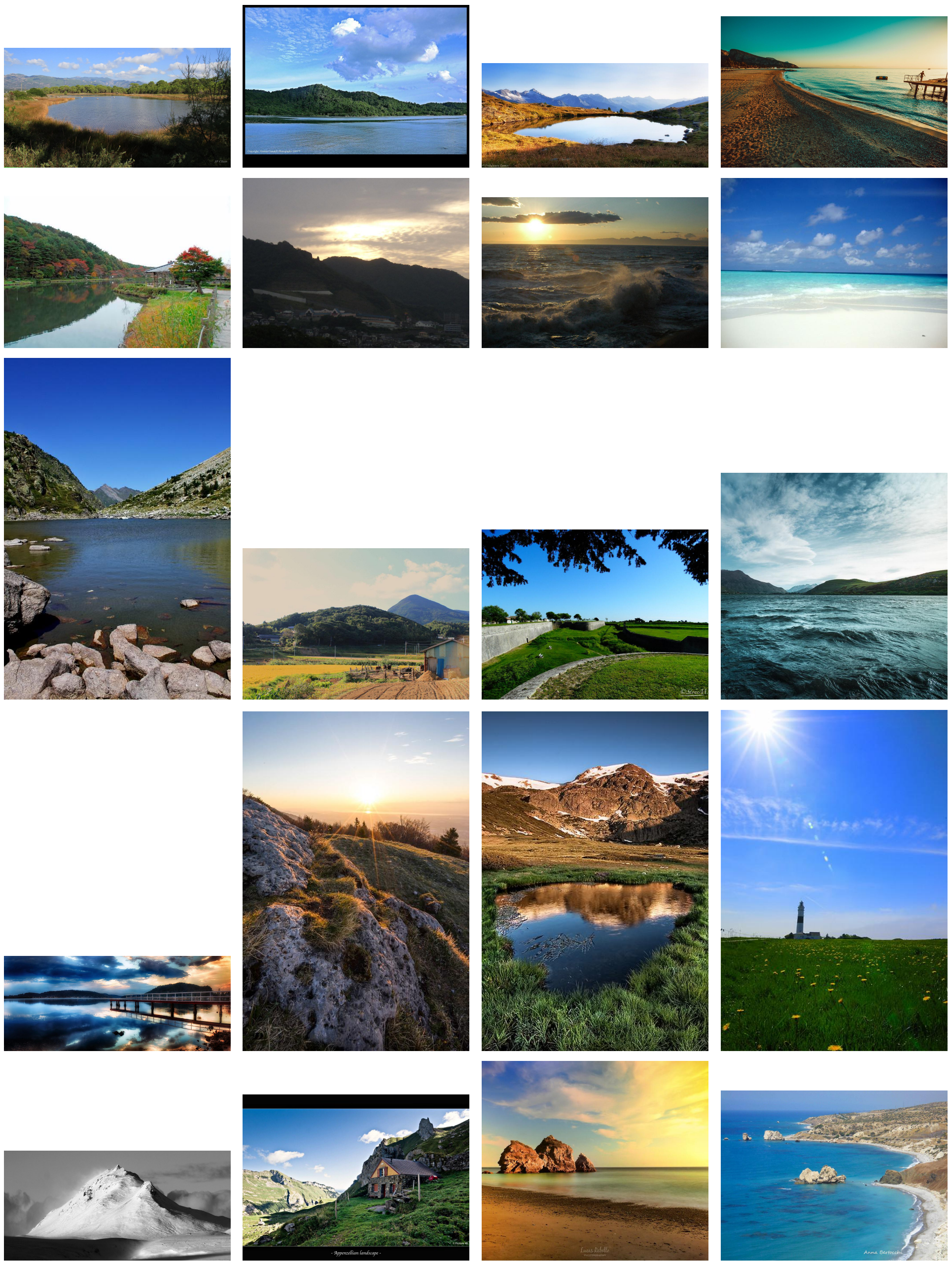
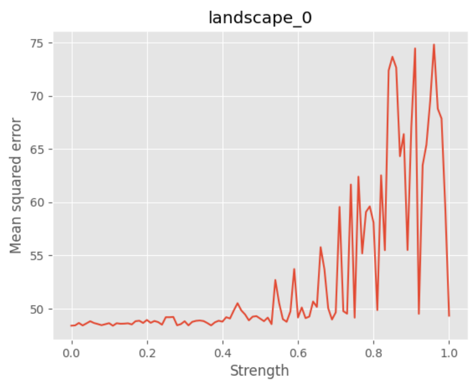
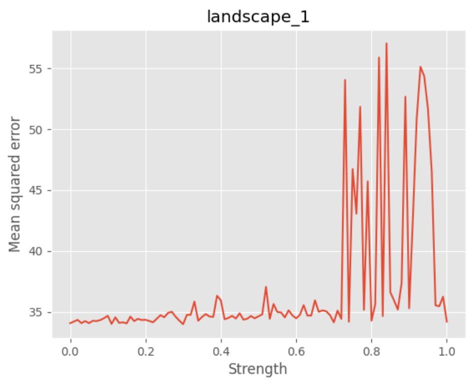


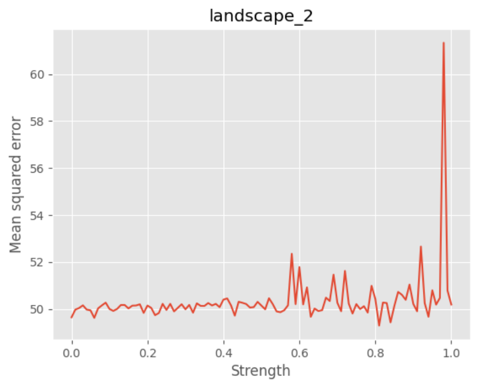
Figure 4.1: The current study utilized a set of randomly selected landscape images for the purpose of conducting strength variation testing. These images were chosen to provide a representative sample of diverse landscape scenarios. The image on the top left is ***Landscape image 0*** and the image number increases row wise to the right with ***Landscape image 1*** until the bottom right image which is ***Landscape image 19***.

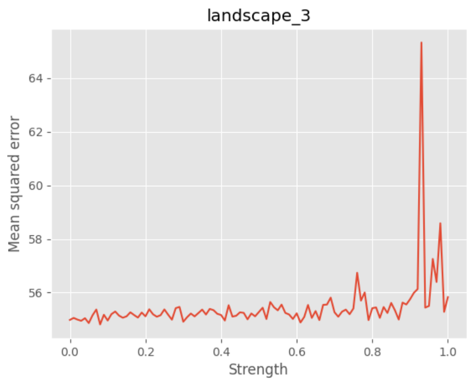
4.2 Pixelchange with respect to strength change

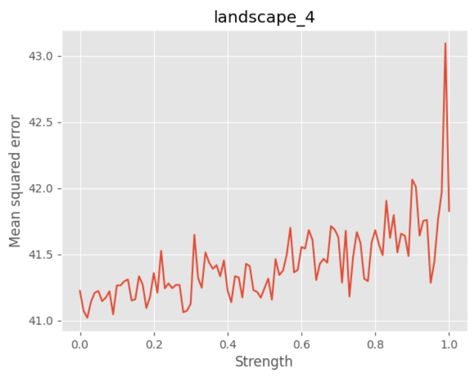
4.2.1 Plots

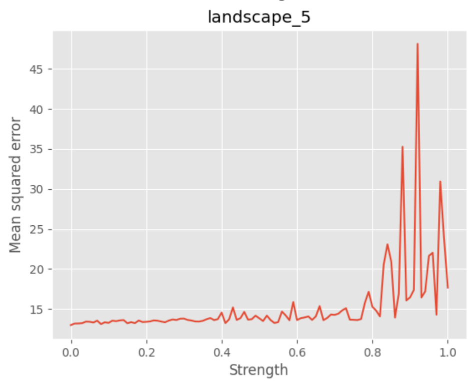


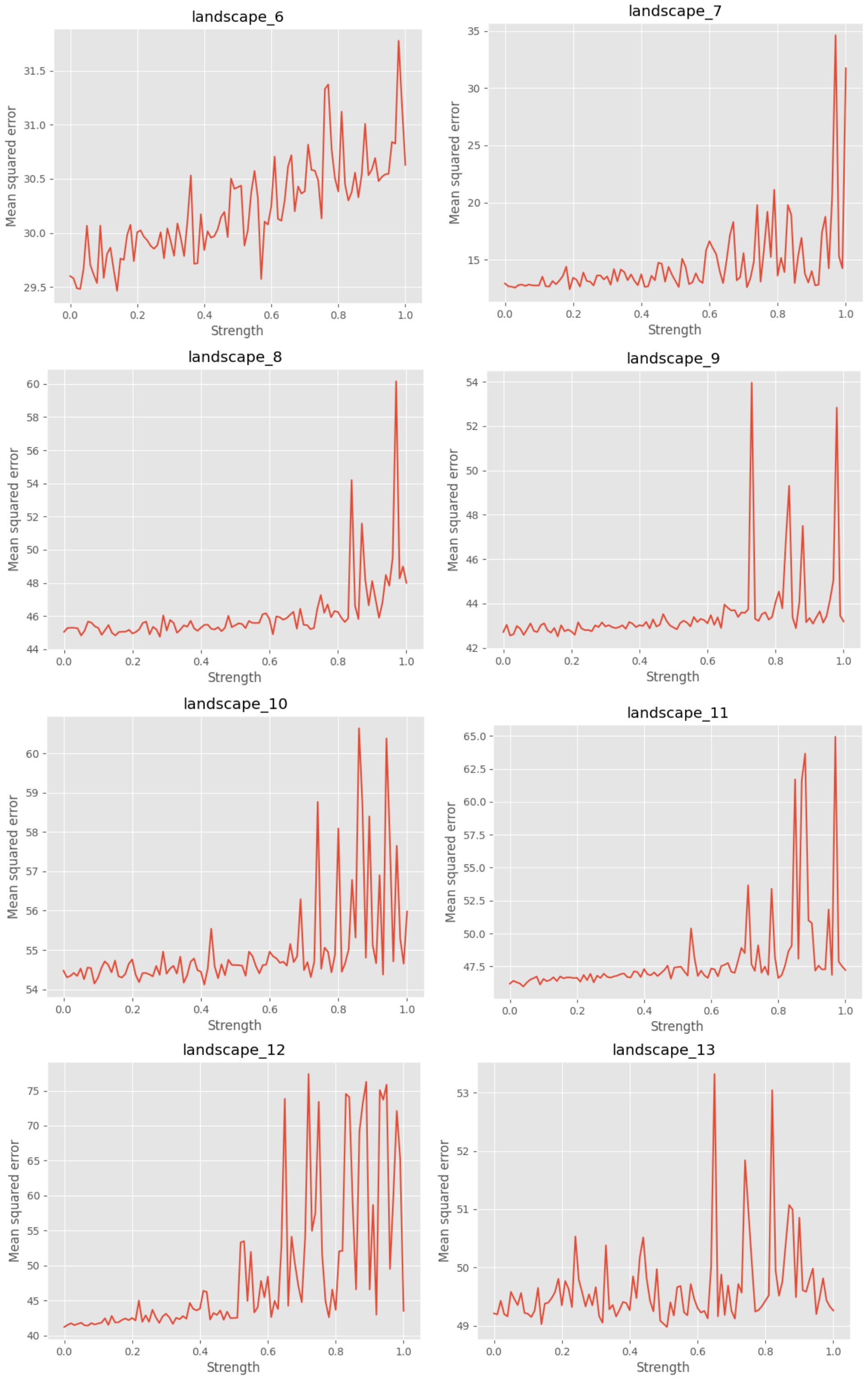











landscape_6
landscape_7
landscape_8
landscape_9
landscape_10
landscape_11
landscape_12
landscape_13
Mean squared error
Strength

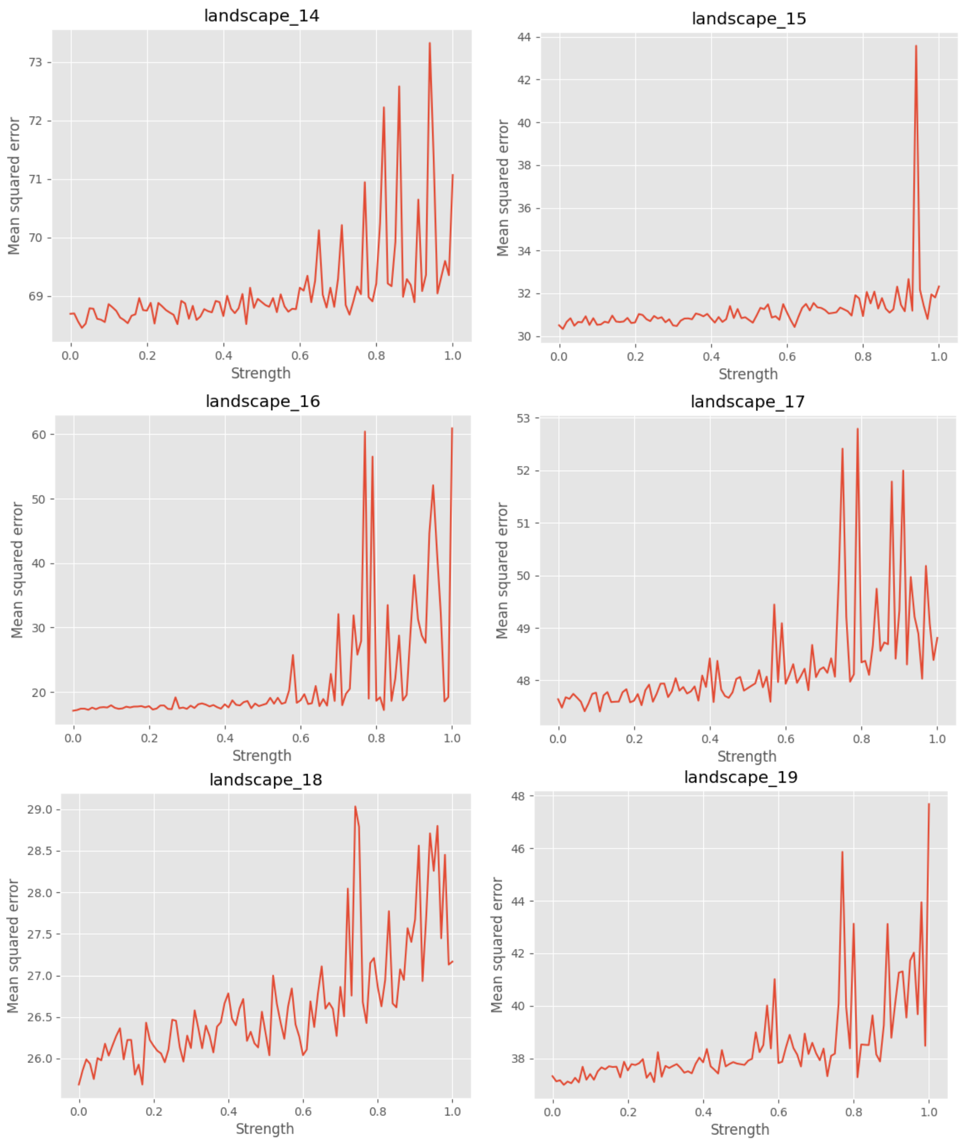


Figure 4.2: The present study conducted a comparison of the pixels outside the inpainting mask of the inpainted images with the original image prior to inpainting. The results indicated a notable trend towards higher mean-squared error (MSE) as the inpainting strength increases. These results were obtained using the set of landscape images depicted in Figure 4.1. A dynamic downscaling method was employed, ensuring that the largest side of each image remained below 650 pixels. Further details regarding the methodology employed in this experiment can be found in Section 3.0.4.

4.2.2 Discussion

The results of the present study, as illustrated in Figure 4.2, provide evidence that as inpainting strength increases, there is a corresponding increase in the mean squared error (MSE) of pixels located outside the inpainting mask. Additionally, the variance of the MSE also increases with increasing inpainting strength. The data also reveals the presence of high spikes in MSE at inpainting strengths of approximately 0.6 and above.

However, it is important to note that the rate of increase in MSE with respect to inpainting strength varies among the images. This variability is not immediately apparent from the plots, as the degree of the spikes for higher inpainting strengths varies for each image, resulting in a range of y-values that depends on the height of the highest spikes. For instance, by comparing the plots for Landscape_6 and Landscape_18, a clear trendline can be identified; however, in other images such as Landscape 1, 2, 4, 5, 8 and 9, the increasing trend is less discernible due to the high variance in the spikes, with MSE values ranging from 15 to 45 in Landscape_5, for instance.

While all of the images demonstrate some increase in MSE with increasing inpainting strength, it is evident that some images exhibit a smaller rate of increase in comparison to others. Further analysis is necessary to identify the specific characteristics of these images that contribute to their lower MSE, as well as to investigate the high spikes in MSE at inpainting strengths of 0.6 and above, as they may indicate a threshold at which inpainting errors become particularly pronounced.

Increasing trendline

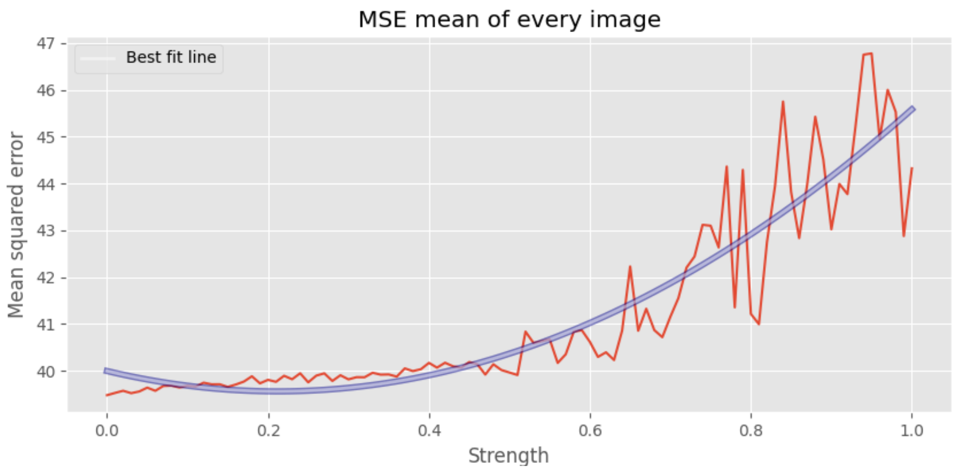


Figure 4.3: Mean of every plot seen in figure 4.2 with the addition of second degree polynomial trendline represented by the function $9.694x^2 - 4.105x + 40$

The mean of the MSE for each image, as a function of inpainting strength, is presented in Figure 4.3. To gain a deeper understanding of the relationship between inpainting strength and mean MSE, a polynomial regression analysis was conducted. The resulting best-fit equation, represented by the polynomial function $9.694x^2 - 4.105x + 40$, is superimposed on the data points in the figure, providing a clear illustration of the trend of the mean MSE as inpainting strength increases.

It is evident from the figure and the best-fit equation that there is a clear positive correlation between inpainting strength and the mean MSE, indicating an increase in the error outside the masked area as strength increases. This highlights the importance of considering the inpainting strength when utilizing this technique, as it has a direct impact on the quality of the inpainted result.

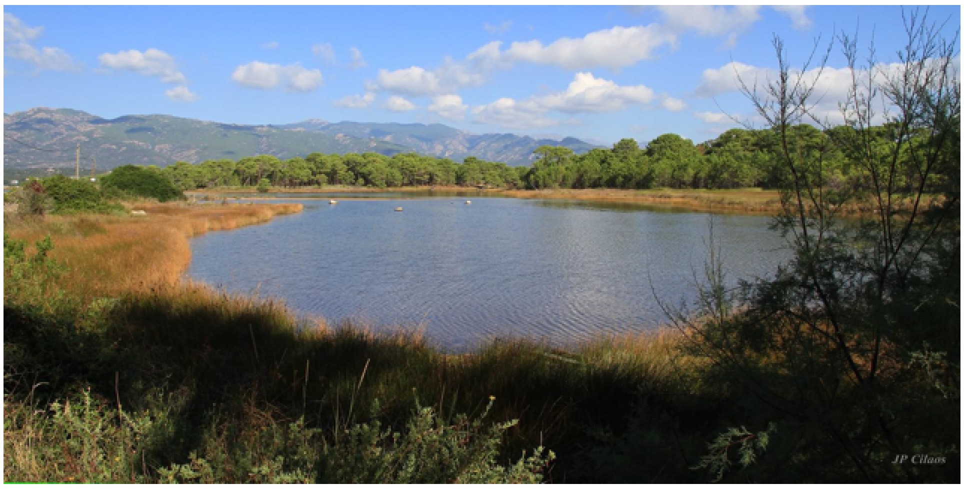
In conclusion, this experiment provides valuable insights into the relationship between inpainting strength and change of pixels outside inpainting mask in image inpainting using diffusion models. The results suggest that there is a clear trade-off between inpainting strength and MSE, and that certain images may be more resilient to inpainting errors. Further research could focus on identifying the specific characteristics of these images and understanding the cause of the high spikes in MSE at high inpainting strengths.

4.2.3 Heatmap visualisation

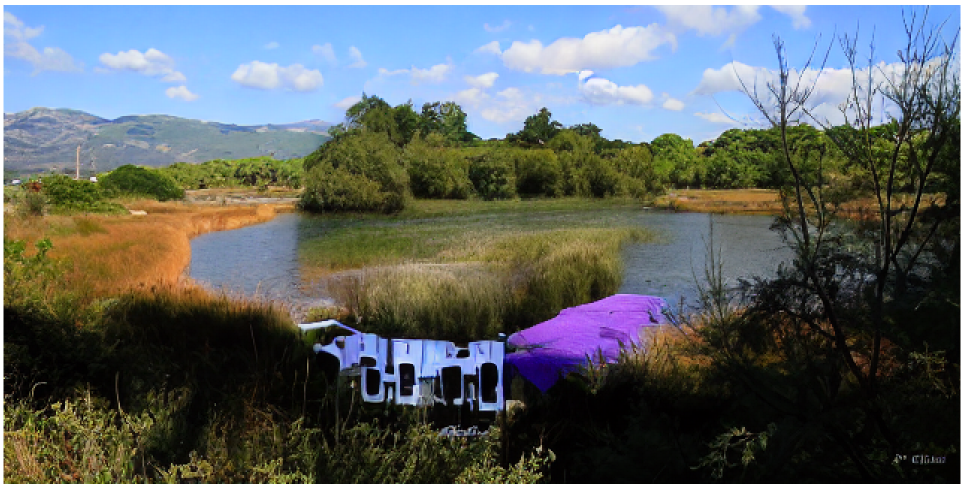
This section will focus on the examination of the pixel-wise distribution of the mean squared error (MSE) for the three inpainting strengths that correspond to the highest MSE values, as observed in the previous analysis. To gain a more detailed understanding of the location and distribution of errors, heatmaps of the pixel-wise MSE were generated for each of these inpainting strengths. These heatmaps provide a visual representation of the MSE across the entire inpainted image, allowing for a more comprehensive assessment of the areas where the change in MSE is most prominent.

In particular, this analysis aims to investigate whether there are any differences in the location of the highest MSE values around the inpainting mask border, as this area is known to be particularly sensitive to inpainting errors. The heatmaps will be analyzed to evaluate the degree of variation in the distribution of the MSE across the images and to identify any patterns or trends in the location of errors. This analysis will provide further insights into the underlying mechanisms that influence the quality of the inpainting results and may inform the development of improved inpainting methods.

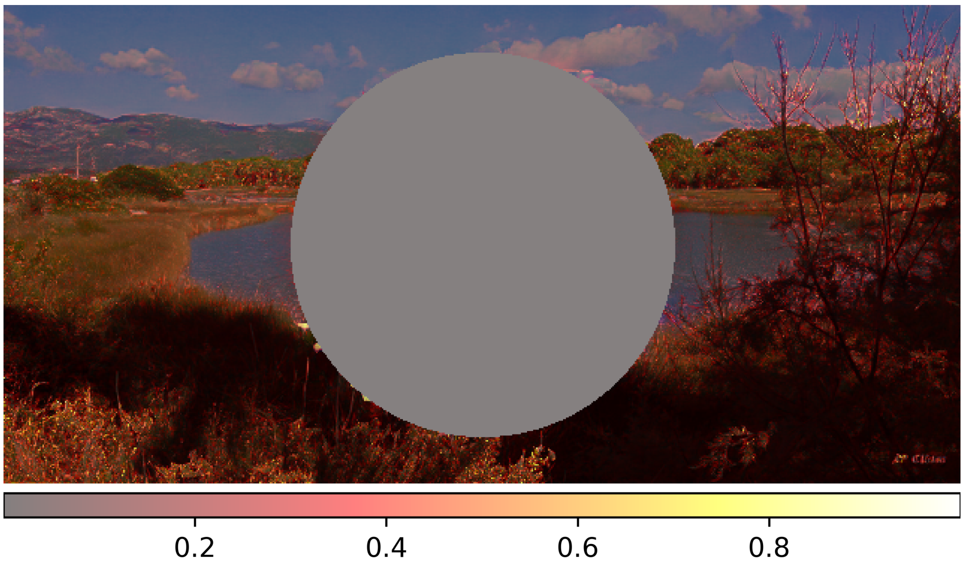
The images presented in figures 4.4, 4.5, 4.6, 4.7, 4.8 and 4.9 were specifically selected from among the three inpainting strengths that correspond to the highest MSE values for each image. These images were chosen due to the observation that they appear to reveal some unique characteristics and patterns regarding the changes in the pixels outside the mask as a result of inpainting.



(a) Original Landscape_0 image downscaled



(b) Inpainted image



(c) Heatmap of pixelwise MSE compared to original image

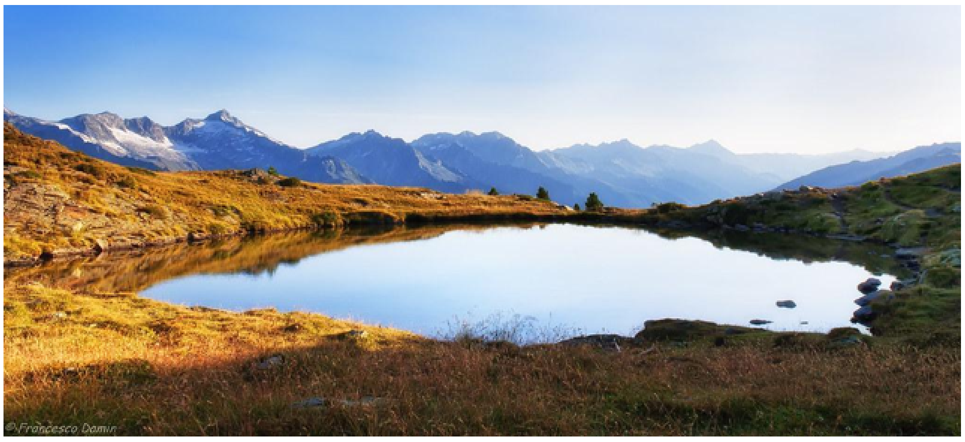
Figure 4.4: Inpainting of **Landscape image 0** done with a strength of 0.84.

Upon closer examination of the heatmap presented in Figure 4.4, it can be observed that the highest levels of inpainting error, as indicated by the yellow color on the colorbar, are concentrated in small areas at the edge of the inpainting mask, specifically between the 6 and 9 o'clock positions. A closer examination of the inpainted area within the mask reveals the presence of blue-colored structures at the lower end of the inpainted region.

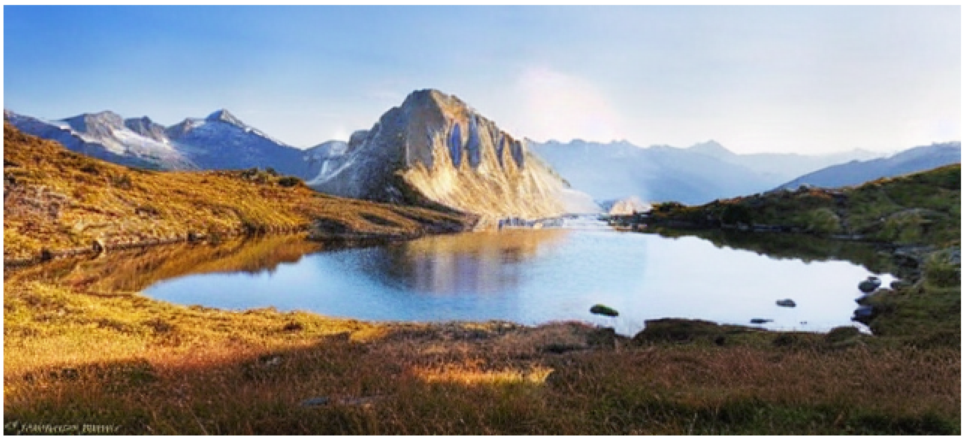
This observation may suggest that the inpainting algorithm, despite the fact that it may have inpainted something that does not fit well with the rest of the image, given its high inpainting strength, is attempting to optimize the transition between the real and inpainted parts of the image. This may result in small areas close to the inpainting mask's edge being altered in order to achieve a better fit. Furthermore, it is also worth noting that the fine-grained structural parts of the image, such as tree leaves, grass, or the mountain ridge in the background, also seem to be affected by the inpainting, although to a lesser extent. These alterations may be due to the algorithm attempting to preserve the overall coherence and continuity of the image, which can be challenging when dealing with such fine-grained details. This highlights the complexity of the inpainting task and the need to consider various factors when developing inpainting algorithms.

This highlights the complexity of the inpainting task and the trade-offs that may be required to achieve optimal results. Further investigation is necessary to better understand the underlying mechanisms that influence the quality of the inpainting results and the ways in which these trade-offs can be minimized.

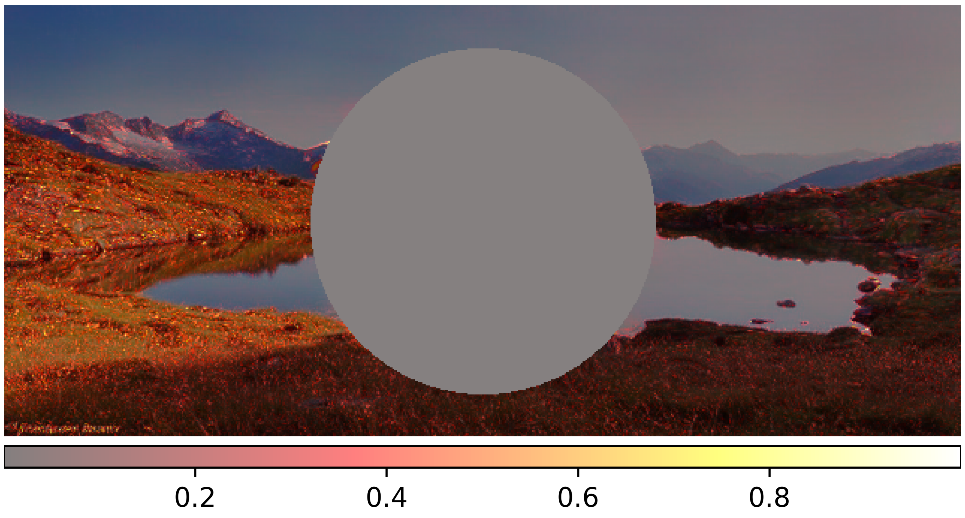
The heatmaps of pixel-wise MSE for these images, will be utilized to gain further understanding of the distribution of errors and the location where the change in MSE is most prominent. This analysis will provide insights into the underlying mechanisms that influence the quality of the inpainting results, and may aid in the development of improved inpainting methods.



(a) Original Landscape_2 image downscaled



(b) Inpainted image



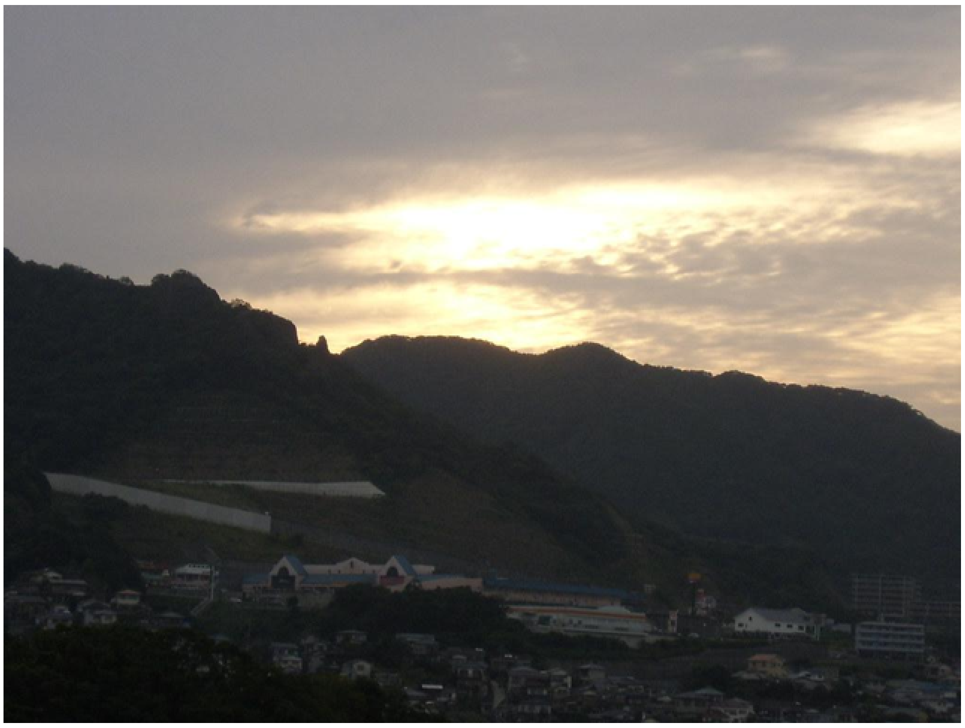
(c) Heatmap of pixelwise MSE compared to original image

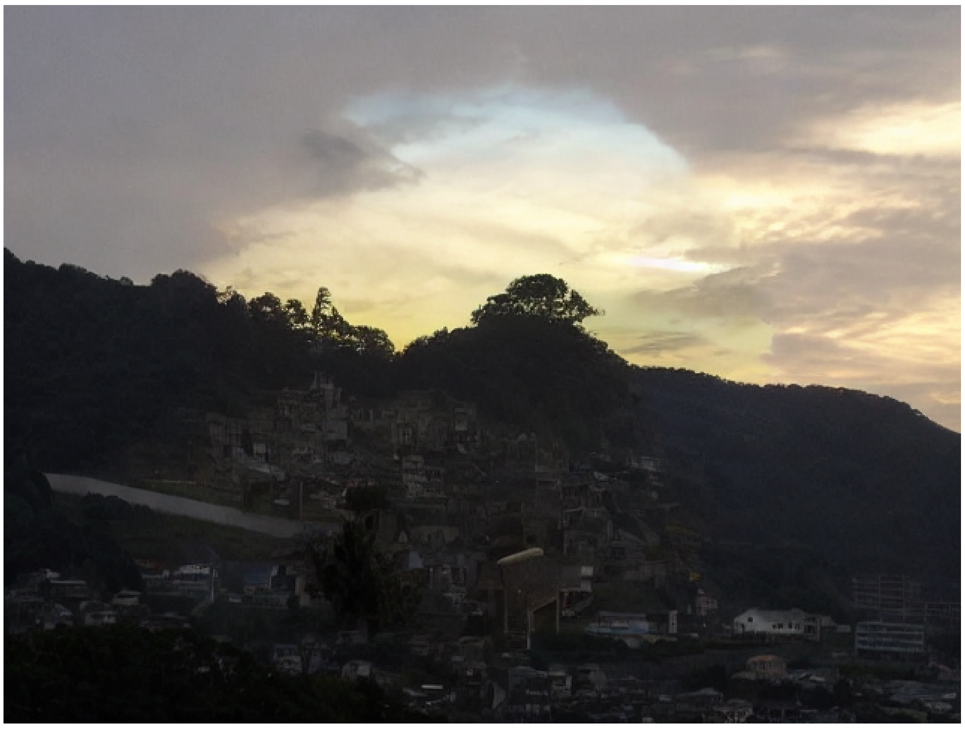
Figure 4.5: Inpainting of **Landscape_2** done with a strength of 0.97.

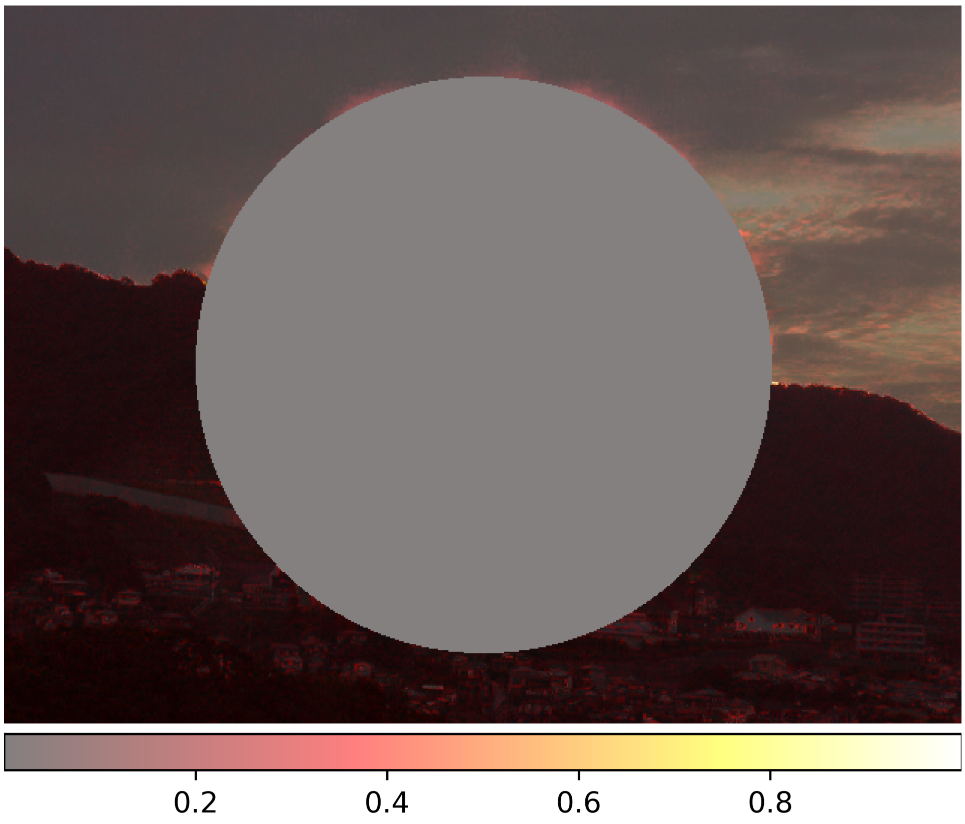
Upon close examination of the heatmap presented in Figure 4.5, it can be observed that the ridges in the image seem to have undergone a higher degree of alteration as compared to the other regions. A pattern that is beginning to emerge is that parts of the image that are characterized by a consistent color over a larger area, such as the water or sky, seem to have undergone minimal alteration as a result of inpainting. This is an interesting observation that warrants further investigation.

Another notable feature is the alteration of the mountain ridge that is located at the edge of the inpainting mask, between the 9 and 12 o'clock positions. The inpainting appears to have resulted in a notable change in this region. Additionally, it can be observed that the atmosphere in the inpainted part does not quite match that of the rest of the image. This can be attributed to the high inpainting strength used, which may have resulted in the algorithm attempting to optimize the transition between the real and inpainted parts of the image.

Furthermore, it is also worth noting that the fine-grained parts of the image, such as the grass on the mountainside to the left, have undergone a moderate alteration across the entire area. This further highlights the complexity of the inpainting task and the need to consider various factors when developing inpainting algorithms. The fine-grained details are particularly challenging to preserve during inpainting.


(a) Original Landscape_5 image downscaled


(b) Inpainted image



(c) Heatmap of pixelwise MSE compared to original image

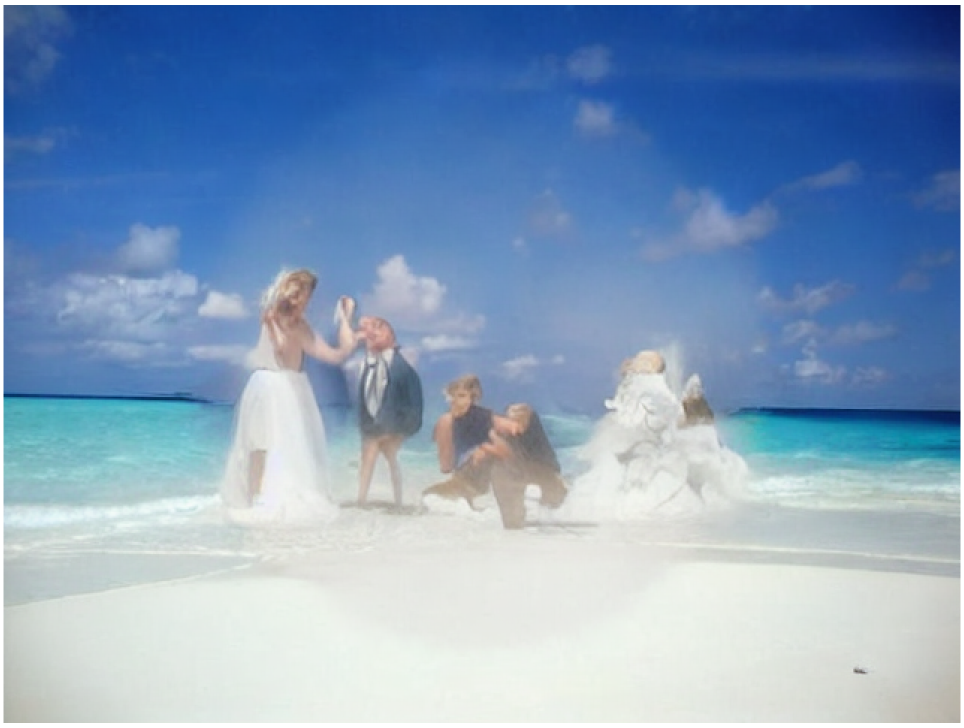
Figure 4.6: Inpainting of **Landscape_5** done with a strength of 0.97

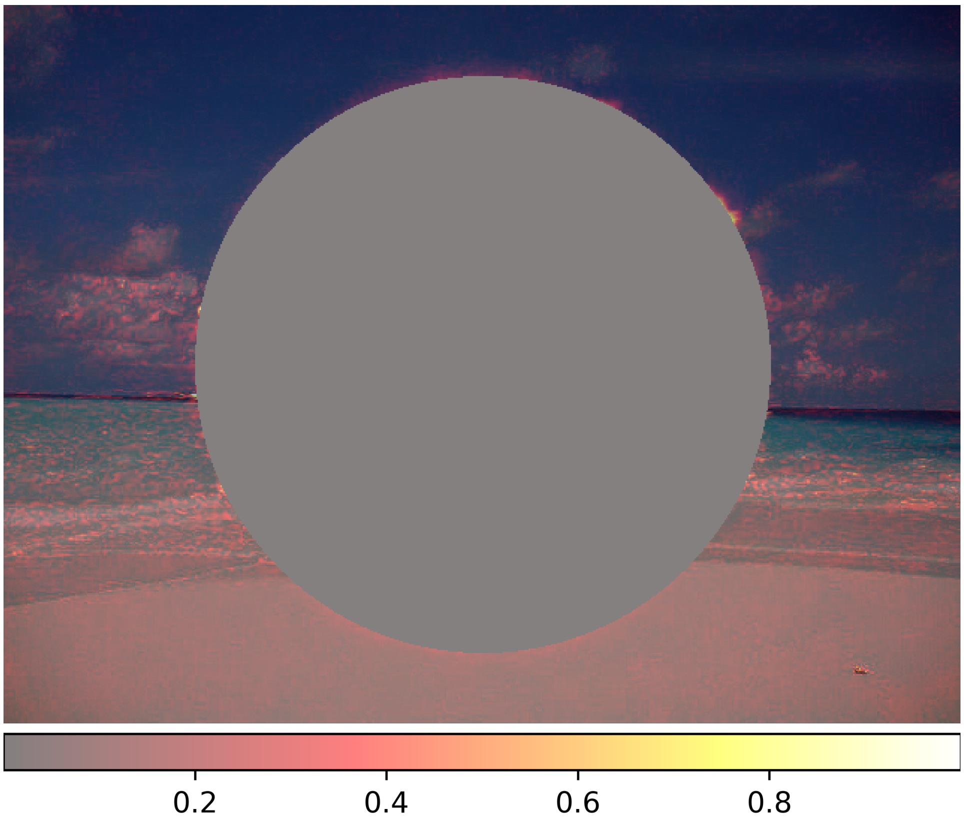
The heatmap presented in Figure 4.6 provides a detailed representation of the pixel-wise mean squared error (MSE) for the image of Landscape 5 inpainted with a strength of 0.97. This heatmap reveals an interesting pattern of alteration that supports the hypothesis that pixels located close to the edge of the inpainting mask undergo higher levels of alteration, particularly when the inpainting strength is high.

When examining the heatmap, it can be observed that despite the fact that the masked area, particularly in the upper part of the inpainting mask, contains image areas that are characterized by consistent color (e.g. the sky), the inpainting mask's edge is surrounded by the red color indicating a higher level of MSE. This suggests that the algorithm is attempting to optimize the transition between the real and inpainted parts of the image at the expense of the overall coherence of the image. This is further supported by the observation that the rest of the sky has undergone minimal alteration, indicating that the algorithm is preserving the consistency of the image in this area.

Additionally, it is worth noting that the ridge of the mountain, as in the previous heatmaps, has undergone a notable change as a result of inpainting. This is likely due to the algorithm attempting to preserve the overall continuity of the image and the fine-grained details present in this area.

(a) Original Landscape_7 image downscaled


(b) Inpainted image



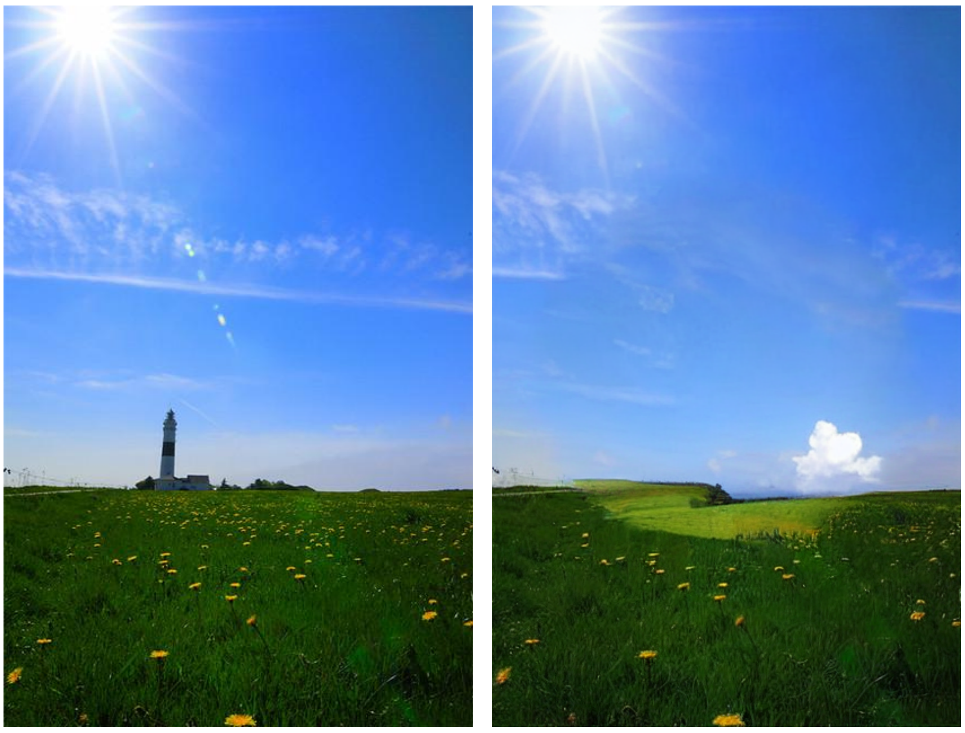
(c) Heatmap of pixelwise MSE compared to original image

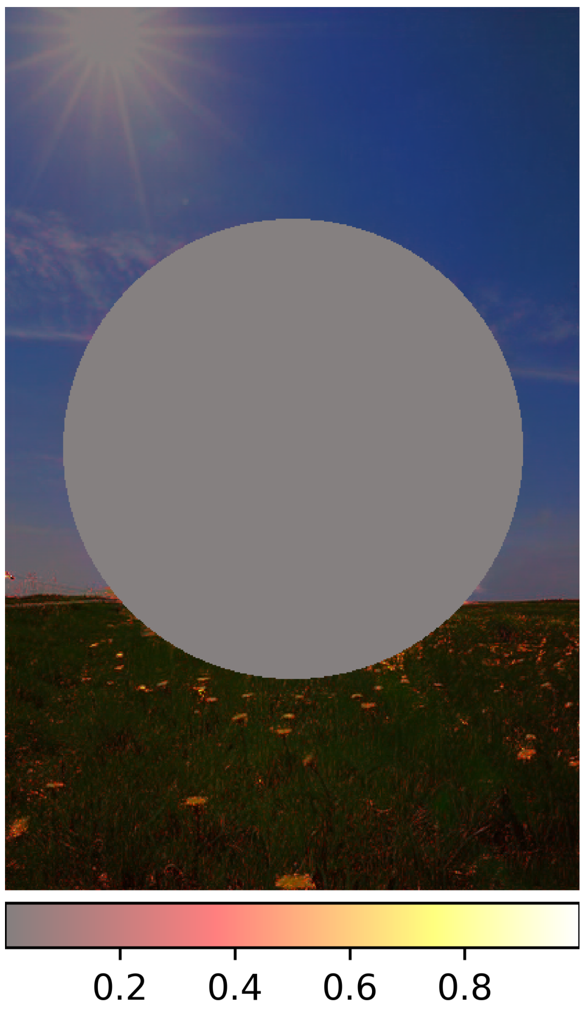
Figure 4.7: Inpainting of **Landscape_7** done with a strength of 0.99.

The heatmap presented in Figure 4.7 provides a detailed representation of the pixel-wise mean squared error (MSE) for the image of Landscape 7 inpainted with a strength of 0.99. This heatmap confirms the observations made in the previous heatmaps.

When examining the heatmap, it can be observed that there is a high level of alteration in certain areas, such as along the edge of the inpainting mask. This suggests that the algorithm is attempting to optimize the transition between the real and inpainted parts of the image at the expense of the overall coherence of the image.

Another interesting observation that can be made is that the change along the horizontal line separating the sea from the sky is not as prominent as the changes observed along the mountain ridges. This raises the question of whether there is a difference between the two lines, such as the mountain ridge being a more complex feature than the sea-sky separating line. Further analysis is needed to investigate the reasons behind this discrepancy.


(a) Original Landscape_15 image downscaled **(b)** Inpainted image



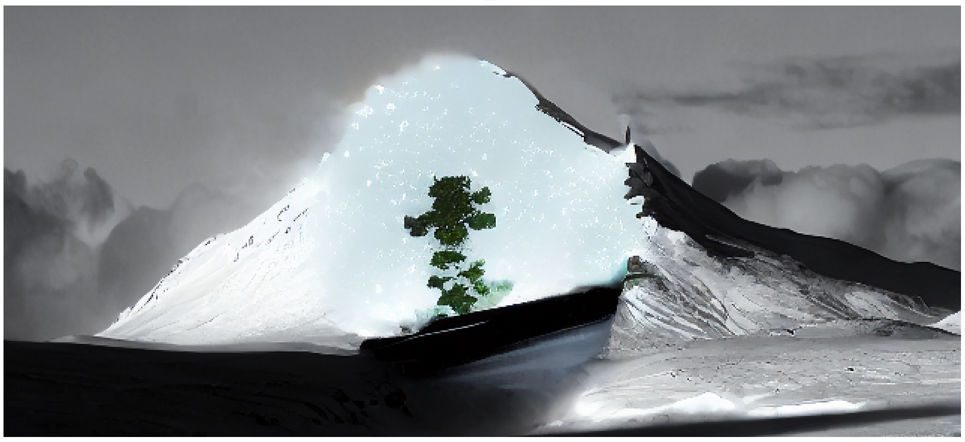
(c) Heatmap of pixelwise MSE compared to original image

Figure 4.8: Inpainting of **Landscape_15** done with a strength of 0.93.

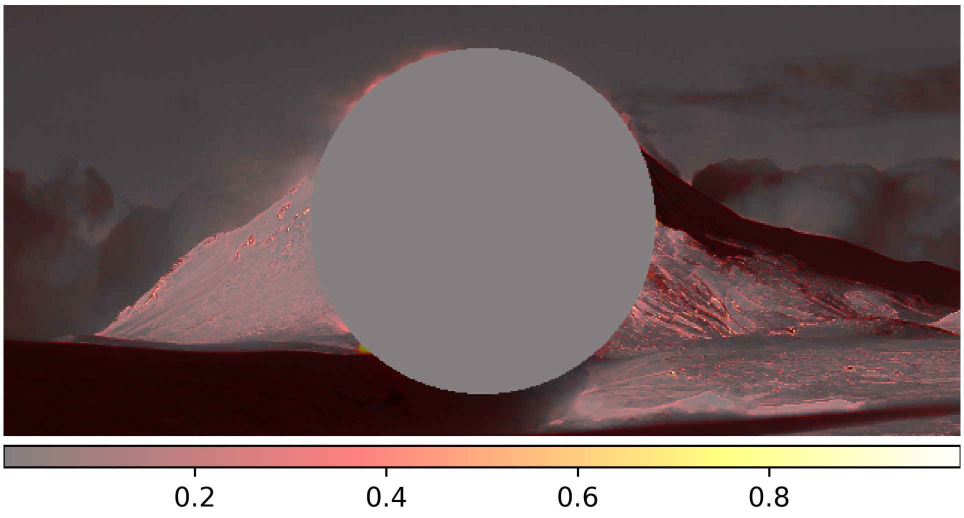
The heatmap presented in Figure 4.8 was selected as an example due to its ability to clearly demonstrate the relationship between the consistency of color within an image and the level of alteration caused by inpainting. As can be observed, areas of the image that exhibit a consistent color tend to undergo less alteration than areas with a higher variation in pixel color.

An interesting observation that can be made is the presence of a relatively bright area of grass within the inpainting mask. Despite this, there is not a significant level of alteration observed in the pixels surrounding this area, which may indicate that the algorithm has determined that a sharp transition of color is the most appropriate fit for this particular image. Additionally, it is also worth noting that the flowers in the image exhibit the highest levels of alteration, which is not surprising given their fine-grained and colorful nature.

(a) Original Landscape_16 image downscaled



(b) Inpainted image



(c) Heatmap of pixelwise MSE compared to original image

Figure 4.9: Inpainting of **Landscape_16** done with a strength of 0.99.

For this last heatmap presented in Figure 4.9 provides a detailed representation of the pixel-wise mean squared error (MSE) for the image of Landscape 16 inpainted with a strength of 0.99. This heatmap, in conjunction with the other heatmaps, continues to confirm the hypothesis that pixels close to the edge of the inpainting mask tend to undergo higher levels of alteration. This is particularly evident in the snow smoke area at the top left of the mountain, which has undergone some alteration in order to optimize the transition between the real and inpainted parts of the image.

Additionally, it can be observed that there is a considerable level of alteration of pixels outside the mask across the fine-grained details of the mountain, such as the mountain ridges. This suggests that the algorithm is attempting to preserve the consistency and continuity of the image while minimizing errors, but this comes at the expense of the overall coherence of the image. This observation is consistent with the previous analysis of the other heatmaps, where it was also observed that pixels in fine-grained and high-contrast areas tend to be more affected by the inpainting process.

Conclusion

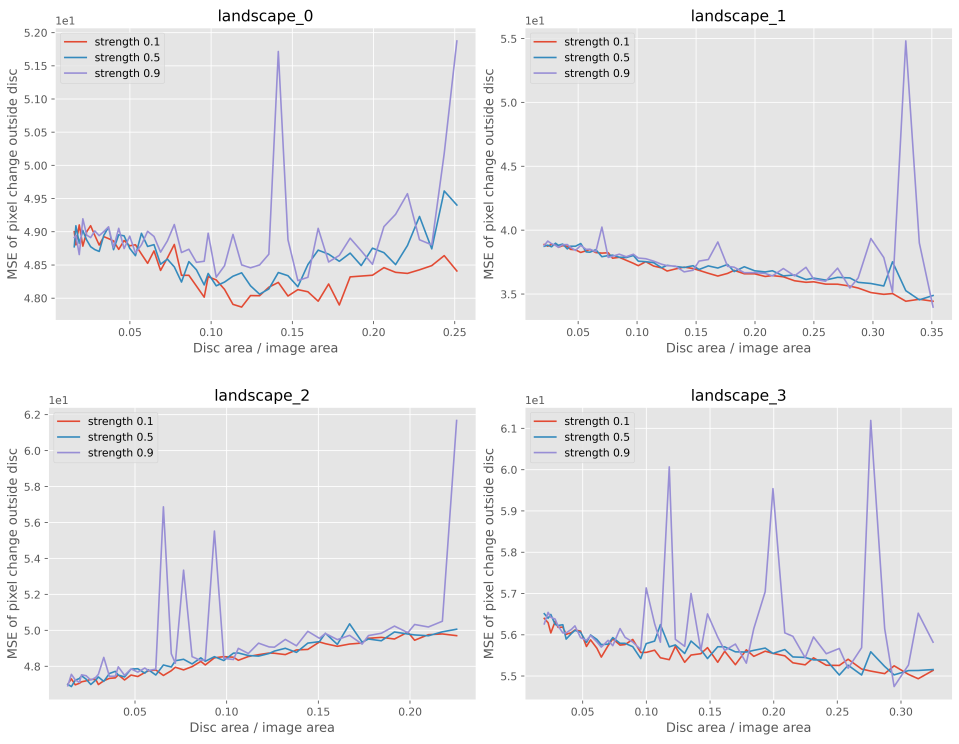
In conclusion, the heatmap analysis of the inpainting experiment has provided valuable insights into the behavior of the inpainting algorithm with regard to the alteration of pixels outside the inpainting mask. The results indicate that areas of the image characterized by consistent pixel color, such as the sky or water, tend to undergo little to no alteration during the inpainting process. Conversely, areas of the image that are more varied in color and texture, such as mountain ridges, tree leaves, and grass, tend to experience higher levels of alteration. Furthermore, the heatmaps have demonstrated that the edge of the inpainting mask is a particularly sensitive area for pixel alteration, with high levels of alteration observed in the immediate vicinity of the mask.

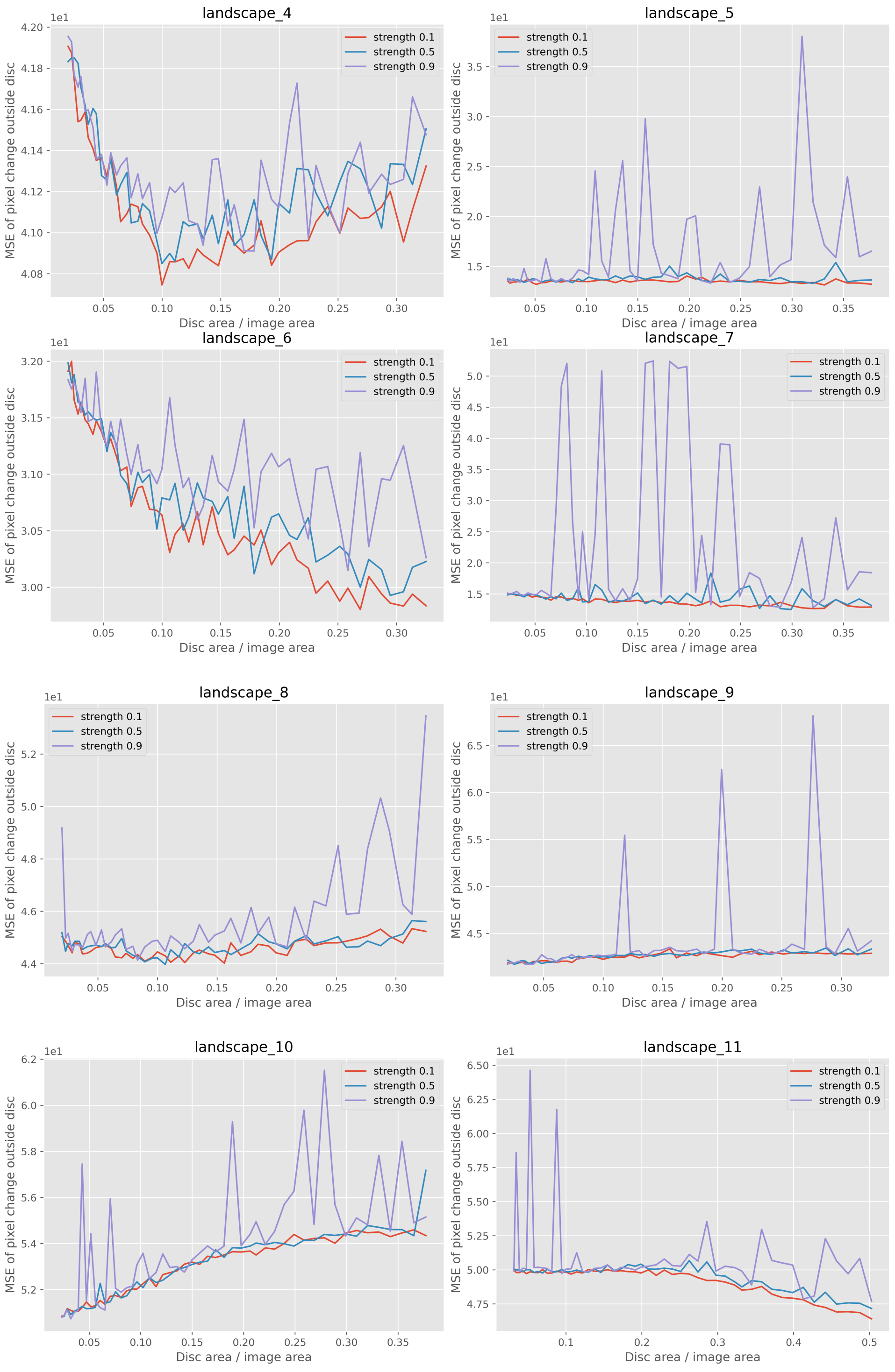
It has been observed that the edge of the inpainting mask tends to be disproportionately affected in comparison to other regions of the image, even when accounting for similarities in color and texture. This observation suggests that the inpainting algorithm places a particular emphasis on preserving continuity at the edge of the mask. Overall, the heatmap analysis has provided valuable insights into the behavior of the inpainting algorithm and has highlighted areas of the image that are most affected by the inpainting process.

In order to further understand the underlying mechanisms of the inpainting algorithm, further research is necessary to identify the specific characteristics of images that contribute to their lower MSE. Additionally, it is crucial to investigate the high spikes in MSE observed at inpainting strengths of 0.6 and above, as well as the reason for the different level of alteration of pixels outside the mask with respect to the image content.

4.3 Pixel change with respect to mask size

4.3.1 Graphical visualisation




landscape_4
landscape_5
landscape_6
landscape_7
landscape_8
landscape_9
landscape_10
landscape_11
1e1
strength 0.1
strength 0.5
strength 0.9
MSE of pixel change outside disc
Disc area / image area

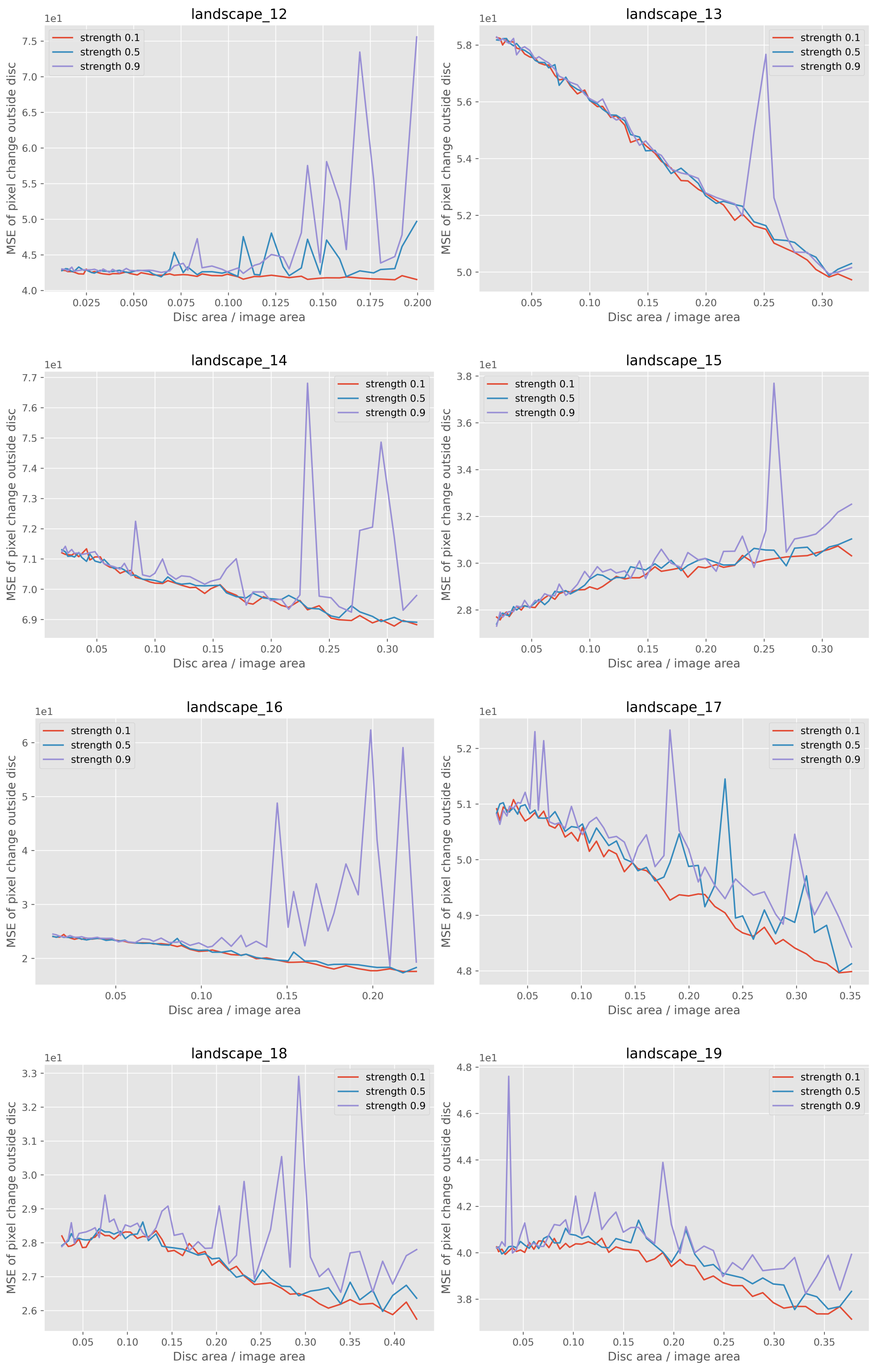

landscape_12
landscape_13
landscape_14
landscape_15
landscape_16
landscape_17
landscape_18
landscape_19
1e1
strength 0.1
strength 0.5
strength 0.9
MSE of pixel change outside disc
Disc area / image area

Figure 4.10

4.3.2 Discussion

As depicted in Figure 4.10, which is outlined in Section 3.0.5, the results of inpainting various landscape images, as illustrated in Figure 4.1, with a range of radii and a fixed strength parameter of 0.1, 0.5, and 0.9 are presented. The primary objective of this experiment was to investigate the relationship between the alteration of pixels outside the inpainting mask and both the inpainting strength and the number of available pixels outside the inpainting mask.

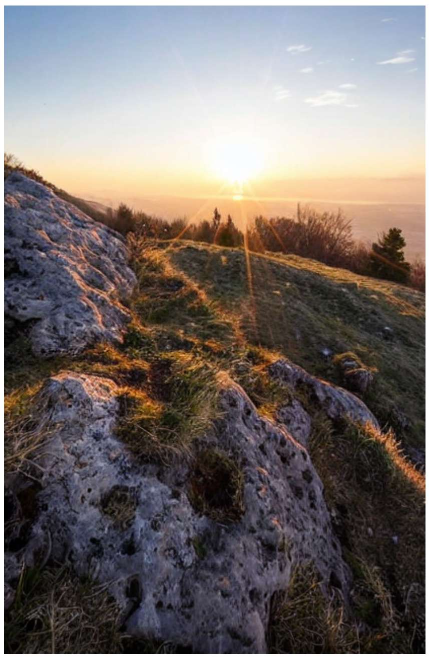
Through incrementally varying the radius of the inpainting mask in 50 steps, it is possible to conduct a supplementary inpainting experiment to examine the effect of different mask sizes on the inpainting process.

As previously established through the plots in Figure 4.2 and 4.2.3, it has become evident that the strength parameter exerts a significant influence on the alteration of pixels outside the inpainting mask. An examination of the results from varying the inpainting mask's radius further confirms this observation, as evidenced by the increased fluctuations in Mean Squared Error (MSE) for pixels outside the inpainting mask for strengths above 0.6.

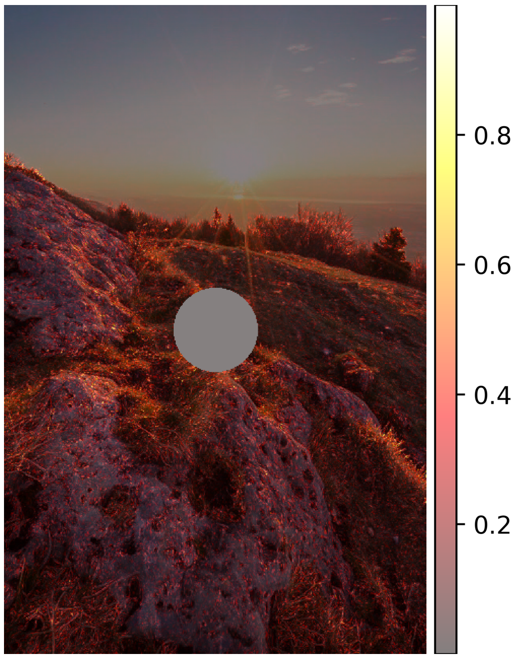
However, it does not appear that changing the radius has a significant impact on the MSE. Nearly all the plots have a minimum to maximum difference in MSE score with respect to the ratio of the disk area to the image area of within a maximum of 3 to 4 MSE score, primarily due to the random spikes observed in the 0.9 strength curve. In comparison to the differences observed in the first experiment, as depicted in Figure 4.2, which could range between 20 to 30 given the spikes in MSE, the differences in this experiment are negligible.

It should also be noted that all the plots in Figure 4.2, heatmap figures from figs. 4.4 to 4.9, and the plots seen for this experiment in Figure 4.10 the error on the y-axis is the mean of the error squared. Therefore, the actual change in practical terms would be the square root of the numbers on the y-axis, which makes the error observed in our plots even more insignificant.

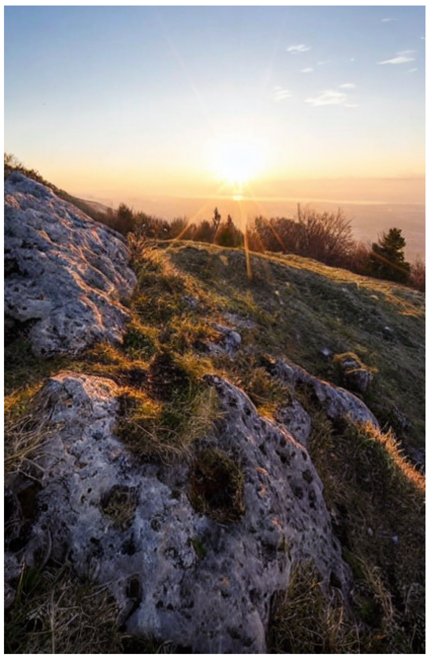
Despite the minimal variations observed in the results, certain trend lines can be discerned in a select number of the inpainted images. Utilizing the knowledge gained from previous experiments, it may be possible to identify the underlying factors contributing to these trends. An examination of Landscape 13 and 17 in Figure 4.10 reveals a striking linear trend, particularly in the case of Landscape_13 seen in the plot.



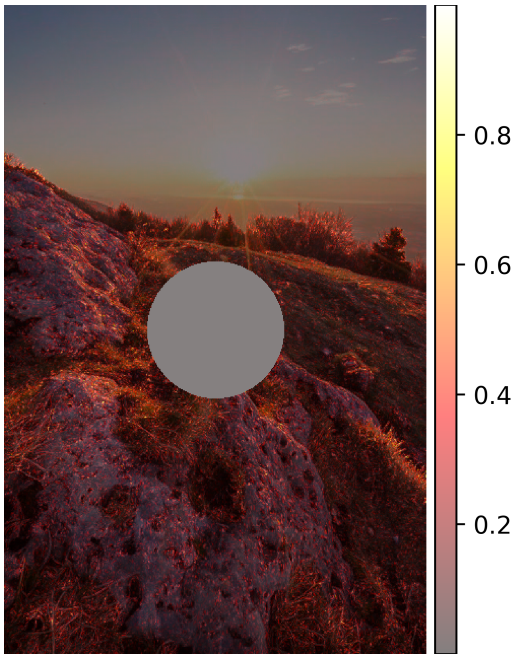
(a) Inpainting with 20% mask radius of smallest image side



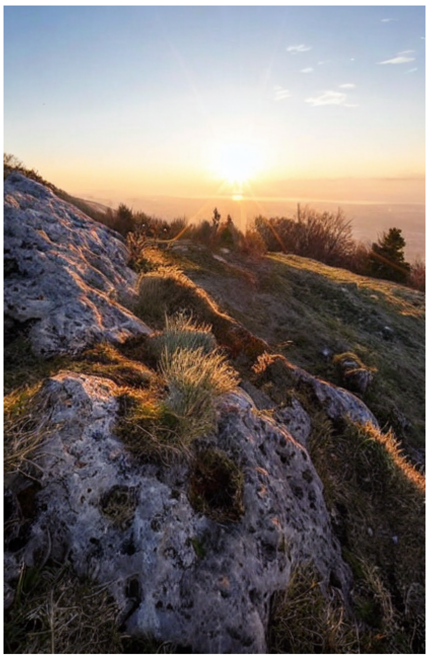
(b) Heatmap of inpainting in 4.11a

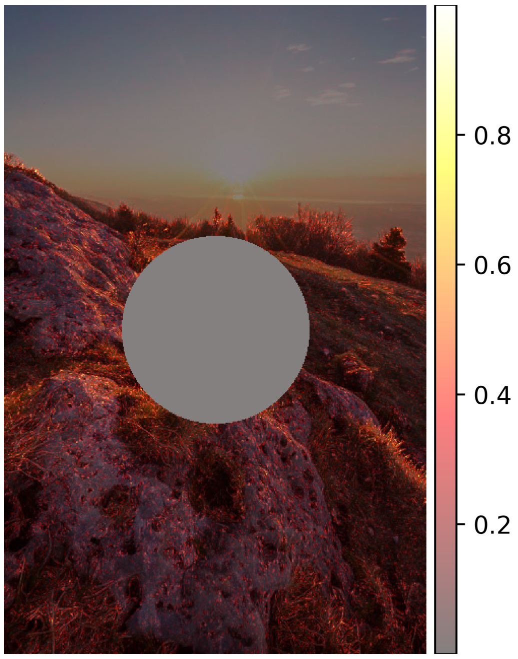


(c) Inpainting with 32% mask radius of smallest image side

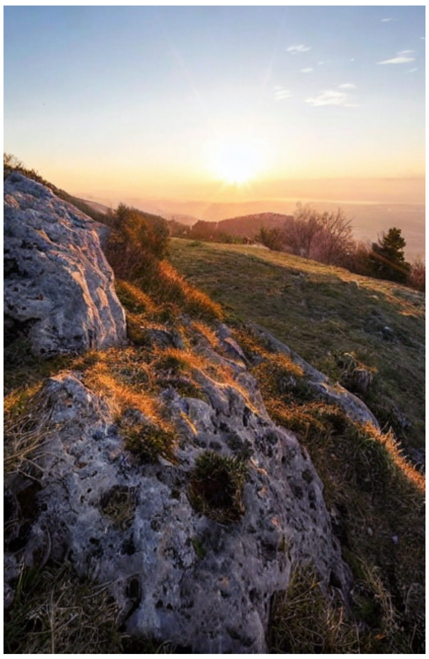


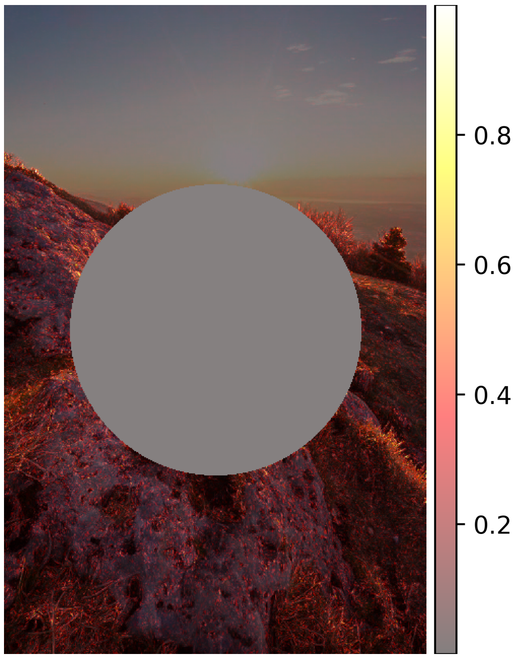
(d) Heatmap of inpainting in 4.11c


(e) Inpainting with 44% mask radius of smallest image side



(f) Heatmap of inpainting in 4.11e


(g) Inpainting with 68% mask radius of smallest image side



(h) Heatmap of inpainting in 4.11g

Figure 4.11: Inpaintings and corresponding heatmaps of Landscape_13, utilizing a strength of 0.5 and a radius percentage of half the smallest dimension of the image, at 20, 32, 44, and 68 respectively.

As demonstrated in Figure 4.11, as the area covered by the inpainting mask increases, fewer pixels are taken into account when calculating the MSE for the change of pixels outside the mask. This phenomenon can be attributed to the mask acting as an area of constant color, such as the sky in the upper part of Landscape_13. As such, the decreasing trend observed in the plot for Landscape_13 may be explained by fewer pixels being considered when calculating the MSE for larger mask radii. Conversely, this phenomenon can also account for the reversed trend in which the MSE exhibits a slight increase as the radius increases. For example, if the inpainting masks as they expand ultimately cover areas of consistent color, such as sky or water, the pixels which previously contributed to lowering the MSE, as observed in the heatmaps in figs. 4.4 to 4.9, will no longer be taken into account, resulting in an increase in the MSE. An example of this increasing trend can be observed in the plot for Landscape_10 in Figure 4.10.

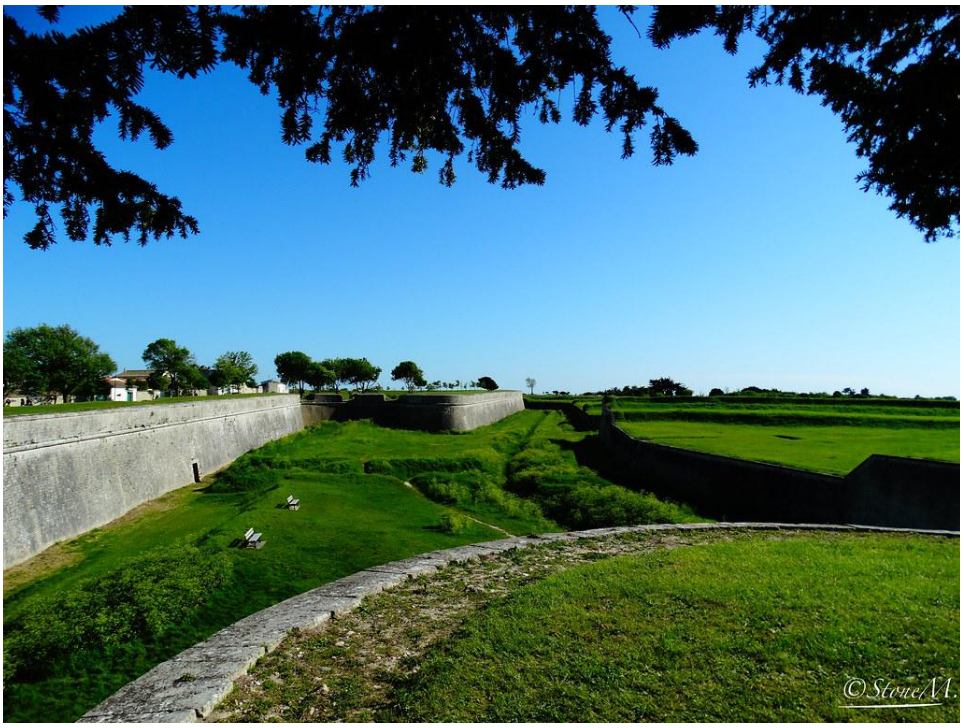


Figure 4.12: Landscape_10

As is evident from the presented image in figure 4.12, a significant proportion of the central region comprises sky. As the radius of the circular inpainting mask increases, a larger portion of the sky will be excluded from the calculation of the MSE for pixels outside the mask. This implies that pixels with higher change will disproportionately contribute to the computation of the mean of the squared errors. It is important to note, however, that the observed trend of change in error, whether increasing or decreasing, is minimal and may be considered negligible.

The analysis of the plots presented in Figure 4.10 reveals an interesting observation with regards to the relationship between inpainting strength and Mean Squared Error (MSE) for pixels outside the inpainting mask. By examining the plots for Landscape images 0, 4, 6, 8, 12, 15, 17 and 18, it can be observed that the curves for inpainting strengths of 0.1, 0.5, and 0.9 overlap initially, as the radius of the inpainting mask is small. However, as the radius increases, the variance in MSE, particularly for inpainting strengths of 0.5 and 0.9, increases. This suggests that for very small inpainting masks, or at least low radius circular inpainting masks, the inpainting strength has a less pronounced effect on the MSE for pixels outside the image.

Furthermore, these plots also confirm the findings from the plots presented in Figure 4.2, which demonstrated that for higher inpainting strengths and inpainting masks with a radius of 80% of half the smallest image side, the MSE outside the image also increases. This is evident by the general increasing distance in MSE for the last plot points for the plots in Figure 4.10.

4.4 Conclusion

The present study aimed to investigate the behavior of an open-source diffusion model inpainting algorithm when presented with varying inpainting strengths and mask radii. However, the results obtained were unexpected and raise important considerations. Our findings indicate that the algorithm not only modifies the pixels within the designated mask, as intended, but also alters pixels outside of the mask, even those that are unrelated to the inpainted subject. This unexpected behavior warrants further investigation and consideration in the field of utilizing diffusion models for inpainting.

The experiments conducted in this study have provided valuable insights into the behavior of the inpainting algorithm when presented with varying inpainting strengths and mask radii. It was observed that areas of the image characterized by consistent pixel color, such as the sky or water, tend to undergo little to no alteration during the inpainting process. On the other hand, areas of the image that are more varied in color and texture, such as mountain ridges and grass, tend to experience higher but still relatively low levels of alteration. The use of heatmaps further highlighted the sensitivity of the algorithm towards the edges of the inpainting mask, with high levels of alteration observed in the immediate vicinity of the mask.

The second experiment revealed that as the radius of the inpainting mask increases, the MSE may fluctuate depending on the content within the excluded pixels. However, it is important to note that any changes in error are so small that they may be considered insignificant.

In addition, it would also be beneficial to study the effect of using different types of masks, such as irregular shaped masks, on the inpainting algorithm's performance. Furthermore, it would be interesting to explore the use of the inpainting algorithm in different fields, such as video inpainting and 3D object inpainting, to understand how it performs in those areas and what are its limitations.

The utilization of diffusion models for inpainting has emerged as a promising avenue of research in the field of deep learning. The algorithm presented by Hugging Face, in particular, has proven to be both user-friendly and efficient when run on a GPU. However, it is worth noting that the lack of proper documentation for the subalgorithms used in the model may present a challenge for those seeking to fully understand and utilize its capabilities.

Overall, this project served as a steep but rewarding learning curve, providing the opportunity to gain knowledge on diffusion models, GPU clusters, Docker, and the ability to solve problems independently. The results of this study have demonstrated the impressive capabilities of diffusion models in the context of inpainting and the potential for future advancements in this field.

Bibliography

[1] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *INTERNATIONAL CONFERENCE ON MACHINE LEARNING, VOL 37*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37, 2015, pp. 2256–2265, 32nd International Conference on Machine Learning, Lille, FRANCE, JUL 07-09, 2015.

[2] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10 684–10 695.

[3] "Imagen: Text-to-Image Diffusion Models." [Online]. Available: https://imagen.research.google/

[4] "DALL·E 2." [Online]. Available: https://openai.com/dall-e-2/

[5] "Midjourney Showcase." [Online]. Available: https://midjourney.com/showcase/recent/

[6] "Stability-AI/stablediffusion: High-Resolution Image Synthesis with Latent Diffusion Models." [Online]. Available: https://github.com/Stability-AI/stablediffusion

[7] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2021.

[8] "Stable Diffusion," Dec. 2022, original-date: 2022-08-10T14:36:44Z. [Online]. Available: https://github.com/CompVis/stable-diffusion

[9] "runwayml/stable-diffusion," Dec. 2022, original-date: 2022-10-18T16:40:30Z. [Online]. Available: https://github.com/runwayml/stable-diffusion

[10] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," 2021. [Online]. Available: https://arxiv.org/abs/2105.05233

[11] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," 2016. [Online]. Available: https://arxiv.org/abs/1606.03498

[12] D. C. Dowson and B. V. Landau, "The Fréchet distance between multivariate normal distributions," *Journal of Multivariate Analysis*, vol. 12, no. 3, pp. 450–455, 1982. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0047259X8290077X

[13] A. Gretton, K. Borgwardt, M. J. Rasch, B. Scholkopf, and A. J. Smola, "A kernel method for the two-sample problem," 2008. [Online]. Available: https://arxiv.org/abs/0805.2368

[14] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[15] A. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *INTERNATIONAL CONFERENCE ON MACHINE LEARNING, VOL 139*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139, 2021, international Conference on Machine Learning (ICML), ELECTR NETWORK, JUL 18-24, 2021.

[16] Karagiannakos, Sergios, Adaloglou, and Nikolaos, "Diffusion models: toward state-of-the-art image generation," *https://theaisummer.com/*, 2022.

[17] C. Luo, "Understanding diffusion models: A unified perspective," 2022. [Online]. Available: https://arxiv.org/abs/2208.11970

[18] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," in *INTERNATIONAL CONFERENCE ON MACHINE LEARNING, VOL 139*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139, 2021, international Conference on Machine Learning (ICML), ELECTR NETWORK, JUL 18-24, 2021.

[19] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013. [Online]. Available: https://arxiv.org/abs/1312.6114

[20] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *MEDICAL IMAGE COMPUTING AND COMPUTER-ASSISTED INTERVENTION, PT III*, ser. Lecture Notes in Computer Science, N. Navab, J. Hornegger, W. Wells, and A. Frangi, Eds., vol. 9351. Tech Univ Munich; Friedrich Alexander Univ Erlangen Nuremberg, 2015, pp. 234–241, 18th International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), Munich, GERMANY, OCT 05-09, 2015.

[21] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 2016. [Online]. Available: https://arxiv.org/abs/1603.07285

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 30 (NIPS 2017)*, ser. Advances in Neural Informa-

tion Processing Systems, I. Guyon, U. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, 2017, 31st Annual Conference on Neural Information Processing Systems (NIPS), Long Beach, CA, DEC 04-09, 2017.

[23] "The Annotated Diffusion Model." [Online]. Available: https://huggingface.co/blog/annotated-diffusion

[24] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen, "Glide: Towards photorealistic image generation and editing with text-guided diffusion models," 2021. [Online]. Available: https://arxiv.org/abs/2112.10741

[25] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes, T. Salimans, J. Ho, D. J. Fleet, and M. Norouzi, "Photorealistic text-to-image diffusion models with deep language understanding," 2022. [Online]. Available: https://arxiv.org/abs/2205.11487

[26] J. Ho and T. Salimans, "Classifier-free diffusion guidance," in *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021. [Online]. Available: https://openreview.net/forum?id=qw8AKxfYbI

[27] "Google Colaboratory." [Online]. Available: https://colab.research.google.com/github/huggingface/notebooks/blob/main/diffusers/in_painting_with_stable_diffusion_using_diffusers.ipynb#scrollTo=byoa1q2zyd6d

[28] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. Van Gool, "Repaint: Inpainting using denoising diffusion probabilistic models," in *2022 IEEE/CVF CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR)*, ser. IEEE Conference on Computer Vision and Pattern Recognition. IEEE; CVF; IEEE Comp Soc, 2022, pp. 11 451–11 461, iEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, JUN 18-24, 2022.

[29] P. Rouzrokh, B. Khosravi, S. Faghani, M. Moassefi, S. Vahdati, and B. J. Erickson, "Multitask brain tumor inpainting with diffusion models: A methodological report," *arXiv preprint arXiv:2210.12113*, 2022.

[30] P. von Platen, S. Patil, A. Lozhkov, P. Cuenca, N. Lambert, K. Rasul, M. Davaadorj, and T. Wolf, "Diffusers: State-of-the-art diffusion models," https://github.com/huggingface/diffusers, 2022.

[31] "rsync." [Online]. Available: https://rsync.samba.org/

[32] "Docker: Accelerated, Containerized Application Development," May 2022. [Online]. Available: https://www.docker.com/

[33] "Landscape Pictures." [Online]. Available: https://www.kaggle.com/datasets/arnaud58/landscape-pictures

[34] "huggingface/diffusers," Jan. 2023, original-date: 2022-05-30T16:04:02Z. [Online]. Available: https://github.com/huggingface/diffusers/blob/09779cbb4046b0afa7cc3da043c928dc4866d59a/src/diffusers/pipelines/stable_diffusion/pipeline_stable_diffusion_inpaint_legacy.py

[35] "VebjornHal/huggingface-diffusion-library-testing: Using huggingface public diffusion library to generate images on the springfield cluster." [Online]. Available: https://github.com/VebjornHal/huggingface-diffusion-library-testing