Ain Shams University
Faculty of Computer & Information Sciences
Scientific Computing Department

Speech2Face Generator

July 2023

Ain Shams University
Faculty of Computer & Information Sciences
Scientific Computing Department

Speech2Face Generator

By

Sara Mohamed Baza	Scientific Computing
Nourhan Mahmoud	Scientific Computing
Ahmed Mohamed	Scientific Computing
Abdelrahman Yasser	Scientific Computing
Abdelrahman Mohamed	Scientific Computing
Ahmed Magdy Abdelrahman	Scientific Computing

Under Supervision of

Associate Prof. Manal Tantawy,
Scientific Computing Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

T.A. Aya Nasser,
Scientific Computing Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

T.A. Mirna Al-Shetairy,
Computer Science Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

July 2023

Acknowledgement

Primarily, we would thank God for being able to complete this project with success. Then We would like to express our thanks and gratitude to our supervisor Dr. Manal Tantawy for her guidance, understanding and support throughout the project.

Also, we would like to show our deep appreciation to T.A. Mirna Al-Shetairy and T.A. Aya Nasser as we were able to successfully complete this project with their help, support and patience.

Finally, we want to thank all our dear friends and colleagues for their valuable suggestions and help.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

In this innovative project, we delve into the realm of artificial intelligence to bridge the gap between spoken words and visual representation. Using advanced deep learning techniques, we have developed a cutting-edge model that can transform human speech into a lifelike image of the speaker's face. Human communication is a complex interplay of verbal and non-verbal cues. While words convey meaning, facial expressions and gestures add depth and emotion to our interactions. [2] Our project aims to unlock the potential of speech by creating a visual counterpart, allowing us to better understand the holistic nature of human expression.

The implications of speech-to-face generation are vast and diverse. From enhancing virtual reality experiences and animated movies to aiding in various fields like law enforcement, this technology holds immense potential. Imagine a world where a simple voice message could be transformed into a vivid visual representation, creating a more immersive and personalized communication experience. [1,3]

In this project, we have made significant strides toward realizing this vision. Through extensive training on large datasets of speech and corresponding facial images, our model has learned the intricate correlations between vocal nuances and facial expressions. The results are remarkable, with generated images closely resembling the speaker's unique features, capturing the essence of their identity.

1.1 Motivation

The project draws inspiration from several compelling motivations, including Enhancing Communication: Communication transcends mere verbal exchanges; non-verbal cues and facial expressions play an indispensable role in conveying meaning. By harnessing the power of generating realistic faces from speech, we seek to enrich communication experiences, particularly in contexts where visual cues are restricted, such as audio-only calls or voice messaging platforms.

Transforming Entertainment Industries: Within animation, gaming, and film industries, voiceover performances constitute a cornerstone of creative expression. The ability to generate visually immersive representations of characters solely based on the performances of voice actors has the potential to revolutionize

entertainment, offering seamless integration of voice and visuals, and elevating the audience's engagement and immersion.

Assisting Forensic Investigations*:* In the realm of forensic investigations, our system can provide invaluable help. By generating facial images from audio recordings, this technology has the potential to aid law enforcement agencies in identifying suspects, reconstructing the appearance of unknown individuals, and potentially uncovering crucial leads in unsolved cases.[4]

Pioneering Technological Advancements: Beyond the immediate applications in communication and entertainment, the development of a reliable and accurate "Speech to Face Generator" system holds transformative potential for various domains. It has the capacity to inspire breakthroughs in human-computer interaction, virtual reality, augmented reality, and numerous other fields, offering unprecedented opportunities for technological advancements and novel use cases

1.2 Problem Definition

For a long time, there have been several threatening crimes, such as extortion with images or death threats, among others. But what all of this has in common is that you may easily obtain the blackmailer's voice or at least a 5-second audio recording. Knowing only the voice of the criminal will not help anyone to catch him, and most of those crimes have been closed to unknown criminals.

Policemen need something more than just a voice, they need at least an image of that criminal. So, our system "speech2face generator" will take this audio and reconstruct a person's facial image from a brief audio recording of that person speaking to know the speaker and prevent the crime.

1.3 Objective

The objective is to meticulously develop and deploy an advanced system capable of generating highly realistic visual representations of individuals based exclusively on their recorded voices. By leveraging state-of-the-art deep learning techniques, sophisticated audio analysis algorithms, and cutting-edge models, our aim is to bridge the perceptual divide between audio and visual information,

thereby pushing the boundaries of audio-visual synthesis. The specific objectives of this project encompass:

- Designing and implementing an intricate deep learning architecture proficient in extracting pertinent features from audio recordings and translating them into detailed facial attributes.
- Training the model using an extensive dataset of audio-visual pairs, ensuring the acquisition of accurate and visually plausible facial generation capabilities.
- Pioneering innovative methodologies to address inherent challenges including limited training data, variations in voice quality, and the diverse range of facial appearances encountered in real-world scenarios.
- Conducting comprehensive evaluations of the system's performance, employing rigorous quantitative metrics and meticulous user studies to quantitatively and qualitatively assess the fidelity and likeness of the generated faces.
- Optimizing the system for real-time or near real-time face generation, thereby fostering practical applications in communication and entertainment industries, where timely and seamless integration of audio and visual components is of utmost importance.
- Demonstrating a steadfast commitment to ethical considerations and privacy protection by rigorously adhering to responsible data usage practices, ensuring robust safeguards against potential misuse or unauthorized access to generated visual content.

1.4 Time Plan

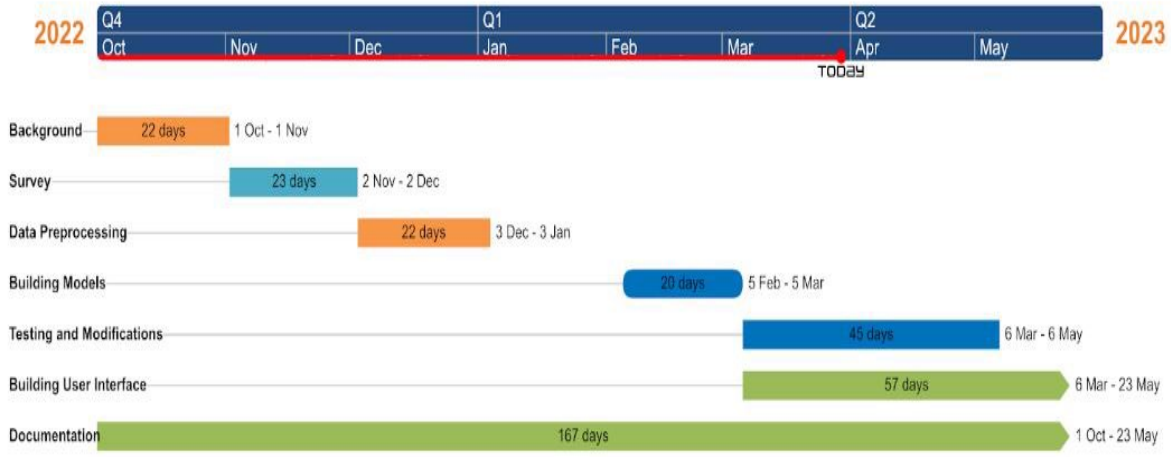


Figure 1.1: Time plan

- 22 days for background establishing.
- 23 days for studying the survey, literature published and related work.
- 22 days for preparing the project data and preprocessing.
- 20 days for building the models and project architecture.
- 45 days for the testing and model modification in parallel with trying another model architecture.
- 57 days for integrating the model and building the user interface.
- Working on the documentation in parallel with the whole work.

1.5 Document Organization

Chapter 2: Background

This chapter includes background about the project, basic concepts, and the related work according to our research.

Chapter 3: Analysis & Design

This chapter includes an overview of the whole system along with the intended user, system architecture, analysis, and design.

Chapter 4: VAE Model

This chapter includes an overview of the dataset used in the model, describes the system functions with details about the implementation of the project's modules and includes some of the experiments that we did through our work to improve the results.

Chapter 5: GAN'S Model

This chapter includes an overview of the dataset used in the model, describes the system functions with details about the implementation of the project's modules, and includes some of the experiments that we did through our work to improve the results

Chapter 6: User Manual

This chapter talks about the needed packages and libraries that must be installed before using the application, and also shows the user how to use it.

Chapter 7: Conclusion & Future Work

This chapter includes the conclusion and results of our work and the future work that may be done based on this project.

References: This chapter includes the papers, books, and links that were used in the survey and in implementation.

Chapter 2: Background

Reconstructing a face from someone's voice is an important project. It could aid authorities in solving crimes by generating a picture of a suspect's face from their voice. It might also be used to create characters more lifelike in movies and video games, and it could help people with disabilities communicate more effectively. This project has the potential to improve our lives in a variety of ways.

2.1 Digital Signal Processing

DSP stands for Digital Signal Processing, which is a technique used to process digital signals using mathematical algorithms. Digital signals are discrete-time signals that are represented as a sequence of numbers, and are commonly used in a wide range of applications, including audio and speech processing, …, etc.
DSP techniques, such as filtering, modulation, compression, and feature extraction, are used to modify and analyze digital signals. To extract valuable information from digital signals.

2.1.1 Audio Signal

Audio signals are a type of analog or digital signal that represents sound waves in the form of electrical or digital signals. Sound waves are created when an object vibrates, causing changes in air pressure that propagate through space as waves. These waves can then be detected by the human ear or by a microphone, which converts the sound waves into an electrical signal.
In analog audio signals, the electrical signal is a continuous representation of the sound wave, varying in amplitude and frequency over time. Analog audio signals are commonly used in traditional audio systems, such as analog radio.
In digital audio signals, the electrical signal is transformed into a series of binary values that represent the amplitude and frequency of the sound wave at discrete time intervals. Digital audio signals are commonly used in modern audio systems, such as CDs, MP3s, and digital radio.

2.1.2 Noise In Audio Signal

The Noise in Audio Signals can degrade the quality of the signal, making it more difficult to analyze and understand.

Common Types of Noise in Audio Signals:

1. White Noise: a type of noise that has a flat frequency response across all frequencies.
2. Pink Noise: a type of noise that has a frequency response that decreases as the frequency increases.
3. Hum: a type of noise that is caused by electrical interference.
4. Hiss: a type of noise that is caused by electronic noise in the recording or transmission system.
5. In Our Project, we assume that the silence or non-speech frames are a type of noise.

Therefore, techniques for reducing or removing noise from audio signals are important in many applications, such as speech recognition, audio compression, and music production. Common noise reduction techniques include filtering, spectral subtraction, and adaptive noise cancellation.

2.1.3 WebRTC VAD

The WebRTC VAD library is a widely used VAD implementation, which is based on machine learning techniques and designed to work well in noisy environments. The algorithm is based on the assumption that speech signals exhibit certain statistical properties that are different from non-speech signals, and uses these properties to distinguish between speech and non-speech.

The WebRTC VAD algorithm works by processing audio signals in short overlapping frames, typically with a duration of 10 to 30 milliseconds. For each frame, the algorithm computes a set of features that capture the statistical properties of the signal, such as the energy, zero-crossing rate, and spectral entropy. Once the features are computed, they are fed into a machine learning model as shown in Figure 2.1, which has been trained on a large dataset of speech and non-speech signals. The model uses the features to classify the signal as speech or non-speech, based on the statistical patterns that are typical of speech signals.

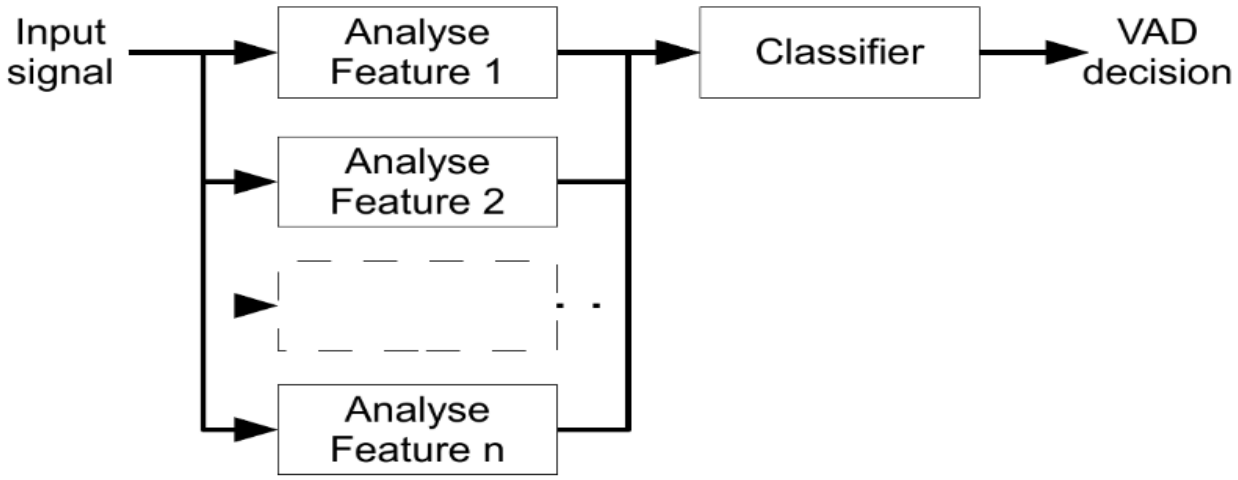


Figure 2.1 WebRTC VAD

2.1.4 Mel-Frequency Cepstral Coefficients (MFCC)

Mel Frequency Cepstral Coefficients (MFCC)[5] is a popular feature extraction technique in speech and audio processing. MFCCs are a concise representation of a signal's spectral features that can be used for speech recognition, speaker identification, and emotion detection.

The computation of mel-frequency cepstral coefficient (MFCC) features typically involves a seven-step process:

1. Pre-emphasis: The first step is to apply a pre-emphasis filter to the signal to boost the high-frequency components and reduce the effect of low-frequency noise.
2. Framing: The signal is divided into short overlapping frames, typically with a duration of 20 to 30 milliseconds. Each frame is usually windowed using a Hamming or Hanning window to reduce spectral leakage.
3. Fourier Transform: A Fast Fourier Transform (FFT) is used to compute the power spectrum of each frame.
4. Mel Filterbank: The power spectrum is then passed through a bank of Mel frequency filters, which are designed to mimic the non-linear frequency resolution of the human ear. The filterbank typically consists of 20-40 triangular filters spaced uniformly on the Mel frequency scale, which is a logarithmic scale that maps frequency to perceptual pitch.
5. Logarithmic compression: The output of each filter is converted to the logarithmic scale to compress the dynamic range of the spectrum.

6. Discrete Cosine Transform: The Discrete Cosine Transform (DCT) is then applied to the log filterbank energies to obtain a set of cepstral coefficients. Typically, the first 10 – 15 coefficients are retained, while the rest are discarded as they contain little useful information.
7. Normalization: The resulting MFCCs are usually normalized to have zero mean and unit variance, to ensure that they are comparable across different speakers and recordings.

The resulting MFCC features capture the spectral characteristics of the signal in a compact form, by representing the power spectrum in terms of a few cepstral coefficients. These features are commonly used as input features for machine learning algorithms such as Hidden Markov Models (HMMs), which are widely used in speech recognition.

2.1.5 Short-Time Fourier Transform (STFT)

STFT[6] is a signal processing technique that was used to analyze the frequency content of a signal over time. The STFT is a Fourier Transform extension, which is a mathematical approach for converting a signal from the time domain to the frequency domain.
The STFT computes the Fourier Transform of the signal's brief overlapping windows, which are often windowed using a Hamming or Hanning window to minimize spectral leakage. For efficient computing utilizing the Fast Fourier Transform (FFT) technique, the length of the window is commonly designed to be a power of two.
The STFT (Short-Time Fourier Transform) is typically computed using the following steps:

1. Divide the signal into overlapping frames of length N and apply a window function to each frame. The window function is usually a symmetric function that tapers the edges of the frame to reduce spectral leakage. Commonly used window functions include the Hamming, Hanning, and Blackman windows.
2. Compute the Fourier Transform of each windowed frame using the FFT (Fast Fourier Transform) algorithm. The FFT is an efficient algorithm for computing the Discrete Fourier Transform (DFT) of a sequence of length N, which has a computational complexity of O(N log N).

3. Compute the magnitude of the complex-valued Fourier coefficients at each frequency and time index. The magnitude represents the strength of each frequency component in the signal at each time index.
4. Optionally, compute the phase of the complex-valued Fourier coefficients at each frequency and time index. The phase represents the relative phase of each frequency component in the signal at each time index.

The generated STFT coefficients describe the frequency content of the signal at each time index and can be utilized for audio analysis, noise reduction, and speech processing, among other things. Other frequency-domain representations, such as the spectrogram, which is a visual depiction of the STFT coefficients, can also be computed using the STFT.

The STFT is a versatile tool for analyzing and modifying audio signals frequently used in signal processing and machine learning.

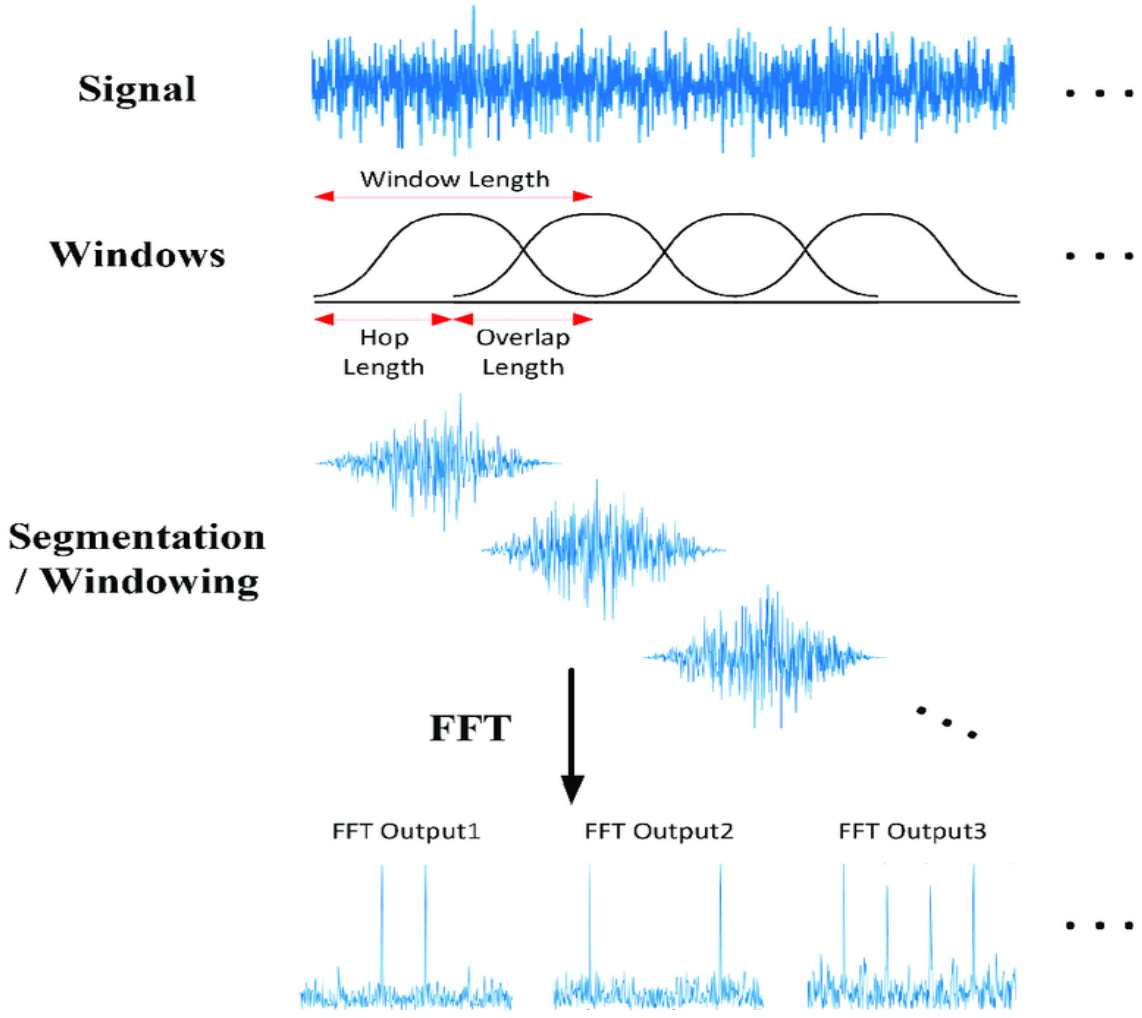


Figure 2.2 STFT

2.2 Neural Networks

2.2.1 Introduction of Neural Networks

A neural network is a machine learning model inspired by the structure and function of the human brain. Neural networks are made up of interconnected nodes called neurons that are arranged in layers. Weights, which affect the strength of the connection between neurons, connect the neurons in a neural network.

The artificial neuron is the basic building element of a neural network; it accepts one or more inputs, conducts a weighted sum of the inputs, applies an activation function to the result, and provides an output. The activation function is typically a nonlinear function, such as the sigmoid or ReLU function, that allows the network to learn complicated patterns in data.

The training phase often incorporates an optimization technique, such as stochastic gradient descent, which iteratively modifies the network weights to minimize a loss function that evaluates the difference between the expected and true output. The loss function is regularly selected based on the task at hand and the type of data being used.

2.2.2 Architecture of Neural Networks

The architecture of a neural network refers to the structure and organization of the network's neurons and layers. Most neural networks can be described in terms of their basic components, which include input layer, hidden layers, and output layer. As shown in figure 2.3

1. **Input layer**: The input layer is the first layer of the neural network, which receives the input data and passes it to the next layer. The number of neurons in the input layer is determined by the dimensionality of the input data.
2. **Hidden layers**: The hidden layers are one or more layers of neurons between the input and output layers. Each neuron in a hidden layer takes the weighted sum of the outputs of the neurons in the previous layer, applies an activation function, and produces an output that is passed to the next layer.
3. **Output layer**: The output layer is the final layer of the neural network, which produces the network's output. The number of neurons in the output

layer is determined by the number of output variables in the problem being solved. For example, in a binary classification issue, the output layer would typically have a single neuron that produces a binary output (0 or 1).

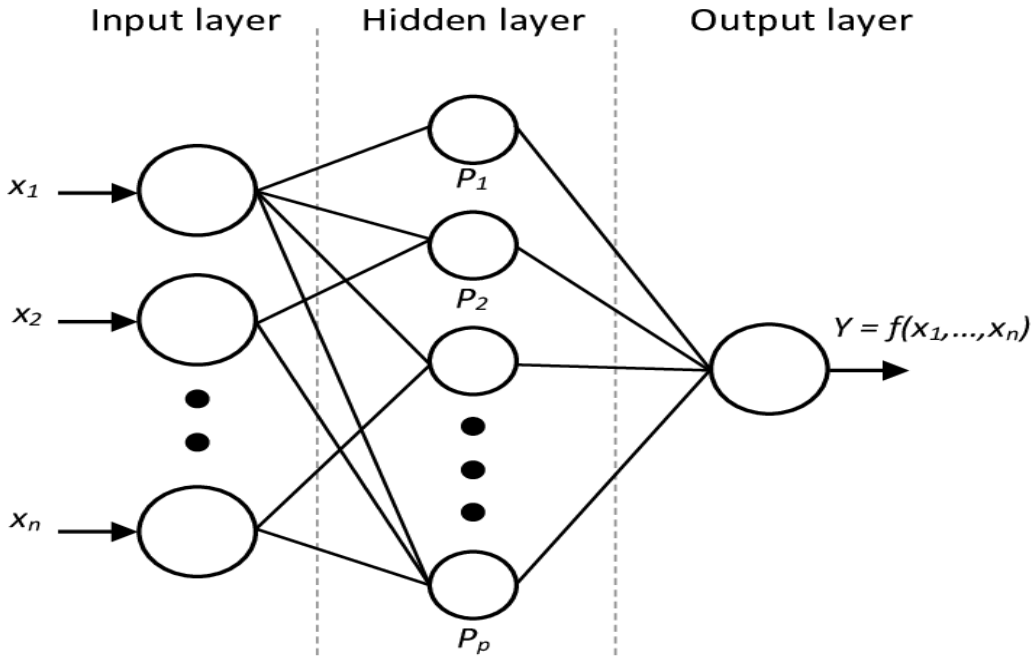


Figure 2.3 Architecture of Neural Networks

2.2.3 Types of Neural Networks

There are several types of neural networks, each with its own unique characteristics and applications. Here are a brief about some of the most commonly used types of neural networks:

1. Feedforward Neural Networks:
   Feedforward neural networks are the simplest type of neural network. They consist of an input layer, one or more hidden layers, and an output layer. The information flows in one direction, from the input layer to the output layer, without any feedback loops. They are commonly used for pattern recognition and classification tasks, such as image classification and speech recognition. As shown in figure 2.4

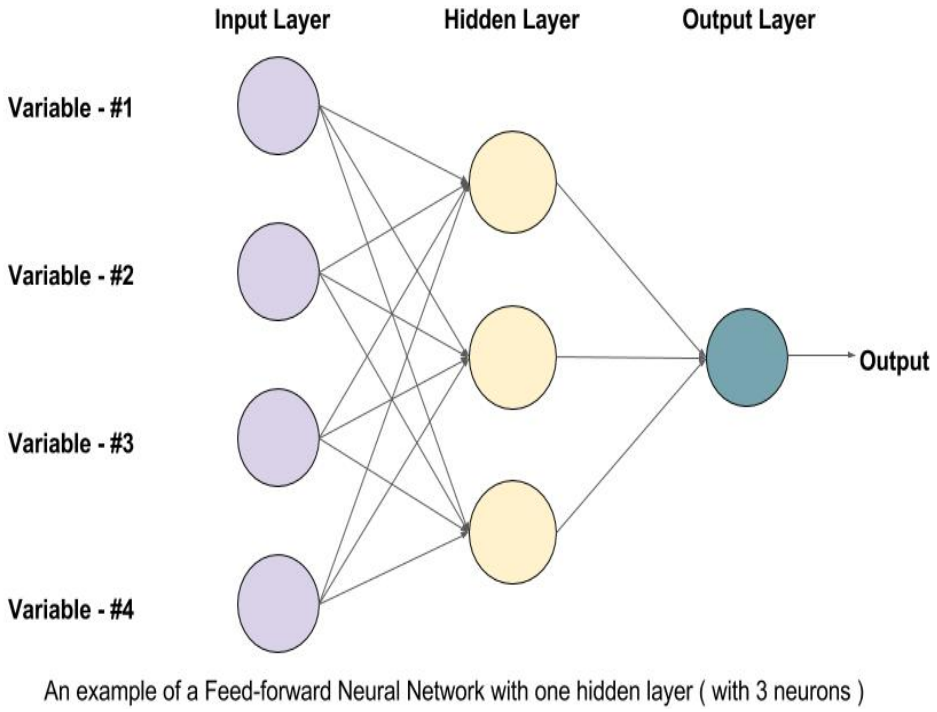


Figure 2.4 Feed Forward Neural Network

2. Convolutional Neural Networks (CNNs):
   As shown in figure 2.5 Convolutional neural networks are a specialized type of neural network that are designed for processing images and other structured data. They use a set of learnable filters to extract features from the input data, and are commonly used in image and video recognition tasks. That is a small brief about CNN, but we will discuss it in more details in 2.2.4 section.

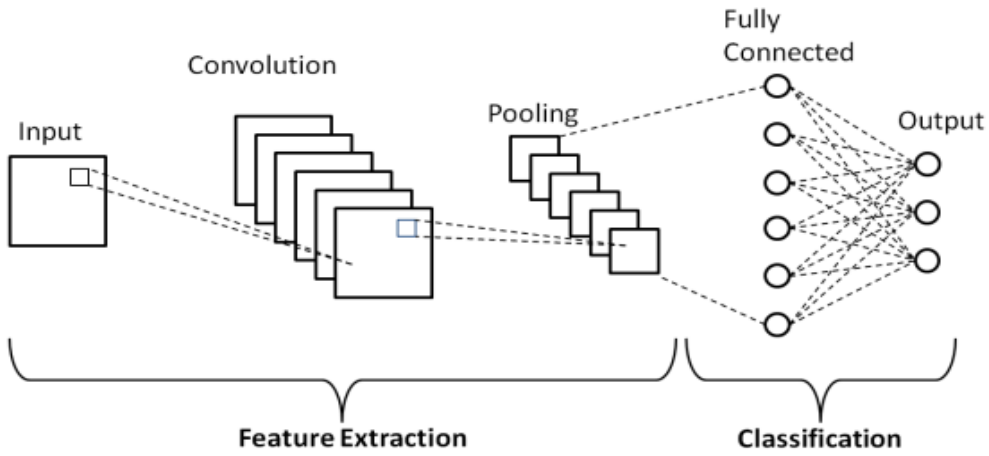


Figure 2.5 CNN Neural Network

3. Recurrent Neural Networks (RNNs):
   As shown in figure 2.6 Recurrent neural networks are designed for processing sequential data, such as time series or natural language data. They maintain an internal state that allows them to process the input data in a time-dependent manner, and are commonly used in speech recognition and language translation tasks.

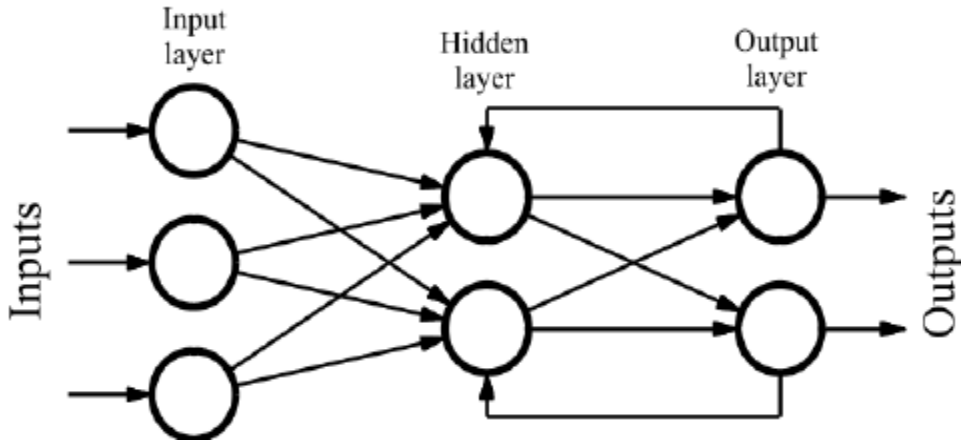


Figure 2.6 RNN Neural Network

4. Long Short-Term Memory (LSTM) Networks:
   As shown in figure 2.7 LSTM networks are a type of recurrent neural network that are specifically designed for processing sequences of data that have long-term dependencies. They are commonly used in speech recognition, language translation, and other natural language processing tasks.

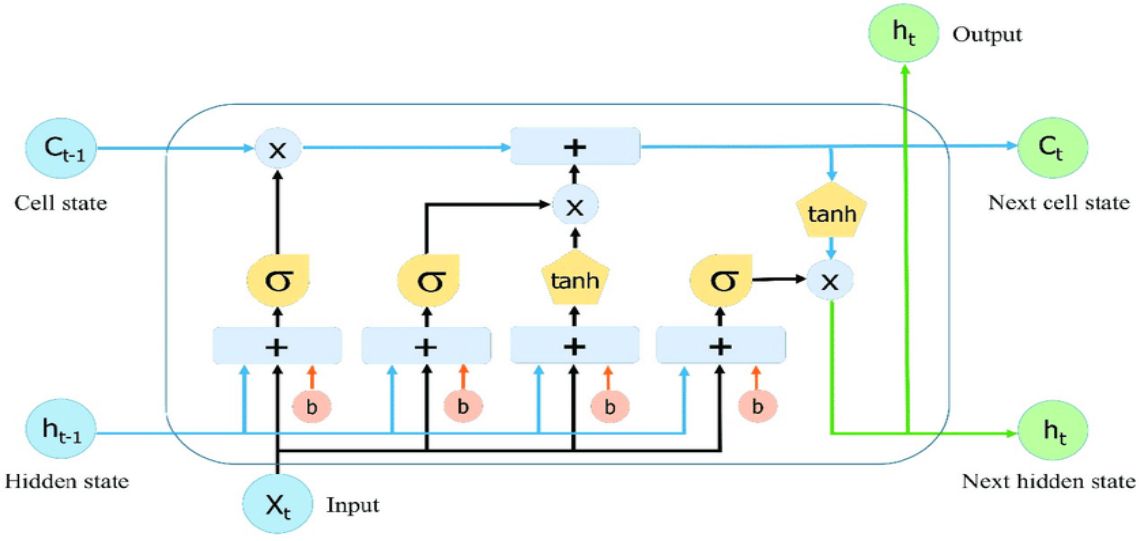


Figure 2.7 LSTM Network

5. Autoencoders:
   As shown in figure 2.8 Autoencoders are a type of neural network that are used for unsupervised learning, where the goal is to learn a compressed representation of the input data. They consist of an encoder network that maps the input data to a latent representation, and a decoder network that reconstructs the input data from the latent representation. They are commonly used in image and audio compression, and anomaly detection. That is a small brief about CNN, but we will discuss it in more details in 2.2.6 section.

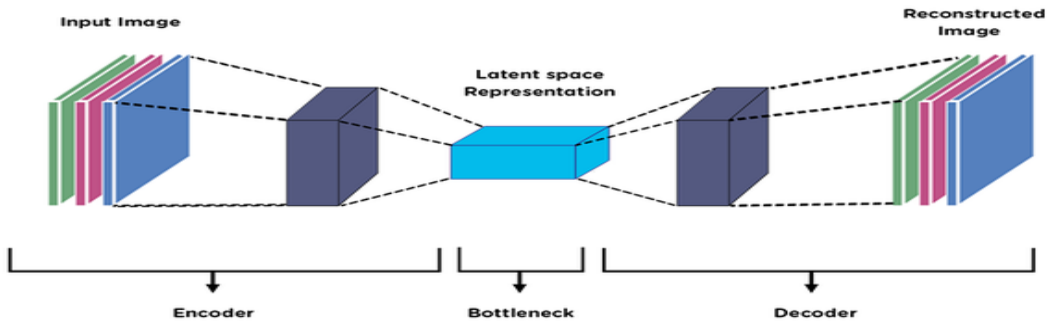


Figure 2.8 autoencoders Network

6. Generative Adversarial Networks (GANs):
   As shown in figure 2.9 GANs are a type of neural network that are used for generative modeling, where the goal is to learn a probability distribution that can generate new samples that are similar to the training data. They consist of a generator network that produces new samples, and a discriminator network that tries to distinguish between the real and generated samples. They are commonly used in image and video generation tasks. That is a small brief about CNN, but we will discuss it in more details in 2.2.7 section.

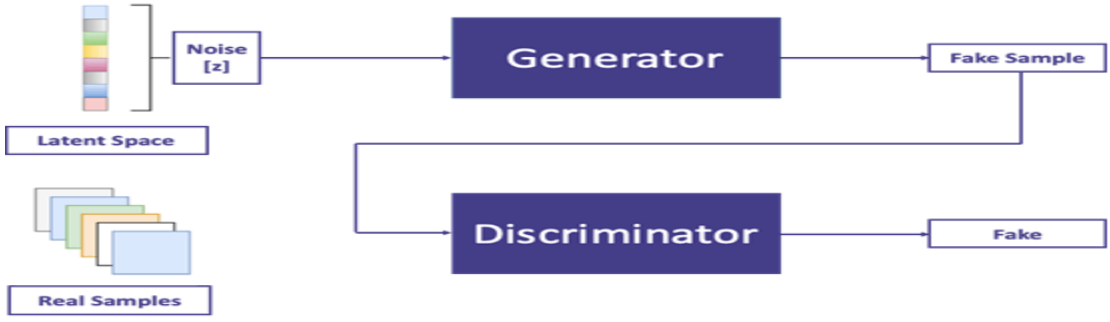


Figure 2.9 GANs Network

2.3 Backpropagation

Backpropagation [7] is a technique that is used to train artificial neural networks by iteratively changing the weights of neural connections. It is a type of supervised learning in which the network is given a set of input-output pairings known as a training set, with the goal of adjusting the network's weights so that it can reliably predict the output for fresh input data that it has not seen before.

The backpropagation algorithm operates by first performing a forward pass through the network, during which the input data is sent forward through the layers of neurons and the output is generated. The discrepancy between the expected and true outputs is then determined for each training example, and this error is utilized to alter the weights of the network's connections. By propagating the error backwards through the network, starting at the output layer and working backwards towards the input layer, the weights are adjusted. At each layer, the error increases by the activation function's derivative, which indicates the output's sensitivity to changes in the input. This derivative is used to update the weights of the previous layer's connections between neurons, reducing inaccuracy in the next iteration. As shown in figure 2.10

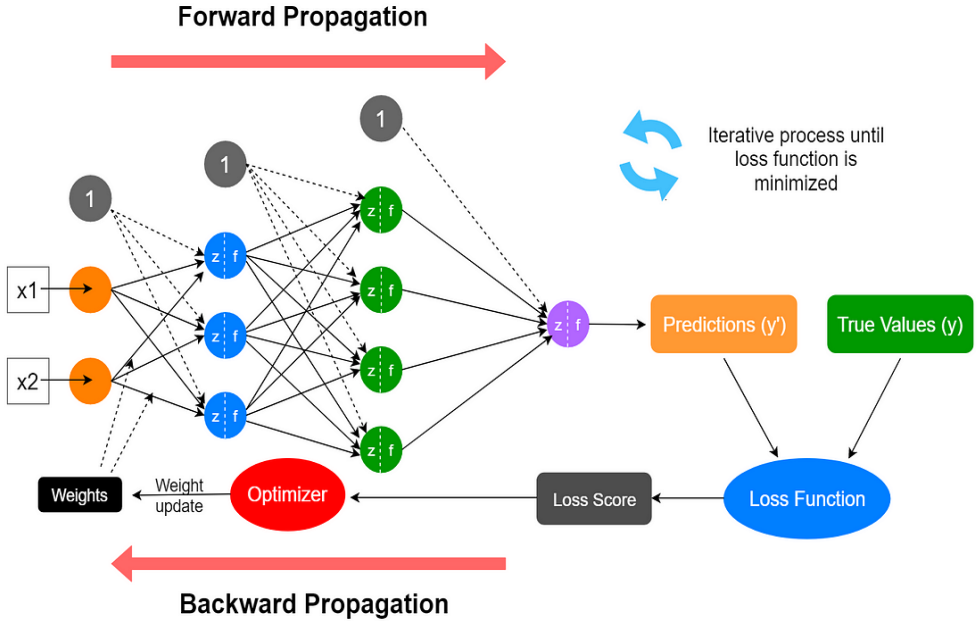


Figure 2.10 Backpropagation

Backpropagation is often done for several iterations, or epochs, until the network converges on a set of weights that produces accurate predictions on the training set. The network is frequently evaluated on a different validation set during training to monitor its performance and minimize overfitting.
Backpropagation is a powerful and extensively used technique for training neural networks that has resulted in numerous artificial intelligence advancements. However, good results might be computationally expensive and require a huge amount of training data.

2.4 Convolution Neural Networks (CNNs)

2.4.1 Convolution Layer

A Convolutional Layer, also known as Conv Layer, is a crucial building block of Convolutional Neural Networks (CNNs) used in computer vision and image processing applications. It performs convolution, a mathematical operation that extracts features from the input image. The Conv Layer consists of a set of learnable filters, also called kernels or weights, which slide over the input image like a window, and each filter is applied to a particular region of the image, producing a feature map that captures the presence of that filter's particular feature in the input image. The Conv Layer is often followed by additional layers such as pooling, batch normalization, or fully connected layers to further process the output of the Conv Layer and improve the performance of the network.

2.4.2 Strides

Stride is a term used in convolutional neural networks to refer to the number of pixels that a filter or kernel is moved horizontally or vertically across the input image during convolution. A larger stride can help to reduce the size of the output feature map and make the network more computationally efficient, but it can also result in information loss because the filter may not be able to capture all the details of the input image. Figure 2.11 likely illustrates an example of how stride affects the output feature map size and information loss.

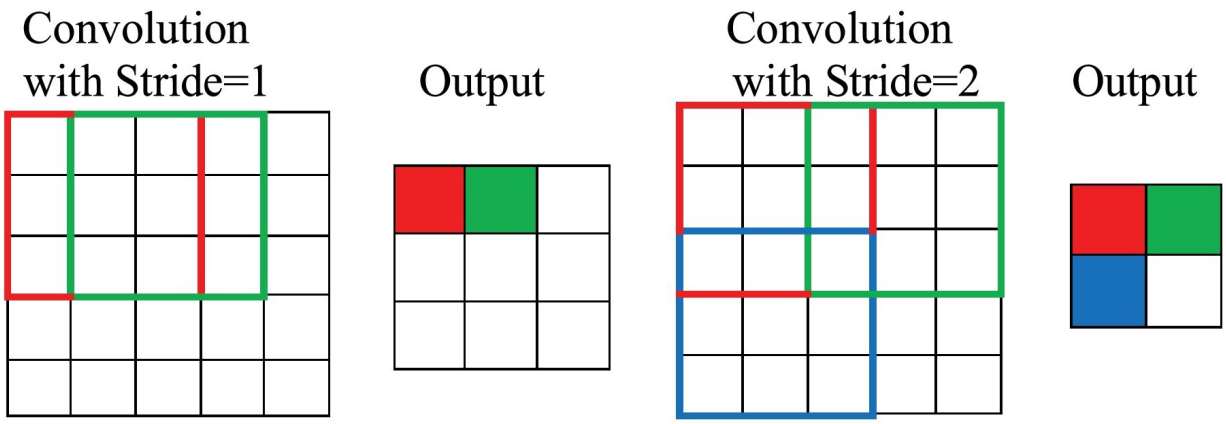


Figure 2.11 Strides

2.4.3 Padding

As shown in figure 2.12 padding is a technique used in convolutional neural networks to add extra border pixels around the input image before applying convolution. The goal of padding is to preserve the spatial dimensions of the input image after convolution and prevent the output feature map from being too small, which can result in the loss of important spatial information.

Figure 2.12 Padding

2.4.4 Pooling Layer

A Pooling Layer is a type of layer in a convolutional neural network that replaces a local neighborhood of pixels with a summary statistic, such as the maximum or average value. It is often used to reduce the spatial dimensions of the feature maps while retaining the most important information, and it can help to prevent overfitting and improve generalization performance. Common types of pooling include max pooling, average pooling, and global pooling. As shown in figure 2.13

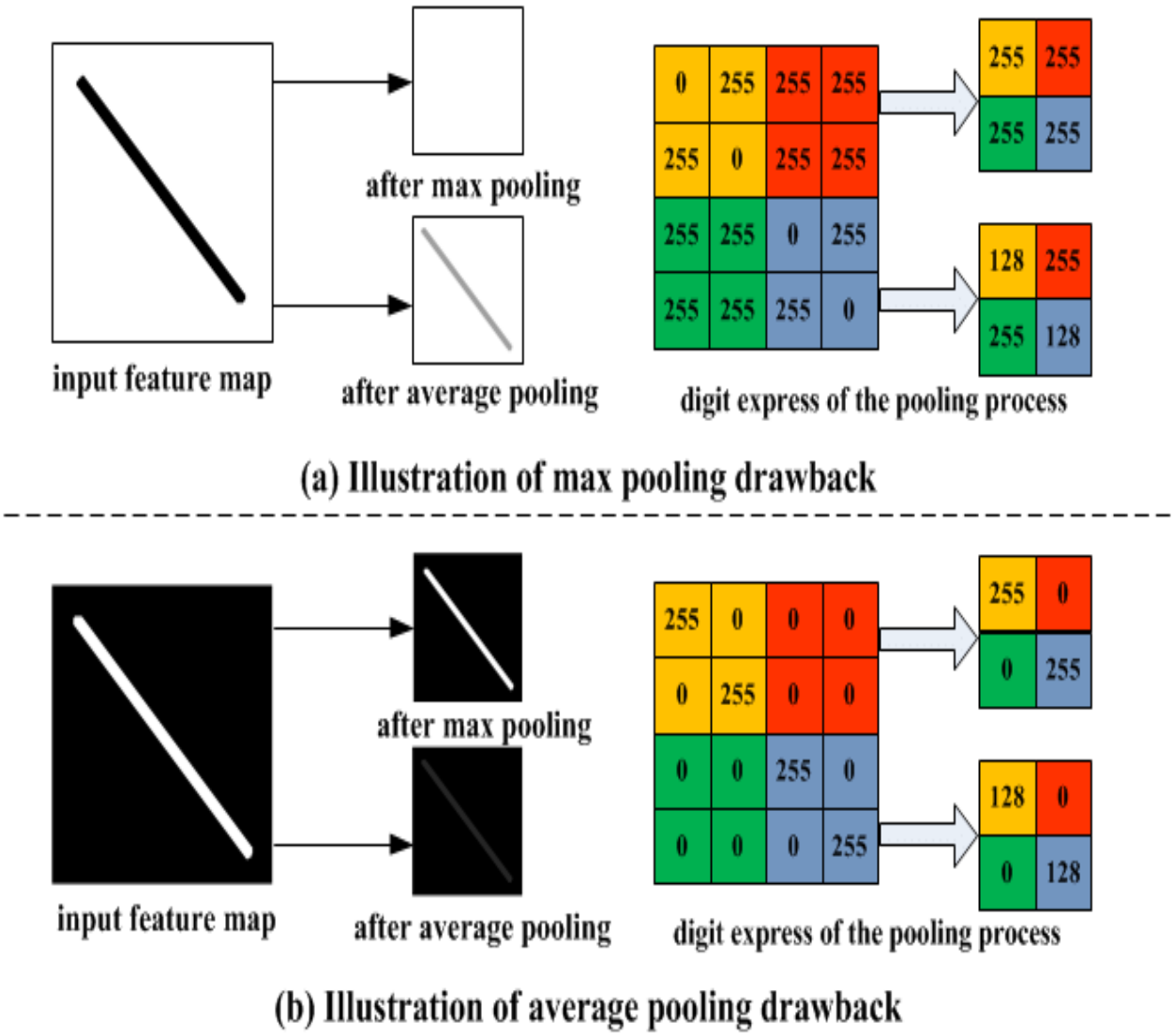


Figure 2.13 Pooling Layer

2.4.5 Activation Functions

Activation functions are mathematical functions that are applied to the output of a neural network's layer to introduce nonlinearity into the model. Activation functions are crucial to neural networks because they allow the network to learn and model complex, nonlinear relationships between inputs and outputs.

2.4.5.1 Tanh (Hyperbolic Tangent)

The tanh function maps input values to a range between -1 and 1, providing stronger nonlinearity than the sigmoid function. As shown in figure 2.14

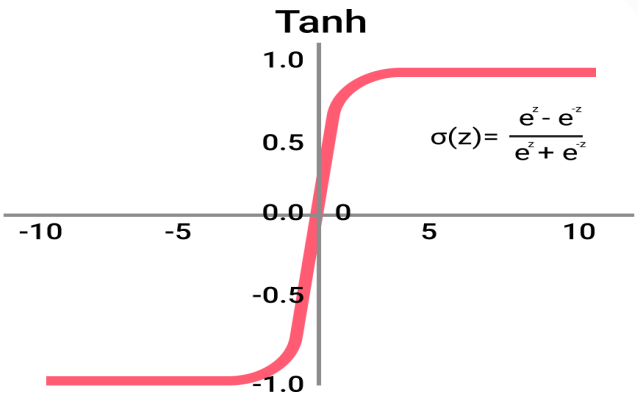


Figure 2.14 Tanh Activation Function

2.4.5.2 ReLU (Rectified Linear Unit)

The ReLU function sets any negative input value to zero and leaves positive input values unchanged. It is one of the most commonly used activation functions in neural networks. As shown in figure 2.15

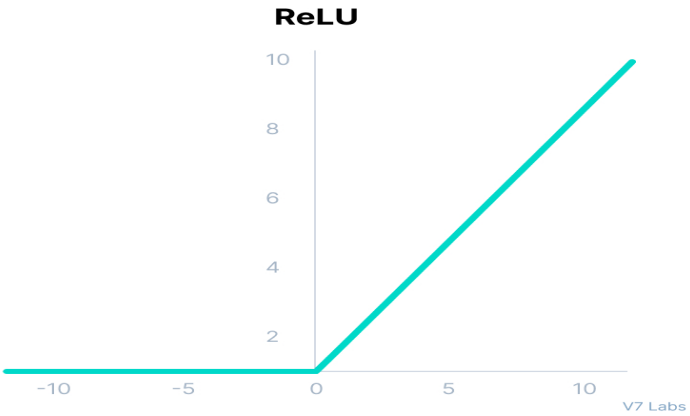


Figure 2.15 ReLU Activation Function

2.4.5.3 Leaky ReLU

Leaky ReLU is a variation of ReLU that allows a small slope for negative input values, which can help to prevent "dead" neurons. As shown in figure 2.16

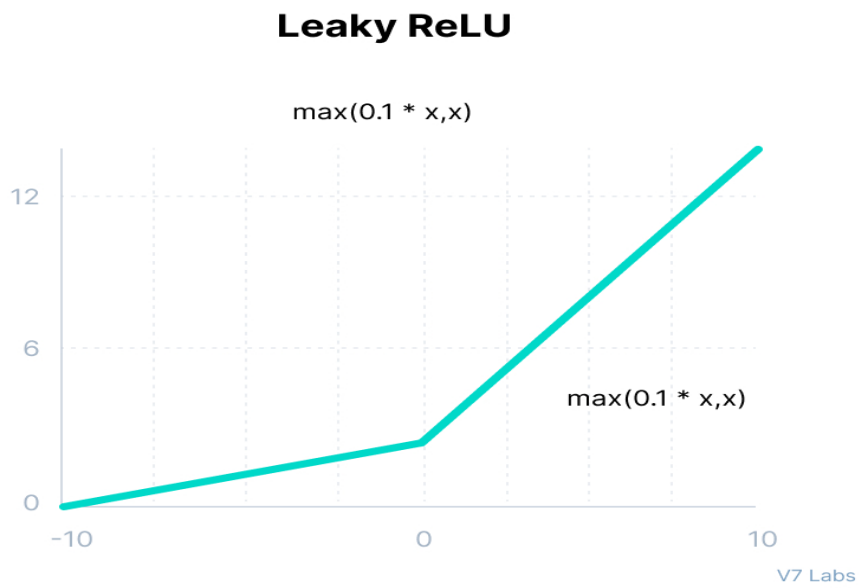


Figure 2.16 Leaky ReLU Activation Function

2.4.5.4 Sigmoid

The sigmoid function maps any input value to a probability between 0 and 1. It is often used in binary classification tasks.

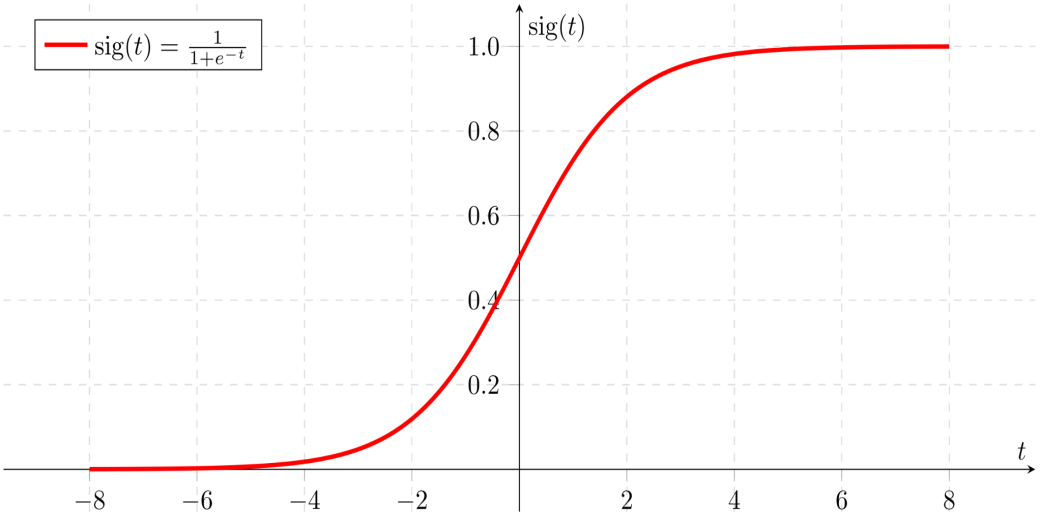


Figure 2.17 Sigmoid Activation Function

2.4.6 Fully Connected Layer

A Fully Connected Layer is a type of layer in a neural network where all the neurons in one layer are connected to all the neurons in the next layer. It involves passing the input through a Linear function, followed by an activation function, and is often used in the final layers of a neural network to make predictions. The number of neurons is a hyperparameter that can be tuned during the model design process. As shown in figure 2.19

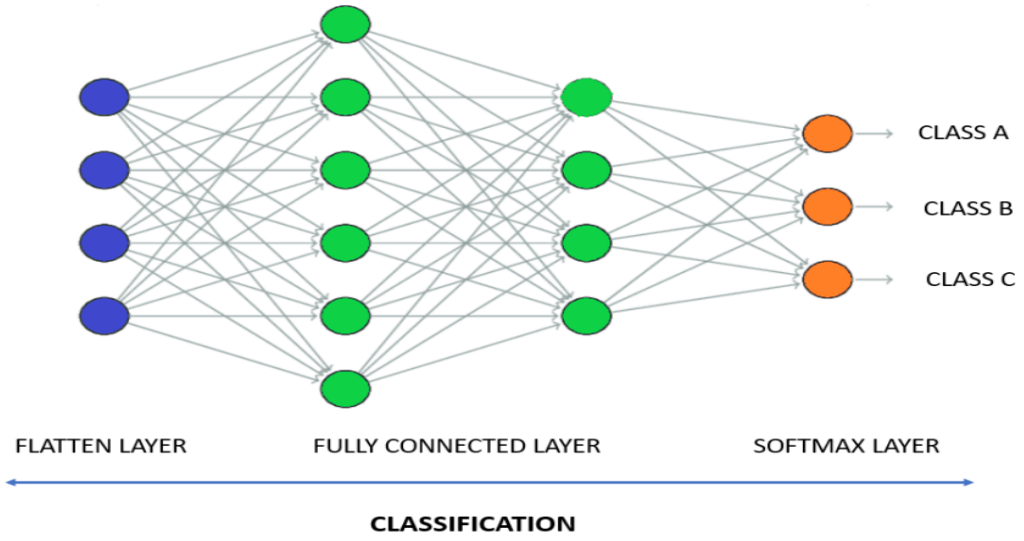


Figure 2.19 Fully Connected Layer

2.4.7 Dropout Layer

A Dropout Layer is a type of layer in a neural network that is used to prevent overfitting by randomly dropping out some neurons during training, as shown in figure 2.20. Dropout encourages each neuron to learn more robust features independently and helps to prevent the network from memorizing noise in the training data. During inference, the Dropout Layer is turned off, and all the neurons are used to make predictions.

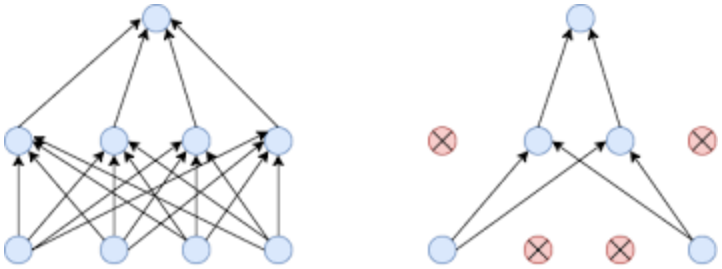


Figure 2.20 Dropout Layer

2.5 Optimizers and Loss Functions

Optimizers and Loss Functions are two important components of a neural network that are used to train the model and optimize its parameters.
Optimizers are algorithms that are used to update the weights and biases of the neural network during training in order to minimize the loss function.
The loss function measures how well the neural network is performing on a specific task, such as classification or regression.

2.5.1 Batch Gradient Descent

As shown in figure 2.21 This is the most basic form of gradient descent, which updates the weights and biases of the neural network after processing the entire training dataset. This can be slow and memory-intensive for large datasets.

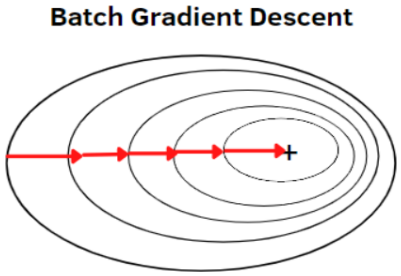


Figure 2.21 Batch Gradient Descent

2.5.2 Mini-Batch Gradient Descent

As shown in figure 2.23 This is a compromise between batch gradient descent and stochastic gradient descent, which updates the weights and biases of the neural network after processing a small batch of training samples. This can be more efficient and less noisy than stochastic gradient descent, while still being faster than batch gradient descent.

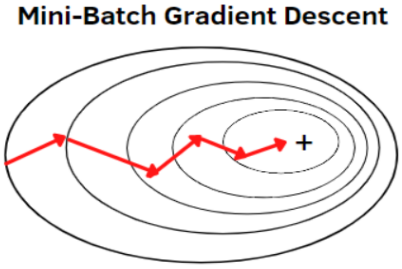


Figure 2.23 Mini-Batch Gradient Descent

2.5.3 Momentum

momentum is a technique used in optimization algorithms, such as stochastic gradient descent, to speed up convergence and help the optimization algorithm escape from local minima. It involves accumulating a moving average of the gradients over time and using this moving average to update the model weights. The momentum term can help to move more quickly in the direction of the minimum and to escape from shallow local minima.

2.5.4 Binary Cross Entropy

This is a loss function commonly used for binary classification tasks that measure the difference between the predicted probability and the true label. As shown in figure 2.24

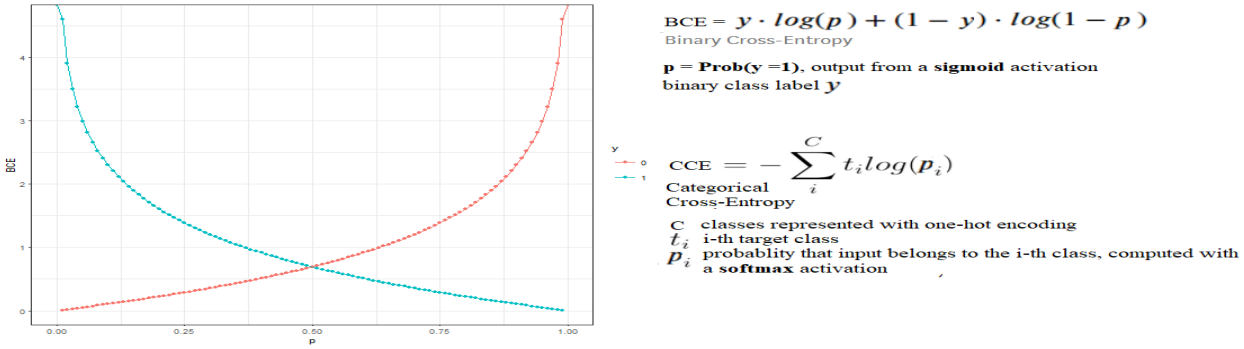


Figure 2.24 Binary Cross Entropy graphs and equations

2.5.5 Negative Log-Likelihood Loss (nll_loss)

nll_loss is the Negative Log Likelihood Loss, which is a commonly used loss function in classification tasks where the output of the model is a probability distribution over the classes. It measures the difference between the predicted probability distribution and the true label distribution and is often used in combination with an optimizer to train the neural network.

2.5.6 Root Mean Square Propagation (RMSprop) Optimizer

RMSprop is an optimization algorithm commonly used in neural networks for stochastic gradient descent. It is an adaptive learning rate method that adjusts the learning rate for each weight in the network based on the historical gradient information.

The RMSprop optimizer can handle non-stationary objectives and noisy gradients, and can converge faster on ill-conditioned problems.

2.5.7 Adaptive Moment Estimation (ADAM) Optimizer

ADAM is an optimization algorithm commonly used in neural networks for stochastic gradient descent. It uses both the first and second moments of the gradients to update the parameters and incorporates bias correction to estimate the moments more accurately.
The ADAM optimizer is computationally efficient and can handle sparse gradients and noisy data, and is less sensitive to the choice of hyperparameters than other optimization methods.

2.6 Generative Adversarial Networks (GANs)

GANs (Generative Adversarial Networks) are a type of generative model that consists of two neural networks: a **generator network** and a **discriminator network**. The **generator network** learns to generate new data that is similar to the training data, while the **discriminator network** learns to distinguish between the generated data and the real data. As shown in figure 2.25

The basic idea behind GANs is to train the **generator network** to generate data that can fool the **discriminator network** into thinking that it is real data.
The **generator network** takes random noise as input and generates a sample of data, which is then fed to the **discriminator network** along with real data. The **discriminator network** then tries to distinguish between the real data and the generated data.

During training, the **generator network** is updated to generate data that is more similar to the real data, while the **discriminator network** is updated to better distinguish between the real data and the generated data. This process continues until the **generator network** is able to generate data that is indistinguishable from the real data, and the **discriminator network** is no longer able to distinguish between the real data and the generated data.

The GAN loss function is based on a minimax game between the **generator** and the **discriminator** networks. The **generator network** tries to minimize the probability that the **discriminator network** correctly classifies the generated data as fake, while the **discriminator network** tries to maximize this probability.

The GAN loss function can be written as:

$$L = -log(D(x)) - log(1 - D(G(z)))$$

Where x is a sample from the real data, z is a random noise vector, G(z) is the generated data, D(x) is the probability that the **discriminator network** correctly classifies x as real, and D(G(z)) is the probability that the **discriminator network** correctly classifies G(z) as fake.

GANs can generate realistic images, text, and other types of data. They have been used in applications such as image synthesis, image-to-image translation, and video prediction. However, GANs can be difficult to train and may suffer from mode collapse, where the **generator network** generates limited types of data that do not cover the entire distribution of the real data.

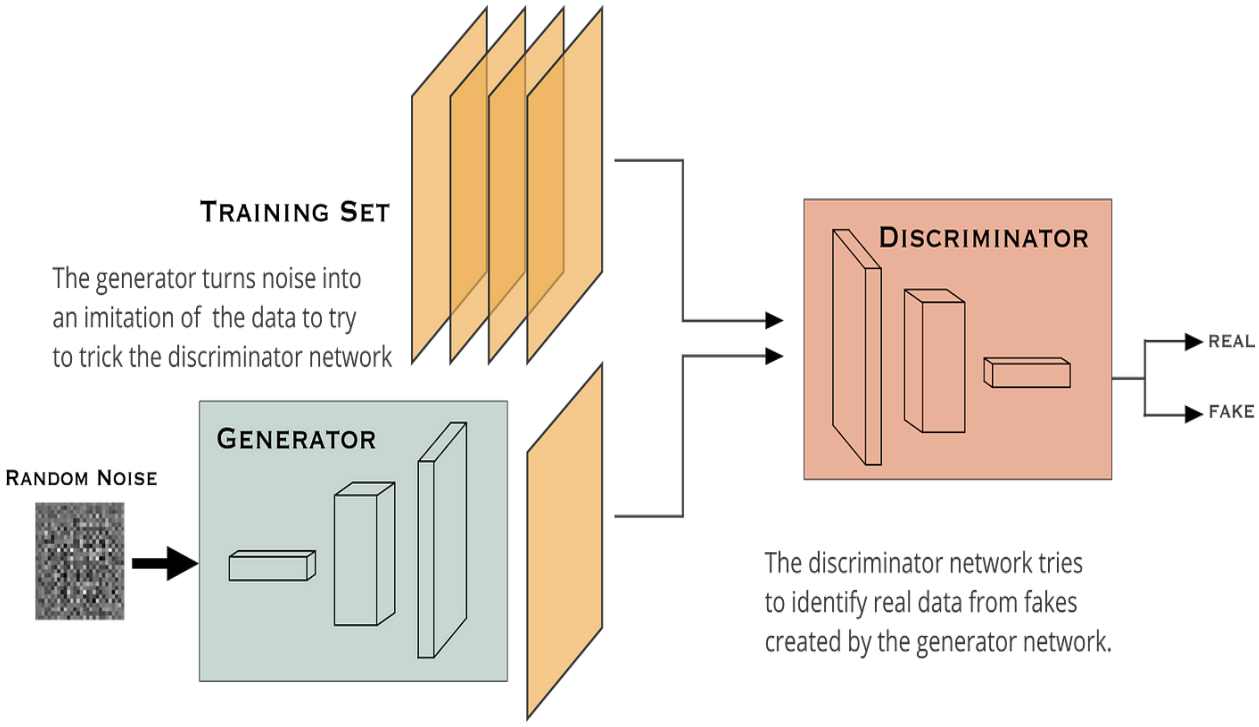


Figure 2.25 GANs Architecture

2.7 Variational Autoencoders (VAE)

VAE stands for Variational Autoencoder, which is a type of generative model that learns to encode and decode data using a neural network. VAEs are a type of autoencoder that are trained to generate data that is similar to the input data, rather than simply reconstructing the input data.

The basic idea behind VAEs is to learn a low-dimensional representation of the input data, called the latent space, that captures the meaningful variations in the data. The VAE consists of an encoder network that maps the input data to the latent space, and a **decoder network** that maps the latent space back to the input data.

During training, the VAE is trained to minimize the reconstruction error, which measures how well the generated data matches the input data. In addition, the VAE is trained to regularize the latent space by encouraging it to follow a specific distribution, typically a Gaussian distribution. This is done by adding a KL divergence term to the loss function, which measures the difference between the learned distribution and the target distribution.

The VAE loss function can be written as:

$$L = reconstructionLoss + KL_{divergence}$$

Where the reconstruction loss measures the difference between the input data and the generated data, and the KL divergence measures the difference between the learned distribution and the target distribution.

The VAE can be used for various applications, such as image generation, text generation, and anomaly detection. In image generation, the VAE can be trained on a dataset of images and used to generate new images that are similar to the training data. In text generation, the VAE can be trained on a dataset of text and used to generate new sentences or paragraphs that are similar to the training data. In anomaly detection, the VAE can be trained on a dataset of normal data and used to detect anomalies in new data that do not fit the learned distribution.

2.8 Previous Studies and Work

2.8.1 VAE Based Study

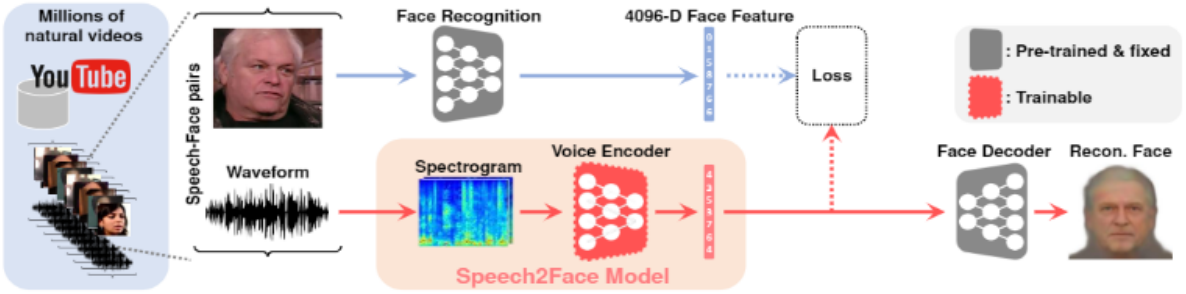
Oh, Tae-Hyun, et al[8] introduce a method to reconstruct facial images of a person from a short audio recording of their speech. The authors trained a deep neural network using millions of natural Internet/YouTube videos to learn voice-face correlations, resulting in a model that can produce images capturing various physical attributes of the speakers. The training is self-supervised, using the natural co-occurrence of faces and speech in Internet videos. The paper evaluates the model by conducting experiments on two datasets, quantifying the accuracy and effectiveness of the reconstructed faces. The authors suggest that this approach can open up new research opportunities and applications, such as assigning faces to machine-generated voices, and validate the existence of cross-modal biometric information between speech and appearance.

2.8.1.1 Dataset

The paper mentions that the model was tested on two datasets: the AVSpeech[9] dataset and the VoxCeleb[10,11,12] dataset. The AVSpeech dataset is used for qualitative evaluation, while the VoxCeleb dataset is used for quantitative evaluation.

2.8.1.2 Model Architecture

The paper describes the model architecture as consisting of two main components: a voice encoder and a face decoder. The voice encoder takes a complex spectrogram of speech as input and predicts a low-dimensional face feature that corresponds to the associated face. The face decoder, on the other hand, takes the face feature as input and generates an image of the face in a canonical form (frontal-facing and with a neutral expression). The voice encoder is a model designed and trained by the authors, while the face decoder is based on a previously proposed model. The specific details of the model architecture are not provided in the given information. As shown in figure 2.26



Layer	Input	CONV RELU BN	CONV RELU BN	CONV RELU BN	MAXPOOL	CONV RELU BN	MAXPOOL	CONV RELU BN	MAXPOOL	CONV RELU BN	MAXPOOL	CONV RELU BN	CONV RELU BN	CONV	AVGPOOL RELU BN	FC RELU	FC
Channels	2	64	64	128	–	128	–	128	–	256	–	512	512	512	–	4096	4096
Stride	–	1	1	1	2 × 1	1	2 × 1	1	2 × 1	1	2 × 1	1	2	2	1	1	1
Kernel size	–	4 × 4	4 × 4	4 × 4	2 × 1	4 × 4	2 × 1	4 × 4	2 × 1	4 × 4	2 × 1	4 × 4	4 × 4	4 × 4	∞ × 1	1 × 1	1 × 1

Figure 2.26 speech2face paper architecture

2.8.2 GANS Based Study

Wen, Yandong, Bhiksha Raj, and Rita Singh[13] introduces a novel task of generating faces from voice for voice profiling. The authors propose a framework based on generative adversarial networks (GANs) to accomplish this task and evaluate the results both qualitatively and quantitatively. The qualitative analysis shows that the framework can map the voice manifold to the face manifold, with many identity associations between the generated faces and input voices. The authors also recommend several quantitative evaluation metrics to assess the model's performance in terms of mapping voices to faces, matching high-level attributes, and matching the identity of the speaker. To test the model, the authors use a voice-to-face matching task and report the matching accuracies on both training and test sets. The paper concludes by acknowledging the potential of the proposed framework, while recognizing the need for further improvements to address issues with the GANs-based output and improve generalizability. Overall, the paper provides insights into the relationship between voice and face modalities and demonstrates the promise of generating faces from voice.

2.8.2.1 Dataset

The paper mentions that the model was trained and tested on Voxceleb[10,11,12] Dataset.

2.8.2.2 Model Architecture

The paper presents a model architecture based on Generative Adversarial Networks (GANs) that consists of two main components: a generator and a discriminator. The generator takes an input vector, which represents the voice manifold, and generates a corresponding face image. The discriminator, on the other hand, takes in both real and generated face images and determines whether they are real or fake. The generator and discriminator are trained iteratively in an adversarial manner until the generator can produce highly realistic face images that can fool the discriminator.

The generator typically consists of multiple layers of 2D deconvolutional layers with Rectified Linear Unit (ReLU) activation functions. The input to the generator is usually a random noise vector or a latent vector that is transformed into a face image. The discriminator, on the other hand, consists of multiple layers of 2D convolutional layers with Leaky ReLU (LReLU) activation functions. The input to the discriminator is a face image, and its output is a probability score indicating whether the image is real or fake. As shown in figure 2.27

Voice Embedding Network			Generator		
Layer	Act.	Output shape	Layer	Act.	Output shape
Input	-	$64 \times t_0$	Input	-	$64 \times 1 \times 1$
Conv $3_{/2,1}$	BN + ReLU	$256 \times t_1$	Deconv $4 \times 4_{/1,0}$	ReLU	$1024 \times 4 \times 4$
Conv $3_{/2,1}$	BN + ReLU	$384 \times t_2$	Deconv $3 \times 3_{/2,1}$	ReLU	$512 \times 8 \times 8$
Conv $3_{/2,1}$	BN + ReLU	$576 \times t_3$	Deconv $3 \times 3_{/2,1}$	ReLU	$256 \times 16 \times 16$
Conv $3_{/2,1}$	BN + ReLU	$864 \times t_4$	Deconv $3 \times 3_{/2,1}$	ReLU	$128 \times 32 \times 32$
Conv $3_{/2,1}$	BN + ReLU	$64 \times t_5$	Deconv $3 \times 3_{/2,1}$	ReLU	$64 \times 64 \times 64$
AvePool $1 \times t_5$	-	64×1	Deconv $1 \times 1_{/1,0}$	-	$3 \times 64 \times 64$
Discriminator			Classifier		
Layer	Act.	Output shape	Layer	Act.	Output shape
Input	-	$3 \times 64 \times 64$	Input	-	$3 \times 64 \times 64$
Conv $1 \times 1_{/1,0}$	LReLU	$32 \times 64 \times 64$	Conv $1 \times 1_{/1,0}$	LReLU	$32 \times 64 \times 64$
Conv $3 \times 3_{/2,1}$	LReLU	$64 \times 32 \times 32$	Conv $3 \times 3_{/2,1}$	LReLU	$64 \times 32 \times 32$
Conv $3 \times 3_{/2,1}$	LReLU	$128 \times 16 \times 16$	Conv $3 \times 3_{/2,1}$	LReLU	$128 \times 16 \times 16$
Conv $3 \times 3_{/2,1}$	LReLU	$256 \times 8 \times 8$	Conv $3 \times 3_{/2,1}$	LReLU	$256 \times 8 \times 8$
Conv $3 \times 3_{/2,1}$	LReLU	$512 \times 4 \times 4$	Conv $3 \times 3_{/2,1}$	LReLU	$512 \times 4 \times 4$
Conv $4 \times 4_{/1,0}$	LReLU	$64 \times 1 \times 1$	Conv $4 \times 4_{/1,0}$	LReLU	$64 \times 1 \times 1$
FC 64×1	Sigmoid	1	FC $64 \times k$	Softmax	k

Figure 2.27 GANs paper architecture

Chapter 3: Analysis & Design

3.1 System Overview

We introduce the entire architecture of our system (*see figure 3.1*). We have two approaches, the first one is Generative Adversarial Networks (GANs) and the other is Variational Autoencoder (VAE).

3.1.1 System Architecture

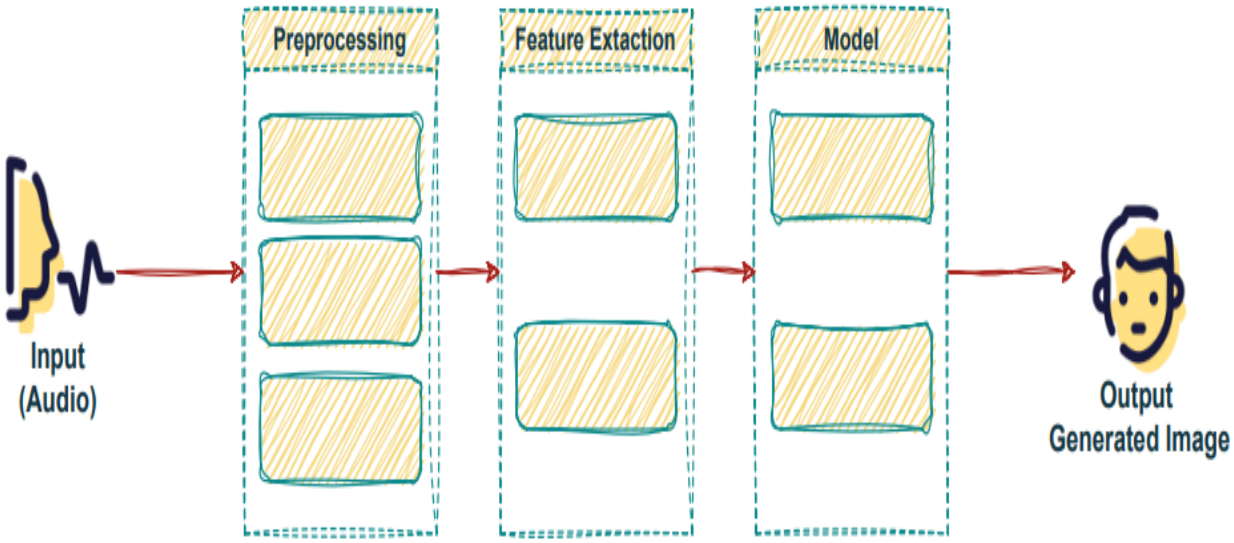


Figure 3.1 System Architecture

First, The input is audio, it gets passed to a preprocessing step to normalize volume, remove noise, and Filter out silence. In the feature extraction step, we extract the log Mel spectrogram. Then feed the model with the output of the previous step. In the end, the output is a generated image for the person who provided the audio input.

3.1.2 System Users

I. Intended Users:
The Reconstructing Face from Voice system is built for end-users who wish to generate a face image from the audio input, such as for police officers, use in entertainment, artistic endeavors, or other creative projects.

II. User Characteristics:
The end-user of the system should have the following minimum requirements:

1. Basic computer skills: The end-user should have basic knowledge of how to operate a computer, such as navigating through the operating system, opening and closing applications, and using input devices like a mouse and keyboard.

2. Knowledge of audio recording: The end-user should have some experience with audio recording, such as understanding how to use a microphone or other recording equipment.

3. Understanding of limitations and potential biases.

4. Minimum application requirements: The end-user should have a device that meets the minimum requirements for installing the application. This could include a specific operating system, processor speed, memory, and storage requirements.

5. No specific hardware requirements: The system does not require any specific hardware to be used, meaning that the end-user can use any device that meets the minimum application requirements.

3.2 Description of method and procedure used

3.2.1 Use Case Diagram

The use case diagram for the Reconstructing Face from Voice system includes a User Actor that is responsible for allowing the user to input audio into the system through two use cases: Record Audio and Upload Audio. These use cases provide the user with two methods for inputting audio into the system, depending on their preferences and available resources.

Once the user has provided the necessary audio input, the system takes the audio and applies a series of processing steps like Normalize volume, remove noise, and Filter out silence. After pre-processing the audio, the system performs feature extraction and generates a fake image as shown in figure 3.2.

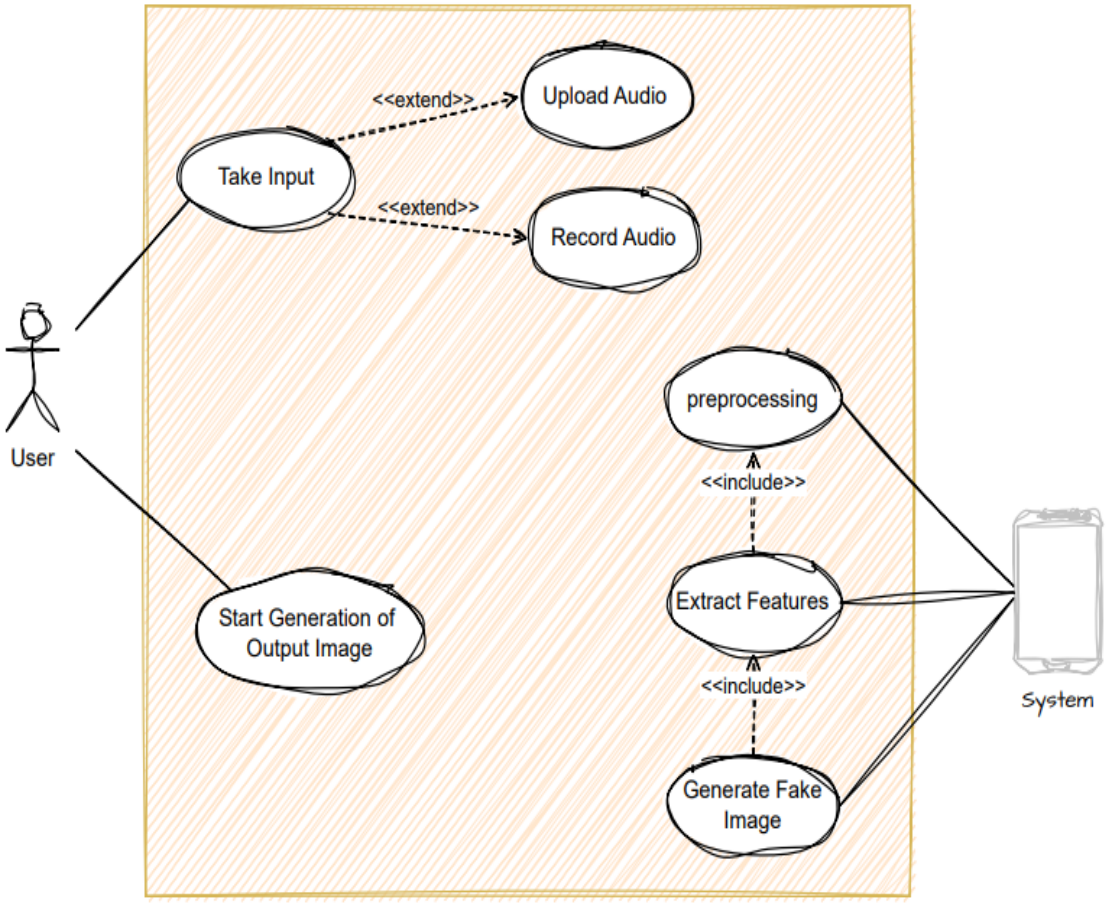


Figure 3.2 System's Use Case Diagram

3.2.2 Flow Of Events

- The user initiates the process by either recording audio or uploading it to the system.
- The system performs preprocessing on the audio, including normalizing the volume, removing noise, and filtering out silence.
- Once the audio has been preprocessed, the system extracts features from it.
- The system generates a fake image based on the extracted features.
- Finally, the system displays the reconstructed face to the user.

3.2.3 Sequence Diagram

In the sequence diagram, as shown in figure 3.3, the User actor initiates the process by either recording audio or uploading it to the system. The system then performs preprocessing on the audio, including normalizing the volume, removing noise, and filtering out silence. Once the audio has been preprocessed, the system extracts features from it and generates a fake image based on those features. Finally, the system displays the reconstructed face to the user.

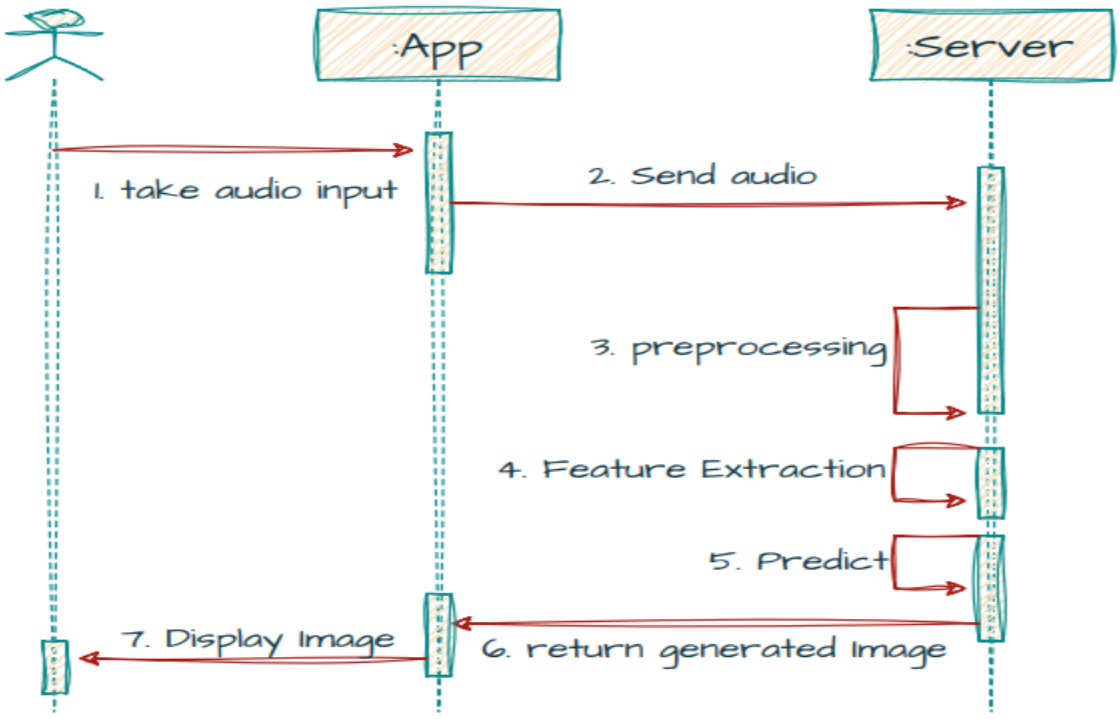


Figure 3.3 System's Sequence Diagram

Chapter 4: VAE Model

4.1 Environment Setup & Tools

Python programming language was used due to its functionalities which were compatible with the fields needed in the project i.e. Deep Learning.

4.1.1 Environment

- Google Colab : Google Colab was used due to the GPU Engine Provided by it on the website, it was more helpful than the localhost because the model needed about 10 GB GPU ram and the localhost had only 6 GB.

4.1.2 Packages and Libraries

1. **PyTorch** : PyTorch is an optimized tensor library for deep learning using GPUs and CPUs.
2. **Keras** : Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow
3. **NumPy** : NumPy is the fundamental package for scientific computing in Python.
4. **Pydub** : Manipulate audio with a simple and easy high level interface.
5. **Os** : This module provides a portable way of using operating system-dependent functionality.
6. **OpenCV** : An optimized Computer Vision library for reading, writing and processing images.
7. **Facemorpher** : A library automatically detects frontal faces and skip images if none is detected.
8. **Librosa** : A python package for music and audio analysis, It provides the building blocks necessary to create music information retrieval systems.
9. **Glob** : A module finds all the pathnames.

4.2 Dataset

AVSpeech dataset was used which is a large-scale audio-visual dataset comprising speech video clips with no interfering background noises. The segments are 3-10 seconds long, and in each clip, the audible sound in the soundtrack belongs to a

single speaking person, visible in the video. In total, the dataset contains roughly 4700 hours of video segments, from a total of 290k YouTube videos, spanning a wide variety of people, languages and face poses.

Unfortunately, only 2611 clips of the data were used due to the difficulty of uploading and extraction of the data on google drive servers because of the bad internet services in Egypt, and after the filtration of the clips using the "facemorpher".

Figure 4.1 AVSpeech Sample

4.3 Data Preprocessing

4.3.1 Audio Data Preprocessing

1. The audio segment was repeated until it reached 6 secs if it was less than that.
2. Then the audio was converted to a wav file and a STFT with a hann window and sampling rate 16000 was applied to it.

4.3.2 Face Data Preprocessing

1. The cropped RGB face images of size 224x224x3 are obtained by similarity transformation.
2. Each pixel in the RGB images is normalized by subtracting 127.5 and then dividing by 127.5.

4.4 Model

4.4.1 VAE Model

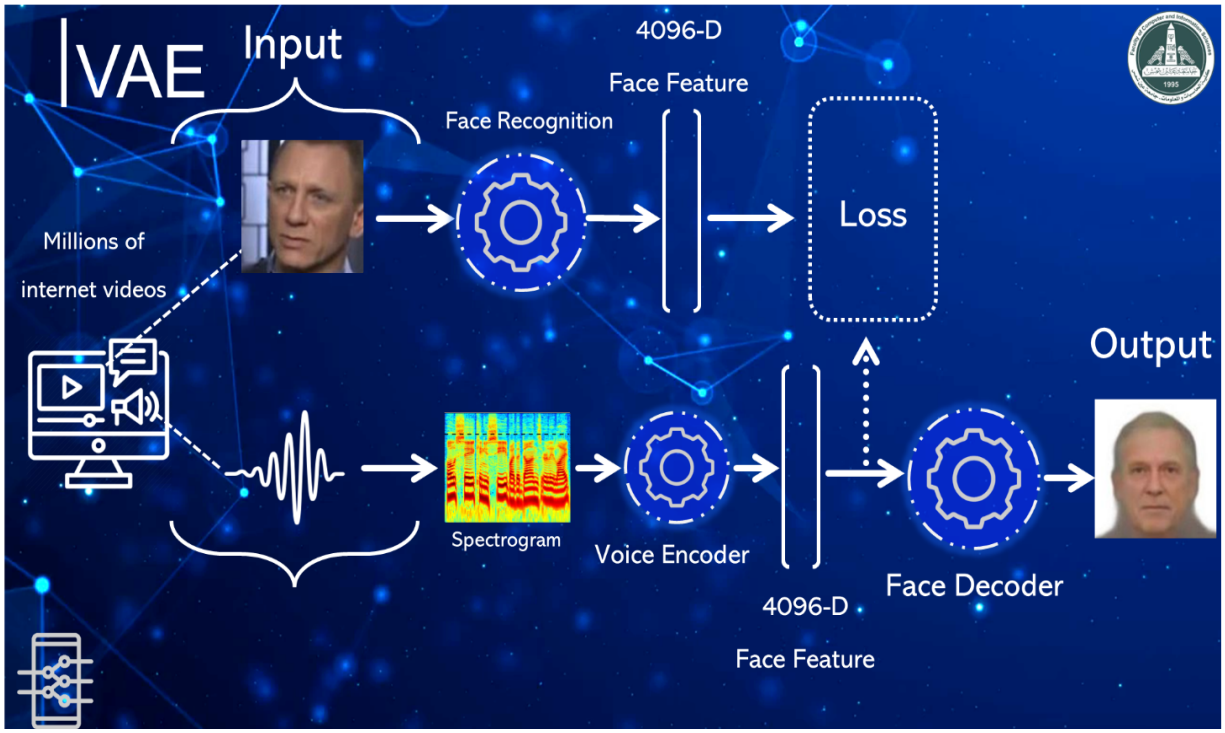


Figure 4.2 VAE Architecture

4.4.2 Voice Encoder

Layer	Input	CONV RELU BN	CONV RELU BN	CONV RELU BN	MAXPOOL	CONV RELU BN	MAXPOOL	CONV RELU BN	MAXPOOL	CONV RELU BN	MAXPOOL	CONV RELU BN	CONV RELU BN	CONV	AVGPOOL RELU BN	FC RELU	FC
Channels	2	64	64	128	–	128	–	128	–	256	–	512	512	512	–	4096	4096
Stride	–	1	1	1	2 x 1	1	2 x 1	1	2 x 1	1	2 x 1	1	2	2	1	1	1
Kernel size	–	4 x 4	4 x 4	4 x 4	2 x 1	4 x 4	2 x 1	4 x 4	2 x 1	4 x 4	2 x 1	4 x 4	4 x 4	4 x 4	∞ x 1	1 x 1	1 x 1

Figure 4.3 Voice Encoder Architecture

- voice encoder, which takes a complex spectrogram of speech as input, and predicts a low-dimensional face feature that would correspond to the associated face
- The output is a feature vector (1x4096)

4.4.3 Face Recognition (VGG-Network)

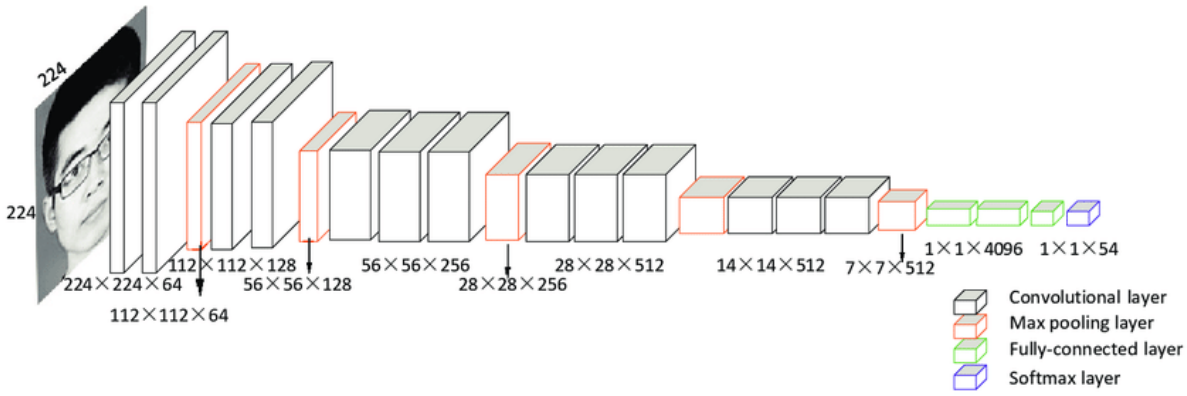


Figure 4.4 VGG-Network Architecture

VGG-Face model[14], a pre-trained face recognition model trained on a large-scale face dataset, and extract a 4096-D face feature from the penultimate layer (fc7) of the network

4.4.4 Face Decoder

which takes as input the face feature and produces an image of the face in a canonical form (frontal-facing and with neutral expression), We trained the model by 120 iterations using 149,000 face images to reconstruct the images from the feature vector (1x4096)D only.

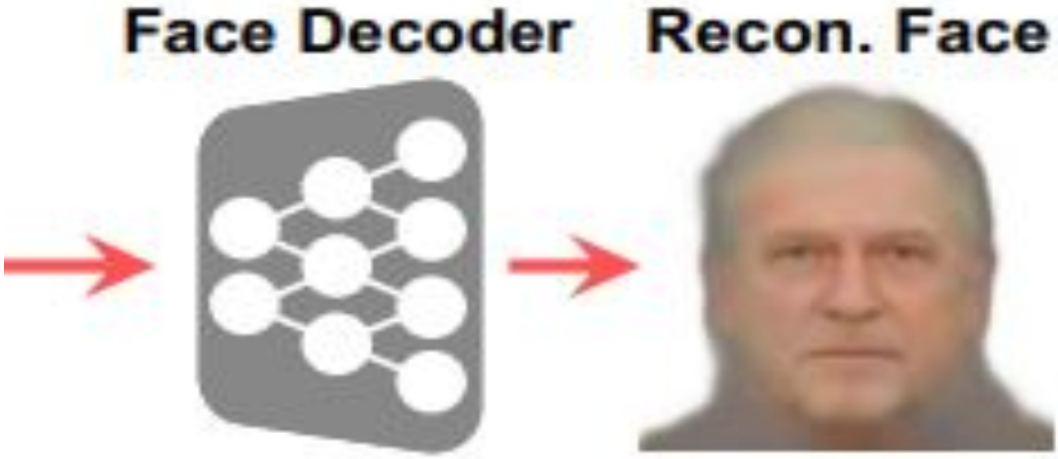


Figure 4.5 Face Decoder reconstruct face

```
Decoder(
  (mlp): Sequential(
    (0): Linear(in_features=4096, out_features=1000, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1000, out_features=50176, bias=True)
    (3): ReLU()
  )
  (texture_gen): Sequential(
    (0): ConvTranspose2d(256, 128, kernel_size=(6, 6), stride=(2, 2), padding=(2, 2))
    (1): ReLU()
    (2): ConvTranspose2d(128, 64, kernel_size=(6, 6), stride=(2, 2), padding=(2, 2))
    (3): ReLU()
    (4): ConvTranspose2d(64, 32, kernel_size=(6, 6), stride=(2, 2), padding=(2, 2))
    (5): ReLU()
    (6): ConvTranspose2d(32, 32, kernel_size=(6, 6), stride=(2, 2), padding=(2, 2))
    (7): ReLU()
    (8): Conv2d(32, 3, kernel_size=(1, 1), stride=(1, 1))
    (9): Sigmoid()
  )
)
```

Figure 4.6 Face Decoder Network

4.5 Experiments & Results

4.5.1 Experiments

We built the same architecture as in paper [8], with a network implemented in TensorFlow and optimized by ADAM with $\beta 1 = 0.5$, the learning rate of 0.001 with the exponentially decay rate of 0.95 at every 10,000 iterations.
First we tried to use the data normally but it produced a blurred image (close to the desired results but blurred), then we tried to use a face morpher to cut the images used in the training process into an image that has the face of the person only but it lead to a black result with some points color points so we didn't consider this solution, after that we increased the amount of data used and it decreased the blur a little bit but we couldn't achieve the desired result due to data problems.

4.5.2 Quantitative Results

We evaluate the performance of the architecture with two commonly used metrics are L1 (mean absolute error), and Cosine Similarity.

4.5.2.1 L1 (mean absolute error)

L1 measures the pixel-wise difference between the real and generated images. A lower L1 score indicates better similarity between the two images.

To calculate L1, we first need to resize both the real and generated images to the same size. We can then calculate the mean absolute error between the two images using the formula:

$$L1 = \frac{1}{n} * sum(|x_i - y_i|)$$

Where n is the total number of pixels in the image, x_i is the pixel value in the real image, and y_i is the pixel value in the generated image.

4.5.2.2 Cosine Similarity

Cosine Similarity measures the similarity between the feature representations of the real and generated images. A higher Cosine Similarity score indicates better similarity between the two feature representations.

To calculate Cosine Similarity, we first need to extract the feature representation of both the real and generated images using a pre-trained feature extractor network. We can then calculate the cosine similarity between the two feature representations using the formula:

$$Cosine\ Similarity = \frac{(u.v)}{(||u||\ ||v||)}$$

Where u and v are the feature representations of the real and generated images, and '.' Denotes the dot product.

Once we have computed these metrics for the 30 generated images, we can use them to evaluate the performance of the generative model. It is important to note that no single metric can fully capture the quality of the generated images, So, use a combination of metrics to get a more comprehensive evaluation.

4.5.2.3 Evaluation Metric

Table 4.1 Our Model compared with Paper's Model

	Our Model	**Paper's Model**
Epochs	DECODER: 130 ENCODER: 10	DECODER: Pre-trained ENCODER: 10
Time	DECODER: 16 H ENCODER: 1 H	DECODER: Pre-trained ENCODER: 45 H
Data Size	15 GB	1.5 TB
Results	L1 Loss : 36650.0	L1 Loss : 8.34

	Cos Sim Loss : 0.01	Cos Sim Loss : 10.92

N.B : Due to a huge problem in data availability and its usage on google drive, the model wasn’t completed.

Chapter 5: GANs Model

5.1 Environment Setup & Tools

We created this project with lots of scripts. We used “python” programming language because it was used in most of the recently published work, also it was the easiest, most useful language to use due to its libraries, as some of them are designed specifically to help develop deep learning and machine learning models.

5.1.1 Environment

Localhost

- While developing our scripts (preprocessing, training, and testing), we tried running them on Google Colab, but our dataset was bigger than 80 GB, so we ran our scripts locally on Jupyter Notebooks.

Table 5.1 Laptop Specs

GPU	RTX 2070 8 GB
RAM	32 GB

5.1.2 Packages and Libraries

1. **PyTorch**: PyTorch is an optimized tensor library for deep learning using GPUs and CPUs.
2. **NumPy**: NumPy is the fundamental package for scientific computing in Python.
3. **Pandas**: Pandas is an open-source, Python package that is most widely used for data science/data analysis and machine learning tasks.
4. **SciPy**: SciPy (pronounced “Sigh Pie”) is an open-source software for mathematics, science, and engineering.
5. **Pydub**: Manipulate audio with a simple and easy high level interface.
6. **OS**: This module provides a portable way of using operating system-dependent functionality.

5.2 Dataset

The Dataset consists of the voice recordings that are from the Voxceleb [1] dataset and the face images that are from the manually filtered version of VGGFace [2] dataset. Both datasets have identity labels. We use an intersection of the two datasets with the common identities, leading to 149,354 voice recordings and 139,572 face images of 1,225 persons. We use the whole dataset for training and testing on any recorded voice, Dataset sample in figure 4.1.

Figure 5.1 VGGFace Sample

5.3 Data Preprocessing

5.3.1 Audio Data Preprocessing

1. Use a voice activity detector interface from the WebRTC project to isolate speech-bearing regions of the recordings.
2. Extract 64-dimensional log mel-spectrograms using an analysis window of 25ms, with a hop of 10ms between frames.
3. Perform mean and variance normalization of each mel-frequency bin.
4. We randomly crop an audio clip around 3 to 8 seconds for training, but use the entire recording for testing. And if the audio is less than 10 seconds, we repeat it until it reaches 10 seconds.

5.3.2 Face Data Preprocessing

1. The cropped RGB face images of size 64 × 64 × 3 are obtained by similarity transformation.
2. Each pixel in the RGB images is normalized by subtracting 127.5 and then dividing by 127.5.

5.4 Model

5.4.1 General Model

Architecture start with a voice recording as input and applied the pre-trained Voice Embedding Network on it, to extract the Voice Embedding vector then the Generator take the Voice Embedding to generate the image and gives the Discriminator a real image and generated image to decide is it real or fake then classifier learns to assign any real face image to its identity label According to the loss function.

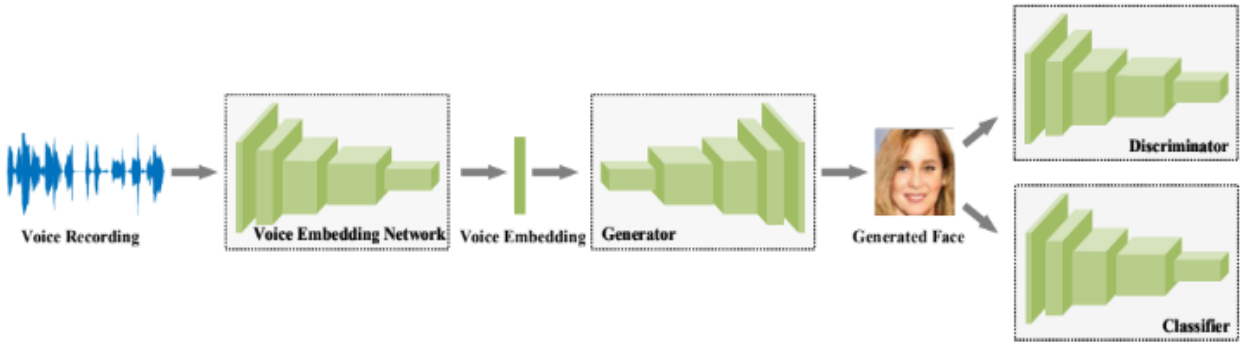


Figure 5.2 General Model

5.4.2 Voice Embedding Network

We used 1D convolutional layers with a kernel size of 3, where the stride and padding are 2 and 1, respectively. Each convolutional layer is followed by a batch normalization layer and Rectified Linear Units (ReLU). The output shape is shown accordingly, where $t_{i+1} = \lceil (t_i - 1)/2 \rceil + 1$. The final outputs are pooled over time, yielding a 64-dimensional embedding like in figure 5.3. We used this model with pretrained weights.

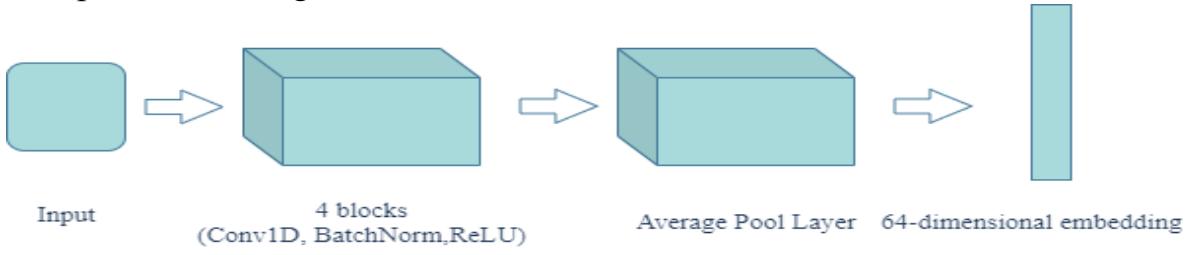


Figure 5.3 Voice Embedding Network

5.4.3 Face Embedding Network

We used a 2D convolutional layer with a kernel size of 1, where the stride and padding are 1 and 0, respectively, as an input layer, but the other convolutional layer with a kernel size of 4, where the stride and padding are 2 and 1, respectively. Each convolutional layer is followed by a Leaky ReLU layer as in figure 5.4. The final output yields a 64-dimensional embedding.

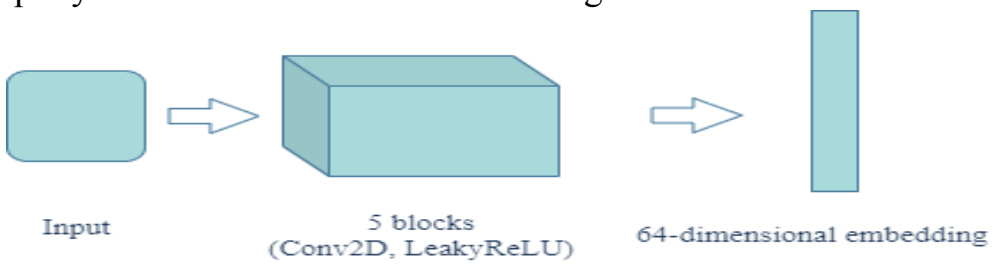


Figure 5.4 Face Embedding Network

5.4.4 Generator

We used a 2D convolutional Transpose layer with a kernel size of 4, where the stride and padding are 1 and 0, respectively, as an input layer, but the other convolutional Transpose layer with a kernel size of 4, where the stride and padding are 2 and 1, respectively. Each convolutional Transpose layer is followed by a Batch Normalization 1D layer and ReLU layer. And the last layer in the model is a 2D convolutional Transpose layer with a kernel size of 4, where the stride and padding are 1 and 0, respectively, followed by a Tanh layer as the last layer. Final outputs yielding a $64 \times 64 \times 3$ image like in figure 5.5.

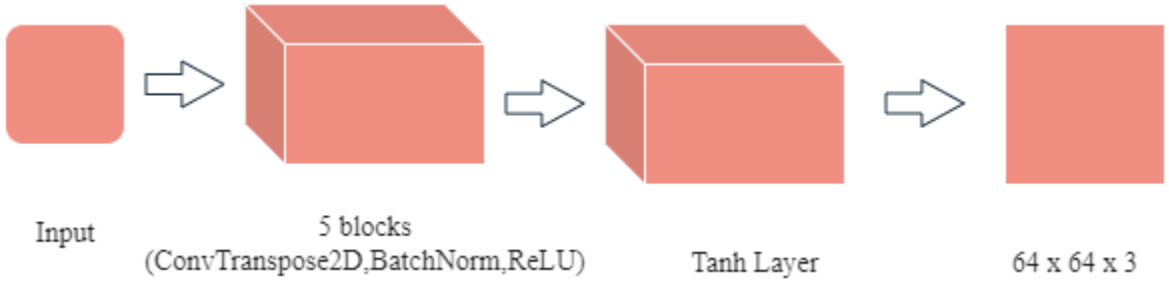


Figure 5.5 Generator Model

5.4.5 Discriminator (Classifier)

We used the results of the generator model, which is a 64 × 64 × 3 image. We feed it to the Face Embedding Network, and get the results of Face Embedding Network, which is a 64-dimensional embedding. And feed the discriminator with this embedding, then classify it if it's real or fake. By only a Sigmoid, take as input the image and output 0 or 1 like in figure 5.6.

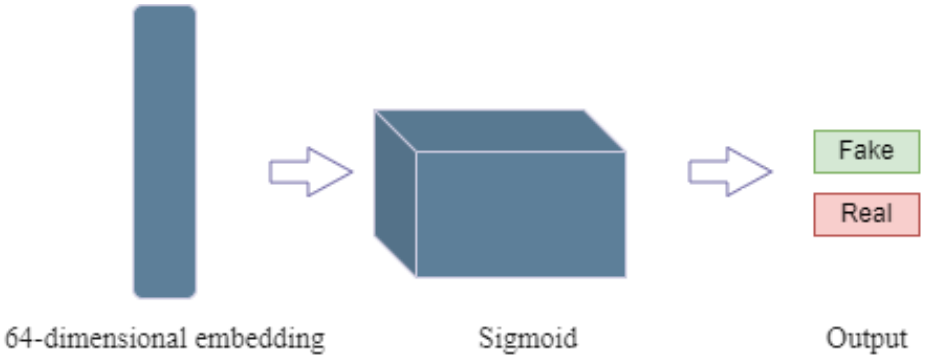


Figure 5.6 Discriminator Model

5.5 Experiments & Results

5.5.1 Experiments

We built the same architecture as in [3] but changed the optimizer and hyperparameters. In [3] they set the 'Optimizer': 'Adam' 'learning rate': 0.0002, 'beta1': 0.5, 'beta2': 0.999. We tried different hyperparameters, but finally we set the 'Optimizer': 'RMSProp', alpha: 0.9, eps: 1e-08, weight_decay=0, momentum=0, centered=False. We also add some extra layers such as we add batch normalization layers before LRelu layers in generators, discriminators, we add tanh layer as the last layer in the generator.

5.5.2 Quantitative Results

We evaluate the performance of a generative model on a sample of 30 generated images, it is common to use a combination of metrics to assess different aspects of the generated samples. Three commonly used metrics are FID (Fréchet Inception Distance), L1 (mean absolute error), and Cosine Similarity.

5.5.2.1 FID (Fréchet Inception Distance)

FID measures the distance between the distribution of real images and the distribution of generated images. A lower FID score indicates better similarity between the two distributions. To calculate FID, we first need to compute the mean and covariance of the feature representations of real and generated images using a pre-trained Inception network. We can then calculate the distance between these two distributions using the formula:

$$FID = || \mu_{real} - \mu_{gen}||^2 + Tr\left(C_{real} + C_{gen} - 2 * \sqrt{(C_{real} * C_{gen})}\right)$$

Where μ_{real} and μ_{gen} are the mean feature representations of the real and generated images, C_{real} and C_{gen} are their covariance matrices.

5.5.2.2 L1 (mean absolute error)

L1 measures the pixel-wise difference between the real and generated images. A lower L1 score indicates better similarity between the two images. To calculate L1, we first need to resize both the real and generated images to the same size. We can then calculate the mean absolute error between the two images using the formula:

$$L1 = \frac{1}{n} * sum(|x_i - y_i|)$$

Where n is the total number of pixels in the image, x_i is the pixel value in the real image, and y_i is the pixel value in the generated image.

5.5.2.3 Cosine Similarity

Cosine Similarity measures the similarity between the feature representations of the real and generated images. A higher Cosine Similarity score indicates better similarity between the two feature representations. To calculate Cosine Similarity, we first need to extract the feature representation of both the real and generated images using a pre-trained feature extractor network. We can then calculate the cosine similarity between the two feature representations using the formula:

$$Cosine\ Similarity = \frac{(u.v)}{(||u||\ ||v||)}$$

Where u and v are the feature representations of the real and generated images, and '.' Denotes the dot product. Once we have computed these metrics for the 30

generated images, we can use them to evaluate the performance of the generative model. It is important to note that no single metric can fully capture the quality of the generated images, So, use a combination of metrics to get a more comprehensive evaluation.

5.5.2.4 Evaluation Metric

Based on the evaluation metric of Fréchet Inception Distance (FID), our general model outperforms other models. However, when considering the metric of L1, the female model demonstrates superior performance. On the other hand, the paper model exhibits greater effectiveness in terms of cosine similarity. As shown in Table 5.2.

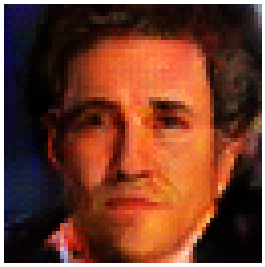
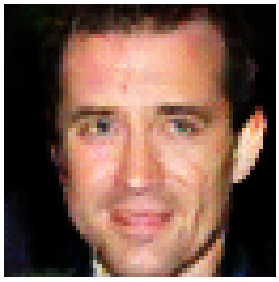
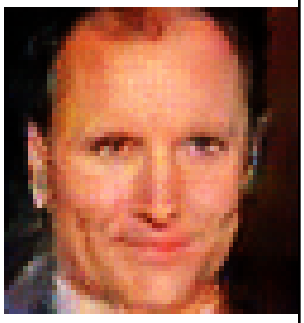
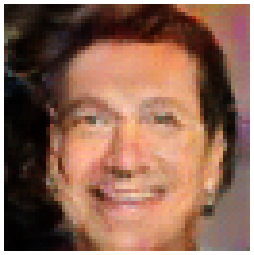
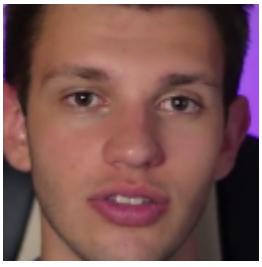
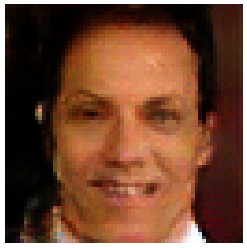
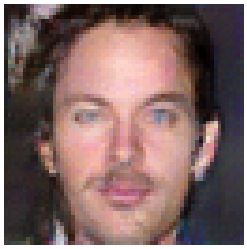
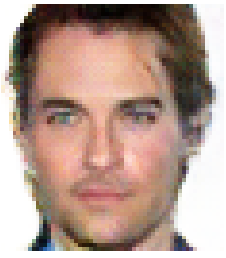
Table 5.2 Evaluation Metric

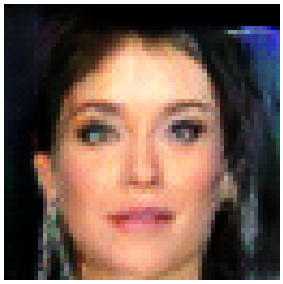
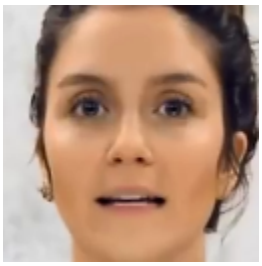
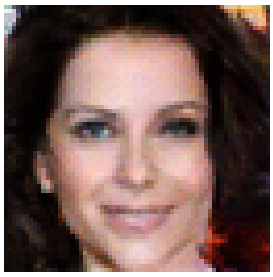
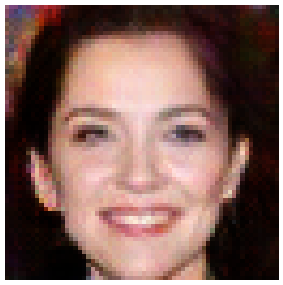
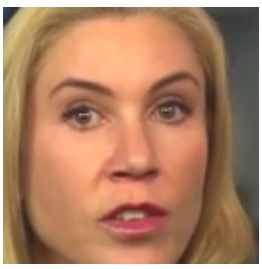
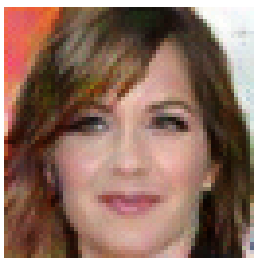
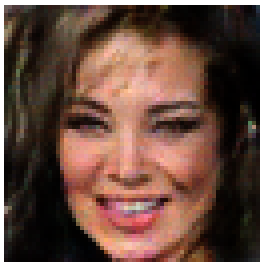
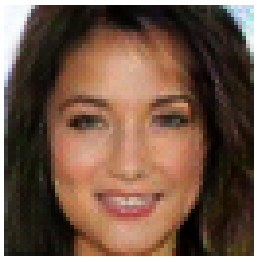
	FID	L1	Cos Similarity
General Model	**114.4445991**	61.9460271	0.244882
Females Model	126.365874	**29.41308**	0.399265
Males Model	129.303104	33.085962	0.2090025
Paper	117.42833	59.494304	**0.33712798**

5.5.3 Qualitative Results

The study was conducted to evaluate the qualitative performance of different generative models in image generation tasks. Specifically, we aimed to compare the quality of the images generated by a general model, a female model, and a male model, in terms of their realism, diversity, coherence, and visual appeal as shown in table 3.

Table 5.3 Qualitative Metric

Real image	General Model	Male model	Female model
			—
			—
			—
			—
			—

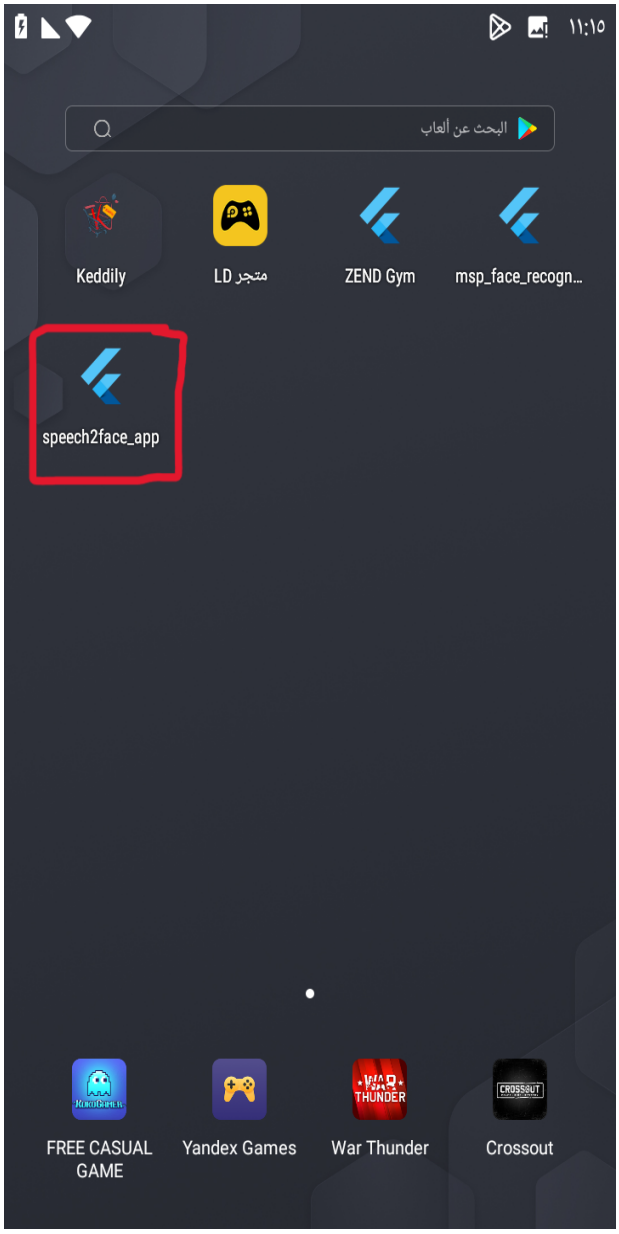
		—	
		—	
		—	
		—	
		—	

Overall after all the previous experiments and results and the comparison between each outcome it was found out that the GANs approach would provide better results specially after specifying a model for each gender so it was the chosen model to be used in our application.

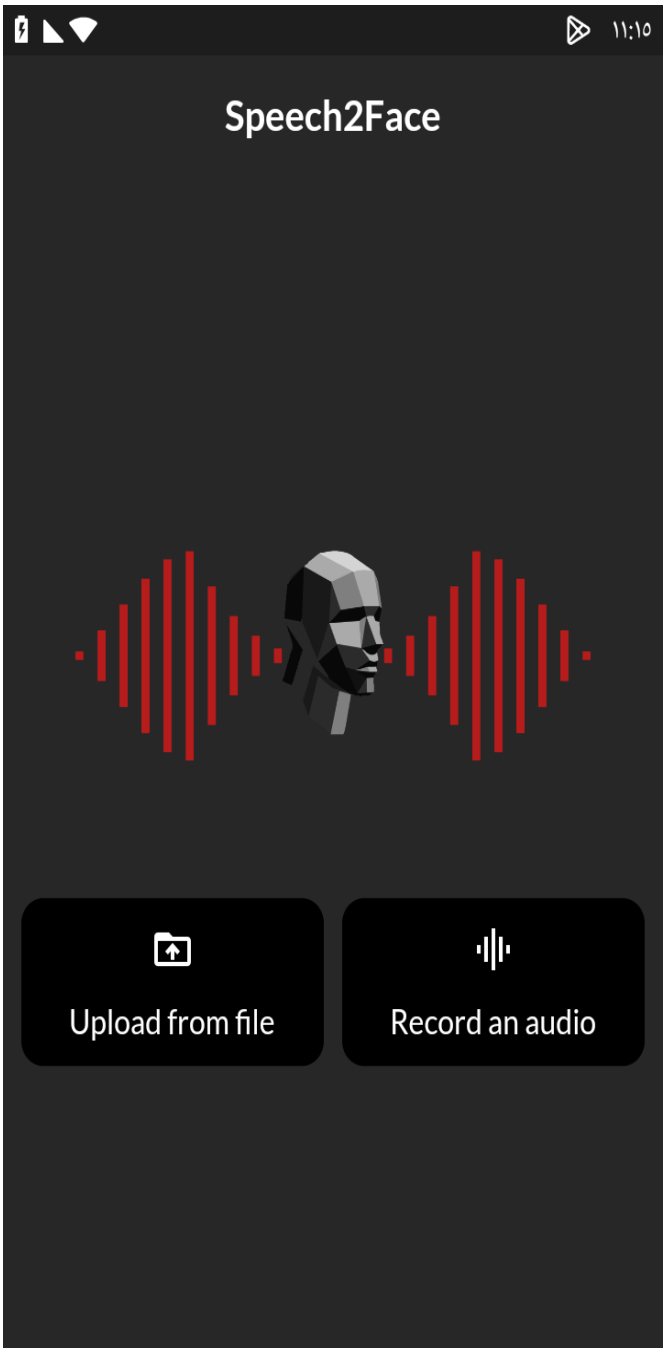
Chapter 6: User Manual

The following steps describe in detail how to operate the speech2face application.

Step 1: install the application on an (android or ios) device.

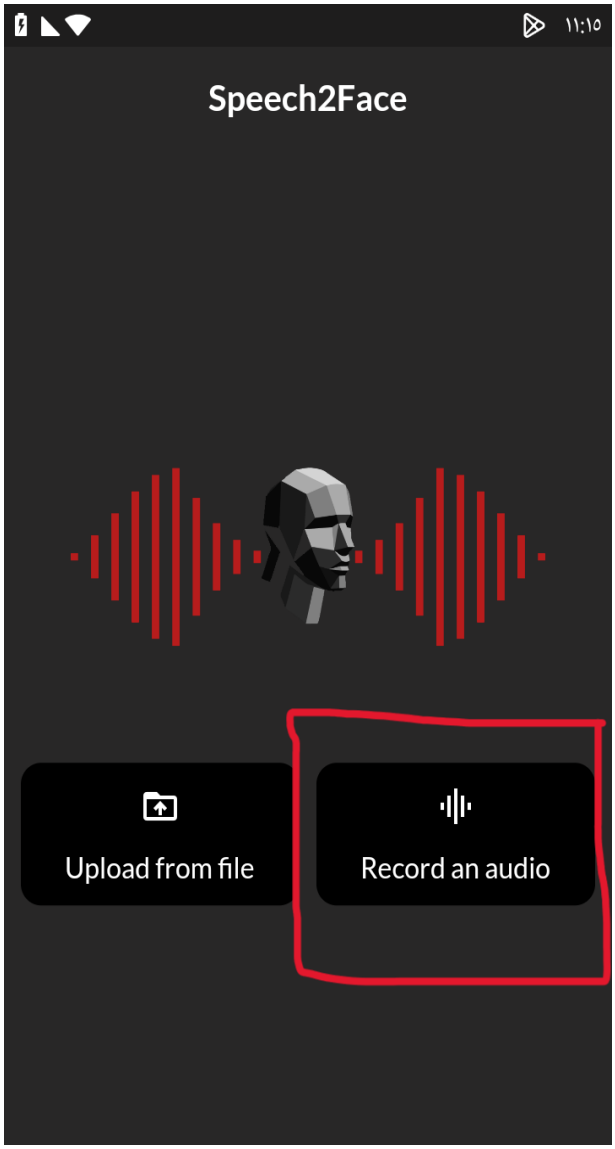


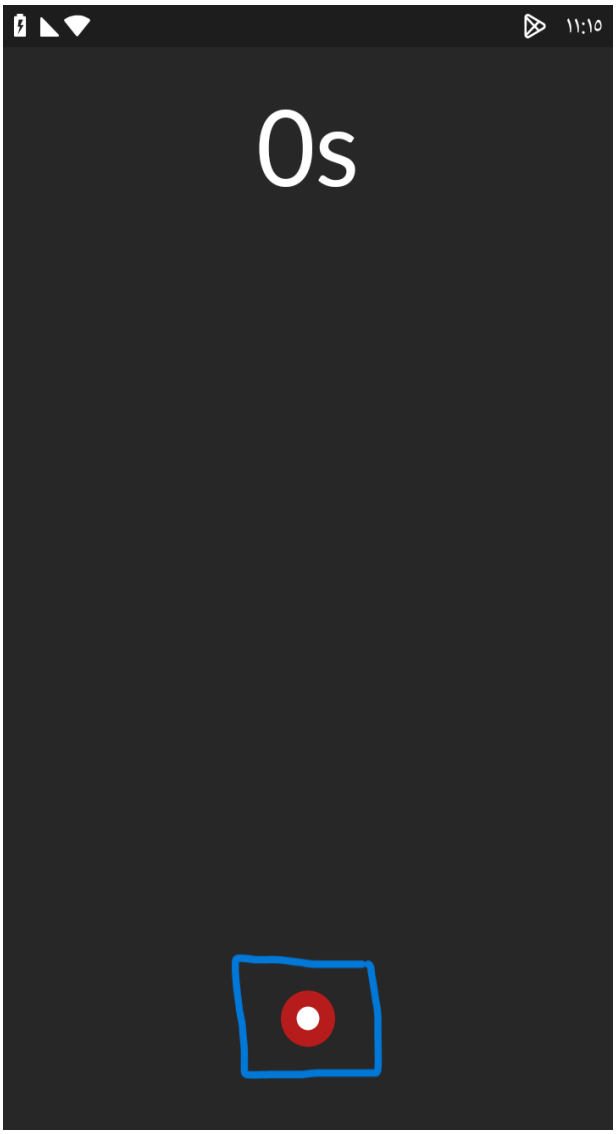
Step 2: Run the application on the mobile device.



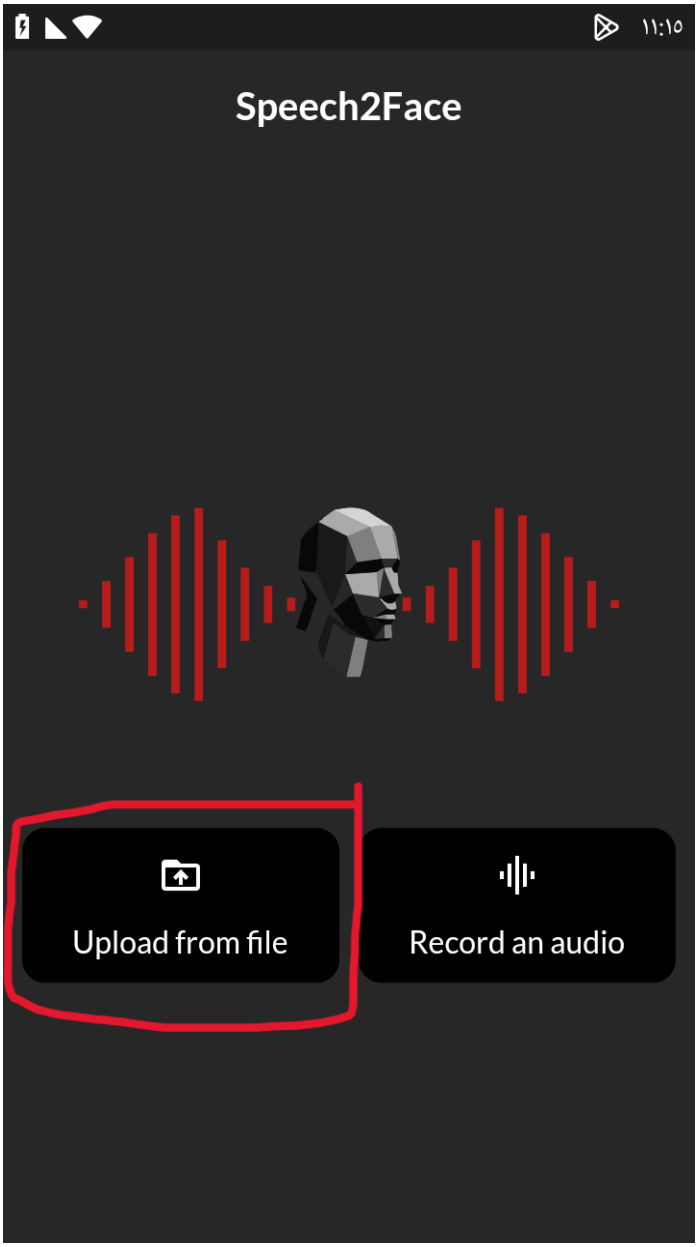
Step 3: Users have 2 options:

Step 3.1: if they press on record an audio, then they will go to screen to record a voice that they want to see the face from it,

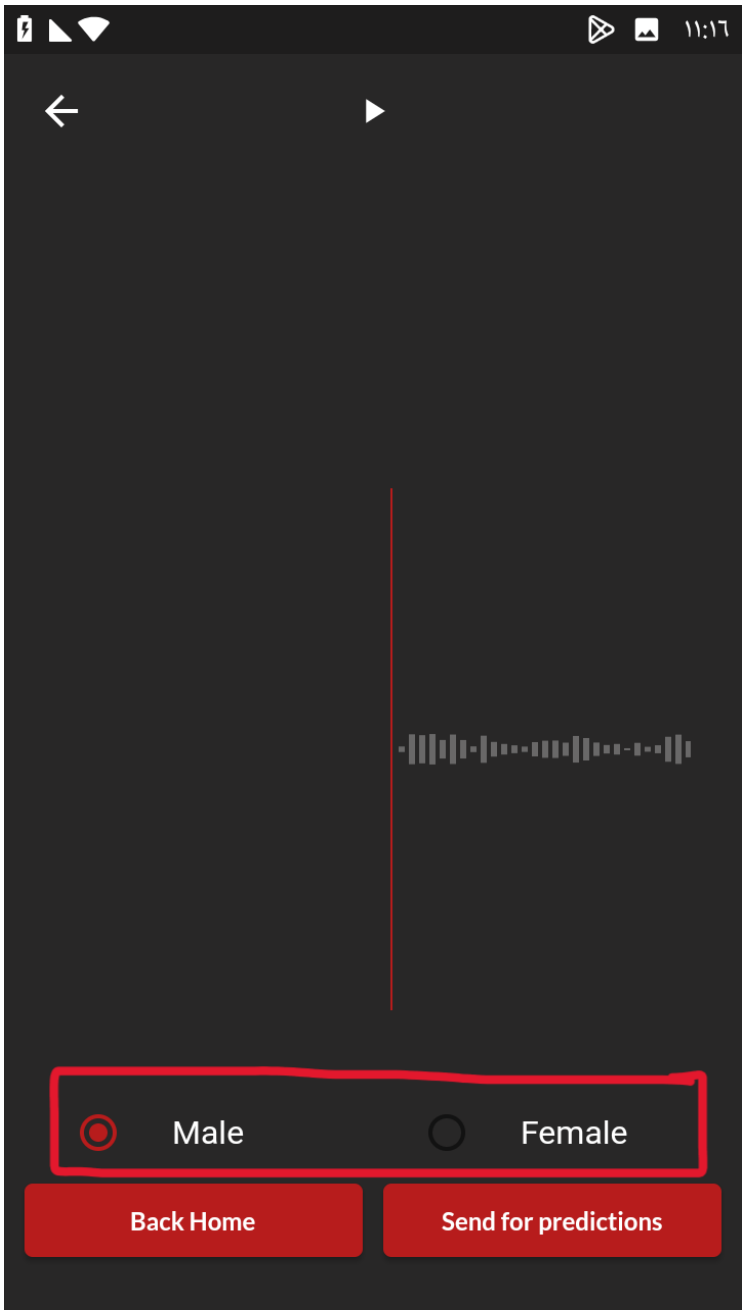




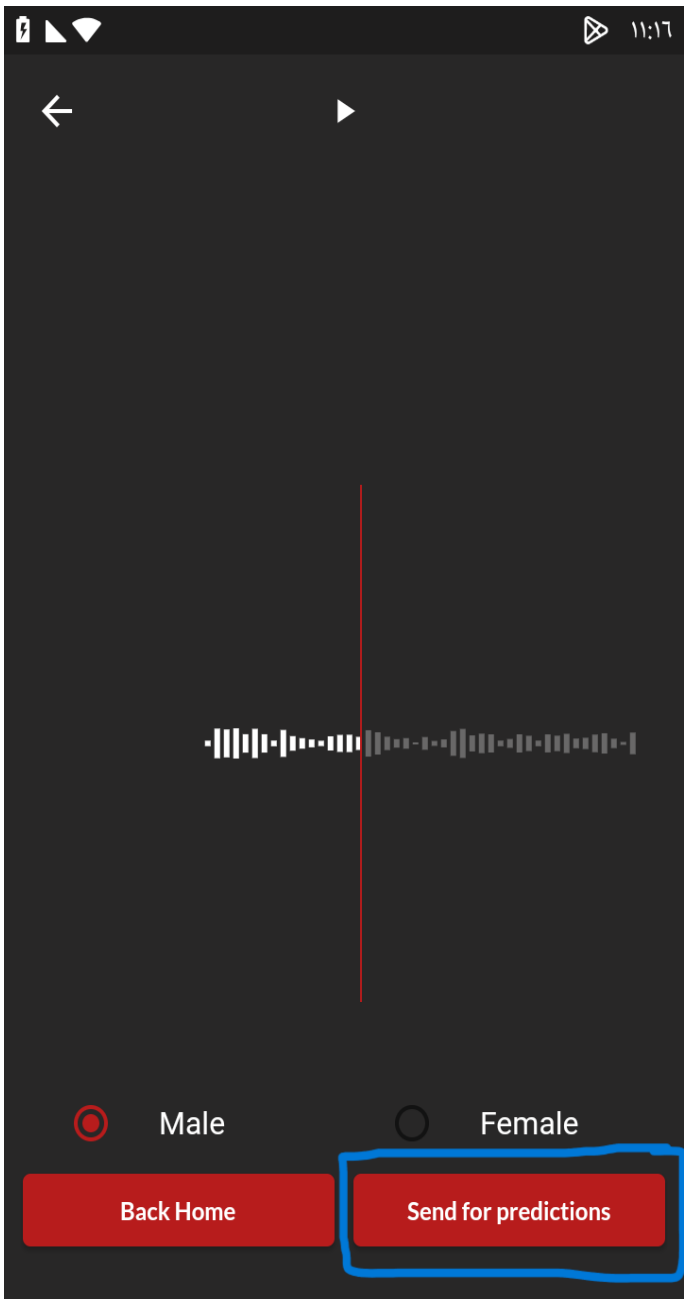
Step 3.2: or they can upload an audio from a file without recording



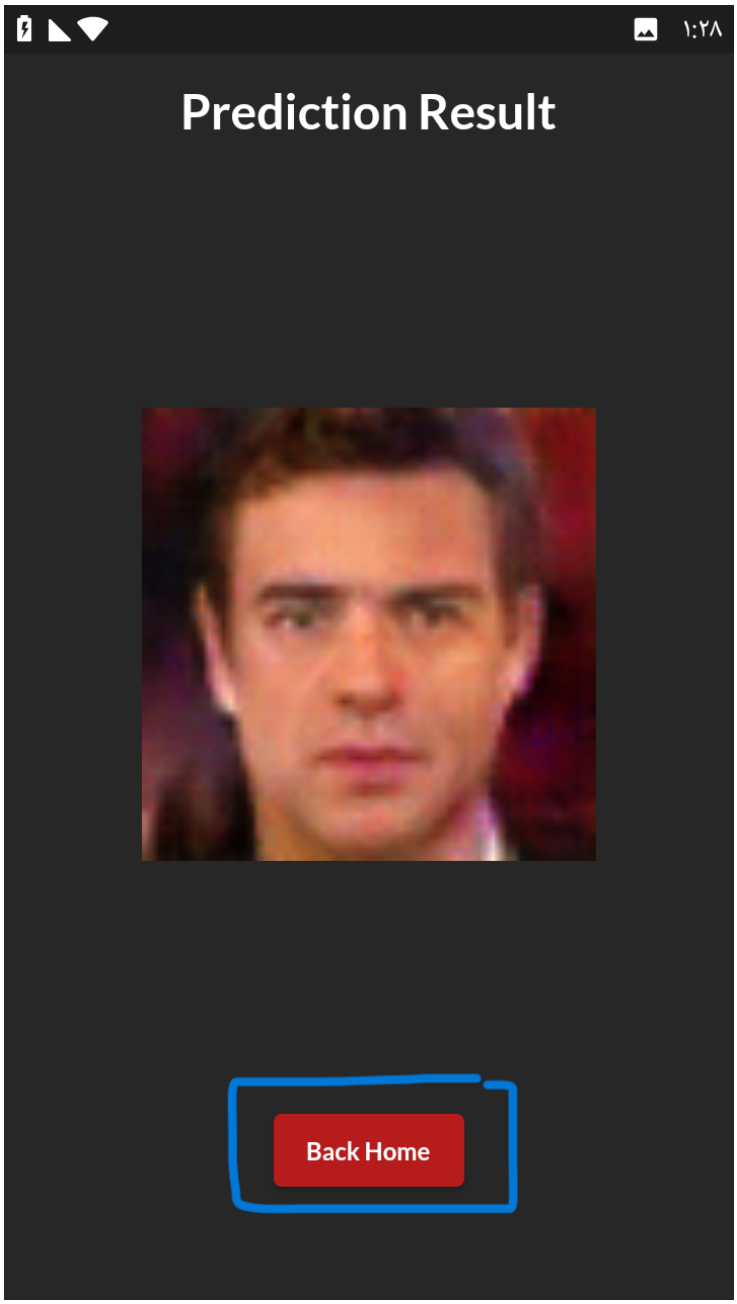
Step 4:choose gender of the person that you want to know their face from audio



Step 5: press on send for prediction to see the result



Step 6: press back to home to operate again



Chapter 7: Conclusion & Future Work

7.1 Conclusion

The aim of this project was to predict the facial features of a person given a voice record. We could achieve this goal through two approaches of generative AI which are VAE and GANs. After a lot of trials and errors in both approaches, we concluded that the amount of data and its organization is a critical step to getting better results and generalizing the model. The VAE Approach required a high-performance GPU to achieve the results so we needed to train it online on "Google Colab" but as mentioned before it also requires a huge amount of well-organized data which we couldn't manage to achieve even if we could upload the data, there were bugs in google drive that prevented us from completing the process. The GANs Approach could be achieved on a local GPU. We faced a gender mismatching problem in some results, so we made a separate model for each gender in order to overcome this problem, this solution didn't only solve the problem but it also achieved great results with better accuracies but it needed to organize the data into males and females again which takes us back to our first conclusion "Data Organization".

Overall after all the previous experiments and results and the comparison between each outcome it was found out that the GANs approach would provide better results specially after specifying a model for each gender so it was the chosen model to be used in our application.

7.2 Future Work

In Encoder-Decoder (VAE) Model we use the AVSpeech dataset and in GANS Model we use the Voxceleb dataset, Gans Model get with us a better result, so we wanted to combine two datasets and train Gans Model with two datasets to get more a better result. But before we combine the two datasets we want to apply on AVSpeech the same preprocessing that applies on Voxceleb to be able to combine this dataset with Voxceleb. We could also try merging the two models VAE and GANs, and test them as one model to enhance the results of our system. Another aspect that could be improved is enhancing our system to be more than image generation, to generate the whole face moving and saying the speech entered into the system.

References

[1] Paul H Ptacek and Eric K Sander. Age recognition from voice. Journal of speech and hearing Research,9(2):273–277, 1966.
[2] Paul Mermelstein. Determination of the vocal-tract shape from measured formant frequencies. The Journal of the Acoustical Society of America, 41(5):1283–1294, 1967.
[3] HM Teager and SM Teager. Evidence for nonlinear sound production mechanisms in the vocal tract. InSpeech production and speech modelling, pages 241–261. Springer, 1990
[4] Rita Singh. Reconstruction of the human persona in 3d, and its reverse. In Profiling Humans from their Voice, chapter 10. springer nature Press, 2020.
[5] Hosom, John-Paul. "Speech Recognition." Encyclopedia of Information Systems, 2003, pp. 155–169, https://doi.org/10.1016/b0-12-227240-4/00164-7
[6] "The Short-Time Fourier Transform | Spectral Audio Signal Processing." Dsprelated.com, 2019, www.dsprelated.com/freebooks/sasp/Short_Time_Fourier_Transform.html.
[7] Pramoditha, Rukshan. "Overview of a Neural Network's Learning Process." Data Science 365, 1 July 2023, medium.com/data-science-365/overview-of-a-neural-networks-learning-process-61690a502fa.
[8] Oh, Tae-Hyun, et al. "Speech2face: Learning the face behind a voice." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.
[9] Ephrat, Ariel, et al. "Looking to listen at the cocktail party: A speaker-independent audio-visual model for speech separation." arXiv preprint arXiv:1804.03619 (2018).
[10] A. Nagrani*, J. S. Chung*, W. Xie, A. Zisserman Voxceleb: Large-scale speaker verification in the wild Computer Science and Language, 2019.
[11] J. S. Chung*, A. Nagrani*, A. Zisserman VoxCeleb2: Deep Speaker Recognition INTERSPEECH, 2018.
[12] A. Nagrani*, J. S. Chung*, A. Zisserman VoxCeleb: a large-scale speaker identification dataset INTERSPEECH, 2017.

[13] Wen, Yandong, Bhiksha Raj, and Rita Singh. "Face reconstruction from voice using generative adversarial networks." Advances in neural information processing systems 32 (2019).
[14] "VGG Face Model" exposing.ai, 2015, https://exposing.ai/vgg_face/