CAIRO UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

Retratista

A Graduation Project Report Submitted

to

Faculty of Engineering, Cairo University

in Partial Fulfillment of the requirements of the degree

of

Bachelor of Science in Computer Engineering.

Presented by

Mohamed Shawky Zaky AbdelAal Sabae

Remonda Talaat Eskarous

Mohamed Ahmed Mohamed Ahmed

Mohamed Ramzy Helmy Ibrahim

Supervised by

Dr. Mayada Hadhoud

26th July, 2021

Abstract

Throughout history, *portraiture* has been a mean of identifying and finding individuals due to various reasons. Aside from being an art to memorialize famous historical characters, it has been used to identify criminals and missing individuals. The main reason behind using portraits for these purposes is that *visual search* is much easier for humans than any other kind of search. Most people can identify a person or an object, they have seen before. Even with the rise of photography, statistics say that drawn portraits are still used in visual search. Recently, portraits developed other applications as well, especially in the study of historical characters and events.

While being important, sketching a complete face portrait can be very cumbersome and time-consuming. Historically, this process required a professional portraitist and could take him weeks to fully sketch an accurate portrait. However, with the rise of digital design application including *Adobe Photoshop*, this process became much easier. Nonetheless, it still requires a professional and can take hours or even days to accomplish. Moreover, the process of refining a drawn portrait can be very tedious and can cause re-sketching in certain cases. On the other hand, the process of sketching a portrait sometimes needs to be extremely fast, in cases of searching for dangerous criminals or missing children.

Retratista addresses both *time* and *accuracy* issues, while keeping the usage as simple as possible for non-professionals. We use the power of modern *generative models* to create an *end-to-end* system for *face portrait generation* from bare description. This description can be a voice, text or manual description. We abstract the complexity of our system from the user by providing a simple *web user interface*. Our system takes just a *few seconds* to generate a complete face. Also, it allows the user to further *refine* the generated face and even render it in *multiple poses* to provide further identification. We include a relatively-sufficient set of *facial attributes* to correctly describe a human face.

الملخص

على مر التاريخ ،كان فن البورتريه وسيلة للتعرف والعثور على الأفراد لأسباب مختلفة. بصرف النظر عن كونه فنًا لإحياء ذكرى الشخصيات التاريخية الشهيرة، فقد تم استخدامه لتحديد المجرمين والأفراد المفقودين. السبب الرئيسي وراء استخدام الصور الشخصية لهذه الأغراض هو أن البحث المرئي أسهل بكثير على البشر من أي نوع آخر من البحث. يمكن لمعظم الناس التعرف على شخص أو شيء ما، رأوه من قبل. حتى مع ظهور التصوير الفوتوغرافي، تشير الإحصائيات إلى أن الصور المرسومة لا تزال تستخدم في البحث المرئي. في الآونة الأخيرة، طورت البورتريهات تطبيقات أخرى أيضًا، لا سيما في دراسة الشخصيات والأحداث التاريخية.

على الرغم من أهميته، إلا أن رسم صورة كاملة للوجه يمكن أن يكون مرهقًا جدًا ويستغرق وقتًا طويلاً. تاريخياً، تطلبت هذه العملية رسام بورتريه محترف ويمكن أن تستغرق أسابيع لرسم صورة دقيقة بالكامل. حاليا ومع ظهور تطبيقات التصميم الرقمي مثل ادب عهتسهب، أصبحت هذه العملية أسهل بكثير. ومع ذلك، لا يزال الأمر يتطلب متخصصًا ويمكن أن يستغرق ساعات أو حتى أيامًا لإنجازه. علاوة على ذلك، قد تكون عملية تحسين الصورة المرسومة صعبة للغاية ويمكن أن تؤدي إلى إعادة الرسم في حالات معينة. من ناحية أخرى، يجب أن تكون عملية رسم البورتريه سريعة للغاية في بعض الأحيان، في حالات البحث عن مجرمين خطرين أو أطفال مفقودين.

يعالج ريتراتيستا مشكلات الوقت والدقة، مع الحفاظ على سهولة الاستخدام قدر الإمكان لغير المتخصصين. نحن نستخدم قوة النماذج التوليدية الحديثة لإنشاء نظام شامل لتوليد صورة الوجه من الوصف العاري. يمكن أن يكون هذا الوصف صوتًا أو نصًا أو وصفًا يدويًا. نحن نعزل تعقيد نظامنا عن المستخدم من خلال توفير واجهة مستخدم ويب بسيطة. يستغرق نظامنا بضع ثوانٍ فقط لتكوين وجه كامل. كما أنه يسمح للمستخدم بتحسين الوجه الذي تم إنشاؤه بشكل أكبر وحتى عرضه في أوضاع متعددة لتوفير مزيد من التعريف. نقوم بتضمين مجموعة كافية نسبيًا من سمات الوجه لوصف الوجه البشري بشكل صحيح.

Acknowledgement

First of all, we would like to express our deep gratitude to our supervisor **Dr. Mayada Hadhoud** for providing us with incredible guidance and support and making this work possible. Also, we like to thank **Prof. Albert Gatt** and **Dr. Marc Tanti** from *University of Malta* for providing us with necessary dataset used in initial experimentation. Also, we would like to thank all the professors, lecturers and teaching assistants in our department for guidance and support throughout the college year. Last but not least, we would like to thank our families for always providing awesome support.

Contents

List of Figures

List of Tables

List of Abbreviation

AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
NLP	Natural Language Processing
CNN	Convolutional Neural Network
GAN	Generative Adversarial Network
API	Application Programming Interface
FFHQ	Flickr-Faces-HQ Dataset
ADA	Adaptive Discriminator Augmentation
FID	Fréchet Inception Distance
LPIPS	Learned Perceptual Image Patch Similarity
AdaIN	Adaptive Instance Normalization
GPU	Graphics Processing Unit
TPU	Tensor processing unit

List of Symbols

$\|\|A\|\|$	Magnitude of a Vector A
$Trace(B)$	Trace of a Matrix B
D	Feature Directions Matrix
d	Single Feature Direction Vector
E	Face Embedding (Latent Vector)
l	Logit (Numerical Value)
l_{diff}	Differentiated Logits (Result of Logits Subtraction)

Contacts

Team Members

Name	Email	Phone Number
Mohamed Shawky Zaky	mohamedshawky911@gmail.com	+2 01271194141
Remonda Talaat Eskarous	remondatalaat21@gmail.com	+2 01554753224
Mohamed Ahmed Mohamed	mohammed.ahmed.m.228@gmail.com	+2 01090666536
Mohamed Ramzy Helmy	mohamedramzy98620@gmail.com	+2 01153302514

Supervisor

Name	Email	Phone Number
Dr. Mayada Hadhoud	mayadahadhoud@gmail.com	+2 01099957167

1 Introduction

In our project, we address the problem of *face portrait generation* from *various descriptions*. These descriptions include voice, textual and manual descriptions. We design our system for complete *end-to-end* portrait generation, with a simple *web user interface*. Moreover, rapid face refinement and rotation is provided to further improve face identification. Our system solves both duration and complexity issues of previous portrait sketching methods, while providing a high fidelity results and keeping simple interface. This eases the usage of portraits in many applications.

Retratista (*our final system*) combines the power of modern *AI* solutions and the great applications of portraiture into a single tool that is available for everyone to use efficiently without experience and with minimum duration.

1.1 Motivation and Justification

Sketching a human face from bare description is a tedious task that requires specialization. Normally, professional portraitists are asked to sketch a face from description. The process of sketching a human face portrait from scratch can involve problems, such as :

- A long time to initially sketch the face that matches the description.
- After being sketched, any refinements in the face might require re-sketching from scratch.
- Sketched faces are not accurate and real enough to be used for facial recognition in both manual and automated (*AI-driven*) visual search.

Despite its difficulty, the process of sketching portrait has many applications, most of which require high accuracy and speed. These applications include :

- Visual search of missing individuals and criminals.
- Visualization of historical and mythical characters.
- Search of surveillance video feeds for individuals' actions.

Here are some statistics from *International Centre For Missing and Exploited Children* that can tell how our system is critically needed :

- In **USA**, an estimated **460,000** children are reported missing every year.
- In **UK**, an estimated **112,853** children are reported missing every year.
- In **Germany**, an estimated **100,000** children are reported missing each year.
- In **India**, an estimated **96,000** children go missing each year.
- In **Canada**, an estimated **45,288** children are reported missing each year.
- In **Russia**, an estimated **45,000** children were reported missing in 2015.
- In **Australia**, an estimated **20,000** children are reported missing every year.
- In **Spain**, an estimated **20,000** children are reported missing every year.

1.2 Project Objectives and Problem Definition

The main focus of **Retratista** is to offer as many utilities as possible to identify a human face from any description. To do this, we have to :

- Translate the input description to a complete human face image.
- Refine the generated face image, as needed, in order to reach the target face.
- Render the final face in multiple poses to provide more face identification.

In order for our product to be valuable, these functionalities have to be done as fast as possible, while maintaining output accuracy and quality. Also, it should be an *easy-to-use* interface with no required experience to use, only enter the description and edit what is needed. The output of our system can, then, be used in multiple applications of visual search and identification.

1.3 Project Outcomes

The project outcome is a web application that simply takes a description as voice, text or manual input and translates it into a complete realistic face portrait within a *few seconds* in a *fully-automated* way. Then, the application gives the user a chance to further refine the facial attributes to match the target face. Furthermore, the generated face can be rendered in multiple poses for better identification. Our system considers 32 facial attributes for generation and 38 facial attributes for refinement.

1.4 Document Organization

The document is organized into 6 chapters along with the appendices. Chapter 1 introduces the main idea of the project, along with the formal definition of the problem, our motivation to work on this problem and the project outcome.

Chapter 2 discusses the market study of our project through identifying the target market and listing our potential competitors. Also, it provides a financial analysis for our product.

Chapter 3 gives the necessary background information, along with previous related work in research literature and shows our adopted approach.

Chapter 4 is the main body of our document, where it discusses, in details, our system architecture and application design. This chapter further explains the implementation details of our modules. Finally, it discusses alternative approaches considered in our experimentation.

Chapter 4 explains the metrics, used to assess our system, and our system results. The testing is described both qualitatively and quantitatively. Also, we show the success and failures cases of our system.

Chapter 5 concludes our document, lists the faced challenges and shows how our work can be extended and improved.

Finally, we include 4 appendices for describing development tools, project use cases, user guide and product feasibility study.

2 Market Feasibility Study

Face Generation is an important field in the market, it is used widely in all camera applications to apply filters on the taken images most of the usage was for entertainment, Although it can be used to have photos for any missing people or criminal and that will increase the chance to find that missing people or the criminal.

2.1 Targeted Customers

Our Project is directed to all governmental and volunteering organizations, that need to generate identical faces of missing people from bare human description of their facial features. It also can be used to identify the ancient kings and queens faces from the Papyrus . We can categorize the different target companies into the following:

- **Criminal identification from witnesses descriptions :**
  - INTERPOL.
  - Different Police Stations.
- **Missing People organizations like :**
  - International Center for Missing and Exploited Children.
  - International Red Cross and Red Crescent Movement.
  - International Commission on Missing Persons *ICMP*
  - The Doe Network.
  - National Missing and Unidentified Persons System.
  - Missing Persons Support Center.
  - LostMissing Inc.
  - Missing People In America.
  - Be United Missing Persons Inc.
  - others.
- **Historical Characters Identification :**
  - Archaeological and Historical Research Facilities.

2.2 Market Survey

As mentioned previously, most Applications that uses face generation is for entertainment, So they depends only on image refinement on limited number of features to generate the image unlike us, We have both image generation and refinement using 34 features for generation and 38 for refinement. Besides that our goal is to find missing person for purpose more important than entertaining. Our biggest competitor is FaceApp, followed by PicsArt, Facetune2 and Booth Apps.

2.2.1 FaceAPP

Figure 2.1: Changing Age using FaceApp application.

FaceAPP is considered Image and video editing mobile application, it generates highly realistic transformations of human faces in photographs using AI and Neural Networks. The Limitation of the App compared with us, it only used for refinement using 14 facial feature which may be not enough for generating face from scratch.

2.2.2 PicsArt

Figure 2.2: image refinement using PicsArt application.

PicsArt is a company that develops online photo and video editing applications, with a social creative community. The platform allows users to take and edit pictures and videos and draw with layers. It only allows 9 features refinement so it's also not the best choice for image generation.

2.2.3 Facetune2

Figure 2.3: image refinement using FaceTune2 application.

Facetune2 is a photo editing application. It's commonly used to enhance the portrait o selfie images and offers 7 feature refinements.

2.2.4 Booth Apps

Figure 2.4: image Editing using PhotoBooth application.

Booth Apps are a collection of applications for image or video editing and enhancement, can add animation to an image and many other entertaining features. Examples for Booth Apps are : Simple Booth Classic, Mini Photobooth, Pocketbooth, Photobooth mini and My Photobooth App. They only consider image enhancement and for entertaining purposes not critical issues like our product.

2.3 Business Case and Financial Analysis

Our product is new in the market which makes our business case a good one. We can provide our services in two forms as following:

- The first is being a software service that is used in police stations so that any missing people or theft happens, they enters the description of that person so they can have an image for him/her to be able to find him faster, Or used in historical institutions that is responsible for finding information about the history from papyrus.
- The second one is a website for general users to be able to use it to generate images for their ancestors who died without having any images, from their parents' descriptions so they can see them, or for any entertaining purposes.

The financial analysis of this application is done after detailed market research. The cost estimation for our software as a service in a specific institution is the cost behind the hardware, our application takes around 1 second on an *Nvidia GTX 1080 ti* for generating the face from any description, but this can extend to be multiple seconds based on the connection. The whole system can be deployed on single PC with 11GB VRAM, around 700$ for the GPU and 1500$ to 2000$ for the whole PC. For scalability, we can use 2 PCs one for generation and the other for Pose estimation to host the whole application. Also, we can scale out our system on more than 2 PCs for improved request parallelism. For the second type of usage we can deploy the system on cloud instance which costs around 150$ per month for these specifications.

3 Literature Survey

In this chapter, we explain the necessary background, as well as the related work in the research literature.

3.1 Background Review

We discuss the basic background knowledge that helps us throughout the document.

3.1.1 Convolutional Neural Networks

First we do a brief introduction for Convolutional Neural Networks (CNN). CNN is a special type of *feed-forward* neural networks, which uses convolution operations to perform the forward computations. They are used in wide range of applications, especially *computer vision*. There are 4 basic types of layers in a CNN that we would like to mention :

1. **Convolution layer** : uses filters (a.k.a. *kernels*) to detect edges and shapes by convolution with the input map. The filter has certain dimensions and scans the map using certain stride. This helps the network to learn sparse connections and local structures.
2. **Pooling layer** : is usually used to reduce the input dimensions through maximizing or averaging using a filter.
3. **Upsampling layer** : increases the dimensions of the input map using traditional upsampling methods, such as bilinear upsampling, or learnable upsampling using *deconvolution*.
4. **Fully-connected layer** : is a normal feed-forward dense layer, usually used in classification.

3.1.2 Generative Adversarial Networks

Generative Adversarial Networks (*GANs*) [1] are generative models that can learn through *adversarial training*. GANs have proven to be very good at generating high dimensional data like images. That's why they are used in wide range of applications including image and video generation, image-to-image translation and others.

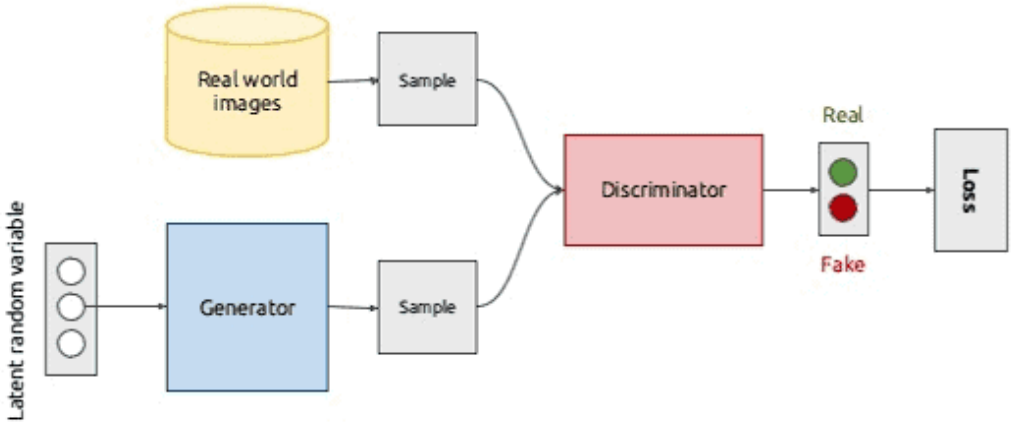


Figure 3.1: The basic architecture of a generative adversarial network (GAN).

The basic architecture of a GAN (as shown in 3.1) consists of 2 main networks. A *generator* takes in random noise (usually *latent random variable*) and convert it into a fake data sample. A *discriminator* judges the quality of the fake data sample by distinguishing between real and fake data samples. The discriminator loss (a.k.a. *adversarial loss*) is used to train both generator and discriminator. So, the whole system can be decomposed into a two-player *minimax* formulated as :

$$min_G max_D V(D,G) = E_{x \sim Pdata}(x)[logD(x)] + E_{z \sim Pz}(z)[log(1 - D(G(z)))] \quad (1)$$

The generator and discriminator networks can be any type of networks. In our case, we use *Convolutional Neural Networks* (CNN), because the output is an image. Consequently, the generator is a CNN that up-samples the latent variable into a complete image. Meanwhile, the discriminator is a classification CNN that encodes the input image and classifies it into real or fake.

The first architecture for a CNN-based GAN goes by the name `DCGAN` [2]. In our work, we focus on a specific type of GANs, named *style-based* (or *latent-based*) GANs.

3.1.3 Latent Space and Entanglement

As mentioned before, we focus on a specific type of generative models named *style-based* GANs. These GANs can have artistic control over their outputs, not just sampling data randomly from a prior distribution. The key idea of these models is to use a *latent variable* to control the output, instead of just considering a random noise. This latent variable is sampled from a *latent space* and, then, used to guide the generation process in the generator network using adaptive instance normalization (*AdaIN*).

AdaIN is a normalization method that aligns the mean and variance of the content features with those of the style features. It's an extension over *Instance Normalization*, which normalizes the input to a single style specified by the affine parameters. It can be formulated as follows :

$$AdaIN(x,y) = \sigma(y)(\frac{x - \mu(x)}{\sigma(x)}) + \mu(y) \quad (2)$$

The latent space is a made-up space, in which the data is encoded in low dimensional representation. For a good model architecture, the latent space is organized, such that similar features are clustered together and changes in low dimensional latent space maps to similar changes in the generated data. Some the most popular and robust latent-based architectures are `BigGAN` [3] and `StyleGAN` [4] [5].

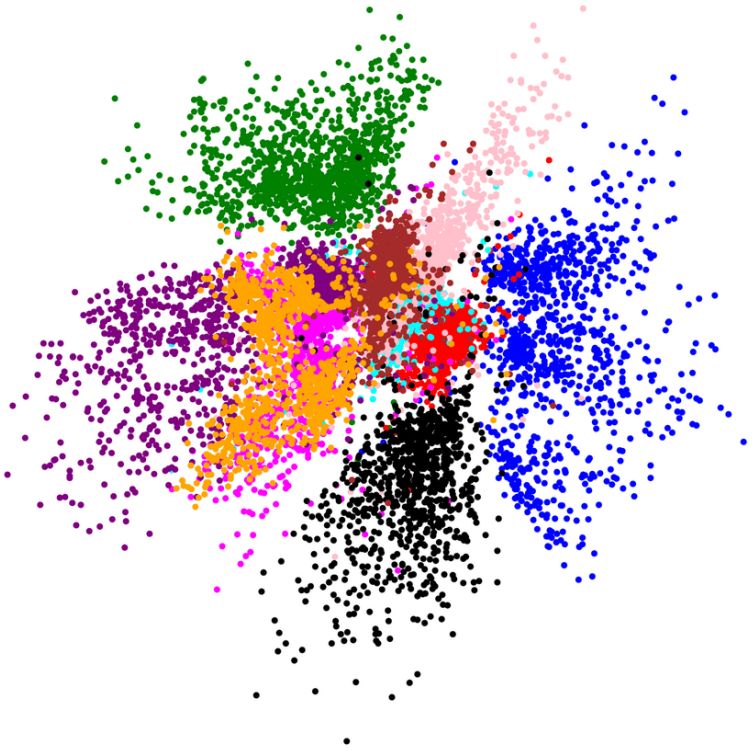


Figure 3.2: An example 2D latent space of MNIST digits dataset.

Figure 3.2 shows a simple $2D$ latent space for `MNIST` dataset for digits classification. We can see that we have 10 clusters corresponding to the 10 digits. Some clusters are more overlapping than others. It's obvious from the figure that we can learn certain directions to move from one cluster to another in the latent space. However, due to overlapping, certain directions might not be clear or even feasible to obtain. This problem is known as **entanglement**, which is one of the most important challenges of our work.

The problem of entanglement arises due to the data and training process of the architecture. Some datasets can show heavy correlation between different features, which causes the latent space to be less organized and the features clusters to be overlapping. Thus, the features are entangled in the latent space and extracting independent directions for every feature is hard or even infeasible. Consequently, moving along a certain direction can cause multiple features to change with certain amounts.

3.1.4 Multi-class Multi-label Text Classification

3.2 Face Modelling and Generation

In this sub-section, an overview is given about face generation and refinement methods.

3.2.1 Face Generation

The problem of human face generation is not new to the *AI* research. Many generative models target the problem of face generation from scratch.

3.2.1.1 StyleGAN

A new generator architecture is proposed that enables controlling the synthesis of the faces by learning the unsupervised separation of high level features. It generates a complete high quality face image controlled by the latent code. It basically works as follows :

- Latent code first is mapped to an intermediate latent space.
- The intermediate latent code controls the generator using adaptive instance normalization.

3.2.1.2 Visual Results

Figure 3.3: Visual results of StyleGAN2.

Figure 3.3 shows visual result samples from `StyleGAN2`, which is the revisited version of `StyleGAN`.

3.2.1.3 StyleGAN latent manipulation

Subsequent research work follows `StyleGAN` to try latent manipulation for more control over the generated faces. These methods include `Image2StyleGAN` [6], `Image2StyleGAN++` [7], `InterfaceGAN` [8], `StyleRig` [9] and `StyleFlow` [10]. All of these work targeted the directed manipulation of *StyleGAN* latent space to change certain facial features. However, none of these methods target complete generation of new face from bare description, which is, also, known as *conditioned sampling*.

On the other hand, our work target face generation from bare description. Consequently, we utilize the concept of directed manipulation to do conditioned sampling, which is described later in this document.

3.2.1.4 Faces à la Carte

To our knowledge, this [11] is the only current open-source research work that addressed the problem of face generation from text. It's based on `StyleGAN2` and uses a general framework to extract the feature directions and reach the target face.

We couldn't replicate their results, because the authors don't provide sufficient information about their method. However, we managed to gain some insights that helped us throughout the project.

Figure 3.4: Visual results of Faces à la Carte.

Figure 3.4 shows the visual results that the authors included in their paper. While being conservative about some of their work, we still consider this as the only former open-source research work related to our problem.

3.2.1.5 NVAE

The paper **NVAE: A Deep Hierarchical Variational Autoencoder** [12] is a recent research work that discusses the usage of Variational Autoencoder (*VAE*) to generate high quality image controlled by a latent variable. It uses normalizing flows to control the output

features. *Normalizing flows* is a method of mapping an unknown distribution to a known distribution (normal or uniform). It's basically used to generate complex distributions from simple normal distributions.

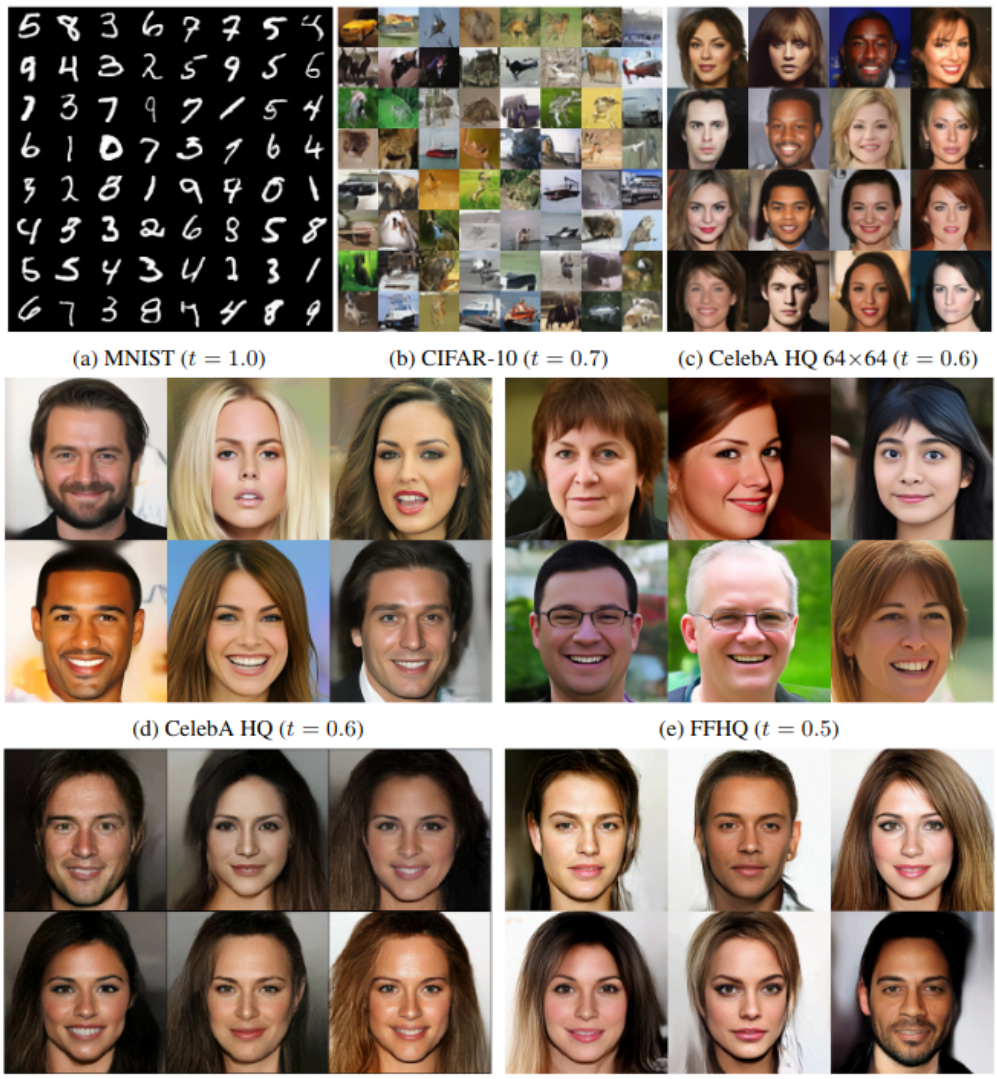


Figure 3.5: Visual results of NVAE.

3.2.2 Face Refinement

The process of face generation can be inaccurate, thus the stage of face refinement is a must, in order to be able to independently edit some of the facial features.

3.2.2.1 AttGAN

`AttGAN` [13] uses an encoder-decoder architecture for the generator, where the decoder takes an attribute vector telling it which attributes are existing and which are not. There's a classifier after the decoder to get the attribute vector of the output image. The decoder is trained to get the same input image if the attribute vector was the input attribute vector and to get an output image with a new attribute vector which makes the classifier get the same attribute vector if the output image was given to it. The architecture is shown in figure 3.6, meanwhile the results are shown in figure 3.7

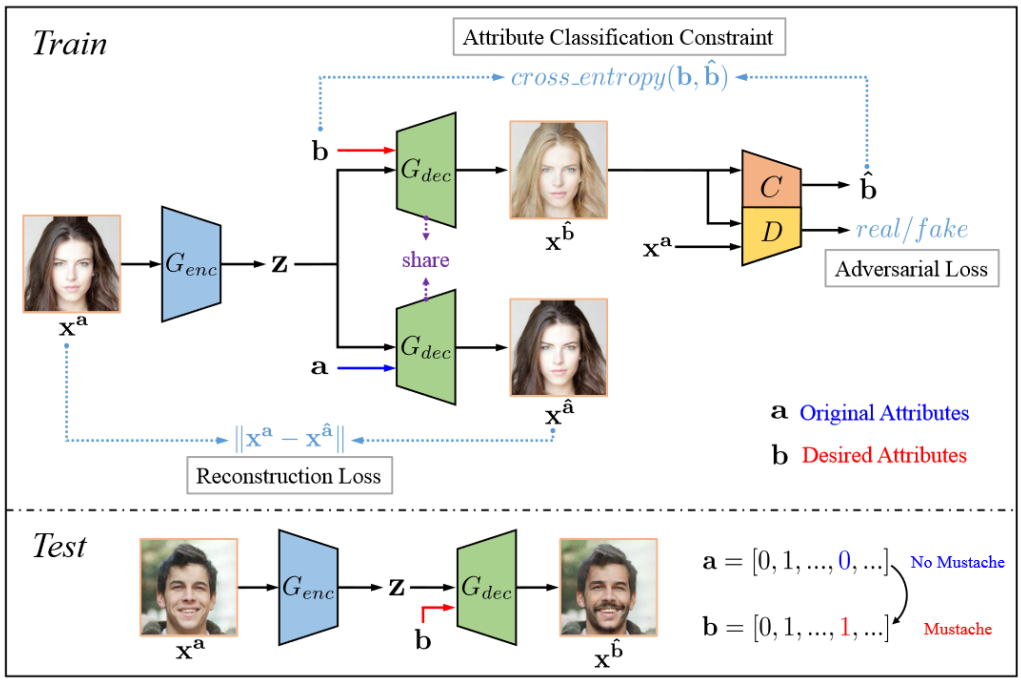


Figure 3.6: Complete architecture of AttGAN.

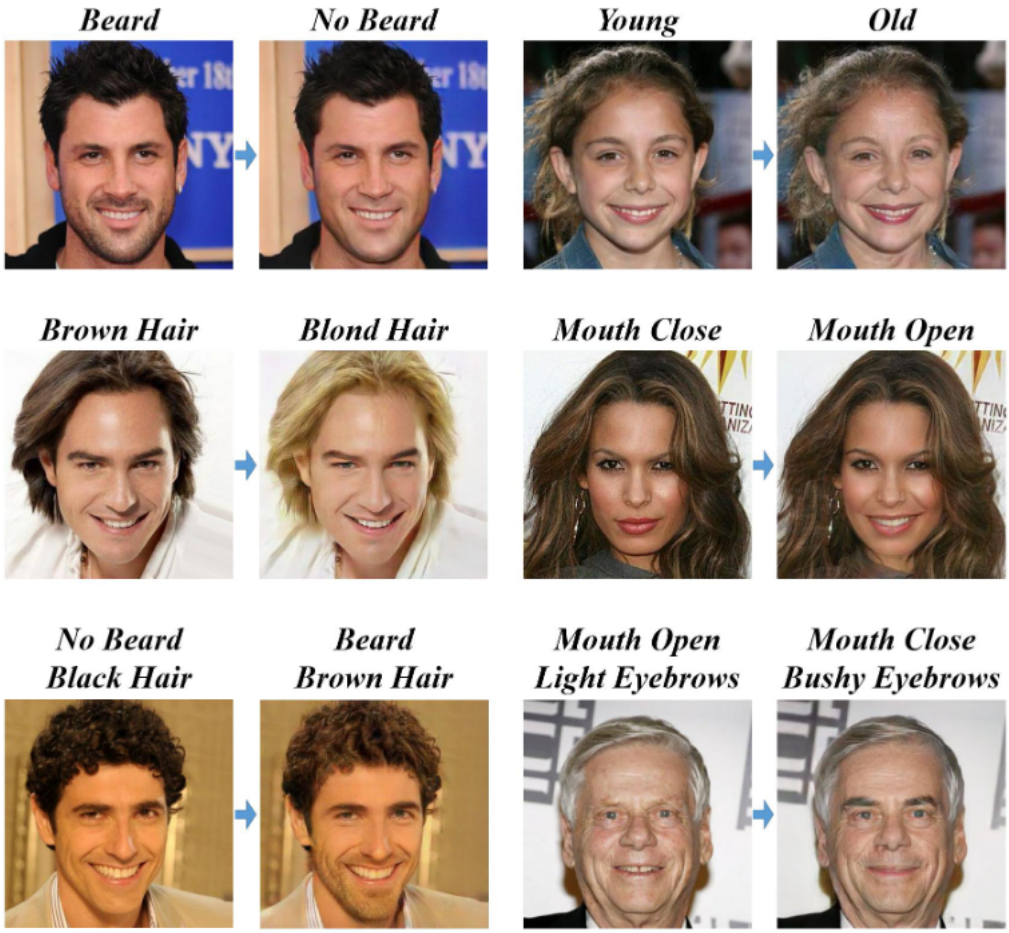


Figure 3.7: Visual results of AttGAN.

3.2.2.2 InjectionGAN

`InjectionGAN` [14] uses the same concepts as AttGAN, but it has one more concept, which is using Feature-wise Linear Modulation (FiLM) layer which successfully infers information from external data and applies it as an affine transformation parameter to vision tasks, so they propose a new type of skip connection layers between encoder and decoder of the generator to add the new attributes shown in figure 3.8. Visual results of InjectionGAN are shown in figure 3.9

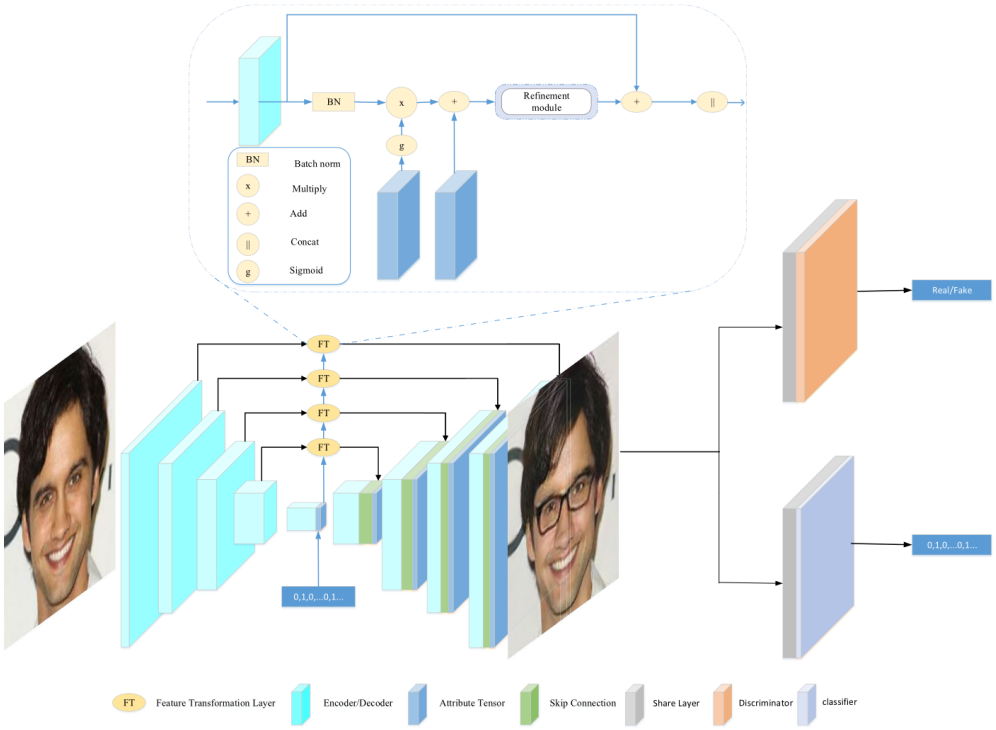


Figure 3.8: Complete architecture of InjectionGAN.

Figure 3.9: Visual results of InjectionGAN.

3.2.2.3 Facelet-Bank for Fast Portrait Manipulation

`Facelet Bank` [15] follows the idea of navigation in GANs latent space, they added a new module called Facelet bank. It's used as skip connections. It takes a bottleneck output from the decoder as input and passing it to a block of CNN layers and add it to the bottleneck output. This summation does the operation of the navigation in the latent space, so the Facelet layer should be trained to get the correct added value to different bottleneck outputs to the same face with changing a specific attribute. Figure 3.10 shows the complete architecture, while figure 3.11 shows the visual results.

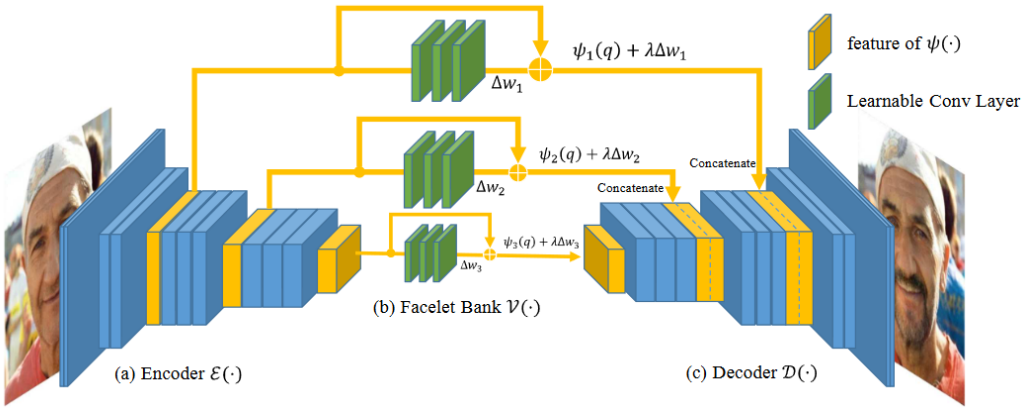
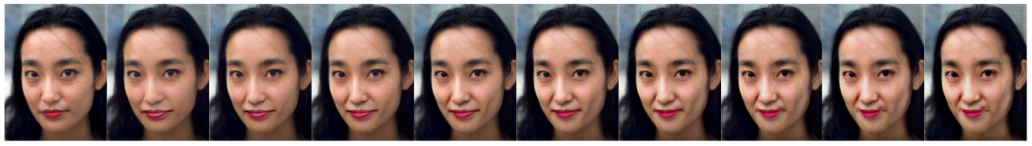
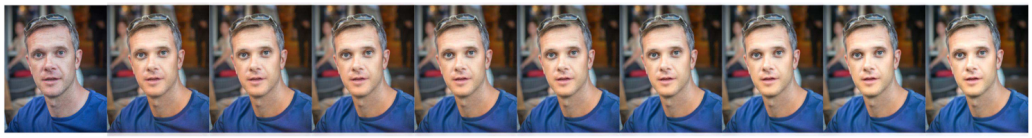


Figure 3.10: Complete architecture of Facelet Bank.

(a) Facial hair



(b) Smile



(c) Younger

Figure 3.11: Visual results of Facelet Bank.

3.3 Face Rotation

In this sub-section, we discuss different methods and approaches to generate multiple head poses from a single image. Note that, we focus on the methods that rotate a face, provided by a single image, with multiple angles of rotation, not just head frontalization. Also, the methods are ordered according to priority based on *performance*, *visual results* and *involved work*.

3.3.1 Rotate-and-Render

3.3.1.1 Brief Description

A DNN-based method under the title of **Rotate-and-Render: Unsupervised Photorealistic Face Rotation from Single-View Images** [16]. The network is based on *3D face modelling* with *unsupervised deep learning*.

3.3.1.2 Workflow

It consists of three stages :

- **3D face fitting** which generates a 3D model of the face and extracts the face texture.
- **Rotate-and-render** which rotates the face in 3D space and renders it in different poses.
- **Render-to-image** which projects the 3D render of the face into a 2D image.

Note : The architecture, also, ensures *cycle-consistency* through re-projecting the face into the initial pose.

3.3.1.3 Visual Results

Figure 3.12: Visual results of Rotate-and-render network.

3.3.2 DepthNets

3.3.2.1 Brief Description

A DNN-based method under the title of **Unsupervised Depth Estimation, 3D Face Rotation and Replacement** [17]. The network is based on *casting* a specific face to a pose provided by another face in an *unsupervised way*.

3.3.2.2 Workflow

It consists of three stages :

- **Landmark extraction** from both faces.
- **Image-to-image translation**.
- **Siamese-like architecture** that compares the generated face with the original one.

3.3.2.3 Visual Results

Figure 3.13: Visual results of DepthNets.

3.3.3 PosIX-GAN

3.3.3.1 Brief Description

A DNN-based method under the title of **PosIX-GAN: Generating multiple poses using GAN for Pose-Invariant Face Recognition** [18]. The network generates the provided faces in *9 angles of rotation* in an *end-to-end manner* that relies on datasets to train GANs.

3.3.3.2 Workflow

It's an end-to-end network that depends on datasets for training.

3.3.3.3 Visual Results

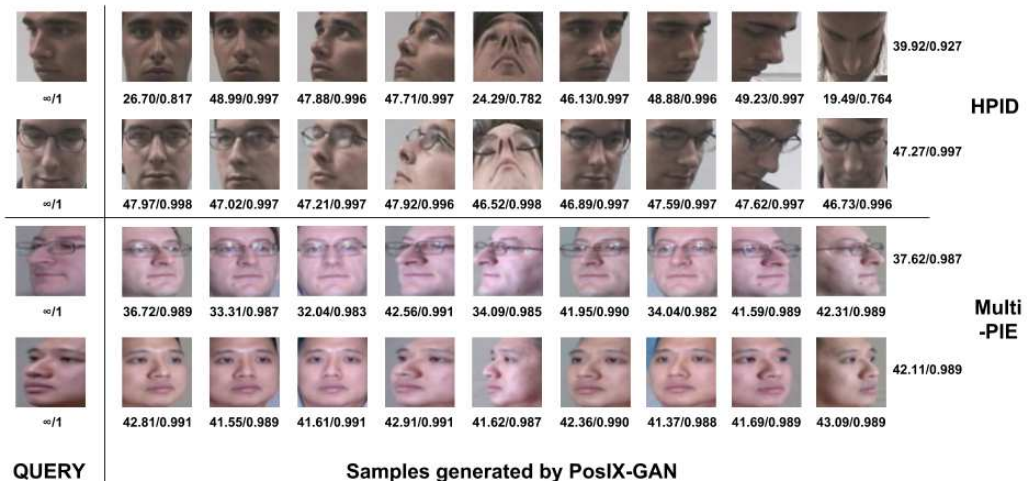


Figure 3.14: Visual results of PosIX-GAN.

3.3.4 CAPG-GAN

3.3.4.1 Brief Description

A DNN-based method under the title of **Pose-Guided Photorealistic Face Rotation** [19]. The network generates different head poses using a *pose-conditioned generator* and *couple-*

agent discriminator that discriminates both identity and pose.

3.3.4.2 Workflow

It's an end-to-end network that depends on datasets for training.

3.3.4.3 Visual Results

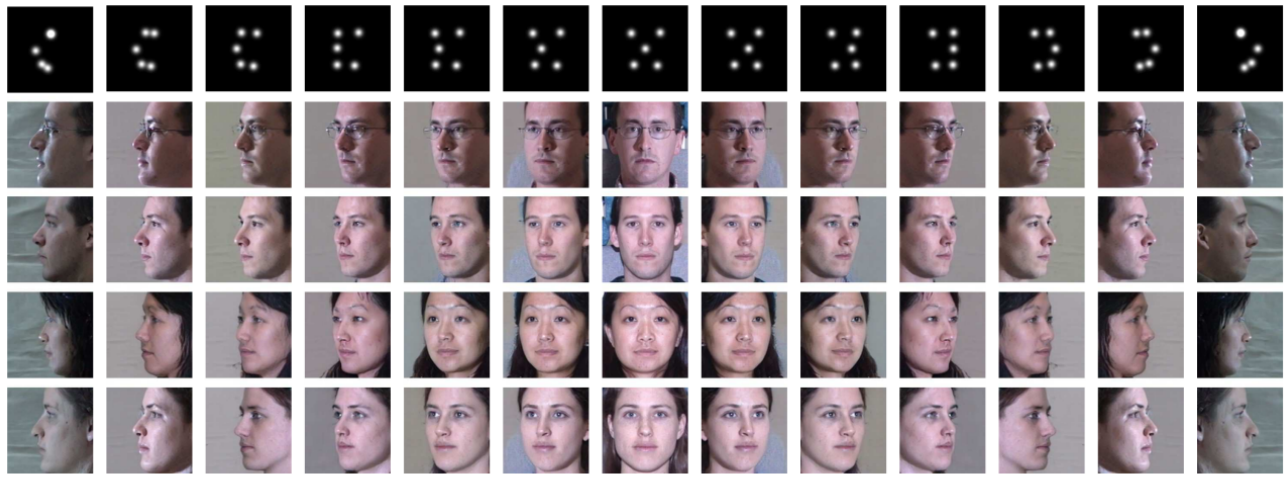


Figure 3.15: Visual results of CAPG-GAN.

3.4 Comparative Study of Previous Work

From our previous discussion, we can see that `StyleGAN2` offers the best quality of faces, while being very flexible. The architecture is used in various research works related to image generation. Other approaches don't have the same flexibility and robustness. That's why we opted to use `StyleGAN2` as our main generative model.

`Faces à la Carte` paper does not provide enough details about their method, as well. This is one of the reasons that encourages us to work on this project, in order to provide are clear pipeline for the problem solution.

Regarding face refinement, the most promising method is `Facelet Bank`, which can be used on high quality image. Also, the *facelet bank* module is flexible enough to handle multiple facial features. However, in our final system, we chose not to go with this architecture for reasons, mentioned later in the document.

Finally, for face rotation, `Rotate-and-Render` and `DepthNets` offer an unsupervised framework for multiple face poses generation. This is better and much easier to adapt than other supervised methods, mainly because most of head poses datasets are not open-source. `Rotate-and-Render` can produce better visual results than `DepthNets`, because it can account for features such as hair through *image inpainting* using *image-to-image translation*. Overall, this yields better visual results.

3.5 Implemented Approach

Our system implements 3 main functionalities, which are **text-to-face generation** (input can be *speech* or *text*), **face refinement** and **multiple head poses generation**. After extensive research and experimentation, we come up with our working pipeline that achieves our project goal.

3.5.1 Face Generation from Description

First of all, we start with our core and the most innovative part of our project. Face generation from bare description is very popular problem, yet few research works target it. To our knowledge, `Faces à la Carte` is the only available research paper that targets this problem. As we mentioned, the authors does not offer enough details about their method. Consequently, we have to utilize the related work to carefully design a fast and robust system for such problem. Our system can take different kinds of descriptions, including audio, text and manual inputs. That's why we have to create multiple stages of input processing and not to use direct end-to-end networks.

The first processing stage is *speech recognition*, which translates the speech into textual description. We opt to use `DeepSpeech2` [20], which is one of the most popular speech recognition architectures.

The second processing stage is *text processing*, which converts the textual description into numerical attributes values. For this stage, we have many options, including classical and DL-based methods. However, we choose to work with `DistilBERT` [21] due to multiple reasons. Basically, `DistilBERT` is a lightweight architecture that does not have a very big memory footprint. It's fast and easily tuned to our needs. Moreover, it does not use *recurrent neural networks*, which are slow at inference time. Finally, we can adapt it to our desired output, which is not just classification, but the actual numerical values of the facial features.

The third and last processing stage is *code generation*, which transforms the numerical attributes values into a face embedding that is used to guide the face generation process. This stage is completely designed from scratch and refined, iteratively, with face generative model (`StyleGAN2`). However, some previous research works are used for guidance, especially `Image2StyleGAN`, which guided our feature directions extraction process. However, our system considers more facial features than any of the previous works.

Finally, we come to the actual face generation model. For this, we opt to use `StyleGAN2`. The reason behind this choice is that `StyleGAN2` is the one of the most powerful image generators that produce high quality images. It's very flexible and can be adapted to many applications. Moreover, it does not required special hardware to be tuned, which is the case in `BigGAN` that requires *Tensor Processing Units* (*TPUs*).

3.5.2 Face Refinement

For the face refinement module, we choose to adapt our work in **face generation** to be used for refinement as well. This is mainly done to avoid including another model, which would have increased the memory footprint. Moreover, latent manipulation of `StyleGAN2` can offer better results than most face refinement architectures with more facial features to

include. Consequently, StyleGAN2 latent manipulation is considered for face refinement with an extended set of facial attributes to refine.

3.5.3 Multiple Head Poses Generation

Although it's possible to extract feature directions for the generated head poses for StyleGAN2 as well, we don't consider using it for face rotation. This is mainly because face pose directions suffer from heavy entanglement with other features. Moreover, these directions don't preserve the face identity and can cause serious illumination problems. Consequently, we only keep this method as a backup plan. However, Rotate-and-Render is our main multiple head pose generation method. We used the same workflow to create our face rotation pipeline and integrate it to our system. The main reason behind choosing Rotate-and-Render is that the whole workflow can be decomposed into separate modules to can be edited and implemented to fit our needs. Also, it is an unsupervised architecture, which makes it easy to work with. Finally, it offers decent results compared to other architectures.

4 System Design and Architecture

In this chapter, we discuss our working pipeline and system architecture in details. Generally, our system takes a speech note, textual description or numerical attributes as an input. It processes the input description and outputs the initial human face portrait that corresponds to the given description. Afterwards, the user is allowed to manually control some facial attributes and morphological features and to rotate the face and render it in multiple poses. In the first section, we give an overview about the final system. Then, we discuss the final system architecture in the second section. In the subsequent sections, each module implementation is discussed in details. In the last section, we discuss the other conducted experiments, why we choose this final system and suggestions that can possibly improve the other experiments.

4.1 Overview and Assumptions

As mentioned above, our system basically enables the user to describe a human face in words or using numerical values and turns it into a full human face portrait that can be manipulated and rendered in multiple poses. The system relies heavily on generative models and text processing, both are iteratively designed to obtain the required results. The overall flow can be described as follows :

- The input speech notes are translated to text.
- The textual description (extracted from speech input or manually entered) is processed to extract the numerical values of the required facial features.
- The numerical values are used generate a face embedding vector that encodes the facial attributes in low dimensional space ($512D$).
- A generative model is specifically designed to translate from the low dimensional embedding into the full face portrait ($1024X1024$).
- The generated face portrait can be further refined by navigating the face embedding space and re-generating the face portrait.
- Once the user settles on the final face portrait, the system can render that face in multiple poses to provide further identification.

The previous flow provides a very versatile framework to generate face portrait and adjust it to your liking. However, there is an extremely large number of facial attributes and morphological features to describe a human face. Consequently, we have to choose a descriptive subset of these attributes to consider in the face description. We consider 32 facial attributes for face description, which are listed as follows :

- Overall face :
  - Gender : Male / Female.
  - Age : Young / Old.

  - Thickness : Chubby / Slim.
  - Shape : Oval / Circular.
  - Skin Color : Black / White.
  - Cheeks : Normal / Rosy.
- Eyes :
  - Color : Black / Blue / Green / Brown.
  - Width : Wide / Narrow.
  - Eyebrows : Light / Bushy.
  - Bags Under Eyes : On / Off.
- Nose :
  - Size : Big / Small.
  - Pointy : On / Off.
- Ears :
  - Size : Big / Small.
- Jaw :
  - Mouth Size : Big / Small.
  - Lips Size : Big / Small.
  - Cheekbones : Low / High.
  - Double Chin : On / Off.
- Hair :
  - Color : Black / Blonde / Brown / Red / Gray.
  - Length : Tall / Short.
  - Style : Straight / Curly / Receding Hairline / Bald / with Bangs.
- Facial Hair :
  - Beard / None.
- Race :
  - White / Black / Asian.
- Accessories :
  - Glasses : Sight / Sun.
  - Makeup : On / Off.
  - Lipstick : On / Off.

4.2 System Architecture

Now, let's discuss our system architecture. The system consists of 6 modules, 3 core modules of the project and 3 auxiliary modules. These modules are deployed in a *web application* to provide an easy-to-use interface for face generation and manipulation. Figure 4.1 shows the complete block diagram of the system architecture. Meanwhile, figure 4.2 shows the application design and how the modules are deployed in a web application. The *core* modules are listed as follows :

- **Text Processing :** processes the input textual description and extracts the corresponding numerical values of facial attributes. This problem is similar to *multi-label text classification*, however the outputs are normalized scores of facial attributes, which are designed carefully to match the *face code generation* process.
- **Face Generation :**
  - **Code Generation :** converts the numerical attributes values to be low dimensional face embedding. This is the most *important* and *innovative* module of our system, because it glues the desired attributes scored with the latent space of the generative model (used to generate the face), resulting in more accurate quality outputs.
  - **Code-to-Face Translation :** translates the low dimensional face embedding into the actual face portrait. For this purpose, we use `StyleGAN2`, which is a *state-of-art latent-based generative model*, whose latent space can be manipulated easily to fit our needs.

Meanwhile, the *auxiliary* modules are listed as follows :

- **Speech Recognition :** translates the input speech to textual description.
- **Face Refinement :** uses the same generative model to manually refine the generated face portrait through navigating the latent space.
- **Multiple Head Poses Generation :** rotates the generated face portrait and renders it into multiple poses.

We discuss each module in more details in the subsequent sections. Also, these modules are organized into a web application for easier usage, as shown in Figure 4.2. The application is divided into :

- **Web (Frontend) :** which contains the user interface and, also, the *speech recognizer*. The speech recognizer is moved to the frontend to reduce the network communication overhead between the web application and the server, as transmitting text is easier than transmitting speech. Moreover, the speech recognizer doesn't require high computational power, so it can be embedded in the web application.
- **Server (Backend) :** which is separated into two servers. First server contains the *text processor* and the *generative model* and serves the requests of face generation and refinement. Second server contains the *pose generator* and serves the requests of face rotation.

The two servers can communicate with each other to exchange the generated face portraits through `TCP sockets`. Meanwhile, the web application communicates and sends requests to the servers through `HTTP REST API`.

4.2.1 Block Diagram

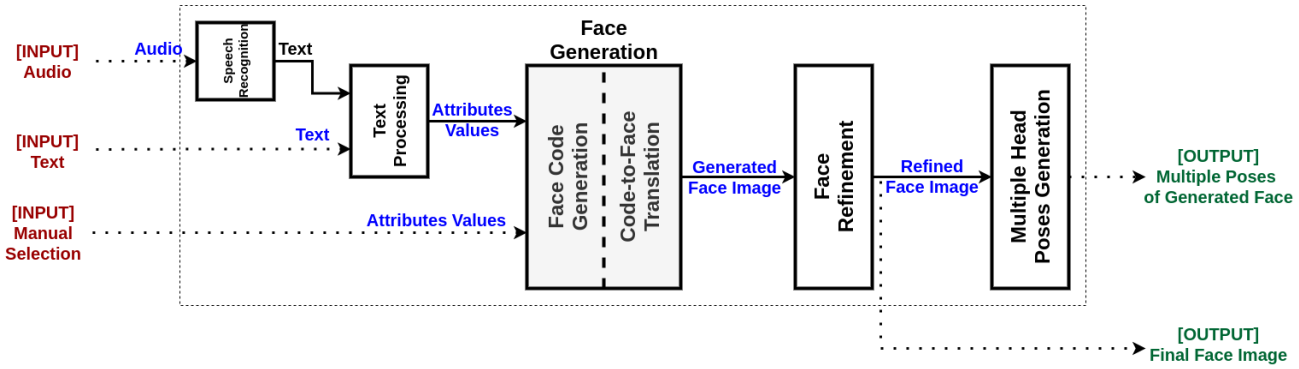


Figure 4.1: Block diagram of complete system architecture

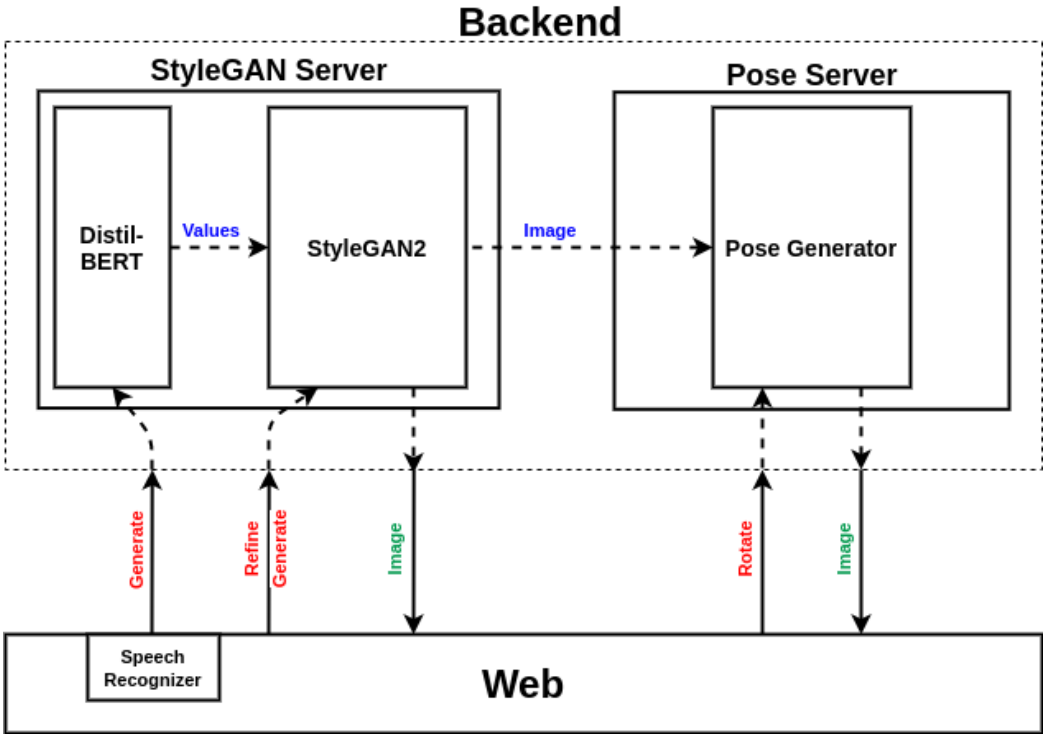


Figure 4.2: Block diagram of application design

4.3 Module 1 : Speech Recognition

This is the first module in our pipeline, its responsibility is to get the textual description of the image from speech. It takes the speech as input then processes it to get its textual content to be used to extract the facial features used to generate the image.

4.3.1 Functional Description

Our aim is to add functionality that the user can enter a speech description of the image to generate it, so we used an automatic speech recognition model based on Deep Speech2 paper, we used the Spectograms features from the audio and applied some preprocessing to those features before training. The target of the speech model is to detect the English spoken words and get their textual meaning so that it can be used to generate the Face Image and output the desired described Face.

- **Input :**
  - speech audio wav.
- **Output :**
  - the spoken words on the input audio.

4.3.2 Modular Decomposition

As mentioned we use deep speech 2 with connectionist temporal classification (CTC) losses as our speech recognition model. The model takes as input features from the audio in the form of spectrograms compressed by frequency and time masking to separate the spectrograms in time and frequency domains.
The model architecture consists of one convolutional layer followed by 3 residual CNNs and 5 bidirectional gated recurrent units GRUs.

The first convolution layer used to compress the audio information in low dimension space, The residual CNN consists of layer normalization over the time axis of the speech then 2 convolution layers summed with the original input to the resCNN.

The bidirectional RNN uses 512 hidden units used to predict the input character at each time step, The last layer is a fully connected layer to out the final predicted character for each input spectrogram.

Connectionist Temporal Classification (CTC) loss function is used to align each character and its location in the audio input file, By summing the probabilities of possible alignments of the input to target and producing a loss value which is differentiable with respect to the model inputs and functions weights. After that those weights are updated using Adam optimizer to minimize this loss value until convergence.

$$L(x, y; \theta) = -\log \sum_{l \in Align(x,y)} \prod_{t}^{T} pctc(l_t|x; \theta) \tag{3}$$

The best aligned character is selected using greedy beam search from all available character to the corresponding input.
The model Was trained and tested on LibriSpeech ASR corpus which consists of 100 hours for training and test set, ”clean” speech for testing. We calculate the word error rate (WER) and character error rate (CER) for the test data to evaluate the model performance.

4.3.3 Design Constraints

As our application is a real time interactive app, It’s a must for the speech recognition model to be very fast to be able to process the input quickly so that we gain the users satisfaction. It was a constraint as we couldn’t use larger model with more accurate results, as it will increase the processing time which is against our needs.

4.4 Module 2 : Text Processing

This module is the first stage in our pipeline that adds the feature of face generation from bare textual descriptions, not just manual manipulators. It's responsible for understanding the input textual description and converting it into facial features logits, where each logit describes how much the generated face should be saturated with. This task is done for 34 different facial attributes where each of them may or may not be entangled with other attributes, such as gender and beard attributes. Attributes may be on different levels as well, some of them are continuous, such as age, some are discrete, such as wearing sunglasses. see figure 4.3 as an example.

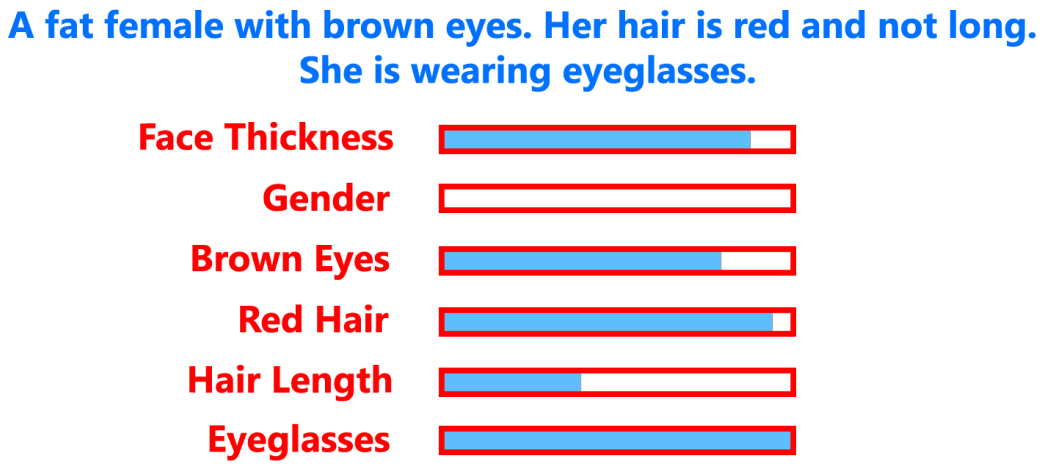


Figure 4.3: Attributes Scores Example

4.4.1 Functional Description

This module takes the input text and tries to

1. extract mentioned facial attributes.
2. extract a score for each mentioned facial attribute referring to the level of saturation of this facial attribute.
3. extract other non-mentioned attributes.

Here's the supported list of facial attributes that can be extracted:

- Eyebrows
  - Bushy Eyebrows — Binary
- Hair Color
  - Black Hair — Binary
  - Red Hair — Binary
  - Blonde Hair — Binary
  - Brown Hair — Binary

- Gray Hair — Binary

- Hair Style
  - Curly-Straight Hair — Continuous
  - Receding Hairline — Binary
  - Baldness — Binary
  - Bangs — Binary
  - Hair Length — Continuous
- Facial Hair
  - Beard — Continuous
- Race
  - Asian Race — Binary
  - Skin Color — Continuous
- General Facial Attributes
  - Face Thickness — Continuous
  - Gender — Binary
  - Age — Continuous
  - Lips Size — Continuous
  - Nose Size — Continuous
  - Ears Size — Continuous
  - Double Chin — Binary
  - High Cheekbones — Binary
  - Pointy Nose — Binary
  - Rosy Cheeks — Binary
- Eyes
  - Black Eyes — Binary
  - Green Eyes — Binary
  - Blue Eyes — Binary
  - Brown Eyes — Binary
  - Eye Size — Continuous
  - Eye Bags — Binary
- Makeup
  - Makeup Saturation — Continuous

  - Lipstick Binary

- Eyeglasses
  - Sight Glasses Binary
  - Sun Glasses Binary

4.4.2 Modular Decomposition

This module is considered as an NLP task that can be tackled using classical approaches, such as Text Parsing and Tagging, and Deep Learning approaches. The final design for this module is to use the power of Deep Learning approaches especially transformers to analyze and understand the text. This is because that the classical approaches failed in such a task as there are a lot of sequence dependencies on the textual description that may arise such as restricting the age of the described person to the range of low-to-medium beard levels and gender attribute, except in the case of age attribute is mentioned explicitly or implicitly in the text, for example ”a boy”, ”a boy with a beard” and ”a boy almost in his thirties”, the first description should be understood as a child or teenager, while other descriptions refer to a middle-aged man. a lot of other dependencies that may arise that make the classical approaches not feasible and makes the deep learning solutions a must to tackle such a task.

The biggest challenge in the Deep Learning approaches is that there are no available datasets mapping textual descriptions to facial features. That's what led us to use synthesize our own dataset that must be much likely to be human-generated. This what made the module to be split into two sub-modules as shown in figure 4.4. First one is the Dataset Synthesis sub-module and the Deep Learning Sequence Model to be trained on the synthesized dataset.

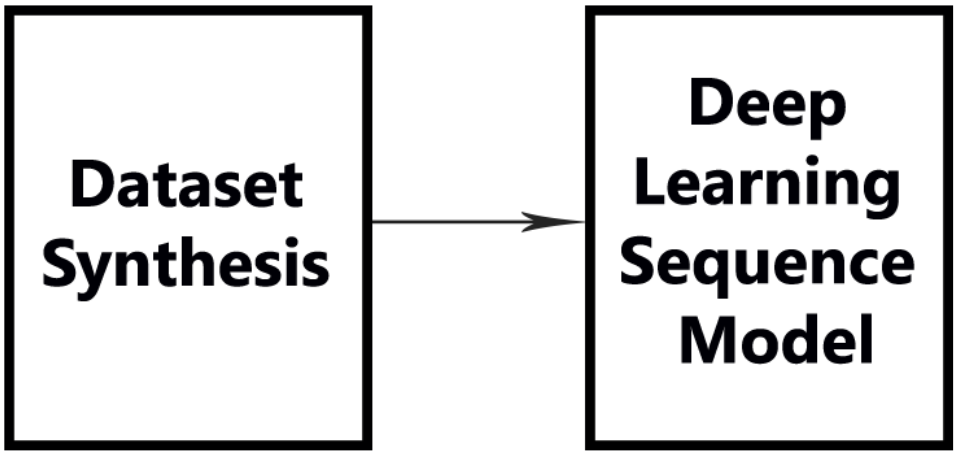


Figure 4.4: Text Processing Pipeline

4.4.2.1 Dataset Synthesis sub-module

This sub-module is the most critical one, as it should be as likely as possible to be human-generated. The approach we used to do so consists of three main stages as shown in figure

4.5 in the reverse order of what needed in the training phase.

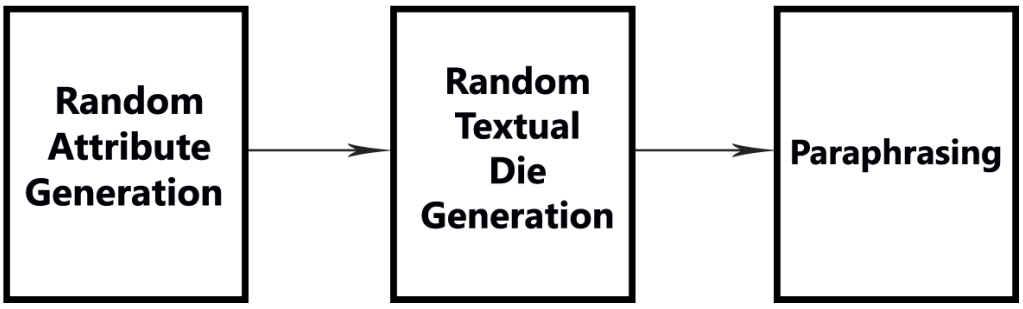


Figure 4.5: Dataset Synthesis Pipeline

1. Random Attribute Generation

   In this stage, we generate any random attribute scores for all 34 attributes. Each attribute is generated with its pre-defined range of scores with one of random different modes

   - Full Random Mode: All attributes will be mentioned in the text with random scores.
   - Half-Length Random Mode: 50% of attributes will be mentioned in the text with random scores, while other 50% will not be mentioned.
   - Short Random Mode: 20% of attributes will be mentioned in the text with random scores, while other 80% will not be mentioned.
   - Very-Short Random Mode: 10% of attributes will be mentioned in the text with random scores, while other 90% will not be mentioned.

   Mentioned Attributes are generated with some restrictions to make their scores consistent with others, such as

   - Females, babies and children cannot have facial hair.
   - Females cannot be bald.
   - Males and babies cannot put makeup or lipstick.
   - Bald people cannot have any hair attribute mentioned, such as hair color or hair style.
   - One hair color at most can be mentioned (i.e. no person can have both yellow and red hair).
   - One eye color at most can be mentioned (i.e. no person can have both blue and black eyes).
   - People with bangs cannot have receding hairline and vice versa.
   - One Eyeglasses type at most can be mentioned (i.e. no person wear both sun and sight glasses).

2. Random Textual Die Generation

   In this sub-module, a textual description is generated in a textual die using the random scores of facial attributes. Attributes are categorized using different categories, such as

   - First Categorization is based on the facial part that is described, such as all hair attributes (Black Hair, Blonde Hair, Gray Hair, Red Hair, Brown Hair, Straight Hair, Hair Length) can be combined in a single description of the hair (e.g. a woman with long, brown and curly hair).
   - Second Categorization is based on the grammatical way that the attribute can be described with. Each attribute can be of one or more of types below
     - With Attributes: attributes that can be described using with statement (e.g. a man with brown hair and blue eyes.)
     - Has Attributes: attributes that can be described using has statement (e.g. a man who has brown hair and blue eyes.)
     - Puts Attributes: attributes that can be described using put statement (e.g. a woman is putting heavy makeup.)
     - Wears Attributes: attributes that can be described using wear statement (e.g. a woman is wearing a glasses.)
     - Full Statement Attributes: attributes that can be described using a full statement (e.g. His hair is brown. His Eyes is blue)
     - Adjective Attributes: attributes that can be described using adjectives (e.g. a blond old man.)

   Each mentioned attribute can have a form of different forms, such as

   - Positive Attributes using Synonyms: describing an attribute with one of its synonyms randomly (e.g. “a man with a thick face.” can be described as a “a chubby man.”).
   - Positive Attributes using Antonyms and Negation: describing an attribute with the negation of one of its antonyms randomly (e.g. “a man with a thick face.” can be described as a “He is not thin” or “a man with non-thin face.” and so forth).
   - Negative Attributes using Antonyms: describing a negative attribute with one of its antonyms randomly (e.g. “a man with no thick face.” can be described as a “a thin man.”).
   - Negative Attributes using Synonyms and Negation: describing a negative attribute with the negation of one of its synonyms randomly (e.g. “a man with no thick face.” can be described as a “He is not chubby” or “a man with non-thick face.” and so forth).

   Each mentioned attribute chooses one of its categories and one of its forms randomly and reserve a slot in the textual die represented in figure 4.6 .

Figure 4.6: Textual Die where each attribute chooses a random slot for itself

3. Paraphrasing

   As we use deep learning approaches, using the textual die to train a model is just making the model learn the die itself, not to generalize and extract the attributes from any other textual description. So, in this sub-module, we tried different approaches to restructure and paraphrase the textual die generated in the previous step, so that the generated description is more likely to be human-generated and to be general. To do so, we tried two approaches as a paraphraser

   - First Approach: Training a Sequence-To-Sequence model using the "entailment" class in Stanford Natural Language Inference Dataset (SNLI) to re-structure any English statement to any other English statement based on the diversity of expressing ways in the English Language
   - Second Approach: Using a random cycle of machine translators (e.g. translates the die in the sequence "English To Arabic To Spanish To Italian To English" or in the sequence "English To Chinese To English"). Using the power of different linguistic ways in different languages makes the paraphrased description more generic.

4.4.2.2 Deep Learning Sequence Model

The main task of Text Processing module is to map input textual description to corresponding attributes scores, so It's a multi-class multi-label NLP problem. Therefore, in this sub-module we trained different deep learning architectures on the previously generated dataset. We tried

- LSTM Architecture: 2 BiLSTMs followed by a linear layer.
- Transformers Architecture: DistelBert-base-uncased, Bert-base-uncased, Albert-base-v2, Roberta-base

Where transformers architectures out-performed, so the final choice was DistelBert-base-uncased, as it's the lightest one out of them with accuracy hit +99% on validation.

4.4.3 Design Constraints

The design constraints of this module are enumerated as follows :

- Human-like Dataset Synthesis: The biggest challenge of this module is that there's no available dataset to tackle such a task. So, we needed to synthesize our own dataset that should be as generic and human-generated as possible, so that what made the paraphraser performance really matters.
- Conditional entailment of different facial attributes: as different facial attribute may affect each other, so the dataset should be generated in a robust way so that it can train the deep learning model to detect such relations, such as entanglement between beard, gender and age.
- Accuracy matters: as this is one of the early modules in the pipeline, the results of all other modules depend on its performance and accuracy. So, lack of accuracy in this module will be catastrophic for the whole pipeline.

4.5 Module 3 : Face Code Generation

Here, we discuss the face code generation from numerical values of facial attributes. This is the most important and innovative module in our system and the first stage of *face generation*. It's worth **noting** that we use both of the terms *"feature"* and *"attribute"* to refer to a facial attribute, like hair color or nose size.

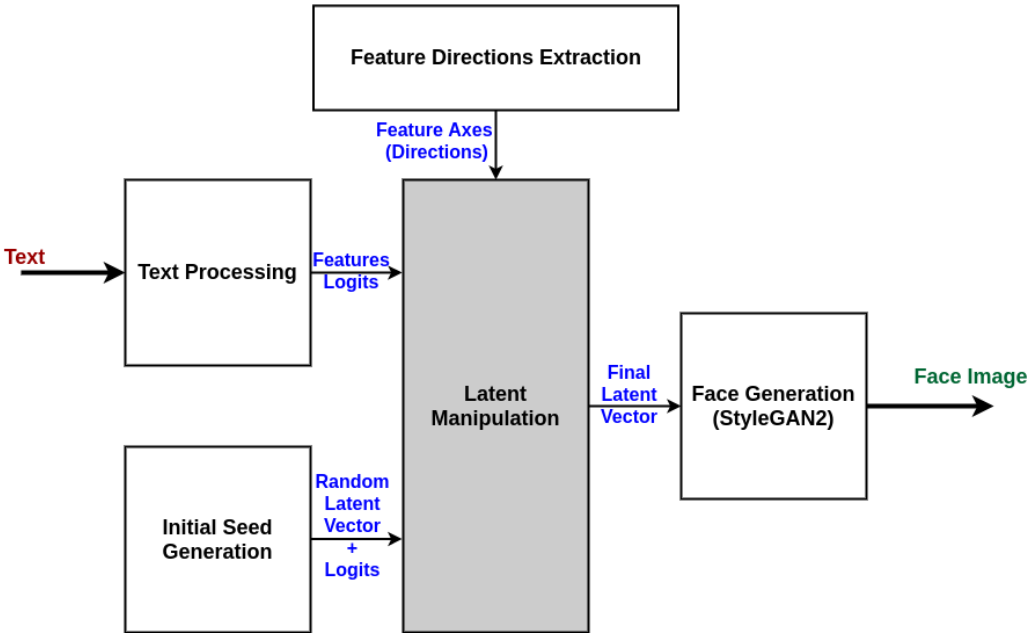


Figure 4.7: Detailed block diagram of the three core modules workflow

4.5.1 Functional Description

Figure 4.7 shows a block diagram of the interaction between the 3 core modules. We can see that the code generation module is the main driver of our face generation process. Generally, it converts the numerical attributes values (a.k.a. *logits*) into a face embedding vector that matches the design of the latent space of the face generator (*StyleGAN2*). Basically, it starts from an initial vector and uses the *required feature values* and *extracted feature directions* to transform this vector into the final latent vector, which is passed to the generative model.

- **Input :**
  - Numerical values of facial features (logits).
- **Output :**
  - Low dimensional face embedding vector (latent vector).

4.5.2 Modular Decomposition

As figure 4.7 tells, the code generation module can be torn down into 3 sub-modules, which are **latent manipulation**, **initial seed generation** and **feature directions extraction**. Each sub-module is discussed in details to show how they integrate to each other to achieve the desired goal.

4.5.2.1 Feature Directions Generation

Since, we use `StyleGAN2` [5] as our generative model, we have a full $512D$ latent space that is used to encode the whole face attributes. The changes in this latent space maps to the generated face image and similar features occupies the same area in the latent space. Consequently, we have to come up with a way to extract the axes (*hyperplanes*) in this latent space to define each of our 32 facial features. These feature directions are, then, used to manipulate the latent vector, in order to map to the required face image.

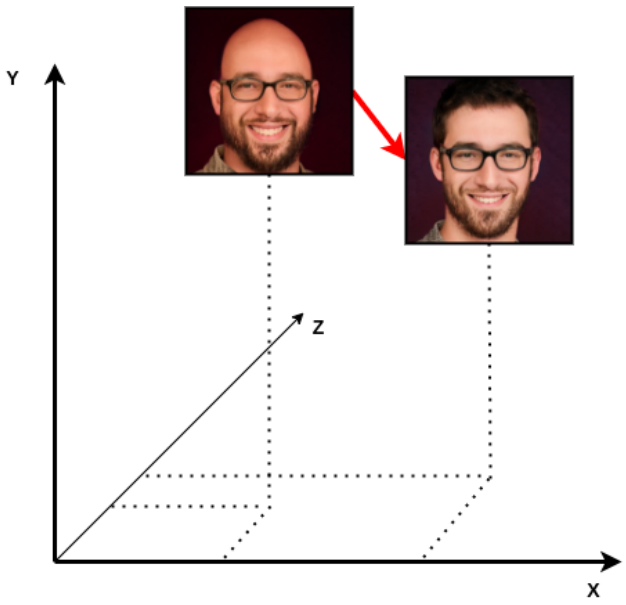


Figure 4.8: Illustration of feature directions in latent space

Figure 4.8 further illustrates the idea of feature directions in the latent space. Here, we plot two face images in a $3D$ latent space. We can see that the difference between the two images in the existence and the absence of the hair, thus the red arrow represents the *baldness* feature direction in that $3D$ latent space (moving along this particular vector causes hair density to change).

Our method of extracting the feature directions (*hyperplanes*) consists of 3 steps :

1. **Code-Image Pairs Generation and Classification :** First, we use `StyleGAN2` to generate a large number of synthetic faces from random latent vectors. After so, we cluster the synthetic images (along with their latent vectors) according to each feature. The clustering can be based on discrete categories (like *hair color* or *race*) or continuous values (like *hair length* or *nose size*). We randomize the synthetic images in each clustering process to have better generalization and to cope with potential generation noise. For classification and regression, we use one of three possible methods, which are **manual labelling**, **classical image processing techniques** and **neural networks**. Thus, the output of this process is different groups of synthetic images sharing common facial features, along with their latent vectors.

2. **Feature Directions Fitting :** Now, we have a set of latent vectors (`X`) and their corresponding feature values (`Y`). It's required to find a set of feature directions that satisfies

the mapping between feature vectors and values. This problem can be formulated as :

$$Y = A_f \cdot X \tag{4}$$

Where A_f is the axis (direction) of feature f.
We can obtain the solution to this equation in a closed form. However, due to the noise in both generation and classification, along with the non-linear nature of the problem, we opt to use *ML* methods, specifically **Logistic Regression** and **SVM** to get an *approximate solution*. Meanwhile, we cannot see any difference between the two methods, as they yield almost the same results.
Finally, the generated feature directions are normalized to unit vectors :

$$A_{unit} = \frac{A}{||A||} \tag{5}$$

3. **Directions Orthogonalization :** Facial features entanglement is one of the most difficult challenged of face generation. Some attributes in the human face tend to be extremely entangled by nature. For example, Asians rarely have curly hair, a woman cannot have beard and a man cannot put on makeup. Since `StyleGAN2` is trained and tuned on **FFHQ** dataset [4], which contains real human faces, it is normal to notice some entanglement between some features. Consequently, the feature directions have to be further disentangled by using *orthogonalization*. The orthogonalization process is done iteratively, starting from the most accurate feature directions. We orthogonalize other feature directions on the accurate ones, so that we have completely independent feature directions, where tuning one direction doesn't affect the others. The directions are orthogonalized as follows :

$$A_{proj} = (A \cdot B_{unit})B_{unit} \tag{6}$$

$$A_{orthogonal} = A - A_{proj} \tag{7}$$

Figure 4.9 visually illustrates the *directions orthogonalization* process on $2D$ vectors.

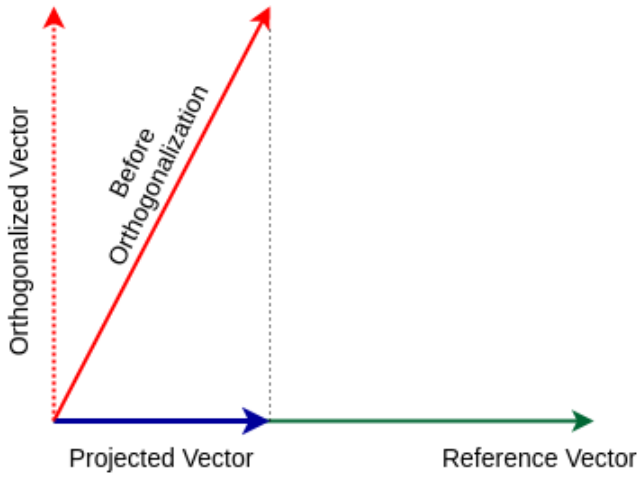


Figure 4.9: Illustration of orthogonalization relative to a reference vector

To ensure convergence to reasonable set of feature directions, we use a threshold margin to stop the orthogonalization process, which is from 85 to 95 degrees (5 degrees on each side of normal angle).

4.5.2.2 Initial Seed Generation

In order to avoid noise and discontinuities in the latent space, we generate an initial random latent vector. This is done by generating a random $512D$ vector and then passing it through the *mapping network* of `StyleGAN2`, which is not invertible. This initial vector is, then, manipulated by sequential navigation along each feature direction (axis) with certain amounts. To get these amounts, we should know the component of the initial vector along each feature direction. We do that by simply performing a dot product between the initial vector and the unit vector of each feature direction. Thus, we have an initial latent vector and the numerical attributes values, it presents.

4.5.2.3 Latent Manipulation

This sub-module ingests all the inputs and produces the required latent vector (*face embedding*) that describes all of the required facial attributes. The inputs to this latent manipulation sub-module are *initial random vector* along with its logits, *text logits* and *feature directions*. The latent manipulation, simply, wants to realize the following transformation on the *initial random vector* :

$$E_{final} = E_{initial} + (l_{text} - l_{rand})D \tag{8}$$

Where E_{final} is the final latent (*embedding*) vector of dimensions $1X512$, $E_{initial}$ is the initial random vector of dimensions $1X512$, l_{text} is the text logits vector of dimensions $1X32$ (remember that we consider 32 facial features), l_{rand} is the logits vector of the initial random vector of dimensions $1X32$ and D is the feature directions matrix of dimensions $32X512$. The transformation includes calculating the difference between the required logits and the random logits and, then, use this difference to move the initial random vector along the feature directions to reach the final latent vector.

It might seem straight forward to perform this transformation. Unfortunately, it's not feasible to perform the transformation using direct matrix multiplication, mainly due to heavy *entanglement* between direction vectors even after *orthogonalization*. Also, the latent space of `StyleGAN2` can be very noisy in certain regions, so transformations have to be done carefully.

Consequently, the processing in this sub-module is done iteratively as follows :

- Both random and text logits are scaled from 0 to 1, which cannot have significant effect, when navigating using unit directions. Consequently, the inputs logits are scaled with the directions scale, which is obtained empirically to be from -4 to 4, as shown in figure 4.10.

- The next step is to get the *difference* between *text logits* and *random logits*, which is of dimensions $1X32$. We call that **differentiated logits**. It's worth noting here that the input text usually contains a *subset* of the facial features. Consequently, *not all* the text logits

are set to specific values. So, when doing the *differentiation*, we set the differentiated logits of the *unmentioned facial features* to 0. So, it can be summarized as follows :

$$l_{diff} = \begin{cases} l_{text} - l_{rand} & l_{text} \neq None \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

- Loop over each direction in feature directions :
  - Multiply the *differentiated logit* corresponding to the *current feature* with its *direction*.
  - Add the product to the current latent vector (starting with the *initial random vector*). The following equation summarizes these steps :

$$E_{next} = E_{prev} + l_{diff}[j] * D[j] \tag{10}$$

  - Finally, to ensure that every transformation is independent of the subsequent transformations and that they are applied sequentially, we re-compute the *produced latent vector logits* and re-differentiate it with the *text logits*. This is done on every iteration of the latent manipulation process.

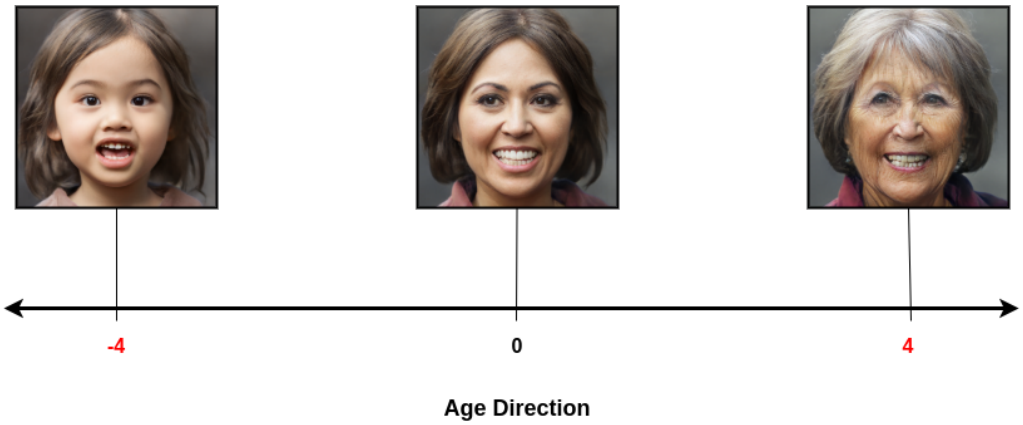


Figure 4.10: Illustration of directions scale using age direction

By applying the previous process, the *numerical value* of facial features extracted from text or manually entered by the user can be converted into a complete face embedding (*latent vector*) matching the required facial attributes. This vector can be passed to `StyleGAN2` to translate it to a complete human face image.

4.5.3 Design Constraints

The design constraints of this module are enumerated as follows :

1. **Facial attributes entanglement** is the main challenge of the face code generation module. Naturally, human face attributes are related to each other. For example, Asians barely have curly hair, no woman cannot have beard and most women have long hair and wear makeup. We mentioned before that `StyleGAN2` is trained and refined on `FFHQ`

dataset, which contains real human faces. Consequently, it's normal to see heavy entanglement between features in the latent space. Due to this *entanglement*, we have to perform extra computations to get decent results.

2. **Random initialization** of the latent vector can cause some issues with the final output, as the vector can be initialized in a noisy area of the latent vector. We try to *limit* the initialization of latent vector to a certain set of random vectors to avoid this effect. This method significantly reduces the *random initialization effect*, however it's not fully cured.

3. **Directions accuracy** can be a challenge as well. Some factors can negatively affect the feature directions accuracy. These factors include **synthetic image clustering** (whether *classification* or *regression*) and **directions fitting** process. We already discussed our solutions to this problem.

4.5.4 Synthetic Image Clustering

As we mentioned before, it's required to cluster the generated synthetic images, in order to be able to fit the feature directions. The *clustering* process can be performed through *classification* or *regression*. For example, features like hair and skin colors should be classified (grouped) into discrete categories, however features like hair length and mouth size should be assigned a continuous value. We do this task using one of three different methods :

1. **Manual labelling** is the first idea to come to our minds. It's straight forward to manually classify a group of faces according to a certain feature. This method is used with some features, however it's very cumbersome and only works for classification.

2. **Classical image processing techniques** are, also, used to classify images based on some features. Mainly, we use these techniques to detect *colors* like eye color and hair color. We use *morphological operators* and *classical segmentation* to detect *the eye* or *the hair* and then retrieve its color.

3. **Deep learning techniques** (*neural networks)* are used to perform regression on the rest of the features. We basically use *facial landmark detection* pretrained networks to detect the important facial landmarks, which is, then, used to calculate *sizes* and *distances*.

4.6 Module 4 : Code-to-Face Translation

Here, we discuss the process of converting the face embedding (*latent vector*) to a complete face image using `StyleGAN2` (our chosen generative model). We discuss our reasons for choosing this particular architecture and how we use it.

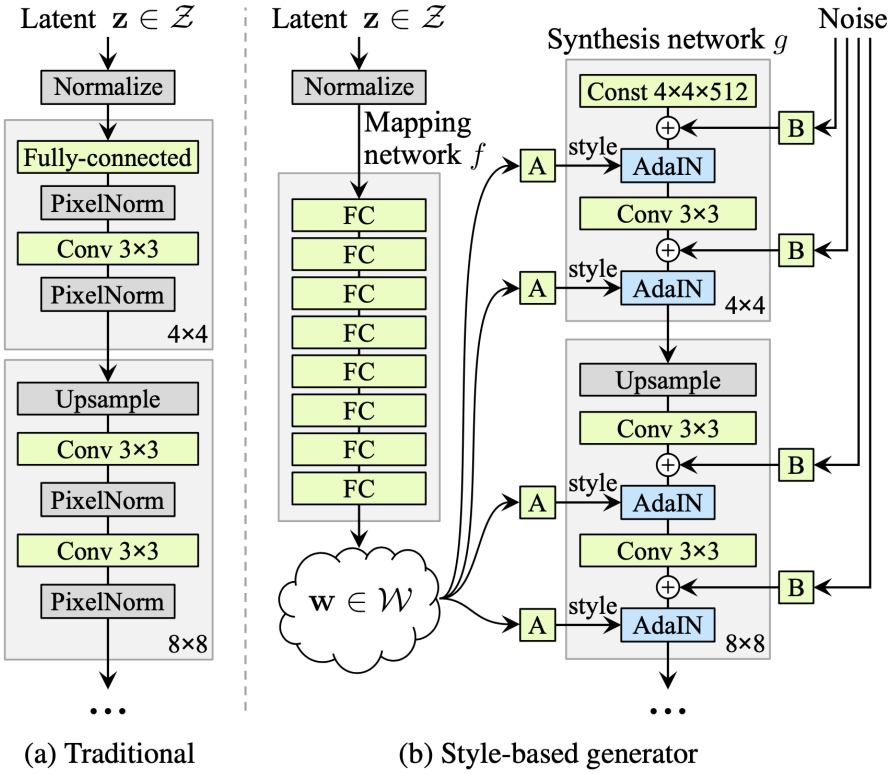


Figure 4.11: Style-based GAN architecture against traditional GAN

4.6.1 Functional Description

This module utilizes the power of *style-based* generative models, specifically `StyleGAN2` [5], to translate the required *latent vector* to a complete *human face image*. *Style-based GANs* (sometimes called *latent-based GANs*) can exert artistic control over the generated content (images, videos, text ..., etc). Consequently, we can tune it to fit our need and, iteratively, design it along with *code generation* 4.5 and *text processing* 4.4, in order to have a complete end-to-end pipeline for *text-to-face generation*.

- **Input :**
  - Low dimensional face embedding vector (latent vector).
- **Output :**
  - Complete human face image (portrait).

4.6.2 Modular Decomposition

As mentioned before, we opt to use *Style-based GANs* to be able to artistically control the output and designed the whole pipeline for *text-to-face generation*. Moreover, we choose

StyleGAN2, because it's one of the most popular and robust Style-based GANs in research literature. Also, it's relatively lightweight compared to other GANs used for the same purposes, but most importantly, StyleGAN2 excels at human face generation based on latent space. Figure 4.11 shows the original architecture of StyleGAN and how it is compared to traditional GANs. StyleGAN generator has two networks as follows :

- **Mapping network** creates nonlinear transformation to the input latent vector z $(512D)$. This transformation is not invertible and results in a $512D$ latent vector w. This latent vector w is expanded into several $512D$ vector using affine transformation, which gives the *extended latent vector* $w+$. The extended latent vector $w+$ dimensions depend on the dimensions of the output image.
- **Synthesis network** generates the synthetic image from *normally-distributed noise* guided by the extended latent vector $w+$.

StyleGAN2 [5] is a newer version that follows the same architecture, but with some modifications to further improve the control over latent space and the quality of the outputs.

So, let's discuss how we adapted StyleGAN2 to our work :

- To provide a high fidelity results, we target $1024X1024$ synthetic images. To achieve this, we have to use an extended latent vector $w+$ of dimensions $18X512$, meaning we repeat the latent vector w 18 times with *affine transformation* for each.
- To further improve feature directions disentanglement, we fine-tune StyleGAN2 using a subset of FFHQ dataset with increasing the weight of *perceptual path regularization* in the loss function. *Perceptual path regularization* in StyleGAN2 loss encourages the smooth mapping between latent and image spaces. So, when increasing it in certain directions, it highly penalizes the deviation between latent and image spaces in these directions giving more organized latent space. To avoid using the whole dataset, we use StyleGAN2 *adaptive discriminator augmentation* (**ADA**) [22] training methodology.
- Finally, we opt to remove the *mapping network* of *StyleGAN2* generator and only use the *synthesis network*. This is mainly because :
  - The mapping network doesn't satisfy the *path length regularization*, so there is no smooth mapping between latent space z and image space (only with latent space w). This is discussed in the original paper [5] and our experiments support that.
  - The mapping network is not invertible, unlike the synthesis network. So, we cannot reverse the transformation from image space to latent space z. That's why we only work with latent space w to test the consistency of the results.
  - Removing the mapping network reduces the computations and the memory footprint, which is crucial in our case.

After the previous modifications, the output model can directly translate the face embedding vector (*latent space*) into a complete human face portrait.

Notice that using this methodology of **code-to-face translation** along with **code generation** makes the sequential navigation in the latent space is *easily invertible*, which eases the

generation and the refinement (discussed in 4.7) of the synthetic face and gives the system versatility and fault tolerance. Figure 4.12 illustrates the idea of *invertibility* of sequential edits on a $2D$ latent space example. Here, we show a simplified $2D$ latent space with 3 feature direction. AB vector represents the *hair color* direction and BC vector represents the *gender* direction. Point A starts with a *blonde girl*, moving along AB vector results in a *girl with black hair* at point B. Then, moving along BC vector gives us a *man with black hair* at point c. Consequently, moving along CA vector, which is the opposite of the resultant of AB and BC, gives us the original face, thus inverting the sequential changes.

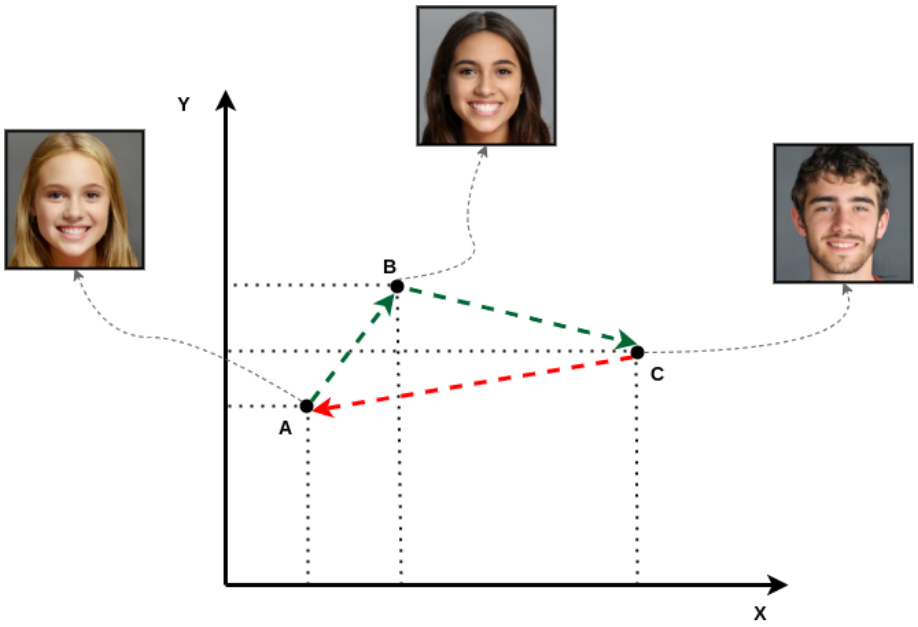


Figure 4.12: Illustration of sequential navigation and invertibility in a $2D$ latent space

4.6.3 Design Constraints

The basic design constraints for this module can be enumerated as follows :

1. **Network size** is surely one of our challenges. `StyleGAN2` has a large memory footprint, as the case with most *deep generative models*. This constrains its training and deployment. We managed to remove the *mapping network*, which gives us a change to use the full *synthesis network*.

2. **Faces dataset** is, also, a constraint, because real images of human faces contains entanglement between facial features (*as discussed before*). So, we have to exert extra effort in the **code generation** module to solve some of this entanglement, emerging from real human faces datasets.

4.7 Module 5 : Face Refinement

Figure 4.12 shows that `StyleGAN2` latent space can be used to perform *directed manipulation* by using the feature directions extract in **code generation** 4.5. We discussed how we use this technique for *conditional sampling* from latent space by starting from an initial seed and editing it until we reach the desired latent vector. Now let's discuss our methodology of applying the same technique is *directed manipulation* of one or more features to be used in *face refinement*.

4.7.1 Functional Description

This module basically gives the users a chance to further refine the generated face portrait to their liking. The user can refine the generated face using the same facial features used in **face generation**, along with some additional features related to *face morphology*, which are hard to describe in words. We adapt the techniques used in *face generation* to perform *face refinement*.

- **Input :**
  - Generated human face image (portrait).
- **Output :**
  - Refined human face image (portrait).

4.7.2 Modular Decomposition

We utilize the same technique used *face generation* to perform *directed manipulation* of one or more features. This is exactly what is required by *face refinement*. Consequently, we opt to stick with our *face generation* pipeline, due to the following reasons :

- Latent manipulation in `StyleGAN2` yields better results than most attribute-editing GANs.
- We use the same network, which significantly reduces the memory footprint.
- Since `StyleGAN2` has an artistic control over its output, we can expand the number of target facial features, which improves the system scalability.

Using latent space navigation, we can easily perform *directed manipulation* on a subset of the facial features, which is required for *face refinement*. The *directed manipulation* over a single facial feature can be formulated as :

$$E_{refined} = E_{old} + \delta_{feature} * d_{feature} \tag{11}$$

Where $E_{refined}$ is the refined latent vector, E_{old} is the old latent vector, $\delta_{feature}$ is the change offset of the facial feature and $d_{feature}$ is the feature direction.

Subsequently, we pass the *refined latent vector* through the *synthesis network* of `StyleGAN2` to produce the new refined face portrait.

Finally, more feature directions are generated for the purpose of *face refinement*, which are hard to describe in words. These features are more related to *face morphology* and listed as follows :

- Distance between eyes.
- Distance between eyes and eyebrows.
- Distance between nose and mouth.
- Eyes opening.
- Mouth opening.
- Smiling or not.

These feature directions are obtained using the same technique described in 4.5, however all of these features are clustered using regression techniques. Overall, we use both facial features used in *face generation* and the new facial features (which sum up to 38 features) to refine the generated face.

4.7.3 Design Constraints

The only constraint, imposed on this module, is the difficulty to generate the feature directions. The new features describe spare facial attributes related to morphology, so it's hard to regress them. We use *facial landmark detection* techniques to detect the distances, in order to fit the feature directions. We couldn't further expand the morphological feature directions, due to *limited resources* and *time constraints*. Our *recommendation* is that using the facial landmarks detection can actually further expand the morphological features, so it's worth spending more time on it.

4.8 Module 6 : Multiple Head Poses Generation

In the previous section we discussed how Face Refinement can be done by changing Facial attributes. One of the important attributes is changing the yaw angle of the generated face to a specific angle. This module is designed specifically for this purpose.

4.8.1 Functional Description

This modules is responsible of changing the yaw pose of the generated Face by providing a specific yaw angle.

- **Input :**
  - 2D Image for the generated Face.
  - Target yaw angle Θ
- **Output :**
  - 2D Image for the generated Face rotated with angle Θ

4.8.2 Modular Decomposition

The Multiple Head Poses Generation is performed on multiple steps:

- **3D Face Fitting :** The first step is generating a 3D Morphable Model (3DMM) from the input 2D Image. The resulting 3DMM is parametrized with the following parameters:
  - $\overline{S}$: The mean 3D Face Mesh
  - A_{exp}: The Eigen vectors responsible for Face Expression
  - A_{id}: The Eigen vectors responsible for the Face Identity
  - α_{exp}: The Face Expression parameters
  - α_{id}: The Face Identity parameters.
  - R: The Euler angles representing the Face Rotation in the 3D space
  - t_{2d}: Face translation matrix in the 2D space
  - f: Face scale parameter in the 2D space

  Thus the resulting 3D Model is represented as follows:

$$S = \overline{S} + A_{id}\alpha_{id} + A_{exp}\alpha_{exp}$$

  To project a 3d vertix of the model to the 2d space, The following equation is used:

$$V_{2d} = f * Pr * R(\overline{S} + A_{id}\alpha_{id} + A_{exp}\alpha_{exp}) + t_{2d}$$

Where Pr is the projection matrix.

In order to construct the 3D model the $R, alpha_{exp}, alpha_{id}, T_{2d}, f$ parameters need to be predicted from the input 2D image. We use a CNN with mobilenet architecture which we trained on AFLW dataset to regress the 3D Parameters by optimizing a parameter distance cost function which consists of the mean squared error between the actual parameters and the predicted parameters. The AFLW dataset consists of facial images and their corresponding 3DMM parameters.

At the end of the first stage we have a 3D model for the input image.

- **Rotating and Rendering the 3D model**: The second stage consists of rotating the resulted 3D model with the target angle in the 3D space and projecting it to the 2D Image. Pytorch's Neural Renderer is used to project the 3D Face Model into a 2D image. The texture information for the 3D model is taken from the input 2D Image by projecting each vertix into the 2D space and getting the corresponding pixel value.

  At the end of this stage we have a 2D Image that has the face model rotated and projected. But the 2D Image for the Face has missing parts which don't exist in the 3D Model such as the hair and teeth. In addition to the occluded parts of the face in the input 2D Image.

- **Face Inpainting**: In this stage, Face Inpainting is performed for the resulted 2D Image from the previous step to fill the missing gaps. The model for Face Inpainting is Cycle-GAN which takes an Input as the Image with gaps and predicts the final 2D Image with the complete face. The CycleGAN model is trained on a generated dataset which is generated by:

  - Generating 3D Model for the face.
  - Rotating the 3D Model with a random angle.
  - Project the 3D Model to 2D space.
  - Generating 3D Model for the 2D Image generated from the previous step.
  - Rotating back the Image with the previous angle to the initial pose
  - Projecting the Image to 2D Space yielding a pair of complete face image and face image with gaps.

  At the end of this stage, The result will be a 2D Image consists of the input face rotated with the input yaw angle.

4.8.3 Design Constraints

In order to design a Pose Generation system suitable for practice and to be integrated with the final application, Some Design constrained has been taken.

- **Models sizes**: Models sizes in this module were constrained to be small to fit into our deployment server which consists of 11 GB Nvidia 1080TI GPU and 16GB ram.

- **Models speed**: Models speed in this module were constrained to be fast so it can be run on CPU. So we made use of MobileNet CNN architecture which is lightweight and fast when it is run on CPU.
- **Output Image Size**: As GAN-based models can take around one month of training on a single GPU when it is trained on images of the 1080x1080 Resolution, We made the output image of our module to be 256x256 in order to make the training feasible given our limited computational power and to make the module fits in memory with the other modules in the system.

4.9 Module 7 : Web Application

As we described above, our users need the UI of the project to be as easy and familiar to them as possible, because they will use it in critical situations, such as a parent describes the criminal who kidnapped his/her child. So, we tried to make the both Front-end Design and Back-end Endpoints to be very simple and isolated from the complex techniques used in the background.

4.9.1 Functional Description

Our Application allows the user to get his/her intended face based on different generation modes that he/she can navigate through in the main page. Our generation modes are

- Generation from Voice Description
- Generation from Textual Description
- Generation using Manual Manipulators

After each mode, the user is allowed to do extra sequential refinements using manual manipulators, and to generate 8 different head poses of the final refined face.

4.9.2 Modular Decomposition

As shown above, we can decompose this module into two sub-modules, Front-end Design and Back-end API

4.9.2.1 Front-end Design

We used VueJS as our front-end framework with Axios as a request handler. Our front-end design consists of 6 pages

- Main Page

  This Page is just a page with single list of generation modes as shown in figure 4.13

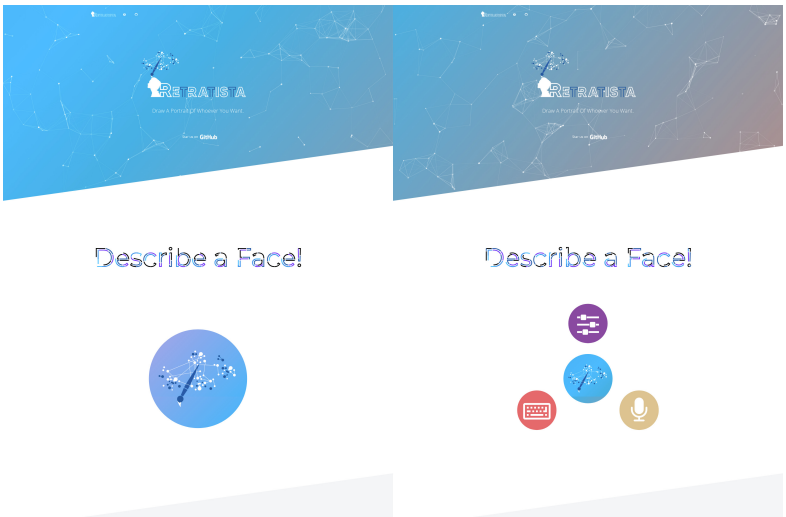


Figure 4.13: Main Page

- Generation From Voice Description Page

  This Page is a page with two different assets (voice and textual input) for the user to describe a face. After generation, it allows the user to refine the generated face or generate head poses, as shown in figure 4.14

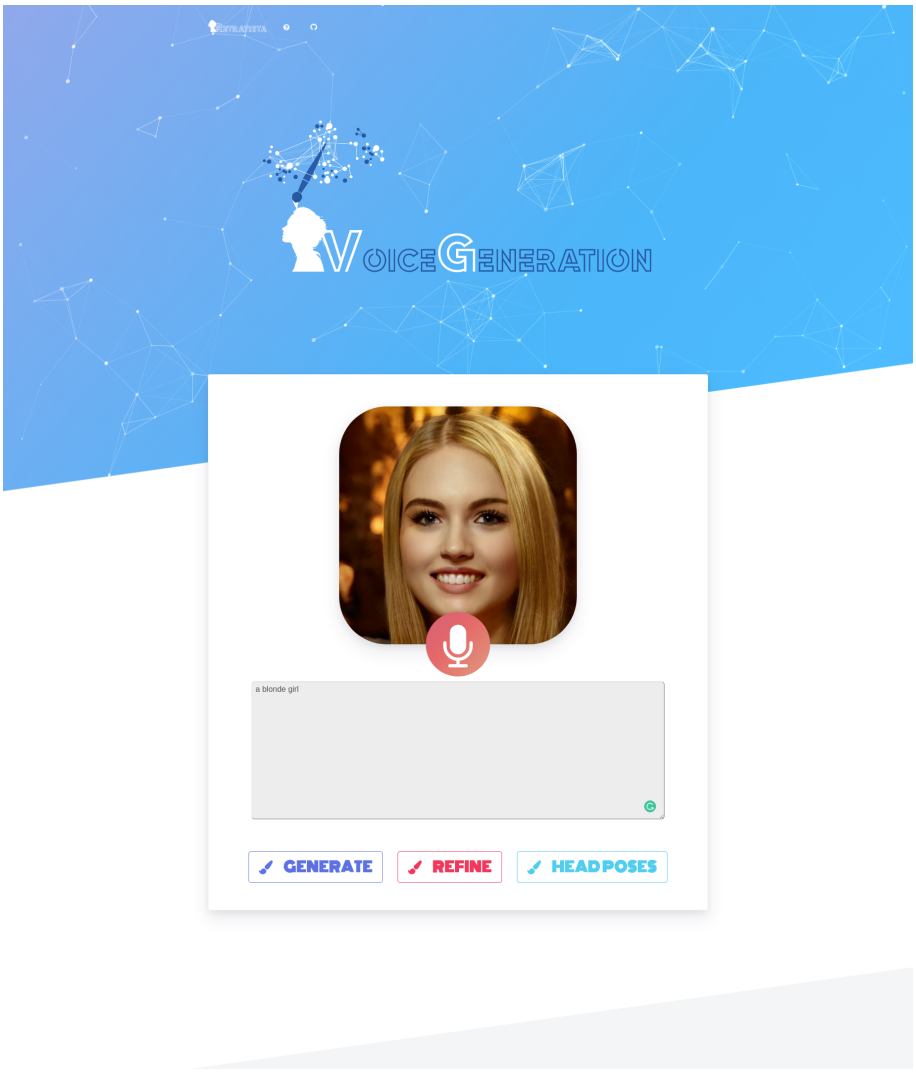


Figure 4.14: Generation From Voice Description Page

- Generation From Textual Description Page

  This Page is a page with one asset (textual input) for the user to describe a face. After generation, it allows the user to refine the generated face or generate head poses, as shown in figure 4.15

Figure 4.15: Generation From Textual Description Page

- Generation/Refinement Using Manual Manipulators Page

  This page allows the user to generate a new face or refine an actually-generated face using maual manipulators, as shown in figure 4.16

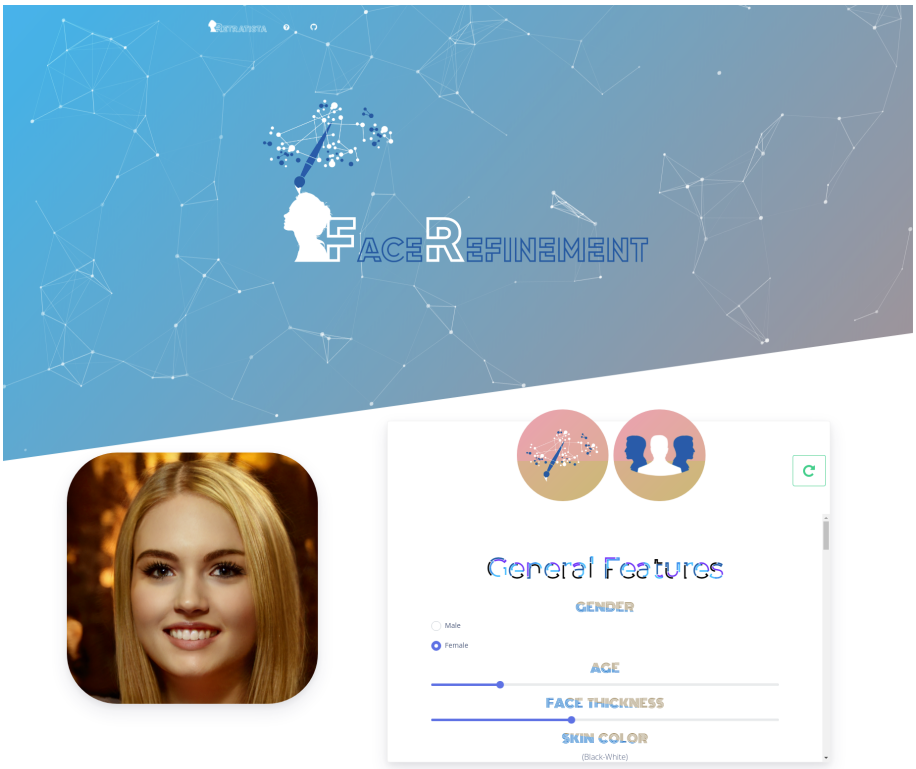


Figure 4.16: Generation/Refinement Using Manual Manipulators Page

- Head Poses Page

  This page generates eight different head poses for extra identification, as shown in figure 4.17

- Help and Demo Page

  This Page is a help aid for the user to know how to use the application and to support the user with some extra information about the project, such as

  - Why is Retratista important?
  - Statistics
  - Project Features
  - How To Use?
  - Result Samples
  - Team

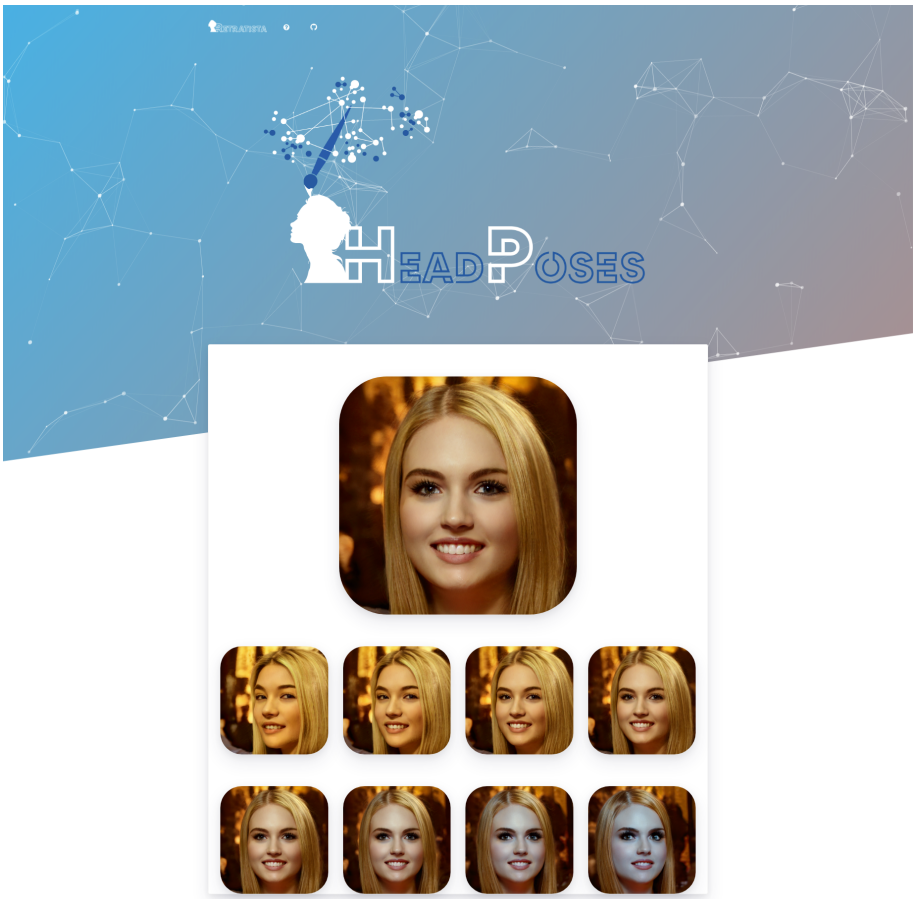


Figure 4.17: Head Poses Page

4.9.2.2 Back-end API

4.9.3 Design Constraints

As described above, the main constraints on this module are

- Simplicity of User Interface.
- Isolation of Complex Techniques used in background.

4.10 Other Approaches

5 System Testing and Verification

In this chapter, we discuss how the system is tested both on module level and integration level. We discuss our *performance metrics* and show the results of the system both *quantitatively* and *qualitatively*. We, also, show how our system performs compared to the baselines. Finally, we include the complexity analysis of our system for both time and memory.

5.1 Testing Setup

The testing environment of the whole system is basically targeting 4 main properties :

1. The quality of the output face images compared to the real human face images.
2. The ability of the system to capture the included facial features in the input description and how they actually map to the output.
3. The smooth and consistent mapping between the edits, imposed on the input description, and the changes of the output face image.
4. The Independence (*disentanglement*) between the different facial features.

We create our testing strategy to assess these 4 properties on the output face images. Also, we compare our results against the following baselines :

- `StyleGAN2` [5] : We compare our results with the original *StyleGAN2* to check the output quality.
- `Faces à la Carte` [11] : This is the only previous research work that attempted *Text-to-Face Generation*.
- `Image2StyleGAN` [6] : Our feature directions extraction methodology is inspired by this work, which is based on the first version of *StyleGAN*.

5.2 Testing Plan and Strategy

To assess the 4 previously-mentioned properties, multiple metrics are used, which are listed as follows :

1. **Fréchet Inception Distance** (*FID*) [23] : This metric is an improvement over the traditional *inception score* to be able to measure the similarities between a set of real and synthetic images. Basically, *inception score* measures the ability of `Inception V3` network [24] to classify a synthetic image into 1000 classes. However, *FID* measures the distance between synthetic and real images. This is done by extracting $2048D$ feature vector from each image using the `Inception V3` network and then calculating the *Fréchet* distance using :

$$d^2 = ||\mathbf{u}_1 - \mathbf{u}_2||^2 + Trace(\mathbf{C}_1 + \mathbf{C}_2 - 2 * \sqrt{\mathbf{C}_1 * \mathbf{C}_2}) \tag{12}$$

Where **u** is the feature-wise mean vector and **C** is the covariance matrix of the feature vector.

2. **Learned Perceptual Image Patch Similarity** (*LPIPS*) [25] : This metric measures the smoothness of the mapping between the latent space edits and the output image changes. This metric takes as an input, two synthetic images. It uses a pretrained neural network to project them to a latent space. Then, it calculates the difference between the two latent vectors, along with the perceptual distance between the two images. Finally, it uses the two distances to calculate the final score. This metric is used in `StyleGAN2` paper to assess the *perceptual path length*.

3. **Edit Consistency Score** : We use this metric to ensure that the final facial attributes values are consistent with the input values after the *latent manipulation* process. This metric is simply calculated by projecting the final latent vector over all feature directions and compare it to the input values.

4. **Directions Disentanglement Score** : The disentanglement between feature directions are assessed by using the *angles* between each pairs of directions. Angles of values 85 to 95 degrees usually indicates low entanglement.

5.2.1 Module Testing

5.2.1.1 Speech Recognition

5.2.1.2 Text Processing

5.2.1.3 Code Generation

To test the quality of the generated feature directions that are used for *code generation*. We use two methods, which are **directions disentanglement scores** and **visual result** of moving along directions.

Table 5.1 shows the angles between the directions of a subset of features. Remember that angles in range 85 to 95 degrees indicate low entanglement. Consequently, we can infer that the number, provided by the table, are reasonable. For example, the angle between *gray hair* and *age* directions is 79.6 degrees, because old people normally have gray hair. Also, men are *not* likely to wear makeup, so the angle between *makeup* and *gender* directions is 107.7 degrees, same for *beard* with men.

Moreover, figure 5.1 shows the *visual results* of moving along some feature directions. We use these results to qualitatively measure the accuracy of the extracted feature directions.

Angles	Age	Gender	Beard	Gray Hair
Age	0.0	92.4	85.8	79.6
Gender	92.4	0.0	80.0	88.6
Makeup	88.0	107.7	100.5	94.5
Hair Length	89.7	95.9	90.6	96.6

Table 5.1: Angles between different feature directions using a subset of the considered facial features (closer to 90 degrees is better).

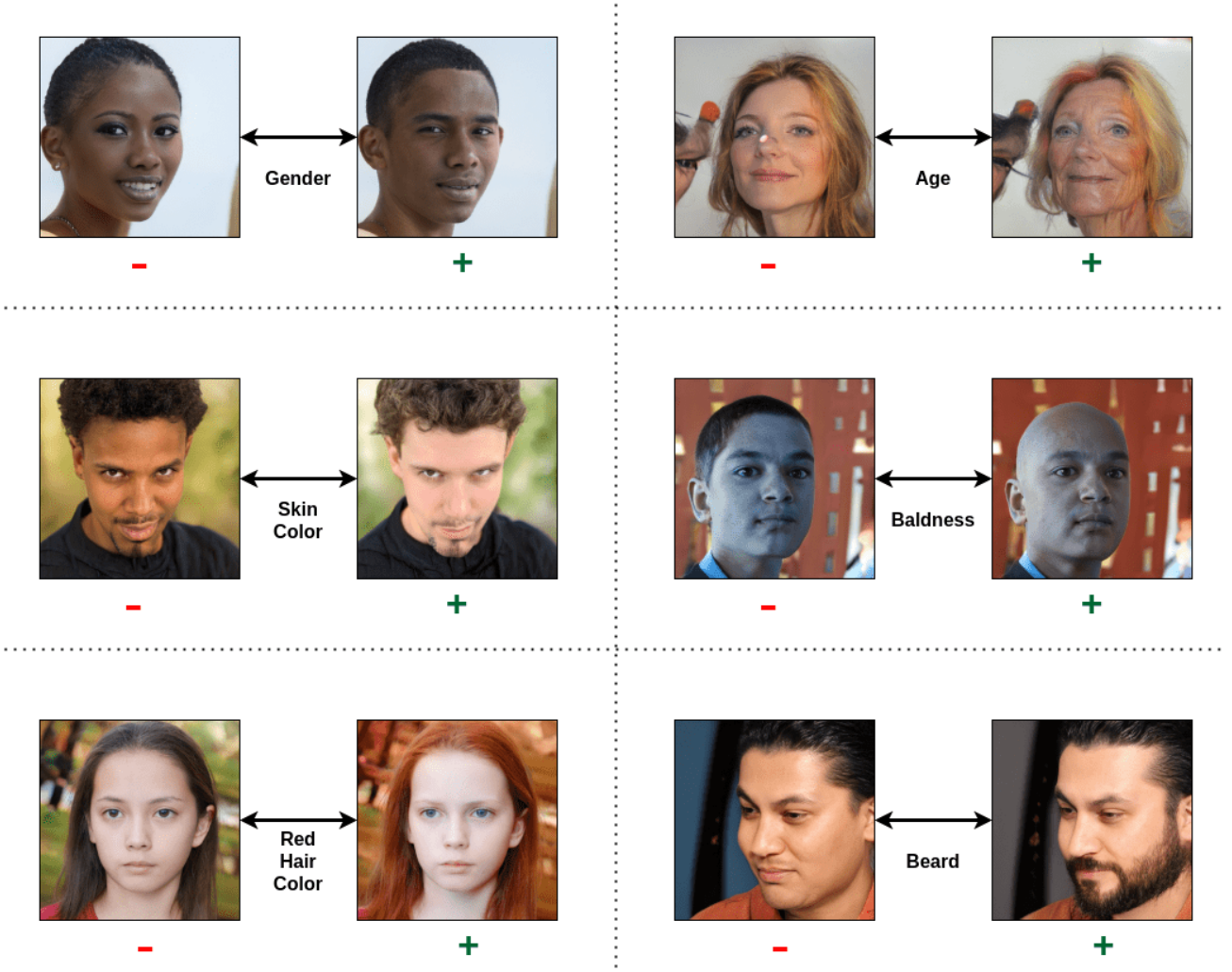


Figure 5.1: The results of moving along some extracted feature directions.

5.2.1.4 Code-to-Face Translation

This module is basically tested using integration with **code generation**, which is shown in the **integration** test. However, we perform testing on this module separately using **edit consistency score** of the facial attributes values of the output image and the input values.

As figure 5.2 shows, the system can convert a random vector (*on the left*) to the final latent vector (*on the right*) driven by the input values (*on top*). Also, we can see the consistency between the required values and the values corresponding to the generated face.

Also, table 5.2 shows the relation between *LPIPS* score (between the intial and output images) and the number of directions, navigated during face generation. We can see that the score increases, as the number of navigated directions increases. This is mainly because the output face image is far from the initial face image. However, we can see that some *anomalies* can occur, like the case of the input text *"Woman with lipstick and rosy cheeks"*, which navigates along only 3 directions, but gives a high *LPIPS* score.

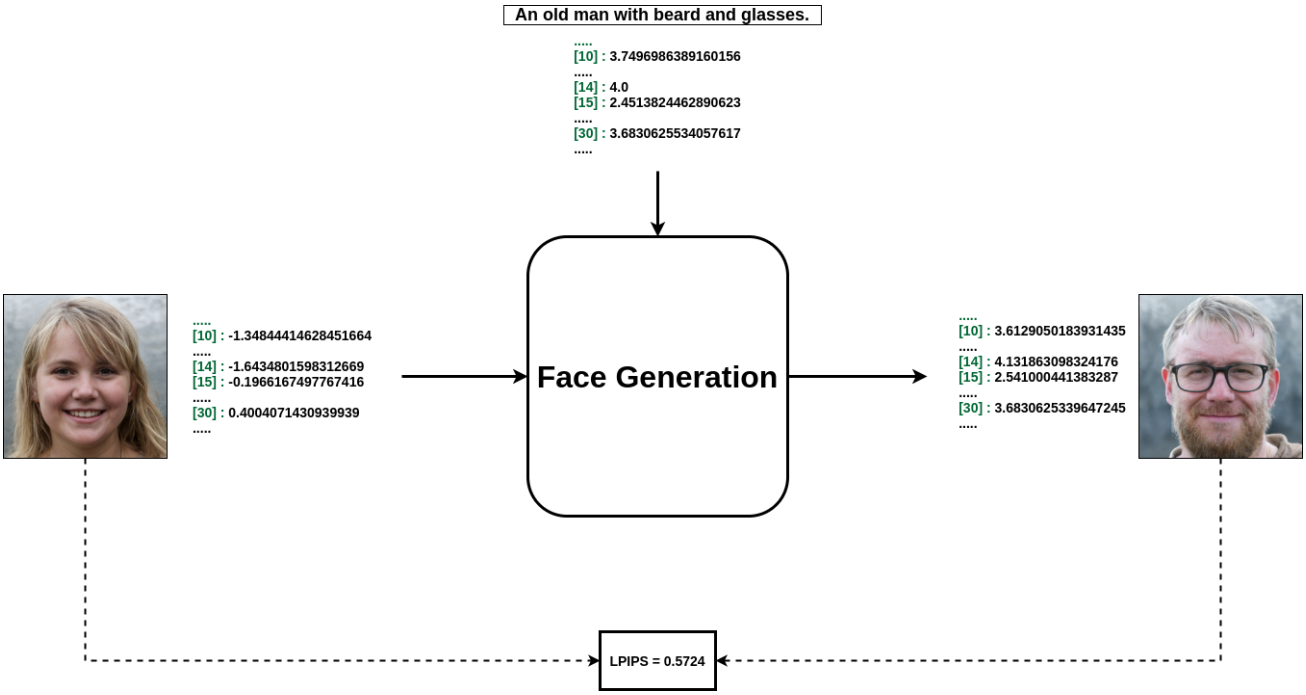


Figure 5.2: An example of the consistency in reaching the required facial attributes starting from initial random vector.

Input Text	Number of Navigated Directions	LPIPS
Female with chubby face	2	0.3905
Woman with long wavy hair	3	0.4412
Old black man with glasses	4	0.5023
Woman with lipstick and rosy cheeks	3	0.5107
Young black man with long hair and beard	5	0.5733
Old man with chubby face and glasses	4	0.4905

Table 5.2: LPIPS values against the number of navigated directions for sample text (lower is better).

5.2.1.5 Face Refinement

Since, we adapt `StyleGAN2` for the refinement process, so this module testing is almost the same as **code-to-face generation** module. However, we focus more on the visuals of the *sequential navigation*, along with the new features related to the face morphology. Figure 5.3 shows the results of sequential navigation given an original synthetic face image. We tried to maintain the independence of sequential direction navigation as much as possible to keep the results visually acceptable. For example, in the first row, we convert from a *"young man with hear and no beard"* into an *"old bald man with beard"* by sequential navigation using *age*, *beard* and *baldness* directions. This yields much better visual results than many recent *attribute-editing GANs*.

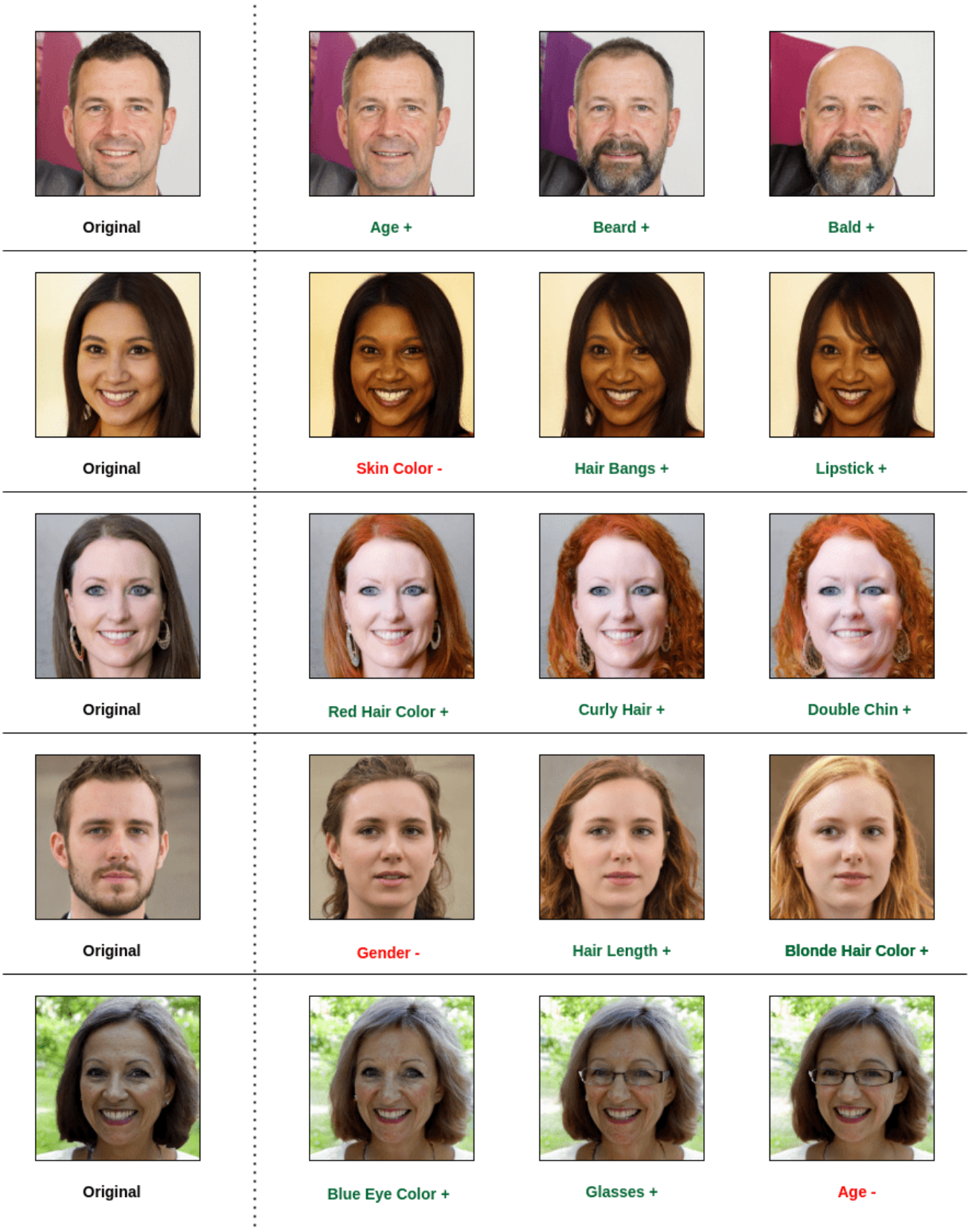


Figure 5.3: The results of sequential navigation along certain feature directions.

5.2.1.6 Multiple Head Poses Generation

5.2.2 Integration Testing

The integration of the whole system into a web application is tested qualitatively and quantitatively. In this section, we show the visual results of our system in an *end-to-end* manner. However, we show the quantitative metrics in the comparison with previous work.

Figures 5.4 and 5.5 show correct results of **face generation** in an *end-to-end* manner using our final *web application*, which contains our complete pipeline. We can see that the results samples are of high accuracy and fidelity. However, of course, the system is *not* 100% accurate. From our extensive testing to the system, we notice $4types$ of failures, described visually in figure 5.6. These failures are listed as follows :

1. Failures due to **contradicting facial features**. As in the *top-left* image, as *rosy cheeks* are very hard to capture with *black skin color*.

2. Failures due to **excessive navigation on directions**. As in the *top-right* image, which is very unclear and of low quality with many visual artifacts.

3. Failures due to **random initialization**. As in the *bottom-left* image, where sometimes bad initial latent vector can cause the output to be visually inconsistent with many visual artifacts.

4. Failures due to **sequential latent manipulation**. As in the *bottom-right* image, where navigation on one direction (*wavy hair*) cancel the navigation on another direction (*short hair*).

Young bald man with beard and glasses.

Young girl with blonde hair and blue eyes.

Young woman with brown wavy hair. She is putting on lipstick and wearing glasses.

Old black man with grey hair.

Figure 5.4: Samples of correctly generated face portrait from textual description.

A bald Asian man with beard. He is wearing glasses.

A white woman with short brown hair. She has hair bangs and is wearing glasses.

An old chubby man with double chin and rosy cheeks. He is wearing glasses and has receding hairline.

A young woman with blonde hair and rosy cheeks. She is putting on slight makeup and her hair is short.

Figure 5.5: Samples of correctly generated face portrait from textual description.

Figure 5.6: Samples of incorrectly generated face portrait from textual description.

5.3 Testing Schedule

Our testing process is scheduled as follows :

- First, we conducted the initial testing on the 3 core modules, while being iteratively designed, until they converged to decent results.
- We, then, did the integration testing on these modules separately.
- After so, the rest of the system modules were designed and tested separately.
- Finally, the whole system was integrated into a single web application and complete testing of the whole system functionalities was conducted.

5.4 Comparative Results to Previous Work

First of all, it's worth mentioning that our system considers *more facial features* than all previous research that worked on latent manipulation. As mentioned before, we compare our results quantitatively with 3 baselines, using **FID score**, **average LPIPS** and **execution time**.

Table 5.3 and plot 5.7 show the comparison of our system to `StyleGAN2` and `Image2StyleGAN`. We couldn't include `Faces à la Carte` in this comparison, as the authors didn't include it in

the paper. Also, we couldn't replicate their work, as they didn't provide any specific details about the implementation. We can see that as the number of test images increases, the overall quality of the images increases (*FID* score decreases). Our system performs better than `Image2StyleGAN`, but worse than the original `StyleGAN2`, because it is just image generation from random vector with no *latent manipulation* (fewer artifacts).

Test Size	StyleGAN2	Image2StyleGAN	Our System
50	129.15	179.58	151.58
100	104.29	150.54	132.54
200	95.25	136.45	114.45
300	92.44	134.04	113.04
400	86.37	121.59	100.59
500	80.09	119.02	99.02

Table 5.3: FID scores comparison on different number of test images (lower is better).

Figure 5.7: Plot of FID score of different pipelines against different number of test images (lower is better).

Next, we compare our system with `Faces à la Carte` using the only metric, the authors provided, which *LPIPS*. We can see from table 5.4 that our system yields lower overall *LPIPS*, which is better. However, we have higher error margin than `Faces à la Carte`.

Our System	Faces à la Carte
0.595±0.008	0.634±0.005

Table 5.4: LPIPS comparison with Faces à la Carte (lower is better).

Finally, table 5.5 shows the execution time of different core stages of our system. We can see that the overall *text-to-face generation* process takes only about 0.2 seconds on an *Nvidia GTX 1080 ti*. However, the web application can take from 1 second up to multiple

seconds to generate a face portrait (from any description) using the same GPU, depending on the *connectivity* with the server. Meanwhile, our *multiple head poses generation* module takes around 5 seconds on the same GPU as well. Most of the application backend runs on *CUDA GPUs*. The whole system without **multiple head poses generation** module occupies around 5.5 *GB* of *VRAM*. Meanwhile, this module occupies up to 4 *GB*. So, in total, the whole system occupies between 9 to 10 *GB* of *VRAM*.

Text Processing	Latent Manipulation	Face Generation
0.048	0.11	0.024

Table 5.5: The execution time of different stages of the core of our system (measured in seconds).

6 Conclusions and Future Work

6.1 Faced Challenges

6.2 Gained Experience

6.3 Conclusions

6.4 Future Work

References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[2] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis, 2019.

[4] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

[5] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan, 2020.

[6] Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2stylegan: How to embed images into the stylegan latent space?, 2019.

[7] Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2stylegan++: How to edit the embedded images?, 2020.

[8] Yujun Shen, Ceyuan Yang, Xiaoou Tang, and Bolei Zhou. Interfacegan: Interpreting the disentangled face representation learned by gans, 2020.

[9] Ayush Tewari, Mohamed Elgharib, Gaurav Bharaj, Florian Bernard, Hans-Peter Seidel, Patrick Pérez, Michael Zollhöfer, and Christian Theobalt. Stylerig: Rigging stylegan for 3d control over portrait images, 2020.

[10] Rameen Abdal, Peihao Zhu, Niloy Mitra, and Peter Wonka. Styleflow: Attribute-conditioned exploration of stylegan-generated images using conditional continuous normalizing flows, 2020.

[11] Tianren Wang, Teng Zhang, and Brian Lovell. Faces à la carte: Text-to-face generation via attribute disentanglement, 2020.

[12] Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder, 2021.

[13] Zhenliang He, Wangmeng Zuo, Meina Kan, Shiguang Shan, and Xilin Chen. Attgan: Facial attribute editing by only changing what you want, 2018.

[14] Chen Ding, Wei Kang, Jiaqi Zhu, and Shuangyan Du. Injectiongan: Unified generative adversarial networks for arbitrary image attribute editing. *IEEE Access*, 8:117726–117735, 2020.

[15] Ying-Cong Chen, Huaijia Lin, Michelle Shu, Ruiyu Li, Xin Tao, Yangang Ye, Xiaoyong Shen, and Jiaya Jia. Facelet-bank for fast portrait manipulation, 2018.

[16] Hang Zhou, Jihao Liu, Ziwei Liu, Yu Liu, and Xiaogang Wang. Rotate-and-render: Unsupervised photorealistic face rotation from single-view images, 2020.

[17] Joel Ruben Antony Moniz, Christopher Beckham, Simon Rajotte, Sina Honari, and Christopher Pal. Unsupervised depth estimation, 3d face rotation and replacement, 2018.

[18] Avishek Bhattacharjee, Samik Banerjee, and Sukhendu Das. *PosIX-GAN: Generating Multiple Poses Using GAN for Pose-Invariant Face Recognition: Munich, Germany, September 8-14, 2018, Proceedings, Part III*, pages 427–443. 01 2019.

[19] Y. Hu, X. Wu, B. Yu, R. He, and Z. Sun. Pose-guided photorealistic face rotation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8398–8406, 2018.

[20] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin, 2015.

[21] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[22] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data, 2020.

[23] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[24] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[25] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018.

A Development Platforms and Tools

A.1 Software Tools

B Use Cases

C User Guide

D Feasibility Study

D.1 Technical Feasibility

To conduct this project knowledge base in the field of image processing and mathematics is required, to build on the knowledge in artificial intelligence, the use of generative adversarial neural networks and supervised learning to build and improve the models that perform the tasks of source separation. Also experience in web designing and launching to represent our product to any user or organization, so that they buy it. We used our devices to train the models besides the online free service; Google Colab.
Sufficient attempts were made in the field of MSS solutions that performed well on the problem. Additionally most of the developed models were guided by published research papers, therefore indicating and emphasize the feasibility of the solution.

D.2 Operational Feasibility

As mentioned before we deal with two types of users, one uses the application for special purposes; finding missing people or identifying the image of some ancient king or queen, so they need to have at least one PC with VRAM 11GB or more, to be able to run the application which will cost around 1500$ to 2000$ or more depends on the number of PCs will be used and their specifications. For the other type of users we can deploy the app for free as a trial, then a little expense is asked from subscribed users so that the deployment cost is covered with profits.

D.3 Legal Feasibility

The development of the project depends on publicly available libraries and software tools, like pytorch, flask and vuejs. All the pretrained models used are open sources and legal to be used for any product. All Datasets used to train any written model are public for any commercial use, or made by us. The devices used to train those models are our own or on Google Colab which is also public for any use.

D.4 SWOT Analysis

D.4.1 *Strength S*

- Fast.
- Automated.
- Easy-to-use.
- Realistic Results.
- Containing a Sufficient set of features to describe a human face.

D.4.2 *Weaknesses W*

- Facial Features Entanglement.
- High VRAM Usage.
- Sometimes Results can be inaccurate.

D.4.3 *Opportunities O*

- No Competitors.
- Fast and Automated.

D.4.4 *Threats T*

- Lake of Datasets.
- High Cost.