Telecom Paris

Project IMA206
SinGAN

Elaborated by:

Haithem DAGHMOURA

Kais ZAIRI

Nourelhouda KLICH

Supervised by:

Arthur LECLAIRE

Yann GOUSSEAU

Academic year :
2023/2024

Abstract

This report presents the implementation and enhancement of SinGAN, a generative adversarial network trained on a single natural image. SinGAN's unique architecture, which captures multi-scale internal statistics of a single image, allows for diverse and high-quality image generation. Our work consists of three primary components: the replication of SinGAN as described in the original paper [1] and its associated GitHub repository [2], an investigation into the impact of hyperparameter modifications and paddining choice impact on the model's performance using the Single Image Frechet Inception Distance (SIFID) metric, and an exploration of three alternative loss functions for the discriminator. The results demonstrate that fine-tuning hyperparameters can significantly influence the quality of generated images, with certain configurations yielding faster results with same visual performance. Additionally, the introduction of new loss functions shows promise in enhancing the model's performance, providing insights into potential improvements over the baseline SinGAN. In fact, substituting the network architecture of the discriminator by a given explicit loss function alleviates the heavy cost of training the discriminator for each scale enhancing the efficiency of the method.

Contents

List of Figures

Introduction

Generative Adversarial Networks (GANs) have emerged as a powerful class of models for image synthesis, capable of generating high-quality, realistic images. Traditional GANs typically require extensive datasets to train models that can produce diverse and accurate outputs. However, the SinGAN model, introduced by researchers, diverges from this approach by training solely on a single natural image. This unique capability stems from SinGAN's architecture, which captures the internal statistics of an image across multiple scales, allowing it to perform various image manipulation tasks such as image generation, super-resolution, and paint-to-image translation.

The primary objective of this project is to implement and enhance the SinGAN model. Our work can be divided into three main components. First, we replicate the original SinGAN as described in the seminal paper [1] and its associated GitHub repository[2]. Second, we investigate the impact of modifying hyperparameters and the choice of padding on the model's performance, utilizing the Single Image Frechet Inception Distance (SIFID) metric for evaluation. Third, we explore the potential benefits of incorporating three alternative loss functions for the discriminator, aiming to improve the model's efficiency and output quality.

After successfully reproducing the paper resutls, we examine the effect of different padding techniques on the quality of generated images. Our experiments reveal that even though this choice wasn't mentioned in the paper, it had a big influence in defining the new generated images structure near borders.

Furthermore, we introduce new loss functions for the discriminator. By substituting the network architecture of the discriminator with explicit loss functions, we aim to alleviate the computational burden associated with training the discriminator at each scale. Our results indicate that these alternative loss functions not only improve the efficiency of the method but also maintain or enhance the quality of the generated images, providing valuable insights into potential improvements over the baseline SinGAN.

This report is structured as follows: we begin with detailing the implementation of the original SinGAN model. The subsequent section presents our experimental results, offering a comprehensive analysis of the impact of our modifications on the model's performance. Finally, we present the new discriminator loss functions and the yielded results.

1 Original SinGan implementation

1.1 Architecture

The SinGAN model employs a multi-scale architecture designed to capture the internal statistics of a single natural image across various scales. This architecture comprises a pyramid of generators $(G_0, G_1, ..., G_N)$ and discriminators $(D_0, D_1, ..., D_N)$, where each generator and discriminator operates at a specific scale of the image pyramid $(x_0, x_1, ..., x_N)$. This hierarchical structure allows the model to generate images with both global coherence and fine details.

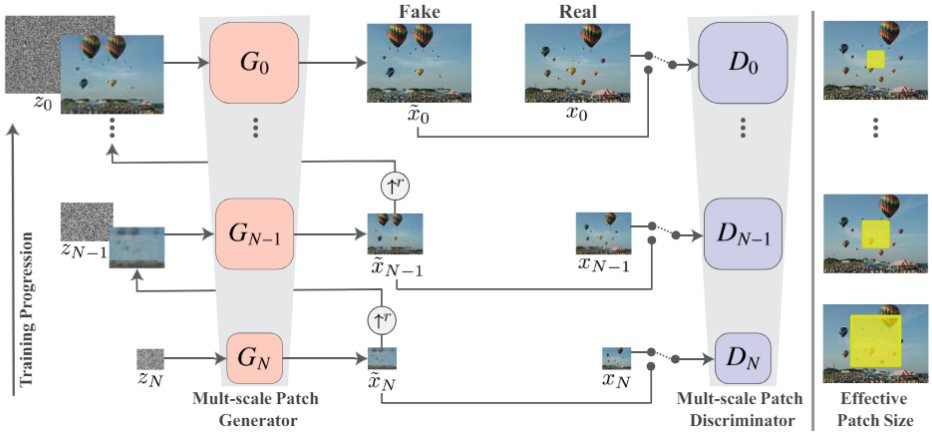


Figure 1: SinGAN's multi-scale pipeline.

Generator

Each generator G_n is responsible for producing realistic image samples at a particular scale n by learning to mimic the patch distribution of the downsampled training image x_n. The image generation process begins at the coarsest scale with G_N and progresses sequentially to the finest scale G_0 (see Figure 1). At the coarsest scale, G_N generates the overall layout and global structure of the image from a random noise input z_N. This can be mathematically represented as:

$$\tilde{x}_N = G_N(z_N)$$

For the finer scales $(n < N)$, each generator G_n refines the image by adding details to the upsampled image from the preceding coarser scale. The input to each generator is a combination of spatial noise z_n and the upsampled image from the previous scale $(\tilde{x}_{n+1}) \uparrow r$:

$$\tilde{x}_n = G_n(z_n, (\tilde{x}_{n+1}) \uparrow r)$$

The generators have a consistent architectural design featuring convolutional layers that enhance the image details and textures. This can be expressed as:

$$\tilde{x}_n = (\tilde{x}_{n+1}) \uparrow r + \psi_n(z_n + (\tilde{x}_{n+1}) \uparrow r)$$

In this expression, ψ_n denotes a fully convolutional network composed of several ConvBlock units.

The key components of the generator are as follows:

- **Head Block**: The head block is the first layer of the generator, consisting of a convolutional layer followed by batch normalization and a LeakyReLU activation function.
- **Body Blocks**: The body of the generator comprises multiple ConvBlock units, which are sequentially stacked. Each ConvBlock includes:
  - A convolutional layer
  - Batch normalization
  - LeakyReLU activation function

  These layers help in extracting and refining features at different levels of abstraction. Importantly, the size of the input image does not affect the number of parameters in the generator, making it adaptable to various image sizes.
- **Tail Block**: The tail block is the final layer of the generator, which converts the feature map back to the image space. It includes a convolutional layer followed by a Tanh activation function to produce the output image.

All generators in SinGAN have a similar architecture consisting of five ConvBlock units, each with convolutional layers of size 3×3, batch normalization, and LeakyReLU activation. The number of kernels per block starts at 32 at the coarsest scale and increases by a factor of 2 every four scales. This ensures that the model does not disregard the noise inputs, which are crucial for generating diverse outputs.

Discriminator

The discriminator in SinGAN, referred to as D_n, is designed to differentiate between real and generated image patches at each scale. Each discriminator has a hierarchical structure comprising multiple convolutional layers organized into blocks. The key components of the discriminator are as follows:

- **Head Block**: Similar to the generator, the head block of the discriminator consists of a convolutional layer followed by batch normalization and a LeakyReLU activation function. This block processes the input image patches to extract initial features.
- **Body Blocks**: The body of the discriminator consists of multiple ConvBlock units, which further process the feature maps to higher levels of abstraction. Each ConvBlock in the body includes:
  - A convolutional layer
  - Batch normalization
  - LeakyReLU activation function
- **Tail Block**: The tail block is the final layer of the discriminator, which reduces the feature map to a single-channel heatmap representing the scores for patch distribution. This heatmap indicates the probability of each patch in the input image being real or generated.

The discriminator is composed entirely of convolutional layers, enabling it to produce this heatmap output, effectively capturing the patch distribution (see Figure 1). The discriminators and generators have the same receptive field, allowing them to capture structures of decreasing size as we move up the generation process.

Remarks

- The effective patch size is the region of the input image that influences the output of a single unit in the convolutional layer. In the multi-scale architecture of SinGAN, the effective patch size decreases as we move up the pyramid. At the coarsest scale, the effective patch size is large, allowing the generator to capture global structures and arrangements of objects. At finer scales, the effective patch size is smaller, focusing on adding finer details and textures to the image.
- Padding plays a crucial role in the convolutional layers of both the generator and discriminator. It determines how the borders of the image are handled during convolution operations. In SinGAN, the choice of padding affects the structure of the generated images near the borders (as we will further discover in the next parts of the report). Proper padding ensures that the patches near the edges of the image are processed consistently with the rest of the image, maintaining visual coherence.

By employing this hierarchical arrangement of generators and discriminators, SinGAN is able to capture the internal statistics of a single natural image effectively, allowing it to perform various image manipulation tasks with high fidelity and diversity.

1.2 Pre-processing

This section outlines the preprocessing steps essential for implementing the SinGAN architecture. The provided functions facilitate normalization, denormalization, GPU operations, image resizing, computing scales, ect... Each function is carefully designed to ensure seamless data handling and processing, which is crucial for the success of the SinGAN model.

1.2.1 Standardization

As it is usually the case, The image data should be Standarized to have values between [-1,1] before conducting the training. In fact, Standardization is essential in Generative Adversarial Networks (GANs) for several reasons. First, by standarizing the data we are ensuring a stabile training. GANs involve simultaneous training of the generator and discriminator. Normalization stabilizes gradient updates, aiding in smoother training and convergence. Second, by doing this we have Consistent Input Scale of values in range [-1, 1] . This is ensured by norm function.

1.2.2 Image Resizing

This is a crucial preprocessing function. In fact, Since we will be training a multiresolution architecture, we need to be able to reshape the image using a scaling factor which is a real number. This makes it quite hard to have a consistant shape when down sampling and then upsampling with the same factor. In fact, you can be accurate and consistent by specifying both scale-factor and output-size. This is an important feature for super-resolution and learning because one must acknowledge that the same output-size can be resulted with varying scale-factors. Best explained by example: if we consider an image of size 9x9 and resize it by scale-factor of 0.5. The Resulting size is 5x5. Afterwards, if we resize with scale-factor of 2 we get resulting size 10x10. However, if we want to resize it to 9x9, we will not get the correct scale-factor which is calculated as output-size / input-size = 1.8. This is one of the main reasons for creating this function. As this consistency is often crucial for learning based tasks.
The resizing is done on the input by giving the scale using cubic interpolation. It first converts the image to uint8 format, resizes it and then converts it back to a Pytorch tensor.

The core function for image resizing is image_resize_in, supporting various interpolation methods and utilizing antialiasing for downscaling. This especially good when we down scale the image to the smallest scale. Since conducting the antialising will ensure that we don't get artificats due to the downsampling.

1.2.3 Fixing scales and sizes

The number of scales and the scaling factor is not fixed beforehand. Before even starting the process of training we must compute these two. This step ensures that we compute an adequate scaling factor depending on the shape of the input in order to get the smallest image to have a dimension of size 25. It starts by computing the number of scales using the intial scale factor which is 0.75. Then it modifies the scaling factor accordingly for the obtained number of scales.

1.3 Training

The training process for SinGAN involves a sequential multi-scale approach, starting from the coarsest scale and progressing to the finest scale. Each scale's GAN is trained and once trained, it remains fixed while the next scale's GAN is trained. This ensures a stable and progressive refinement of the generated images.

1.3.1 Training Objectives

The training loss for each GAN at scale n is composed of an adversarial loss term $L_{adv}(G_n, D_n)$ and a reconstruction loss term $L_{rec}(G_n)$. The combined objective is:

$$\min_{G_n} \max_{D_n} L_{adv}(G_n, D_n) + \alpha L_{rec}(G_n) \tag{1}$$

where α is a weighting factor that balances the two loss components.

Adversarial Loss

Each generator G_n is paired with a discriminator D_n, which classifies each overlapping patch of its input as real or fake. The adversarial loss L_{adv} penalizes the distance between the distribution of patches in the real image x_n and the generated image $\tilde{x}_n$. The WGAN-GP loss is used to improve training stability, where the final discrimination score is the average over the patch discrimination map.

$$L_{adv} = -\mathbb{E}[D_n(x_n)] + \mathbb{E}[D_n(\tilde{x}_n)] + \lambda \mathbb{E}[(\|\nabla D_n(x)\|_2 - 1)^2] \tag{2}$$

where λ is the gradient penalty coefficient.

Reconstruction Loss The reconstruction loss L_{rec} ensures that there exists a specific set of input noise maps that can reproduce the original image x. This feature is essential for various image manipulation tasks. The reconstruction loss is defined differently for the coarsest scale N and the finer scales $n < N$.

For the finest scale N, the reconstruction loss is:

$$L_{rec} = \|G_N(z^*) - x_N\|_2^2 \tag{3}$$

where z^* is a fixed noise map .

For finer scales $n < N$, the reconstruction loss is:

$$L_{rec} = \|G_n(0, (\tilde{x}_{n+1}) \uparrow_r) - x_n\|_2^2 \tag{4}$$

where $(\tilde{x}_{n+1}) \uparrow_r$ is the upsampled image from the previous scale.

1.3.2 Training Procedure

The training procedure involves the following steps for each scale:

- **Noise and Image Initialization**: Initialize noise maps and set up the real image for the current scale. For the initial scale, random noise maps are generated and padded appropriately. For subsequent scales, noise is generated similarly, but images from previous scales are used to provide a starting point for the current scale. The previous scale's generated image is combined with new noise to form the input for the current scale.
- **Generator and Discriminator Initialization**: The generator and discriminator for the current scale are initialized with specific configurations. The number of feature channels is set based on the current scale level, and padding layers are applied to ensure proper handling of image borders.
- **Adversarial Training**: For each scale, the generator G_n and discriminator D_n are trained adversarially. The discriminator tries to distinguish real patches from fake ones, while the generator aims to fool the discriminator. This involves generating a fake image using the generator and updating the discriminator multiple times .
- **Reconstruction**: The generator is also trained to reconstruct the original image from a specific set of noise maps, ensuring the model can effectively manipulate images. The reconstruction loss ensures the existence of noise maps that can reproduce the real image.
- **Forward Pass and Loss Calculation**: Perform forward passes through the generator and discriminator to compute the adversarial and reconstruction losses. The generator minimizes the adversarial loss to fool the discriminator by generating realistic patches, while also minimizing the reconstruction loss to reproduce the original image from noise maps.
- **Backpropagation and Optimization**: Backpropagate the adversarial and reconstruction losses through the generator and discriminator networks. Update their parameters using optimizers (Adam) with learning rate schedulers that adjust the learning rates during training. The learning rates for both generator and discriminator are scheduled to change at specific milestones.

- **Scale Transition**: Once training for the current scale is complete, the trained generator and discriminator models are saved. The process then transitions to the next finer scale, initializing new noise maps .
- **SIFID Calculation**: Calculate the Single Image Frechet Inception Distance (SIFID) score at each epoch to evaluate the quality of generated images. This involves using a pre-trained InceptionV3 model to extract features and compute the Frechet Distance between the feature distributions of real and generated images.
- **Iteration Process**: The above steps are repeated for a specific number of iterations for each scale. At each epoch, noise maps are generated or updated, fake images are produced by the generator, and both generator and discriminator are optimized. The progress is periodically saved, and SIFID scores are computed to track the improvement in image quality.

By following this detailed multi-scale training approach, SinGAN effectively captures the internal statistics of a single natural image, enabling it to generate high-quality and diverse outputs. The sequential training ensures that each scale builds upon the previous ones, progressively adding finer details and textures to the generated images.

1.3.3 Training results and discussion

During the training process, various images are plotted and saved to monitor the model's performance and evaluate the quality of the generated images. The evaluation is primarily conducted using the Single Image Frechet Inception Distance (SIFID) score, which provides a quantitative measure of the visual similarity between real and generated images.

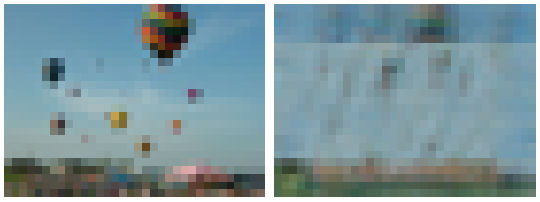
At each scale of the training process, the following images are saved to assess the progress and quality of the generated images:

- **Real Image**: The actual input image at the current scale. This image represents the ground truth that the generator is trying to imitate at each scale.
- **Fake Image**: The image generated by the generator at the current scale after its training. This image is compared with the real image to visually assess the quality and similarity of the generated output.

Figure 2 shows the real input images and the generated images from the generator after training for 2000 iterations per scale. Each pair of subfigures represents the real and generated images at a specific scale, ranging from scale 0 to scale 8. The real images are the downsampled versions of the original input image at different scales, while the generated images are produced by the generator after training.

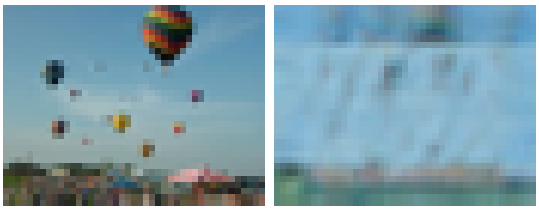
Observations:

- **Initial Scales (0-2):** At the coarsest scales, the generated images (Figures 2(a)-(c)) show a significant level of abstraction and blurriness. This is expected as the generator is learning to capture the overall structure and layout of the image rather than fine details.
- **Mid Scales (3-5):** As the training progresses to finer scales (Figures 2(d)-(f)), the generated images start to capture more details and textures. The overall quality of the generated images improves, and they become more recognizable and closer to the real images.
- **Finer Scales (6-8):** At the finest scales (Figures 2(g)-(i)), the generated images are almost indistinguishable from the real images. The fine details and textures are well captured, demonstrating the effectiveness of the progressive training approach.
- **Improvement with Iterations:** The quality of the generated images improves with the increase in the number of iterations per scale. Training the generator for 2000 iterations per scale allows it to learn the intricate details of the image, resulting in high-quality outputs at finer scales.



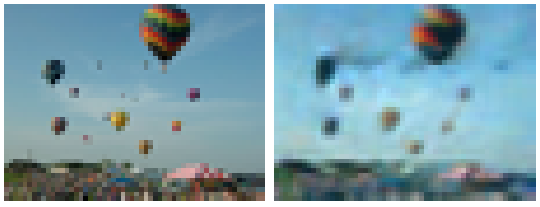
Real Fake

(a) Images at Scale 0



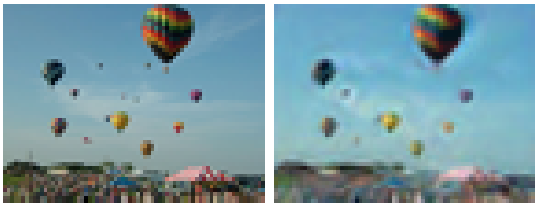
Real Fake

(b) Images at Scale 1



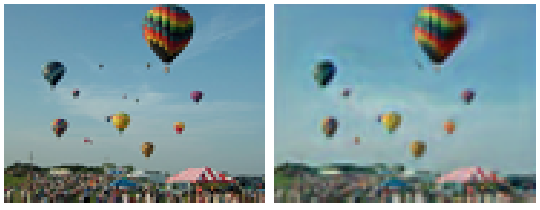
Real Fake

(c) Images at Scale 2



Real Fake

(d) Images at Scale 3



Real Fake

(e) Images at Scale 4

Real Fake

(f) Images at Scale 5


Real


Fake

(g) Images at Scale 6


Real


Fake

(h) Images at Scale 7


Real


Fake

(i) Images at Scale 8

Figure 2: Comparison of Real and Fake Images at Each Scale

The quality of the generated images is evaluated using the SIFID score, which measures the distance between the feature distributions of real and generated images. The steps involved in the evaluation process are as follows:

- **Feature Extraction**: Use a pre-trained InceptionV3 model to extract features from both real and generated images.
- **Distance Calculation**: Compare the feature distributions using the Frechet Distance to compute the SIFID score.
- **Score Plotting**: Plot the SIFID scores over epochs to visualize the improvement in image quality during training.

Figure 3 presents the Single Image Frechet Inception Distance (SIFID) scores for each scale over 2000 iterations. The SIFID score is used to evaluate the quality of the generated images by measuring the similarity between the real and generated images at each scale. Lower SIFID scores indicate higher quality and more realistic generated images.

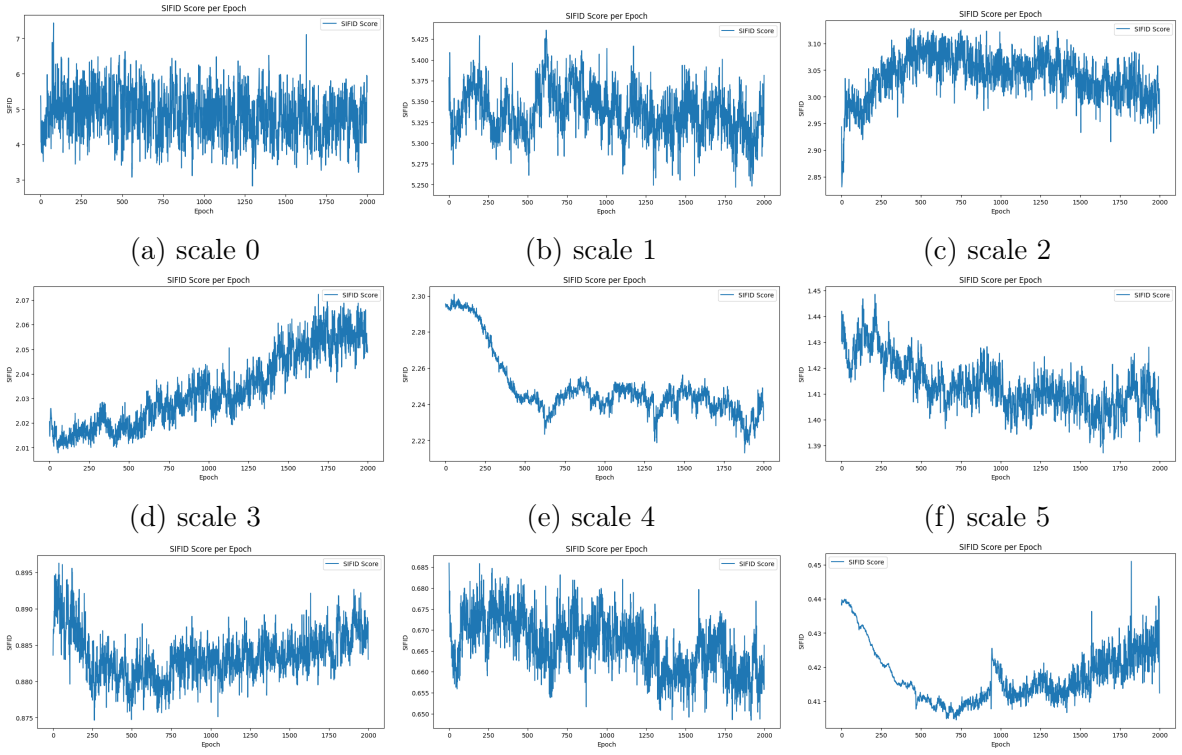
Observations:

- **Initial Scales (0-3):** At the initial scales (Figures 3(a)-(d)), the SIFID scores exhibit higher values and greater fluctuations. For example, at scale 0, the SIFID scores range between 3 and 6. As the training progresses within these scales, a gradual reduction in the variability of SIFID scores is observed, with scale 1 showing scores mostly between 1 and 3. This suggests an initial convergence towards capturing the global structure of the image.

- **Mid Scales (4-6):** In the middle scales (Figures 3(e)-(g)), there is a noticeable decrease in the average SIFID scores, and the fluctuations become less pronounced. For instance, at scale 4, the SIFID scores drop to a range between 0.5 and 2. By scale 5, the scores range from approximately 0.2 to 1.2. This trend signifies that the generator is progressively refining its ability to generate images with more detailed textures and features.

- **Finer Scales (7-8):** At the finer scales (Figures 3(h)-(i)), the SIFID scores are at their lowest and exhibit minimal variability. For example, at scale 7, the scores range between 0.1 and 0.6, and at scale 8, the scores range between 0.1 and 0.5. The low SIFID scores indicate that the generated images are highly similar to the real images, demonstrating the generator's ability to reproduce fine details and intricate textures effectively.

- **Progressive Improvement:** Across all scales, the overall trend of decreasing SIFID scores with increasing scale number emphasizes the effectiveness of the multi-scale training approach. Each scale builds upon the previous one, gradually improving the quality and realism of the generated images. The progressive reduction in SIFID scores from scale 0 to scale 8 highlights how the model successfully captures both global structures and fine details of the input image.



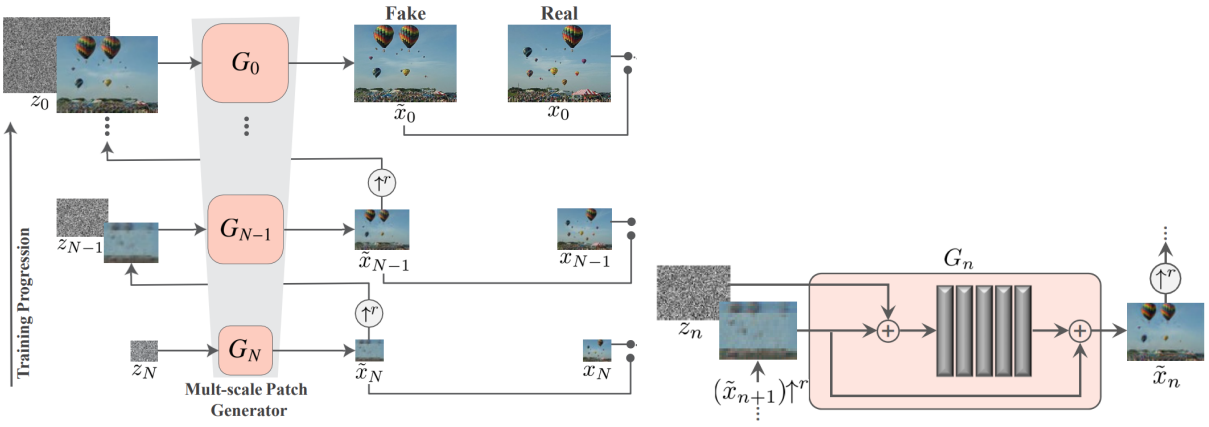
(a) scale 0 (b) scale 1 (c) scale 2

(d) scale 3 (e) scale 4 (f) scale 5

(g) scale 6 (h) scale 7 (i) scale 8

Figure 3: SIFID scores

1.4 Generating images



(a) Generation pipeline (b) Single scale generation

Figure 4: Generation process

After training on a single natural image, SinGAN is capable of generating new images that are both diverse and high-quality. The generation process of SinGAN mirrors its training process. At each scale n of the generative process, the image from the previous scale, $\tilde{x}_{n+1}$, is upsampled and combined with a random noise map, z_n. This mixture is then processed through a series of five convolutional layers, which output a residual image. This residual image is added back to the upsampled image, resulting in the final output for that particular scale, denoted by $\tilde{x}_n$.

$$\begin{cases} \tilde{x}_N = G_N(z_N) \\ \tilde{x}_n = G_n(z_n, (\tilde{x}_{n+1})\uparrow^r) \quad \forall n < N \end{cases} \tag{5}$$

It's important to note that at the scale N, the process begins with an empty image composed of zeros. This empty image is then combined with a random noise map, setting the initial conditions for the generative process.

The design of this generative pipeline is key to SinGAN's ability to produce images that not only vary significantly from one another but also retain the intricate textures and overall structure of the original training image. This ensures that the generated images are not just random assortments of pixels, but coherent samples that closely resemble the source image in style and content.

The figure below demonstrates the results of our generation function compared to the implementation provided on GitHub.

(a) Generation with original implementation (b) Generation with our implementation

Figure 5: Generated samples

1.5 Change of boundary conditions

1.5.1 From Zero-pad to Circular-pad

In the provided paper [1] and its official implementation [2], there is no justification for using zero-padding on the image before passing it through the generator. In our initial experiments, we replaced zero-padding with circular-padding to observe the effects.

1.5.2 Results and discussion

The figure 6 illustrates the results given with Circular-padding.

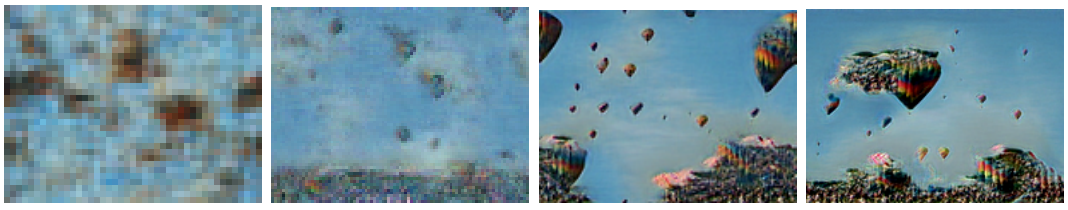


Figure 6: Results with Circular-padding in scale 1, 4, 6, 7

The generated images exhibited the same elements from the real image but in a disordered manner. For instance, some images showed people placed near balloons, parts of balloons on the ground, or sections of the sky misplaced on the ground. This experiment highlights an important observation: zero-padding enables the model to learn the boundaries of the image, thereby maintaining the structural integrity of the real image. The results suggest that zero-padding plays a crucial role in preserving the spatial relationships and overall coherence within the generated images.

2 Training with substitute discriminators

The traditional GAN framework employs a discriminator network to distinguish between real and generated (fake) images, guiding the generator to produce more realistic images. However, training the discriminator at multiple scales in SinGAN can be computationally expensive and time-consuming. To address this issue, we propose a novel approach by replacing the discriminator network with different loss functions.

2.1 Discriminator Replacement with Frechet Distance Loss

The first loss function is based on the Frechet Distance, calculated directly between the real and fake images.

This replacement aims to simplify the training process and reduce computational overhead. By computing statistical measures such as the mean and covariance of image patches, we capture essential information about the image's texture and structure at different scales. The Frechet Distance between these statistics for real and generated images serves as a measure of similarity, guiding the generator in the absence of a traditional discriminator.

2.1.1 Frechet Distance Loss Calculation

The Frechet Distance, is a measure of similarity between two probability distributions. In the context of image generation, it can be used to compare the statistical properties of real and generated images. Specifically, we calculate the Frechet Distance between the mean and covariance of patches extracted from real and generated images.

Given two sets of image patches, one from the real image and one from the generated image, we first compute their respective mean vectors and covariance matrices. Let μ_r and Σ_r represent the mean vector and covariance matrix of patches from the real image, and μ_f and Σ_f represent those from the generated image.

The Frechet Distance $d^2(\mu_r, \Sigma_r, \mu_f, \Sigma_f)$ is defined as:

$$d^2(\mu_r, \Sigma_r, \mu_f, \Sigma_f) = \|\mu_r - \mu_f\|^2 + \mathrm{Tr}(\Sigma_r + \Sigma_f - 2(\Sigma_r\Sigma_f)^{1/2}) \tag{6}$$

where $\|\mu_r - \mu_f\|^2$ is the squared Euclidean distance between the mean vectors, and Tr denotes the trace of a matrix.

2.1.2 Patch Extraction Process

To compute the Frechet Distance, we first extract all possible patches from both the real and generated images. Let I_r and I_f denote the real and generated images, respectively. For each

image, we extract patches of size $k \times k$ with a stride 1.

- Extract patches $\{P_r^i\}_{i=1}^{N_r}$ from the real image I_r, where N_r is the total number of patches.
- Extract patches $\{P_f^i\}_{i=1}^{N_f}$ from the generated image I_f, where N_f is the total number of patches.

Note that at each scale we have $N_r = N_f$ and the extraction process is done using nn.Unfold function.

2.1.3 Mean and Covariance Computation

For the set of patches extracted from the real image, we compute the mean vector μ_r and covariance matrix Σ_r as follows:

$$\mu_r = \frac{1}{N_r} \sum_{i=1}^{N_r} P_r^i \tag{7}$$

$$\Sigma_r = \frac{1}{N_r - 1} \sum_{i=1}^{N_r} (P_r^i - \mu_r)(P_r^i - \mu_r)^T \tag{8}$$

Similarly, for the set of patches extracted from the generated image, we compute the mean vector μ_f and covariance matrix Σ_f as:

$$\mu_f = \frac{1}{N_f} \sum_{i=1}^{N_f} P_f^i \tag{9}$$

$$\Sigma_f = \frac{1}{N_f - 1} \sum_{i=1}^{N_f} (P_f^i - \mu_f)(P_f^i - \mu_f)^T \tag{10}$$

2.1.4 Implementation of square root matrix in pytorch

Since there is no predefined function in pytorch that computes the square root of a matrix, and since we need it to compute the loss we proceeded by doing it ourselves. To implement the square root of a matrix in PyTorch, we utilize the eigen decomposition method, which allows us to compute the square root of the covariance matrix while ensuring that gradients can propagate through this operation (`require_grad=True`). Here's the step-by-step process:

1. Compute Eigen Decomposition: First, compute the eigenvalues and eigenvectors of the covariance matrix Σ_f. In PyTorch, this can be achieved using `torch.linalg.eigh()` function[6].

2. Handling Numerical Stability: Due to numerical stability issues, eigenvalues may have small negative values, which should not be the case of the covariance matrix that is always semi-definite positive. To ensure numerical stability, we replace any negative eigenvalues to zero, before taking the square root.

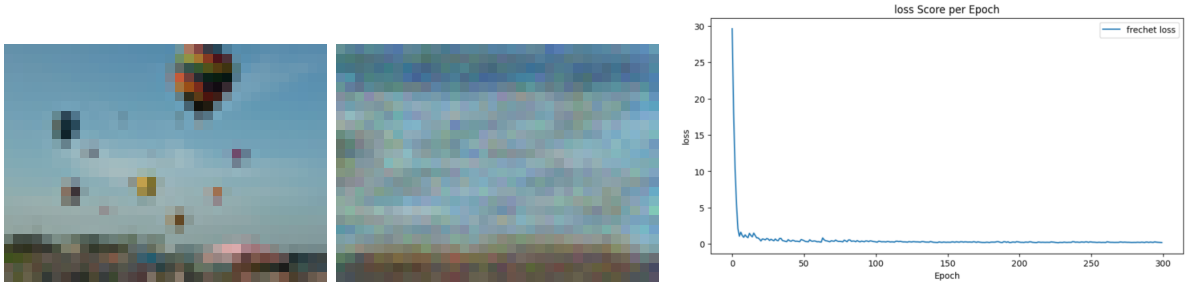
3. Square Root Computation: Compute the square root of the eigenvalues by taking the square root of the clipped eigenvalues.

4. Transforming Back to Original Space: Reconstruct the square root matrix by multiplying the square root of the eigenvalues with the eigenvectors. This operation transforms the square root from the eigenvector space back to the original space.

By implementing the square root of the covariance matrix in this manner, we enable the use of the Frechet Distance as a loss function in the SinGAN framework, facilitating efficient training and optimisation of the generative model.

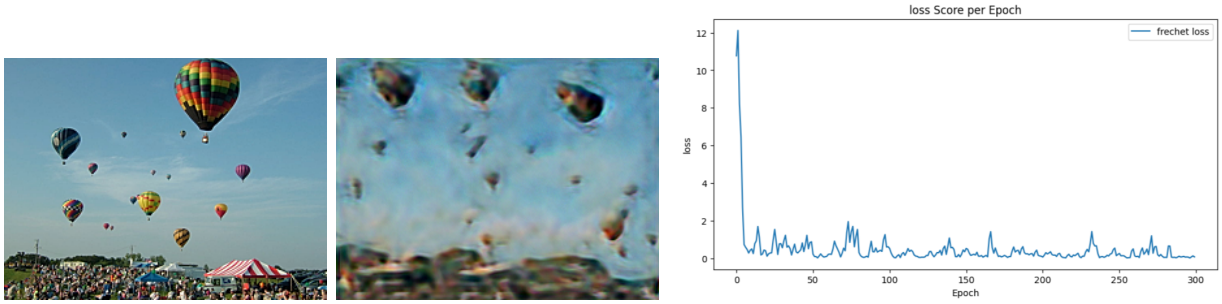
2.1.5 Results and discussion

In order to be able to visualise the results, we proceeded by saving the image generated by the last trained generator in each scale in addition the real image at the same scale. This enables us to visually assess the capabilities of our generator at the end of the training phase for each generator. In the example we will be providing, we have 8 scales. We will show visualise the real and fake images in addition to the obtained loss function in that specific scale. Afterwards, we will plot all the losses obtained across all the scales.



(a) Original image (b) fake image image (c) loss

Figure 7: Generated images with Frechet Distance Loss at scale 1



(a) Original image (b) fake image image (c) loss

Figure 8: Generated images with Frechet Distance Loss at scale 8

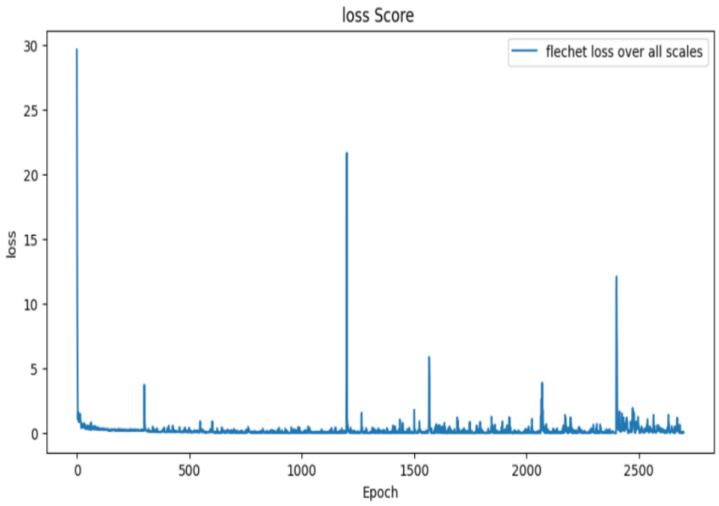


Figure 9: Frechet Distance Loss across all the scales

To begin, it is important to mention that for this loss, the training is done much more faster. This is expected because we have much less parameters to update. Namely, we have dropped all the learning part of the discriminator. In addition, by setting the number of epochs in each scale to 300 we notice that the loss converges very fast and that this number of epochs is more than sufficient. In contrast, in the default implementation of SinGAN we needed around 6000 updates of the generator and 6000 updates of discriminator to reproduce the paper results.

Although the computations were much more efficient, we notice that we sacrificed a huge part of the image appeal. Thus we lost our generation capabilities and the generated images after all the training are not natural. Leading us to conclude that this loss is inadequate for this kind of training. This is to be expected, since we are comparing the distribution of the patches using only the mean and the covariance matrix. This results lead us to try out other substitutes for the discriminator that we will describe in the next sections

2.2 Discriminator Replacement with Nearest Neighbor Patch Loss

2.2.1 Nearest Neighbor Patch Loss

As mentioned in the previous section, the Frechet Distance Loss, which aligns the distribution between patches from the real and fake images, is not sufficient to generate sophisticated results, despite its computational efficiency and speed. To address this, we are introducing a more complex calculation in the form of a new loss function. We propose the **Nearest Neighbor Patch Loss (NNPL)** to tackle this issue. For each patch x_i from the fake image, we assign the patch y_{j*} from the real image that has the minimum distance among all the patches y_j.

$$NNPL(x, y) = \frac{1}{N_x} \sum_{i=1}^{N_x} \min_{j \in \{1,\dots,N_y\}} ||x_i - y_j||^2 \tag{11}$$

where,

- x : fake image
- y : real image
- N_x : number of patches of the fake image x
- N_y : number of patches of the real image y
- x_i : patch number i from the fake image
- y_j : patch number j from the real image

The NNPL measures how well the generator is able to take each patch from the real image and find a place for it in the fake image such that it looks natural and maintains the overall structure and texture of the real image. So, in essence, the NNPL is a way to quantify how good the generator is at creating fake images that convincingly resemble the real one. It's like a scorecard for the generator's performance. The lower the score (loss), the better the performance.

2.2.2 Implementation of the Loss using Pytorch

In the implementation step of this loss we need to be attention at some key points:

- One import point to notice is that there some variables (patches of the real image) that are calculated only one time at each scale so we can store them for faster time or simply define the loss as a class and define these variables as attributes of that class

– We need to recall the same extraction patches using the function `torch.nn.Unfold()` provided by Pytorch[4].

– In order to calculate the distances between the patches we will use the function `torch.cdist()` that calculate the distance between each pair of patches and return a matrix of distances[5].

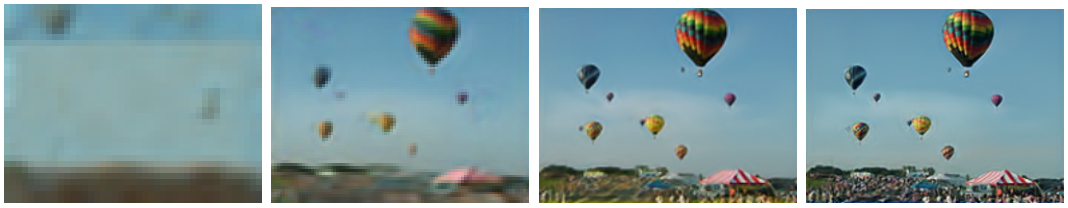
– Finally calculate the minimum distance along each line and return the mean distance.

A significant challenge we encountered when calculating the matrix distance was the large number of patches. With an image size of (3, 256, 256), using a patch size of 11 × 11 with a stride of 1, we ended up with approximately 60,000 patches. Consequently, the resulting distance matrix was a 60,000 × 60,000 matrix, which could potentially exhaust all available memory.
There are several alternatives to address this issue:

1. **Increase the stride:** By increasing the stride to 3 or 5, we can reduce the number of patches.
2. **Random selection of patches:** Another option is to select a random subset of patches for the distance calculation.
3. **Grouping patches:** The last, and potentially most effective solution, involves dividing the patches from the fake image into smaller groups. The calculation is then performed separately for each group. The individual losses are summed and finally divided by the total number of patches. This approach should yield the best results.

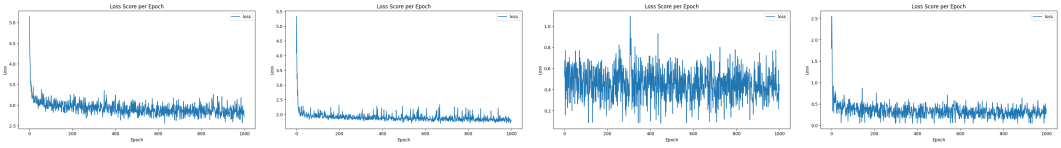
2.2.3 Results and discussion

The figures 10 and 11 illustrate the results with the Nearest Neighbor Patch Loss.



(a) scale 1 (b) scale 4 (c) Scale 7 (d) Scale 8

Figure 10: Generated images with NNPL Loss in different scales



(a) scale 1 (b) scale 4 (c) Scale 7 (d) Scale 8

Figure 11: Evolution of NNPL Loss in different scales

Examining the images generated at different scales, we observe satisfactory results where the images retain the same structure and texture as the real ones. The loss curves further demonstrate these results, showing a decrease in the loss function over the epochs. Although there are some oscillations due to the inherent randomness in the generation process, the overall trend is a decreasing loss across most scales, indicating the effectiveness of our proposed method.

2.3 Discriminator Replacement with Aligned Nearest Neighbor Patch Loss

2.3.1 Aligned Nearest Neighbor Patch Loss

There is an issue with the previous loss function when the distributions of the real and fake patches are not aligned. As a result, the entire fake image may converge to a small portion of the real image that has the least distance. To enhance the previous loss function, we propose aligning the distributions of the real and fake patches as a preprocessing step before calculating the NNPL loss. The new loss is called **Aligned Nearest Neighbor Patch Loss (ANNPL)** and can be expressed mathematically as follows:

$$x' = \phi(x) = \Sigma_y^{1/2}\Sigma_x^{-1/2}(x - \mu_x) + \mu_y \tag{12}$$

$$ANNPL(x, y) = NNPL(x', y) \tag{13}$$

2.3.2 Implementation of the Loss with Pytorch

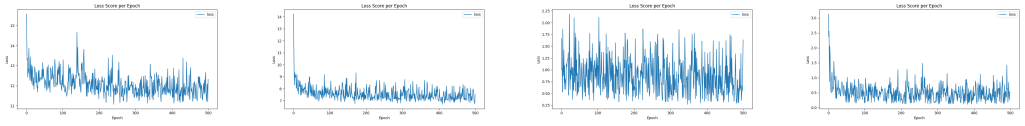
The implementation of this loss in PyTorch is quite similar to the Nearest Neighbor Patch Loss, with just a few additional lines of code. Theoretically, this loss should yield better results, but we were unable to achieve this due to instability issues arising from the calculation of the square root matrix and its inverse at each step. In practice, we opted to use the `torch.nn.BatchNorm1d()` function [7] to Batch Normalize the patches, aligning all patches to have zero mean and unit variance.

2.3.3 Results and discussion

The figures 12 and 13 illustrate the results with the Aligned Nearest Neighbor Patch Loss.

(a) scale 1 (b) scale 4 (c) Scale 7 (d) Scale 8

Figure 12: Generated images with ANNPL Loss in different scales



(a) scale 1 (b) scale 4 (c) Scale 7 (d) Scale 8

Figure 13: Evolution of ANNPL Loss in different scales

The images generated using this loss are not entirely convincing, but they are the best we could achieve. While we obtain a generally good image, many details are lost. The loss curves are similar to those of the previous loss function, exhibiting the same trends and remarks. Although the overall shape of the loss function decreases over epochs, some oscillations remain due to the randomness inherent in the generation process.

Conclusion

This project successfully implemented and enhanced the SinGAN model, a generative adversarial network trained on a single natural image, achieving notable improvements and insights. Our work focused on replicating the original SinGAN model, exploring the effects of hyperparameter and padding modifications, and introducing alternative loss functions for the discriminator.

We faithfully replicated the original SinGAN model, establishing a baseline for subsequent enhancements. This replication verified the model's ability to generate high-quality images from a single input. We discovered that hyperparameter tuning and the choice of padding significantly affect the quality of generated images. Specifically, transitioning from zero-padding to circular-padding highlighted the importance of padding in preserving image structure and coherence, especially near borders.

The Frechet Distance Loss aimed to reduce computational overhead by comparing statistical measures between real and generated images. While it improved training efficiency, it did not maintain the visual quality of generated images. The Nearest Neighbor Patch Loss (NNPL) improved image quality by aligning patches from real and generated images, better preserving the texture and structure of the real images compared to the Frechet Distance Loss. The Aligned Nearest Neighbor Patch Loss (ANNPL), by aligning patch distributions before calculating the NNPL, this approach further enhanced image quality. However, it faced instability issues due to the complexity of distribution alignment. Our results demonstrate that replacing the discriminator with explicit loss functions can significantly enhance computational efficiency while maintaining or improving the quality of generated images. These alternative loss functions offer promising avenues for improving the baseline SinGAN model.

In conclusion, this project not only reproduced the results of the original SinGAN but also introduced significant enhancements that improve both the efficiency and quality of image generation. The insights gained from this study contribute valuable knowledge to the field of single-image generative adversarial networks and suggest potential directions for future research and development.

References

[1] SinGAN: Learning a Generative Model from a Single Natural Image, by Tamar Rott Shaham, Tali Dekel, Tomer Michaeli

[2] https://github.com/tamarott/SinGAN/tree/master

[3] https://pytorch.org/docs/stable/index.html

[4] https://pytorch.org/docs/stable/generated/torch.nn.Unfold.html

[5] https://pytorch.org/docs/stable/generated/torch.cdist.html

[6] https://pytorch.org/docs/stable/linalg.html

[7] https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm1d.html