Ministère de
l'Enseignement Supérieur
et de la Recherche
Scientifique
Université de Carthage
Ecole Polytechnique de
Tunisie

وزارة التعليـم العالي و
البحث العلمي
جامعـة قرطاج
المدرسـة التونسية للتقنيات

Engineering Internship Report

SC-FEGAN : Face Editing Generative Adversarial Network with User's Sketch and Color

1^{th} July - 31^{th} August, 2023

Elaborated by:
Razi HAMDI
Third Year Engineering Student- SISY

Within:

Supervised by: **Mr. Ahmed BADAOUI** **CEO - Vistasy Clinic**

Academic Year
2023 - 2024

Acknowledgement

I would like to express my heartfelt appreciation to **Mr.Ahmed Badaoui**, the CEO of **Vistasy Clinic** and a Computer Vision Engineer. Working closely with him during my internship has been an enriching experience that has greatly enhanced my knowledge in the field. His guidance, expertise, and support have been invaluable in shaping my understanding of computer vision and its applications in generative AI.

I also want to express my gratitude to The **Ecole Polytechnique De Tunisie**, for providing me with the opportunity to undertake this internship. The education and skills I gained at **EPT** laid a strong foundation for this experience, and I am thankful for the dedication of all my teachers who have played a significant role in my development.

Lastly, I wish to thank my family and friends for their unwavering support and encouragement throughout this internship journey.

Abstract

The main goal of this project is to implement a novel face editing system that can generate high-quality synthetic images using intuitive user inputs, such as free-form masks, sketches, and color data. The system is based on a generative adversarial networks (GANs)[1] architecture which is a type of neural network architecture that consists of two networks.

The use of GANs allows the system to generate high-quality synthetic images that are visually appealing and semantically meaningful.

The system can be used for a variety of image editing tasks, including image completion, object removal, and style transfer, and can be trained end-to-end using a combination of loss functions that help to improve the quality and realism of the generated images. The authors demonstrate the effectiveness of the proposed system through a series of experiments and comparisons with state-of-the-art image editing methods.

Keywords : Generative Adversarial Networks, Neural Network, face editing, sketch, free-form mask , color .

Contents

List of Figures

Chapter 1

General Context

1.1 Introduction

Generative Adversarial Networks, or GANs for short, are a class of machine learning models introduced by Ian Goodfellow and his colleagues in 2014. GANs are a type of generative model designed to generate new data samples that are similar to a given dataset. They have found applications in a wide range of fields, including computer vision, natural language processing, and generative art.
Within this type of machine learning model, my engineering internship at Vistasy Clinic aims to implement a scientific paper for editing images under user control.

1.2 Host Company: Vistasy Clinic

Vistasy Clinic is a DeepTech startup that leverages AI to revolutionize healthcare and promote well-being. This startup is composed of researchers, engineers (including Computer Vision Engineers, Integrators, and Embedded Systems Engineers), and business developers. The philosophy of this company is dedicated to creating simulations for people, enabling them to make better and wiser decisions through cutting-edge technology. Finally, the main products of Vistasy Clinic are smart devices named VMirrors, offering a wide range of aesthetic options suitable for the patient and patient-doctor relationship management.
My engineering internship objective is to implement one of the VMirror's features under the supervision of Mr. Ahmed Badaoui.

1.3 Internship scope

This project consists of understanding and implementing, with PyTorch, a scientific paper named SC-FEGAN: Face Editing Generative Adversarial Network with User's Sketch and Color [2] using GANs techniques to generate realistic edited face images based on user guidance.

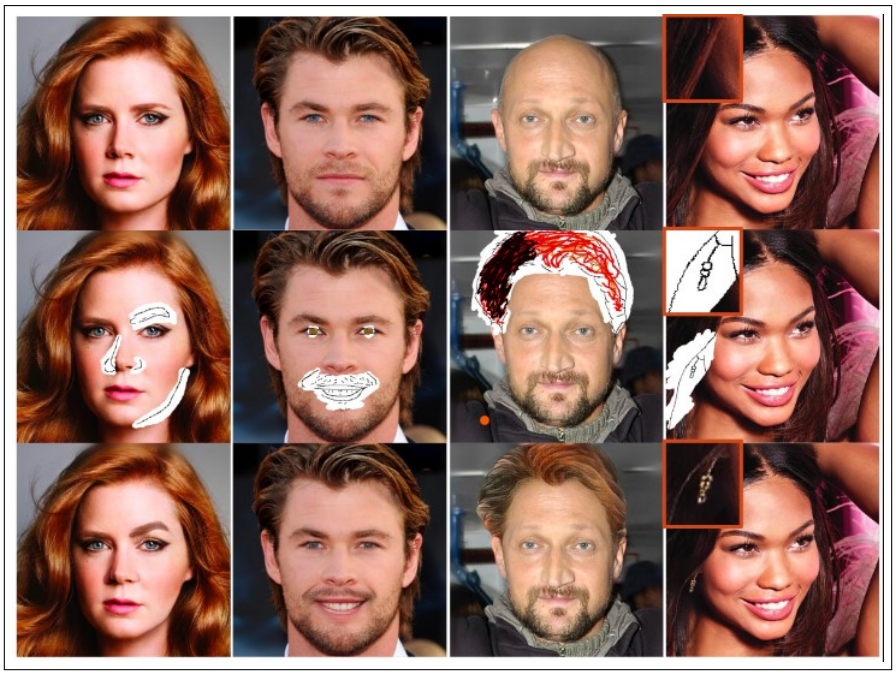


Figure 1.1: Results From Paper

Figure 1.1 represents the original image, a masked image with regions to be edited, and finally, the output of the model with edited regions.
The scope of this internship is to reach the quality of this generated images. ;

Chapter 2

State of the art

2.1 Introduction

The purpose of this chapter is to provide a summary of a bibliographical review on the SC-FEGAN paper's approach. Firstly, we will present an overview of the Generative Adversarial Networks (GANs) framework, which consists of two innovative models: the generator and the discriminator. We will also present the adversarial process involved in training these models.
Finally , we will provide an overview of SC-FEGAN state-of-the-art , along with its custom architecture and various loss functions.

2.2 Generative Adversarial Network's (GANs)

2.2.1 Introduction

Generative Adversarial Networks, or GANs , are an approach to generative modeling using deep learning methods, such as convolutional neural networks.
In general, generative modeling is an unsupervised learning task in machine learning that automatically involves discovering and learning the regularities or patterns in input data. The aim is for the model to generate or output new examples. On the other hand, GANs are a clever way of training a generative model by framing the problem as a supervised learning problem with two sub-models generator and discriminator:

Generator : is a neural network that generates new data instances that are similar to the training data. The generator takes a random noise vector as input and produces a new data instance as output. The goal of the generator is to produce data that is indistinguishable from the real data, so that the discriminator cannot tell the difference between the two.

Discriminator : is a neural network that evaluates the authenticity of a given data instance. The discriminator takes a data instance as input and produces a binary output, indicating whether the input is real or fake. The goal of the discriminator is to correctly

classify data into real and fake data, so that it can distinguish between the two.

The generator and discriminator are trained together in a zero-sum game, where the generator tries to produce data that can confuse the discriminator, and the discriminator tries to correctly classify if the data is real or fake. This adversarial process drives both models to improve their methods until the generated data is indistinguishable from the real data.
In other words, the generator G is trained to capture the data distribution of real samples, while the discriminator D is trained to estimate the probability that a given sample originated from the training data rather than from G.. This is an adversarial process.

2.2.2 Adversarial Process

The adversarial process leads to the training of a generator that can produce data samples that closely match the real data distribution, making GANs a powerful tool for generative modeling and data synthesis.
During training, the generator and discriminator are in a constant battle, where the generator strives to improve its ability to generate realistic data to trick the discriminator, while the discriminator tries to enhance its skill in distinguishing real data from fake data.In other words, the generator tries to diminish the discriminator's ability to accurately classify its generated samples, while the discriminator tries to enhance its accuracy in distinguishing between real and fake samples. In this case, the objective function of GANs becomes a minimax game between the generator G and the discriminator D.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.1}$$

$p_{\text{data}}(x)$ represents the data distribution, and $p_z(z)$ represents the noise distribution.

Algorithm 1 Minibatch Stochastic Gradient Descent Training of Generative Adversarial Nets

Require: Number of training iterations
Require: Number of discriminator update steps, k
1: **for** each training iteration **do**
2: **for** k steps **do**
3: Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
4: Sample minibatch of m examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
5: Update the discriminator by ascending its stochastic gradient:
6: $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D(x^{(i)}) + \log\left(1 - D\left(G(z^{(i)})\right)\right)\right]$
7: Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
8: Update the generator by descending its stochastic gradient:
9: $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G(z^{(i)})\right)\right)$
10: **Note:** The gradient-based updates can use any standard gradient-based learning rule. In our experiments, we used momentum.

The training process of GANS is represented with algorithm 1

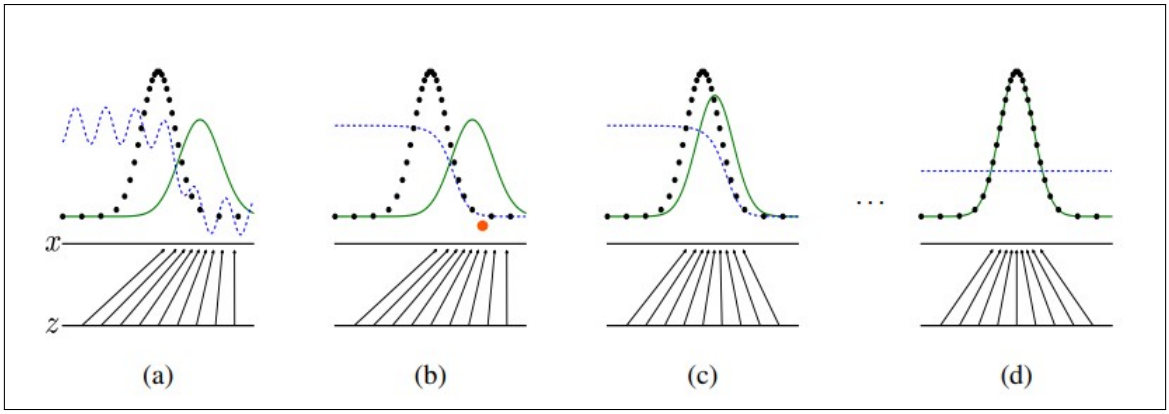


Figure 2.1: the Distribution of Real sample and Fake sample over Training

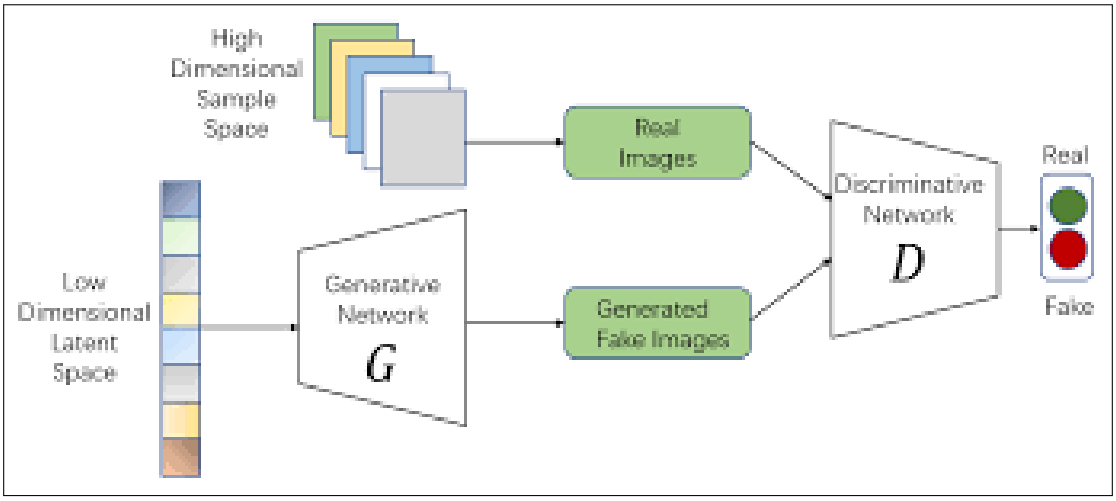


Figure 2.2: Summary for Generative Adversarial Networks

2.3 SC-FEGAN Approach

2.3.1 Introduction

The practice of changing images manually has been around for a long time. Mostly, it involved using handcrafted techniques rather than deep learning approaches. This approach is seen in commercial image editing softwares .

Most of these software tools have set ways of doing things, if you want to make changes to an image, you need to know what you are doing and use a combination of steps to achieve good results. This can be a lot of work, and it requires a good deal of expertise. So, for people who are not experts or do not have time, the traditional way of doing things is not very helpful. It's also a bit boring if you want to make really good changes to your pictures.

But, in addition to the old-fashioned methods, there have been some recent advances in GANs research. These have led to new ways of completing, changing, and transforming images. They work by training GANs on big sets of pictures.One of the novel AI approaches is SC-FEGAN.

2.3.2 Approach

The approach of the SC-FEGAN paper is to propose a neural network-based face image editing system that can generate high-quality synthetic images with realistic details. The system is designed to take as input a combination of sketch and color data, and can handle incomplete or missing image data using a combination of loss functions, including the adversarial loss, per-pix loss, total variation loss,style loss, and perceptual loss.
The system is based on a generative adversarial network (GAN) architecture, which consists of a generator network and a discriminator network. The generator network takes as input an incomplete image, a binary free-form mask,color map , and a sketch image, It generates a completed image that matches the missing regions of the input image. The discriminator network tries to distinguish between real and fake images, and provides feedback to the generator network to improve the quality and realism of the generated images.
In the next subsections, we will discuss the inputs of the generator and how to create them for training.

2.3.3 Inputs of generator

Incomplete Image

The incomplete image input is an image with a missing or incomplete region that needs to be completed by the generator . It serves as the basis for the generated image, and helps the generator to understand the context and content of the missing region. The incomplete image input can be any type of image and it can be edited using a combination of free-form mask, sketch, and color inputs.

Figure 2.3: Incomplete image

Free-form Mask

The free-form mask input is a type of mask that allows users to indicate the missing region of the input image using a free-form shape. Unlike traditional masks, which are typically rectangular or circular in shape, free-form masks can take on any shape or size, allowing users to provide more precise and detailed input to the generator. This type of masks allows users to quickly and easily indicate the desired shape and location of the missing region.

Figure 2.4: Free-form mask

Binary Sketch

The binary sketch input is a black and white image that describes the structure of the missing or incomplete region in the input image. It helps the generator to understand the shape and layout of the missing region, and guides the generation of the completed image. The sketch is an allows users to quickly and easily indicate the desired shape and structure of the missing region.

Figure 2.5: Binary sketch

Color Map

color map input is a color image that provides additional color information for the missing or incomplete region in the input image. It helps the generator network to generate a completed image that matches the color and texture of the surrounding regions, and ensures that the generated image is visually coherent and realistic.

Figure 2.6: Color map

2.3.4 Data Preparation

For the training of our networks, we utilized the CelebA-HQ dataset, which comprises 25,556 high-resolution celebrity images. These images were resized to 512x512 pixels. The 25,556 images serve as target data. From these images, we generated input data for our generator For the training process, including free-form masks, sketches, and color maps.

Free-form masks generation

When training on the facial images, we randomly applied a free draw mask with eye-positions as a starting point in order to express complex parts of the eyes. We also randomly added hair mask using GFC[3] .
GFC is a type of generative model that is specifically designed for face segmentation tasks. It is trained on a large dataset of face images and is able to generate realistic hair masks that can be used to complete incomplete images. By using GFC in the free-form masking process, SC-FEGAN is able to generate high-quality synthetic images with realistic hair details.
Details of complete free-form masks are described in Algorithm 2.

Algorithm 2 Free-form Masking with Eye-Positions

$maxDraw, maxLine, maxAngle, maxLength$ are hyperparameters
$GFCHair$ is the GFC for getting the hair mask of the input image
$Mask \leftarrow$ zeros($inputSize$, $inputSize$)
$HairMask \leftarrow GFCHair(InputImage)$
$numLine \leftarrow$ random.range($maxDraw$)
for $i \leftarrow 0$ to $numLine$ **do**
    $startX \leftarrow$ random.range($inputSize$)
    $startY \leftarrow$ random.range($inputSize$)
    $startAngle \leftarrow$ random.range(360)
    $numV \leftarrow$ random.range($maxLine$)
    for $j \leftarrow 0$ to $numV$ **do**
        $angleP \leftarrow$ random.range($-maxAngle, maxAngle$)
        if j is even **then**
            $angle \leftarrow startAngle + angleP$
        else
            $angle \leftarrow startAngle + angleP + 180$
        $length \leftarrow$ random.range($maxLength$)
        Draw a line on $Mask$ from point ($startX$, $startY$) with angle and length.
        $startX \leftarrow startX + length \cdot \sin(angle)$
        $startY \leftarrow startY + length \cdot \cos(angle)$
    Draw a line on $Mask$ from an eye position randomly.
$Mask \leftarrow Mask + HairMask$ (randomly)

Sketches and color maps generation

Sketches are generated using the HED[4] (Holistically-Nested Edge Detection) which is a type of convolutional neural network (CNN) that is specifically designed for edge detection tasks. It is trained on a large dataset of images and is able to detect edges in an image with high accuracy. HED is unique in that it uses a holistic approach to edge detection, meaning that it considers edges at multiple scales and resolutions simultaneously. This allows it to detect edges that may be missed by other edge detection methods.

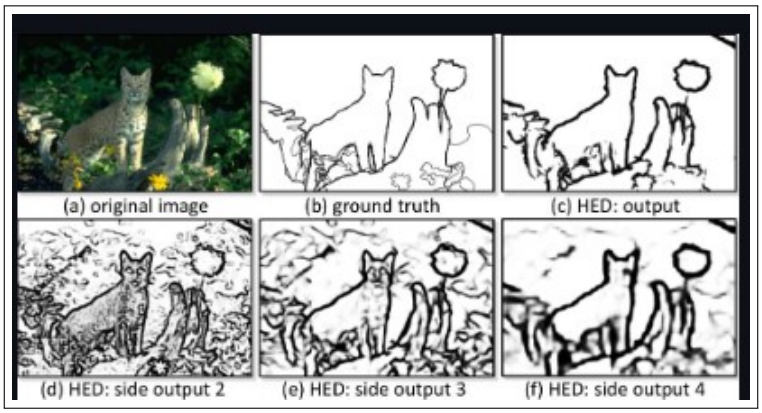
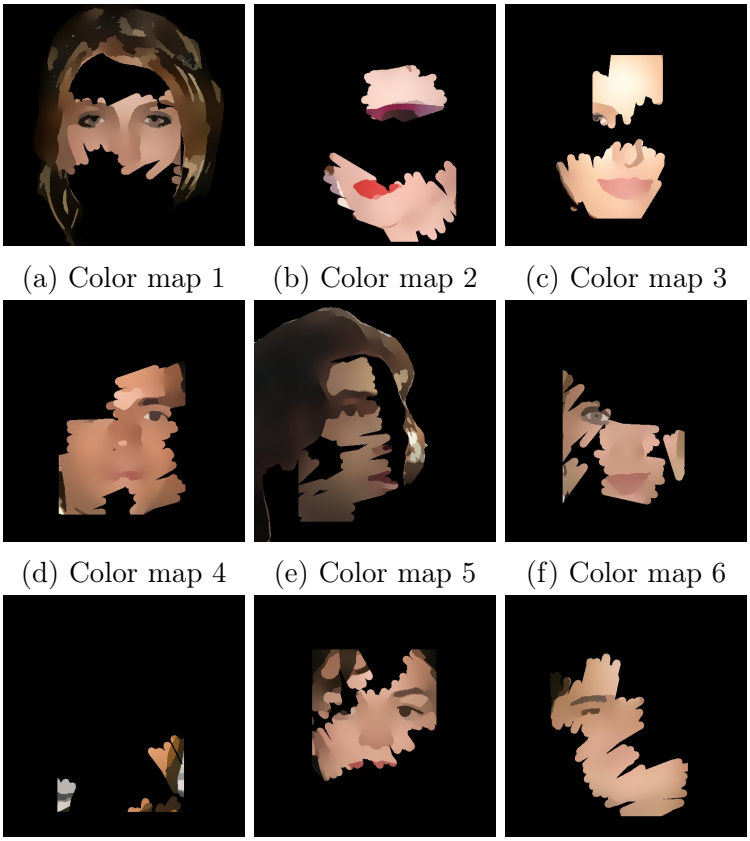


Figure 2.7: Example of HED Outputs

the generation of color maps inputs takes a series of steps . Initially, the images un-

derwent smoothing by applying a median filter with a size of 3, followed by 20 iterations of a bilateral filter to further reduce detail. Subsequently, After that, GFC was used to segment the face, and each segmented parts were replaced with the median color of the corresponding parts.



(a) Color map 1 (b) Color map 2 (c) Color map 3

(d) Color map 4 (e) Color map 5 (f) Color map 6

(g) Color map 7 (h) Color map 8 (i) Color map 9

Figure 2.8: Sample of color map inputs

2.3.5 Comparaison with approaches

Compared to other image completion methods, SC-FEGAN has several advantages. First, it is able to generate high-quality synthetic images using intuitive user inputs, such as sketches and colors.Also it is able to generate realistic results,even when large portions of the image are missing and finally SC-FEGAN allows users to generate images that reflect their own creative vision by using free-form masking rather than being limited by predefined input formats.

To summarize, what makes SC-FEGAN the best completion approach are the architectures of its generator and discriminator, as well as the losses of these two models that we will discuss in the next chapter.

Chapter 3

MODELS And LOSSES

3.1 Introduction

In this chapter, we will present the architectures of the generator and discriminator, as well as the techniques used to make these models more stable and performant like spectral normalization and local response normalization . We will also present the losses used and discuss the role and impact of each loss on the generated image.

3.2 Models Architectures

3.2.1 Definitions

U-net architecture

The U-Net[5] architecture is a deep learning model primarily used for semantic segmentation tasks, but also can be adapted for image completion and other computer vision tasks. The U-net architecture consists of :

- **Encoder (Contracting Path):**This part of the U-Net captures context and extracts features from the input image through a series of convolutional layers followed by ReLU activations and max-pooling layers. The number of feature maps increases as you go deeper into the network.
- **Bottleneck (Bridge):**At the bottom of the U-Net is a bottleneck layer that compresses spatial information.
- **Decoder (Expansive Path):** The decoder part of the U-Net up-samples the feature maps, gradually increasing the spatial resolution. Feature maps from the contracting path are concatenated to those in the expansive path to refine output.
- **Final Layer:**The U-Net typically ends with a convolutional layer using a sigmoid activation function to produce a segmentation map or a completed image.

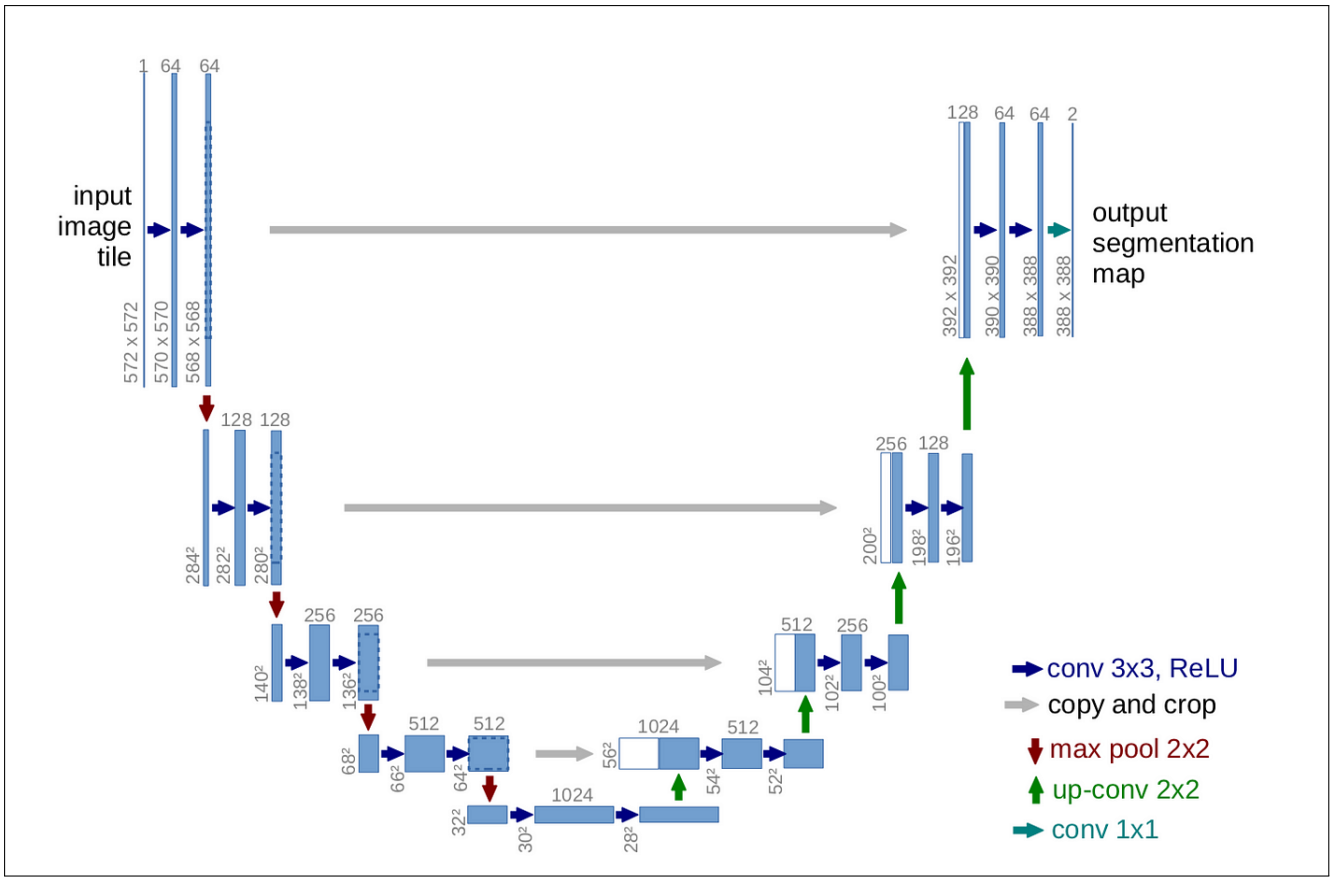


Figure 3.1: Classic U-net architecture

Local Response Normalization

The primary purpose of LRN is to normalize the activity of neurons within the same local region of a feature map in a convolutional layer. This normalization can help improve the generalization and performance of the network by ensuring that the neurons do not become too activated or saturated.
The normalization formula used in LRN can vary, but a common formula is as follows:

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1+n/2)} (a^i_{x,y})^2 \right)^{\beta}$$

where:

N is the total number of kernels in the layer

k, n, α, β are hyperparameters

$a^i_{x,y}$ is the activity of a neuron computed by applying kernel i at position (x,y)

$b^i_{x,y}$ is the response-normalized activity

Gated convolution

Gated convolution[6] is a type of convolutional layer that is designed to selectively filter information from the input. It is composed of two parallel convolutional layers, one for the input and one for the gate, which controls the flow of information.

The use of this convolution has several advantages. First, it allows the network to selectively filter information from the input, which can help to reduce noise and improve the quality of the output. Second, it allows the network to learn more complex features by selectively combining information from different parts of the input. Finally, it can help to reduce the number of parameters in the network, which can improve training speed and reduce overfitting.

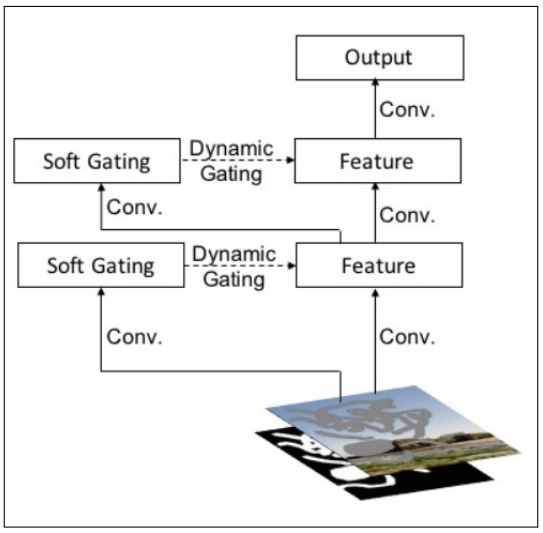


Figure 3.2: Gated Convolution

Spectral Normalization

Spectral normalization is a weight normalization technique that is used to stabilize the training of Generative Adversarial Networks (GANs). It works by normalizing the weight matrix of each layer by its spectral norm, which is the maximum singular value of that matrix. This normalization ensures that the transformation of each layer does not become too sensitive in any one direction, which can help stabilize the training of GANs. The utility of spectral normalization lies in its ability to improve the performance of GANs by constraining the Lipschitz constant of the discriminator, which in turn helps to prevent the discriminator from overpowering the generator. This can lead to better quality generated images and more stable training of GANs.

$$\bar{W} = \frac{W}{\sigma(W)}$$

Where

W is the weight matrix
$\bar{W}$ is the normalized weight matrix.
$\sigma(W)$ is the singular value of W

PatchGAN

PatchGAN is a type of discriminator used in GANs for image synthesis tasks. It is designed to classify whether each patch of an image is real or fake, rather than classifying the entire image as real or fake. This allows the discriminator to provide more detailed feedback to the generator, which can help to improve the quality of the generated images.

The PatchGAN architecture consists of a series of convolutional layers followed by a sigmoid activation function. The output of the network is a matrix of values between 0 and 1, where each value represents the probability that a given patch of the input image is real or fake.

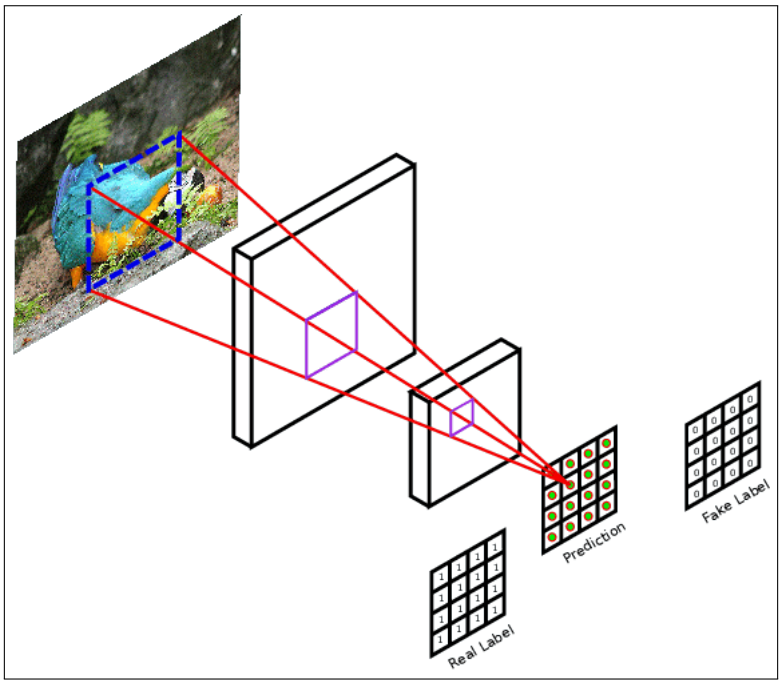


Figure 3.3: Patch Classification

3.2.2 Generator

the generator in SC-FEGAN is based on the U-net architecture and consists of 16 convolutional layers. The generator takes as input a tensor of size 512x512x9, which includes an incomplete RGB channel image with a removed region to be edited, a binary sketch that describes the structure of removed parts, an RGB color map, a binary mask, and noise.

The encoder of the generator downsamples the input 7 times using 2 stride kernel convolutions, followed by dilated convolutions before upsampling. The decoder uses transposed convolutions for upsampling.Then, skip connections were added to allow concatenation with previous layer with the same spatial resolution. We used the leaky ReLU activation function after each layer except for the output layer, which uses a tanh function.

All convolution layers use gated convolution with 3x3 size kernel, and local signal normalization (LRN) is applied after feature map convolution layers excluding other soft gates. LRN is applied to all convolution layers except input and output layers.

The generator is trained with several loss functions, including per-pixel losses, perceptual loss, style loss, total variance loss, and the generic GAN loss function. The generator is also designed to replace the remaining parts of the image outside the mask with the input image before applying the loss functions to it. This replacement allows the generator to be trained on the edited region exclusively.

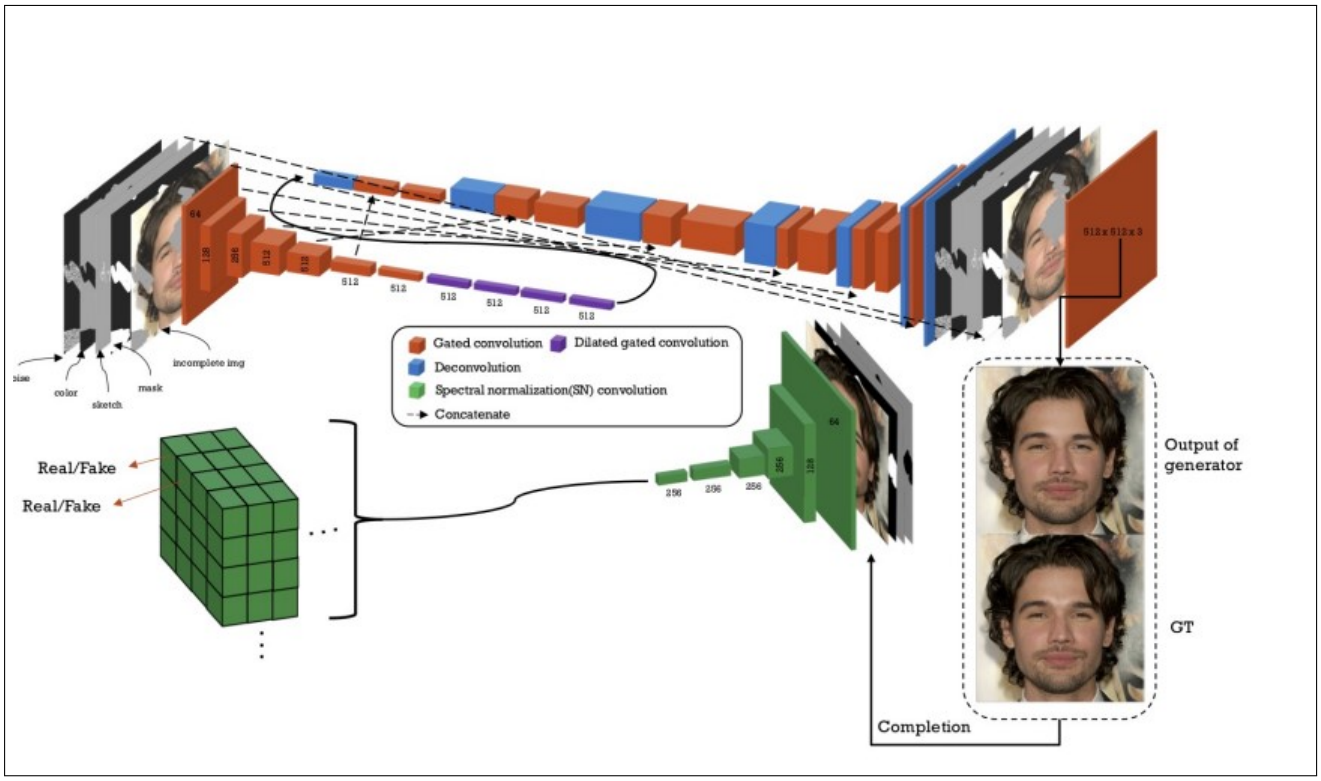


Figure 3.4: Generator Architecture

3.2.3 Discriminator

SC-FEGAN uses a SN-patchGAN[6] discriminator, this type of discriminator is designed to address and improve on the awkward edges of generated images.

The SN-PatchGAN discriminator is a modification of the PatchGAN discriminator that

uses spectral normalization to constrain the Lipschitz constant of the discriminator, this helps to prevent the discriminator from overpowering the generator and can lead to better quality generated images and more stable training of GANs

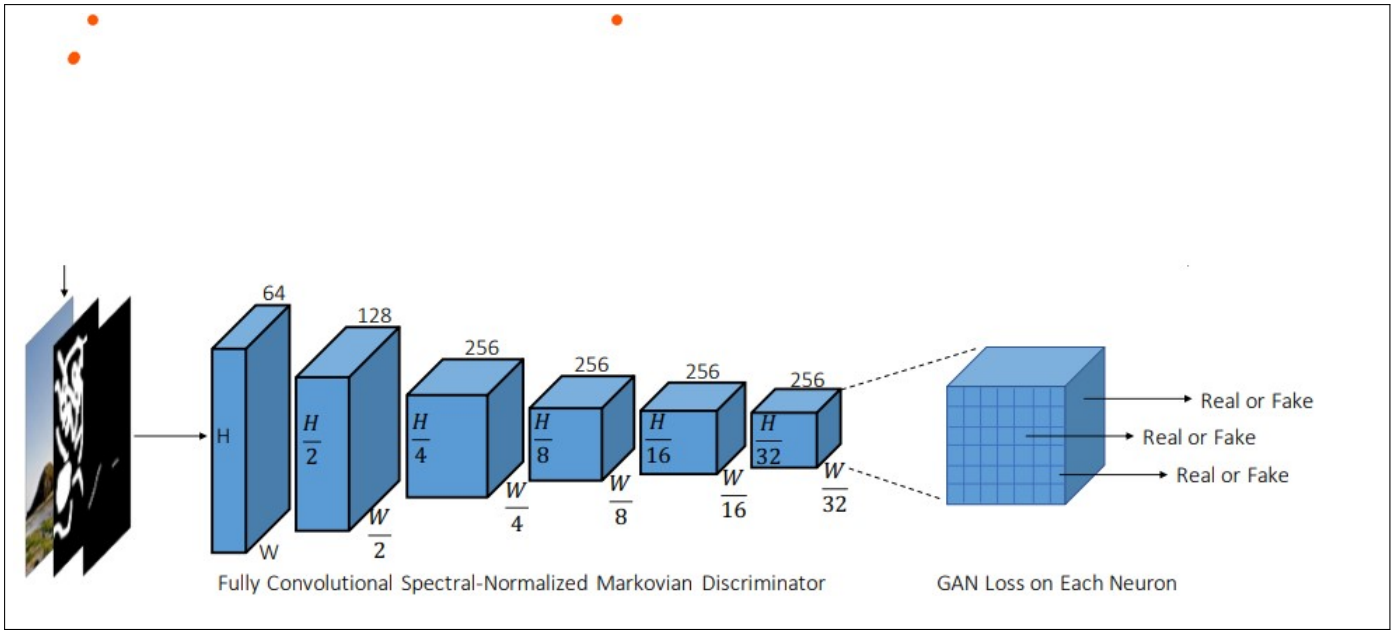


Figure 3.5: Discriminator Architecture

3.3 Loss Functions

SC-FEGAN uses several loss functions to train the generator and discriminator networks. These include per-pixel losses, perceptual loss, style loss, total variance loss, and the generic GAN loss function.
Each of these losses has a specific purpose, and the combination of these losses helps generate high-quality images. In addition to the model architectures, these losses make the SC-FEGAN approach better than previous approaches. We will discuss each of these losses.

3.3.1 Generator losses

Per-pixel loss

Per-pixel loss is a loss function used to measure the difference between the generated image and the ground truth image at the pixel level. It is computed as the L1 distance between the generated image and the ground truth image, and is used to ensure that the generated image is similar to the ground truth image in terms of color and intensity. The per-pixel loss is an important component of the overall loss function used to train the generator network in SC-FEGAN.
By minimizing the per-pixel loss, the generator network is able to learn to fill in the

missing or incomplete regions of the image

$$L_{\text{per-pixel}} = \frac{1}{N_{\text{Igt}}} \sum_k M \odot \|I_{\text{gen}} - I_{\text{gt}}\|_1 + \alpha \frac{1}{N_{\text{Igt}}} \sum_k (1 - M) \odot \|I_{\text{gen}} - I_{\text{gt}}\|_1 \tag{3.1}$$

where, N_a is the number elements of feature a, M is the binary mask map and I_{gen} is the output of generator and I_{gt} the ground truth. We used the factor $\alpha > 1$ to give more weight to the loss on the erased part.

Perceptual loss

Perceptual loss[7] is a loss function used to measure the difference between the generated image and the ground truth image in terms of perceptual similarity. It is computed by projecting the images into feature spaces using a pre-trained VGG-16 network, and then computing the L1 distance between the feature maps of the generated image and the ground truth image. The perceptual loss is used to ensure that the generated image is perceptually similar to the ground truth image, even if the pixel values are not exactly the same.
By minimizing the perceptual loss, the generator network is able to learn to produce images that are visually similar to the ground truth images, even if they are not identical at the pixel level.

$$L_{\text{percept}} = \frac{1}{N_{\Theta_q(I_{gt})}} \sum_q \|\Theta_q(I_{gen}) - \Theta_q(I_{gt})\|_1 + \frac{1}{N_{\Theta_q(I_{gt})}} \sum_q \|\Theta_q(I_{comp}) - \Theta_q(I_{gt})\|_1 \tag{3.2}$$

Here,$\Theta_q(x)$ is the feature map of the q-th layer of VGG16 given that input x is given and I_{comp} is the completion image of I_{gen} with the non-erased parts directly set to the ground truth. q is the selected layers from VGG-16,and we used layers of pool1, pool2 and pool3.

Style loss

Style loss is a loss function used to measure the difference between the generated image and the ground truth image in terms of style similarity. It is computed by comparing the Gram matrices of the feature maps of the generated image and the ground truth image, which capture the correlations between different feature maps. The style loss is used to ensure that the generated image has a similar style to the ground truth image, even if the content is different.The Gram matrix is a matrix that captures the correlations between different feature maps in a convolutional neural network.
By minimizing the style loss, the generator network is able to learn to produce images that have a similar texture, color, and other stylistic features to the ground truth images.

$$L_{\text{style}}(I) = \sum_q \frac{1}{C_q C_q} \frac{1}{N_q} \|G_q(I) - G_q(I_{gt})\|_1 \tag{3.3}$$

where the $G_q(x) = (\theta_q(x))^T(\theta_q(x))$ is the Gram matrix to perform an autocorrelation on each feature maps of VGG-16. When the feature is of shape $H_q \times W_q \times C_q$, the output of Gram matrix is of shape $C_q \times C_q$.

Total Variation Loss

Total variation loss is a regularization term used in image processing to reduce the presence of checkerboard artifacts in the output image. It is computed as the sum of the total variation loss in the column direction and the row direction. The total variation loss is defined as the L1 distance between adjacent pixels in the image.
The role of total variation loss is to improve the quality of the generated images by reducing the presence of checkerboard artifacts that may appear in the output image. $L_{tv} = L_{tv-col} + L_{tv-row}$ is total variation loss.
Where:

$$L_{\text{tv-col}} = \sum_{(i,j)\in R} \frac{1}{N_{\text{comp}}} \|I^{\text{i,j+1}}_{comp} - I^{\text{i,j}}_{comp}\|_1 \tag{3.4}$$

$$L_{\text{tv-row}} = \sum_{(i,j)\in R} \frac{1}{N_{\text{comp}}} \|I^{\text{i+1,j}}_{comp} - I^{\text{i,j}}_{comp}\|_1 \tag{3.5}$$

Adversarial loss (Hinge Loss)

Hinge loss, also known as the margin loss or max-margin loss, is a preferred loss function for image completion tasks over the classic Generative Adversarial Network (GAN) loss for several reasons. Firstly, it offers improved training stability, addressing issues like vanishing gradients that can hinder GAN training. Secondly, it effectively mitigates mode collapse, a common problem in GANs where they generate a limited set of samples, by encouraging the generation of diverse samples through margin maximization between real and generated data distributions. This makes hinge loss a valuable choice for smoother training dynamics and better performance in image completion applications.

$$L_{G-H} = -\beta \mathbb{E}\left[D(I_{\text{comp}})\right] + \epsilon \mathbb{E}[D(I_{\text{gt}})^2] \tag{3.6}$$

Where, L_{G-H} is the generator hinge loss, β and ϵ are hyperparamaters.

Total Generator Loss

The total generator loss is a weighted sum that encompasses all the previous losses. This loss is the main key to generating realistic images.

$$L_G = L_{per-pixel} + \sigma L_{percept} + L_{G-H} + \gamma(L_{style}(I_{gen}) + L_{style}(I_{comp})]) + \nu L_{tv}$$

(3.7)

3.3.2 Discriminator Loss

The total discriminator loss is a combination of the hinge loss and the gradient penalty loss .

$$L_D = \mathbb{E}[1 - D(I_{\text{gt}})] + \mathbb{E}[1 + D(I_{\text{comp}})] + \theta L_{GP} \tag{3.8}$$

Gradient penalty

The gradient penalty is an additional regularization term added to the training process. The gradient penalty encourages the Lipschitz continuity of the discriminator, preventing it from being too sensitive to small perturbations in the input samples. This helps stabilize the training and improves the overall quality of generated samples.

$$L_{GP} = \mathbb{E}[(\|\nabla_U D(U) \odot M\|_2 - 1)^2] \tag{3.9}$$

Here, **U** is a data point uniformly sampled along the straight line between discriminator inputs from I_{comp} and I_{gt}. This term is critical to quality of synthetic image in our case.

We used $\sigma = 0.05, \beta = 0.001, \gamma = 120, \nu = 0.1, \epsilon = 0.001, \theta = 10$.

Chapter 4

IMPLEMENTATION AND RESULTS

4.1 Introduction

In this chapter, we will implement the architectures of the previous models and their associated loss functions. Additionally, we will create a custom dataset class that includes methods for preprocessing the dataset. All of this will be accomplished using object-oriented programming with PyTorch. Furthermore, we will discuss the custom architectural changes made to both the generator and discriminator, highlighting the impact of these modifications on their performance.
We will also provide an overview of the training process, showcasing how the quality of the generated images improves over time. Finally, we will present the results obtained from the most recently saved model.

4.2 Implementation Details

4.2.1 Hardware Environment

The successful implementation of SC-FEGAN relies heavily on the computational resources available for training and inference.
In this subsection, we will provide an overview of the hardware environment utilized during the internship.

Training Envirement

The primary training environment employed for this internship was Amazon Web Services (AWS) SageMaker. The SageMaker instance was equipped with four NVIDIA V100 GPUs.
These GPUs are renowned for their high computational capacity and efficiency, making them well-suited for accelerating the training process of our project. The parallel processing capabilities of the V100 GPUs significantly expedited the convergence of our models, reducing training times and enabling experimentation with larger and more complex architectures.

Testing Envirement

For testing and fine-tuning purposes, a Kaggle environment was leveraged, featuring two NVIDIA T4 GPUs. These GPUs, although less powerful than the V100s, were suitable for smaller-scale experiments and rapid Envirement.

4.2.2 Software Envirement

This subsection offers an in-depth overview of the software components and tools leveraged throughout the project.

Programming language

In this project, Python was chosen due to the rich ecosystem of deep learning libraries and frameworks. Python's simplicity, readability, and strong community support played a crucial role in simplifying and expediting the implementation process.

Deep Learning Framework: PyTorch

PyTorch provided an intuitive and efficient platform for defining and training complex neural network architectures. The framework's extensive library of pre-built modules and utilities, as well as its automatic differentiation capabilities, expedited the development of the GAN model.

Libraries Used

Several essential libraries complemented the PyTorch framework, enhancing our capability to manipulate and process data, visualize results, and optimize the training process. These key libraries include:

- **Matplotlib:** Matplotlib played a pivotal role in visualizing training progress, displaying generated images, and providing insights into the GAN's progress.
- **torchvision:** As an extension of PyTorch, torchvision supplied critical tools for working with image datasets and applying data augmentation techniques. This library streamlined the preprocessing of the face dataset and ensured compatibility with PyTorch's data loading mechanisms.
- **OpenCV:** OpenCV, a renowned computer vision library, was instrumental in various image preprocessing tasks. It provided tools for resizing, data augmentation, and image manipulation, contributing to the overall quality of the training data.
- **os:** The built-in Python library os was used for file and directory operations, simplifying tasks such as dataset management, model checkpointing, and result logging

4.3 Data Preprocessing

This subsection outlines the construction of the dataset named Custom-fegan-dataset and the data preprocessing steps undertaken to prepare it for training.
This dataset incorporates various input components, such as image inputs, color inputs, mask inputs, sketch inputs, and noise inputs, along with corresponding ground truth images. A total of 25,556 samples were organized within distinct subdirectories for efficient data management.
Data preprocessing steps were applied to the dataset, including image loading, conversion of mask and sketch inputs to grayscale, and data normalization. These preprocessing steps ensure that the input data is appropriately formatted and scaled for GAN training. Each data sample consists of a generator input, a mask input, a sketch input, and a ground truth image.

```
class Custom_fegan_dataset(Dataset):
    def __init__(self, root_dir):
        ...

    def convert_to_grayscale(self, image):
        ...

    def __len__(self):
        ...

    def generator_input(self,image_input,mask_input,sketch_input,
   color_input,noise_input):
         ...


    def __getitem__(self, idx):
        ...
```

Listing 4.1: Custom Dataset

4.4 Models Training

This subsection provides an overview of the training process, including key hyperparameters, model initialization, and the role of the DataLoader.

Dataset Loading

The custom dataset, **Custom-fegan-dataset**, is loaded from a specified directory, containing image inputs, color inputs, mask inputs, sketch inputs, noise inputs, and corresponding ground truth images.

Dataloader

The DataLoader is used to efficiently load and manage the dataset during training. It handles data loading in batches, shuffling, and parallelism (if multiple GPUs are avail-

able). This ensures that the model is trained with batches of data, preventing memory issues and introducing randomness to improve generalization.

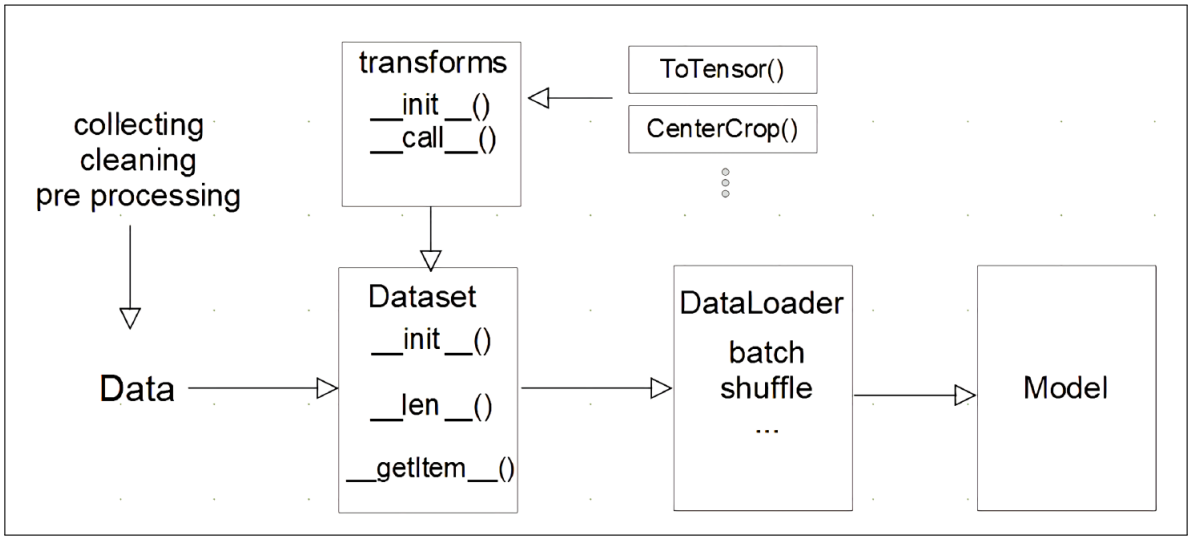


Figure 4.1: Summary for Data Loading to the Models during Training

Optimizers Setup

Optimizers for both the generator and discriminator are created. Adam optimizers are used with specific learning rates for each network. Learning rate schedulers are also applied to adjust the learning rates during training.

Weight Initialization

The generator and discriminator models are initialized with weights using a custom weight initialization function. This ensures that the neural networks start with appropriate initial conditions for training.

Training Loop

The training process involves an iterative loop over the dataset. During each iteration, generator input and ground truth images are used to update both the generator and discriminator. Loss functions are calculated to guide the optimization process.

Visualization

Intermediate and final results are visualized, showing generated images alongside ground truth images. Visualizations help in understanding the evolution of the GAN's performance during training.

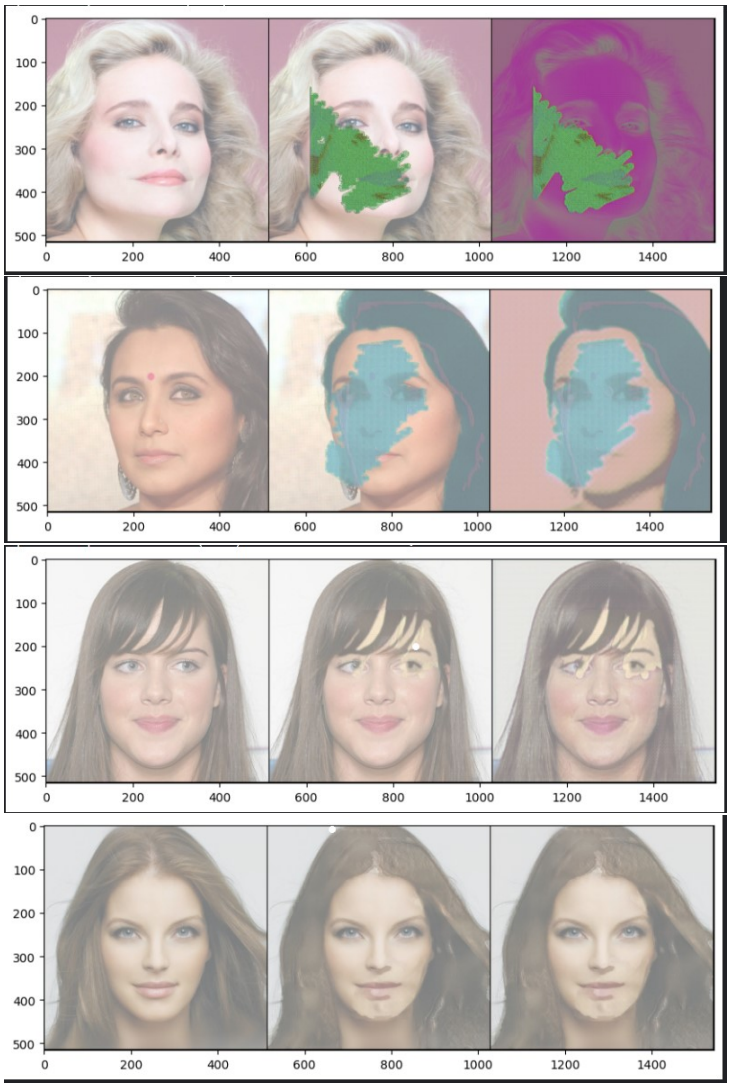


Figure 4.2: Example of image generation during training

Note : The first column represents the ground truth images , the second represents the completed image and the last column represents the generated image.

4.5 Results and Evaluation

In this section, we present the results obtained from the trained **SC-FEGAN** model for face generation.
These results include images generated from the test set using the latest model provided by the supervisor, visualizations of the training progress through generator and discriminator loss plots.

4.5.1 Results

In this subsection, we showcase the visual outcomes of the SC-FEGAN model, which is the latest model provided by the supervisor. These results include images generated from the test set, offering insights into the model's performance in producing realistic facial images.

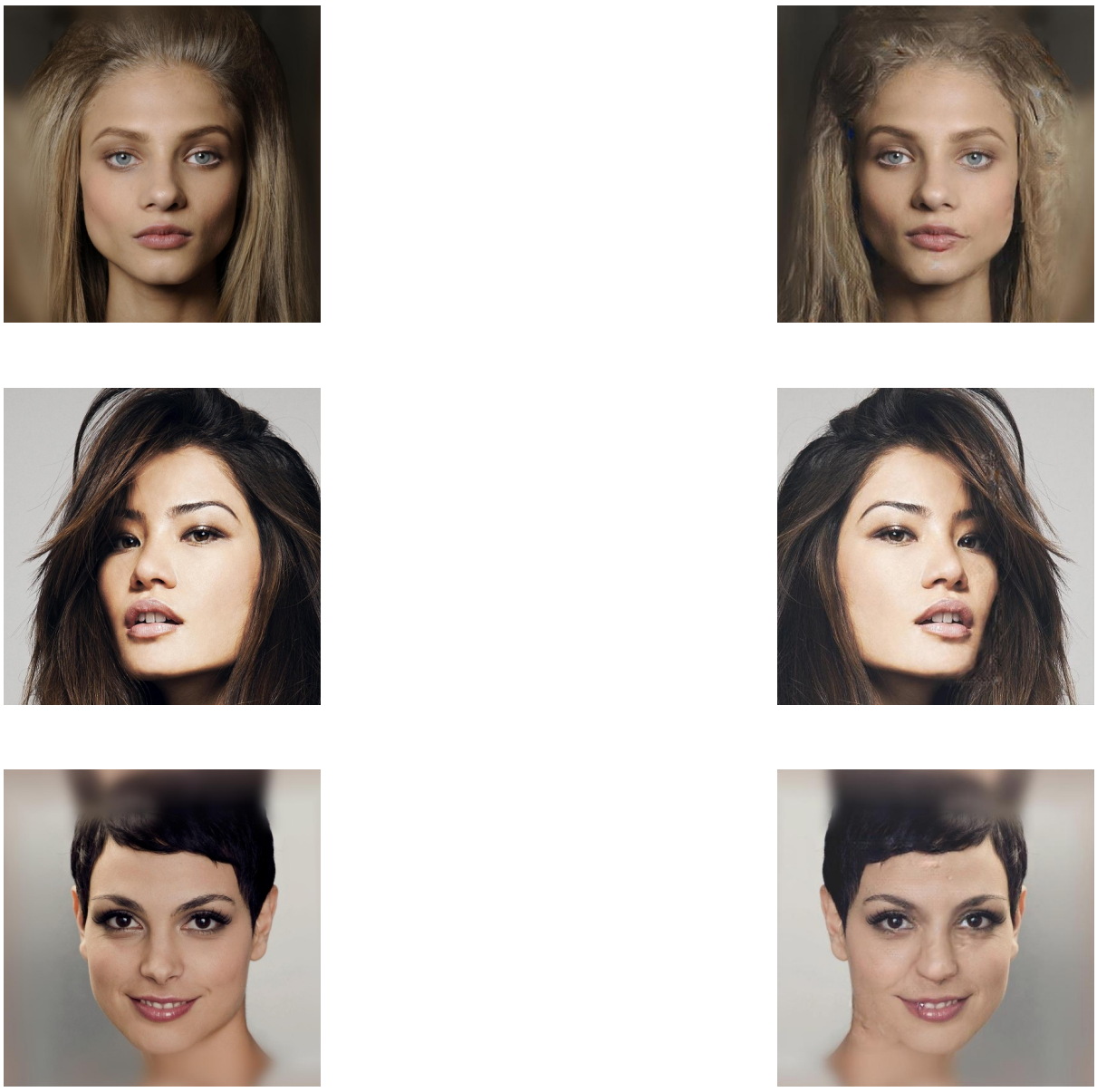


Figure 4.3: Comparison between Ground truth Images and Generated Images

The quality of the generated images can potentially be further improved through continued training. Our model may benefit from additional training iterations, allowing to capture finer details . It can produce even more realistic results.

4.5.2 Evaluation: Loss Monitoring

In this subsection, we delve into the evaluation of the GAN model's performance by monitoring the generator and discriminator loss values over the training epochs.

Generator loss

The generator loss curve visually illustrates how well the generator network is learning to produce realistic facial images. It measures how close the generated images are to real images. A lower generator loss indicates that the generator is getting better at mimicking real images.

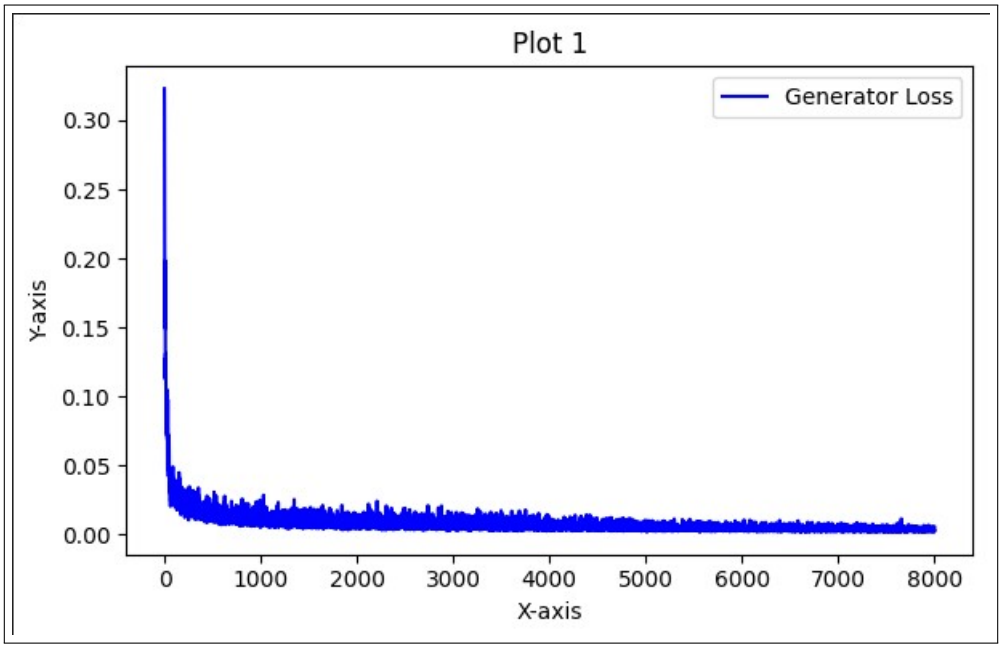


Figure 4.4: Generator loss

The generator loss curve exhibits a general decreasing trend, showcasing the model's improvement. However, occasional fluctuations in loss values are normal and are part of the training dynamics.

Discriminator loss

The discriminator loss curve represents how well the discriminator network is performing in its task of distinguishing between real and generated images. A decreasing discriminator loss suggests that the discriminator is becoming more skilled at telling real from fake images.

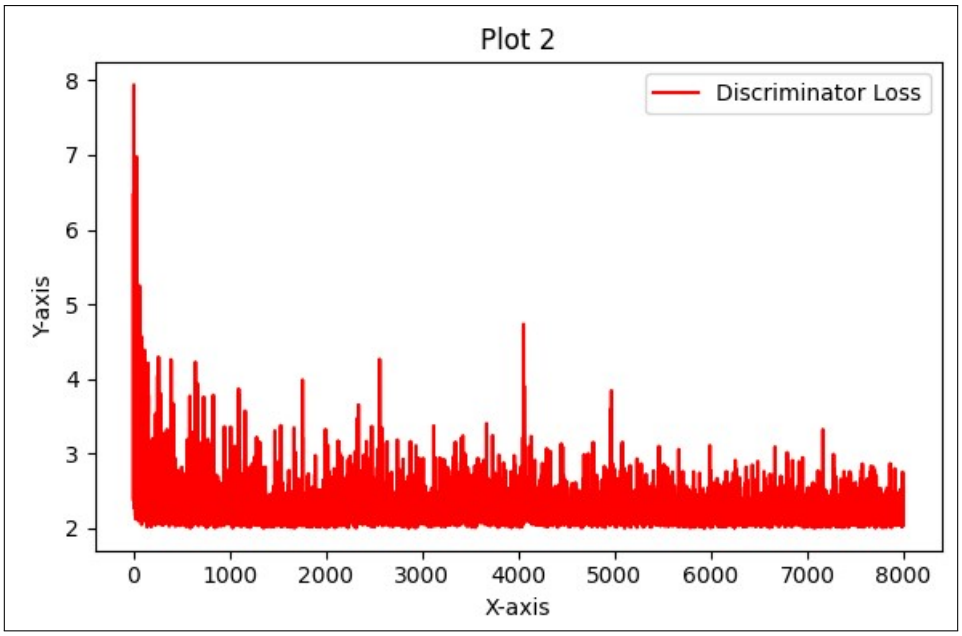


Figure 4.5: Discriminator Loss

Similar to the generator loss curve, the discriminator loss may experience occasional fluctuations. This behavior is expected in the adversarial training setup as the discriminator responds to changes in the generator's performance.

Note: It is important to mention that the loss plots presented here are generated from a generator and discriminator trained on a batch of 400 samples. Training with the entire dataset would significantly extend the training time, as each epoch already requires a substantial amount of computation. Parallel training on 4 GPUs has been utilized to expedite the process, but even then, each epoch can take more than two hours.

General Conclusion and perspectives

In this general conclusion, we encapsulate the key findings and outcomes of the project. We successfully navigated through the implementation and evaluation of SC-FEGAN: Face Editing Generative Adversarial Network with User's Sketch and Color. Notably, our robust hardware environment enabled us to manage the computational demands of SC-FEGAN training. The creation of a diverse custom dataset laid the foundation for generating realistic and diverse facial images. Training dynamics, characterized by decreasing generator loss and occasional discriminator loss fluctuations, were observed. The showcased results from the latest model underscored the SC-FEGAN approach potential for producing high-quality facial images.

In terms of future perspectives, several promising directions emerge from this project. Firstly, fine-tuning and optimizing the SC-FEGAN model offer avenues for achieving even higher image quality and realism. Additionally, considering the utilization of larger and more diverse datasets can lead to the generation of more detailed and diverse facial images. Beyond face generation, the principles explored in this project can extend to various applications, such as art generation and data augmentation.

Bibliography

[1] Ian J. Goodfellow Author. *Generative Adversarial Networks*. arXiv preprint arXiv:arxiv-1406.2661. 2014. URL: https://arxiv.org/abs/1406.2661.

[2] Jongyoul Park Author Youngjoo Jo. *SC-FEGAN: Face Editing Generative Adversarial Network with User's Sketch and Color*. arXiv preprint arXiv:arxiv-1902.06838. 2019. URL: https://arxiv.org/abs/1902.06838.

[3] Sifei Liu Author Yijun Li. *Generative Face Completion*. arXiv preprint arXiv:arxiv-1704.058388. 2017. URL: https://arxiv.org/abs/1704.058388.

[4] Zhuowen Tu Author Saining Xie. *Holistically-Nested Edge Detection*. arXiv preprint arXiv:arxiv-1504.06375. 2015. URL: https://arxiv.org/abs/1504.06375.

[5] Philipp Fischer Author Olaf Ronneberger. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. arXiv preprint arXiv:arxiv-1505.04597. 2015. URL: https://arxiv.org/abs/1505.04597.

[6] Zhe Lin Author Jiahui Yu. *Free-Form Image Inpainting with Gated Convolution*. arXiv preprint arXiv:arxiv-1806.03589. 2019. URL: https://arxiv.org/abs/1806.03589.

[7] Alexandre Alahi Author Justin Johnson. *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*. arXiv preprint arXiv:arxiv-1603.08155. 2016. URL: https://arxiv.org/abs/1603.08155.