CH AAZEEN AHMAD
01-1314191-010

Face-GAN: Face image enhancement using Generative Adversarial Networks

Bachelor of Science in Computer Science

Supervisor: Mr. Abdur Rehman

Department of Computer Science
Bahria University, Islamabad

6th May, 2023

Certificate

We accept the work contained in the report titled "Face-GAN :Face image enhancement using Generative Adversarial Networks", written by Mr. Ch Aazeen Ahmad as a confirmation to the required standard for the partial fulfillment of the degree of Bachelor of Science in Computer Science.

Approved by . . . :

Supervisor:: Mr Abdul Rahman

Internal Examiner: Name of the Internal Examiner

External Examiner: Name of the External Examiner

Project Coordinator:: Ms.Maryam Khalid Multani

Head of Department:: Dr Arif ur Rehman

6st May, 2023

Abstract

The primary objective of this project is to construct a deep learning-based Generative Adversarial Network. Super-resolution, the process of enhancing the resolution of low-resolution images, has garnered significant attention in the field of computer vision. In recent years, Generative Adversarial Networks (GANs) have emerged as a powerful tool for achieving remarkable improvements in super-resolution tasks. This project focuses on leveraging GANs and deep learning techniques to address the challenge of super-resolving low-resolution images.The proposed approach involves training a GAN architecture consisting of a generator and a discriminator. The generator network aims to transform low-resolution images into high-resolution counterparts, while the discriminator network learns to distinguish between generated high-resolution images and real high-resolution images. This adversarial training setup fosters a competitive learning process, driving the generator to produce increasingly realistic and visually appealing results. By leveraging the outcomes from the first phase, I progressively refine the generator's performance, ultimately leading to superior results compared to its previous state. Despite significant advancements in single image resolution through the use of fast and deep neural convolution networks, a central challenge remains unresolved: effectively preserving fine texture details while accurately addressing substantial objects. In light of this, I propose SR-GAN—an innovative Super-Resolution (SR) Generative Adversarial Network (GAN).The results obtained demonstrate significant improvements over traditional interpolation-based methods and state-of-the-art super-resolution techniques. Overall, the proposed approach holds promise for various applications, including medical imaging, surveillance systems, and digital entertainment, where high-resolution imagery plays a vital role.

Acknowledgments

In the name of Allah, the most beneficent and the most merciful. I am highly grateful to the One who created us and blessed us with a privileged life. I like to thank our parents who have always been a pillar of strength and support. I extend my heartfelt appreciation to my supervisor, Mr. Abdur Rehman, for valuable guidance, support, encouragement throughout our final year project. His insightful feedback and constructive criticism helped me to stay on track and complete my project successfully. Thank you, Sir Abdul Rahman, for being such an integral part of my journey and for helping me to grow and learn. I am truly grateful for your support. Last but not the least, i would like to appreciate our friends who make study and hard times less challenging. May Allah (SWT) guide me to the right path.

CH AAZEEN AHMAD
Bahria University, Islamabad Pakistan

6th May, 2023

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

SRGAN	Super-Resolution Generative Adversarial Networks
ANN	Artificial Neural Network
CNN/ConvNet	Convolutional Neural Network
ML	Machine Learning
DL	Deep Learning
GAN	Generative Adverserial Networks
SR	Super Resolution
AI	Artificial Intelligence
ResNet	Residual Network
PReLU	Parametric Rectified Linear Unit
VGG	Visual Geometry Group

Chapter 1

Introduction to Generative Adverserial Networks

Generative Adversarial Networks(GANs), an approach to generative modelling employing deep learning techniques, such as convolutional neural networks, is called generative adversarial networks (GANs).Generative modelling is a machine learning task that involves automatically identifying and learning the regularities or patterns in input data so that the model can be used to produce new examples that could have been reasonably derived from the original dataset.

GAN are basically two neural networks fighting against each other. It consists of two networks Generator and Discriminator . Generator generates specific data based on the distribution of opportunities and the analyst tries to predict the weather data from the input database or generator. Generator rather than trying to add generated data to deceive the discriminator.

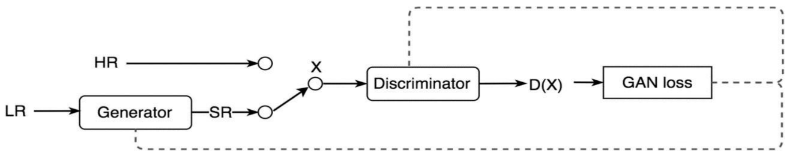


Figure 1.1: Generative Adversarial Network (GAN) Structure

Generative Adversarial Networks incorporates a family of ways to learn production models where the way to use a computer is based on the concept of the game.A Generator (G) capable of producing samples from data distribution is what GANs aims to accomplish, with encrypted conversion vectors from lower-dimension latent (Z) space to high-resolution data space (X). Normally, hidden carriers are sampled from Z using a uniform or standard distribution. To G training, Discriminator (D) is trained to differentiate real training samples from which the Generator produced the fake samples. Thus, Discriminator returns the value D (x) [0,1] which can be defined as an opportunity that the input sample (x) is the actual sample from the data distribution. For this purpose, G is trained to block D by making better samples of the actual training samples, while D is continuously pointed to differentiate real from fake samples.

In extreme cases, G does not have direct access to real samples from the training set, as it only reads in its interaction with D. If G can accurately match the actual px data distribution, then D will be very confused, predicting 0.5 of all input samples Both D and G using deep dividing networks, usually containing of many discussion layers, and fully connected.

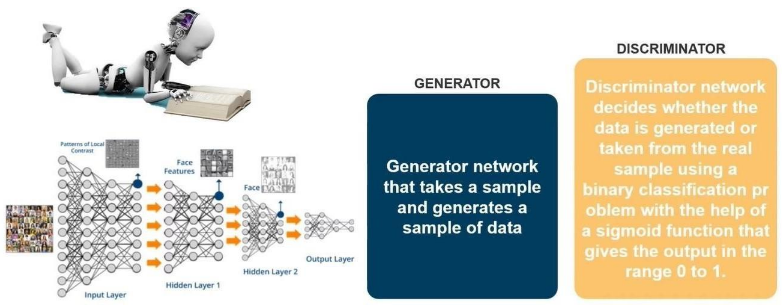

Patterns of Local Contrast
Face Features
Faces
Input Layer
Hidden Layer 1
Hidden Layer 2
Output Layer
GENERATOR
Generator network that takes a sample and generates a sample of data
DISCRIMINATOR
Discriminator network decides whether the data is generated or taken from the real sample using a binary classification pr oblem with the help of a sigmoid function that gives the output in the range 0 to 1.

Figure 1.2: What are GANS(www.youtube.com/watch?v=5g1eXmQtl0Et=391s)

1.1 Background

GANs are a model architecture for training a generative model, and it is most common to use deep learning models in this architecture.

The GAN architecture was first described in the 2014 paper by Ian Goodfellow, et al. titled “Generative Adversarial Networks.”

A standardized approach called Deep Convolutional Generative Adversarial Networks, or DCGAN, that led to more stable models was later formalized by Alec Radford, et al. in the 2015 paper titled “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.

Generative adversarial networks represent a form of artificial intelligence algorithm specifically devised to address the generative modelling quandary. The primary objective of a generative model is to analyze a set of training instances and acquire knowledge about the underlying probability distribution that gave rise to them. Subsequently, Generative Adversarial Networks (GANs) possess the capability to produce additional instances by leveraging the estimated probability distribution. Although generative models grounded in deep learning are prevalent, GANs stand out as one of the most triumphant examples, particularly due to their remarkable capacity to generate lifelike, high-resolution images. GANs have demonstrated successful applications across a wide array of tasks, primarily in research domains, yet they continue to present distinct challenges and avenues for further investigation owing to their reliance on game theory, in contrast to the predominantly optimization-based approaches found in other generative modelling methodologies.

1.2 Problem Description

The vast majority of prior work for this problem focus on how to increase the resolution of low-resolution images which are artificially generated by simple bilinear down-sampling (or in a few cases by blurring followed by down-sampling).We show that such methods fail to produce good results when applied to real-world low-resolution, low quality images.

- Traditional image super-resolution techniques, such as bicubic interpolation or up-sampling, can result in images that are blurry or lack fine details. These techniques simply increase the size of the image without adding any additional information or details, resulting in a loss of image quality and realism.
- traditional image super-resolution techniques may not be able to effectively handle complex image features, such as textures or patterns.
- Traditional image super-resolution techniques may require manual intervention or parameter tuning, which can be time-consuming and may not always result in optimal performance.

To circumvent this problem, we propose a two-stage process which firstly trains a High-to-Low Generative Adversarial Network (GAN) to learn how to degrade and downsample high-resolution images requiring, during training, only unpaired high and low-resolution images. Once this is achieved, the output of this network is used to train a Low-to-High GAN for image super-resolution using this time paired low- and high-resolution images. Our main result is that this network can be now used to efectively increase the quality of real-world low-resolution images. We have applied the proposed pipeline for the problem of face super-resolution where we report large improvement over baselines and prior work although the proposed method is potentially applicable to other object categories.

1.2.1 Why unsupervised Generative model for image super resolution

Supervised learning often surpasses human accuracy in trained models, leading to its integration into various products and services. However, the learning process itself falls short of human abilities. It heavily relies on human supervisors to provide output examples for each input, and existing approaches often require millions of training examples to out-perform humans. To address these limitations, researchers focus on unsupervised learning, particularly using generative models like generative adversarial networks (GANs). GANs offer potential solutions and present core research problems related to convergence in games. Unsupervised learning encompasses various algorithms with different goals, such as clustering and dimensionality reduction. Generative modeling is another approach where a model aims to approximate an unknown data distribution as closely as possible. Traditional methods face challenges in designing tractable density functions or computationally efficient approximations. GANs, an implicit generative model, generate samples directly from the model distribution. Although GANs have excelled in generating high-quality images, they continue to attract interest for their applications beyond straightforward generation.

Model follows unsupervised learning model as a generative model when we talk about adversarial the model will train in adversarial setting and network only means for the training of model we use neural networks as artificial intelligence algorithms.In GANS there is generator network that takes sample and generates sample of data and after this the descriminator network decides weather the data is generated or taken from the real sample using the binary classification problem with the help of the sigmoid function that gives output as 0 or 1 the generative model analyses the data in such a way that after the training phase the probability of descriminator making the mistake maximizes and descriminator on the other hand baased on a model that will estimatethe probability that the sample is coming from the real data or from the generator

We have training data which is given to give the real sample and generator network is going to generate the sample from the random noise or the examples and then it will goes to the descriminator network where it is going to check weather the sample that is oming is real or fake,that is how GANS actually work.

1.3 Motivation

The motivation behind this project is to gain knowledge about Deep Learning and Artificial intelligence and to help society by providing our product. This project will help different markets like surveillance, security, medical diagnosis, space exploration. We have to learn that how we trained a machine that can learn from experience. So, we opted this deep learning which is sub part of machine learning field for project.

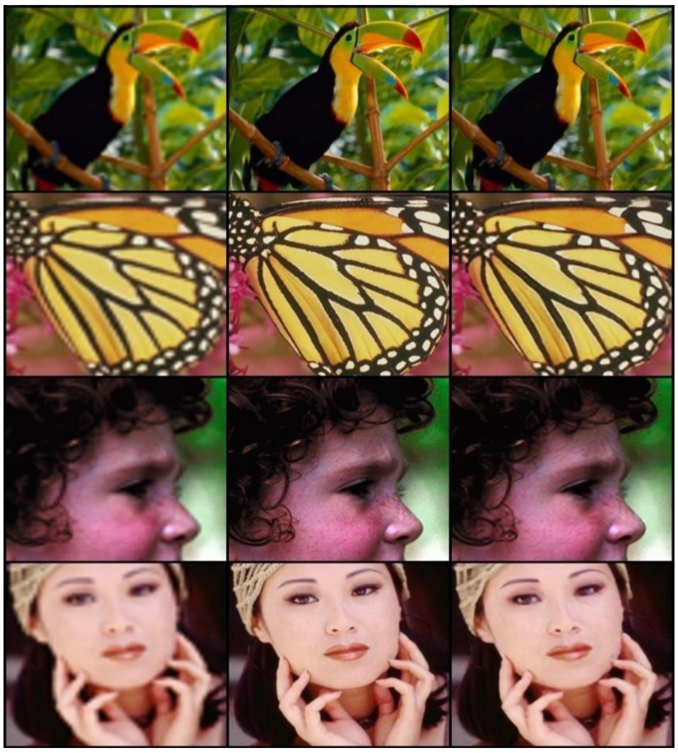


Figure 1.3: 1st column: Interpolated LR images, 2nd column:original HR images, 3rd column: SRGAN image.

1.4 Objective

To design a model based on Super resolution GANs, that will super resolute the given image from user using neural networks.

- Develop model from scratch to enhance images.
- Enchance the quality of given image to end confusion/blurriness.
- To develop application to generate high resolution images.

Numerous GAN-based techniques have been developed for manipulating data since the development of GANs. Facial images GANs are able to produce more realistic faces when compared to conventional algorithms, however most of them are unable to preserve the identity of the individual being transformed.

1.5 Project Scope

- This project is related to face only because descriminator is trained on high resolution faces. So, it is limited to generate high resolution faces only.
- Collecting a dataset of low-resolution face images, Preprocessing the dataset, Training a GAN model, Evaluating the performance of the GAN model
- It is widely used in various fields of like image processing, Surveillance and security, Gaming and animation etc

1.6 Methodology

Throughout the training process, a high-resolution (HR) image undergoes a transformation into a low-resolution counterpart. Subsequently, a Generative Adversarial Network (GAN) is employed to enhance the low-resolution images and elevate them to a super-resolution level. To achieve this, a discriminator is utilized to discern between the high-resolution images and provide feedback to both the discriminator and generator through adversarial network loss backpropagation. The network architecture for both the generator and discriminator primarily consists of convolution layers and parameterized Rectified Linear Units (ReLU). .

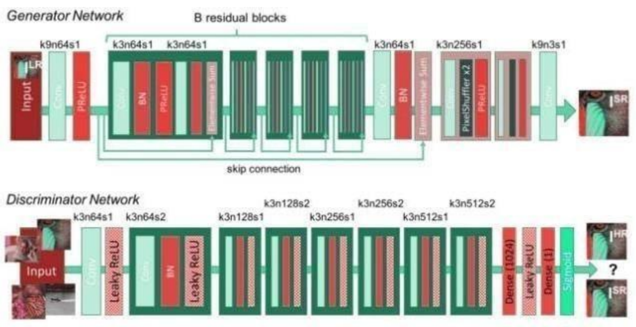


Figure 1.4: Generator network

Figure 1.5: SRGAN results

As shown above, SRGAN enhances visual appeal by incorporating additional intricate elements when compared to a comparable design lacking GAN integration

1.6.1 Generator

The generator architecture employed in our approach consists of 16 residual blocks with an identical layout. The structure of these residual blocks can be observed in the Figure below. To enhance the generation of details, we propose utilizing dense blocks instead. Our rationale behind this modification is driven by the fact that, since the primary objective of this network is to generate super-resolution images, employing conventional residual blocks alone proves insufficient. In the SR-ResNet, 16 residual blocks are combined into a larger block, with a skip connection that connects the first and last blocks, aiming to improve gradient flow. However, this approach may not fully exploit the benefits of skip connections. Our proposition is to employ a novel network architecture called DenseNet. Differing from ResNet, DenseNet employs concatenation rather than direct summation of feature maps. Consequently, each layer receives the feature maps from all preceding layers as input. This is represented by the equation: li = max (0, wi × [l1, l2, ..., li-1] + bi), where [l1, l2, ..., li-1] denotes the concatenation of feature maps generated in the previous convolutional layers 1, 2, ..., i-1. The structure of DenseNet establishes short paths between each layer and all other layers, facilitating the smooth flow of information within deep networks. Additionally, DenseNet exhibits the advantage of parameter reduction through feature reuse, resulting in lower memory consumption and computational requirements for achieving high performance. Consequently, we employ dense blocks as the fundamental building blocks in our generator network..

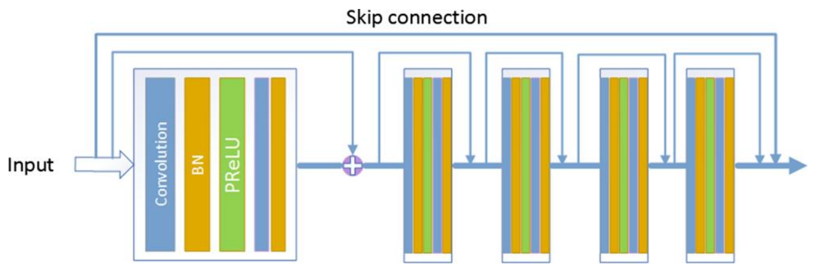


Figure 1.6: The structure of the residual blocks

1.6.2 Discriminator

In addition to enhancing the structure of the generator, we have also made improvements to the discriminator based on the spectral normalization GAN approach. The objective is to train the discriminator network to differentiate between the generated super-resolution (SR) image and the original high-resolution (HR) image. The discriminator architecture depicted in the above figure follows the design proposed by SRGAN. It consists of eight convolutional layers, each utilizing a 3x3 filter kernel. The number of filter kernels increases from 64 to 512 with a doubling rate. To obtain the probability of sample classification, two dense layers and a sigmoid activation function are employed. Throughout the network, we utilize the LeakyReLU activation with a slope of 0.2 and avoid the use of max-pooling. To enhance the discriminator, we replace the standard discriminator with the spectral normalization discriminator, denoted as DSN. Spectral normalization can be explained as follows: We consider a simple discriminator constructed as a residual network with the input x: f(x,) = F(x, Wi) + x, where = W1, W2, ..., Wi represents the set of learning parameters. The function F(x, Wi) denotes the residual mapping to be learned, such as F(W2h(W1x)), where h represents the activation function. For simplicity, we omit the bias terms for each layer. The operation F(x) is performed using a shortcut connection and element-wise addition. The final output of the discriminator is obtained as D(x,) = H(f(x,)), where H corresponds to an activation function chosen by the user based on the desired divergence or distance measure.

1.6.2.1 How to train

Taining happens in 2 phases:

In first phase we train discriminator and freeze the generator which means that the training for the generator turns false and the network will only do the forward pass and there will not be any back propagation,basically the discriminator is trained in real Data and check if it can predict them correctly and same with fake data to identify them fake.

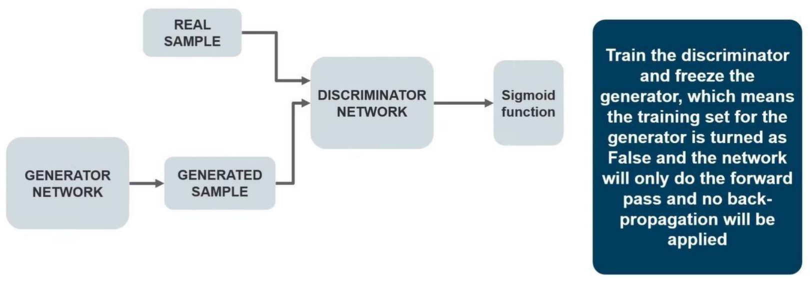

REAL SAMPLE
DISCRIMINATOR NETWORK
Sigmoid function
GENERATOR NETWORK
GENERATED SAMPLE
Train the discriminator and freeze the generator, which means the training set for the generator is turned as False and the network will only do the forward pass and no back-propagation will be applied

Figure 1.7: Training discriminator

In second phase we train generator and freeze the discriminator, , we adopt a training approach where the discriminator is frozen while focusing on training the generator. we obtain results from the generator and utilize them to refine the model further, aiming to deceive the discriminator more effectively and achieve improved output quality.

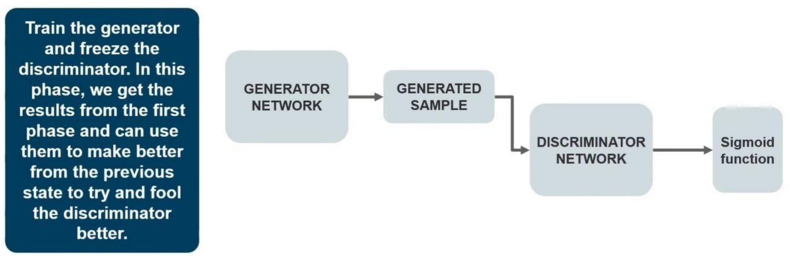

Train the generator and freeze the discriminator. In this phase, we get the results from the first phase and can use them to make better from the previous state to try and fool the discriminator better.
GENERATOR NETWORK
GENERATED SAMPLE
DISCRIMINATOR NETWORK
Sigmoid function

Figure 1.8: Training Generator

To understand it better:

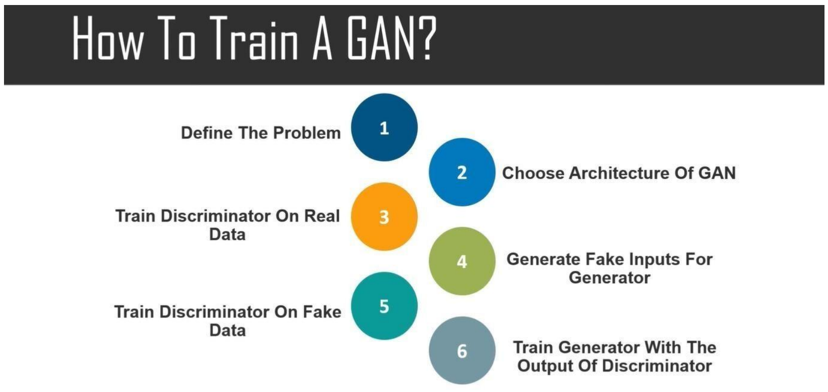

How To Train A GAN?
1 Define The Problem
2 Choose Architecture Of GAN
3 Train Discriminator On Real Data
4 Generate Fake Inputs For Generator
5 Train Discriminator On Fake Data
6 Train Generator With The Output Of Discriminator

Figure 1.9: Training Generator

1.7 Summary

GANs are a type of generative model that operates based on principles from game theory. They have demonstrated remarkable practical success in generating realistic data, particularly in the realm of image generation. However, training GANs remains a challenging task. On the other hand, Super-resolution (SR) imaging refers to a set of techniques aimed at enhancing the resolution of an imaging system. These techniques involve reconstructing a higher-resolution image or sequence from observed low-resolution (LR) images. With over three decades of development, both multi-frame and single-frame SR techniques have found significant applications in our daily lives. SRGAN, a specific type of GAN, can be utilized to enhance the quality of our images.

Chapter 2

Literature Review

In this chapter, we would be reviewing the work (similar to ours) of others extensively and we would try to find gaps and/or shortcomings of those works. We would also try to explain how our proposed application would fill those gaps by avoiding the flaws existing on the current platforms.

Photographs record valuable moments of our life. With the popularization of mobile phone cameras, users enjoy taking photographs even more. However, current cameras have limitations. They have to reconstruct a complete and high-quality image from a set incomplete and imperfect samples of the scene. The samples are often noisy, incomplete in color and limited in the resolution and the dynamic range. Image enhancement methods attempt to address the issues with color rendition and image sharpness.

2.0.1 Related work

Image enhancement has been studied for a long time. Many operations and filters have been proposed to enhance details, improve contrast and adjust colors. Wang et al proposed a method for enhancing details while preserving naturalness. Aubry et al proposed local Laplacian operator for enhancing details. Most of these operations are algorithmic and based on heuristic rules. Bychkovsky et al. proposed a learning-based regression method for approximating photographers' adjustment skills. For this purpose, they collected a dataset containing images before and after adjustments by photographers. The convolutional neural networks (CNNs) have become a major workhorse for a wide set of computer vision and image processing problems. They have also been applied to the image enhancement problem. Yan et alproposed the first deep-learning-based method for photo adjustment. Gharbi et al. proposed a fast approximation for existing filters. Ignatov et al took a different approach by learning the mapping between a mobile phone camera and a DSLR camera. They collected the DPED dataset consisting of images of the same scene taken by different cameras. A GAN model was used for learning the mapping. Chen et al. approximated existing filters using a fully convolutional network. It can only learn existing filters and cannot do beyond what they can do. All these methods are supervised and require paired images while ours is unpaired. The unpaired nature eases the process of collecting training data.

Table 2.1: GAN-synthesized images used for super-resolution

	GAN Model	Technique
1	MSGAN	Lesion-Focused SR method
2	SRGAN	Progressive upscaling method to generate true colors
3	ESRGAN	Slices from 3 latitudes are used for SR
4	NESRGAN	Noise and interpolated sampling
5	MedSRGAN	Residual whole arbitary-scale super resolution
6	FPGAN	Use a divide-and-conquer manner with multiple subbands in the wavelet domain
7	End to End GAN	Uses a hierarchical structure

2.1 Types of GANs:

2.1.1 Super-Resolution Generative Adversarial Networks (SRGAN)

Super-resolution (SR) is upsampling a low-resolution image into a higher resolution with minimal information distortion. Since researchers had access to machines strong enough to compute vast amounts of data, significant progress has been made in the super-resolution field, with bicubic resizing, efficient sub-pixel nets, etc

2.1.2 Wasserstein GAN (WGAN).

WGAN is another improvement to the family of generative adversarial networks that improves model stability and defines a loss function that takes into account the degree of difference between the probability distributions of real and fake images. The critic is altered to produce a real/false score instead of the probability of being real or fake

2.1.3 Deep Convolutional GAN (DC-GAN

DC-GAN is a GAN model that uses a deep CNN for each generator and discriminator model. Using the Generator-Discriminator framework, this layout essentially uses a CNN to provide snapshots from noisy data that belong to a particular distribution.

2.1.4 GAN with interpolation and conditional latent space (GAN-INT-CLS).

It is a GAN model that is trained on common text and image pairs, as well as for each image, additional text embeddings are included to help the discriminator easily distinguish whether it is real or fake.

2.1.5 Progressively Growing GAN (PRO-GAN)

This is an advance GAN helps in generating high-resolution images by constantly adding new layers to the model and helping the generator and discriminator to train gradually instead of learning everything at the same time. In PRO-GAN, Multi Scale

2.1.6 Gradient PROGAN (MSG-PROGAN)

This is an upgraded version of PRO-GAN that uses gradients along with a single generator and discriminator that will use multi-level connections for better results.

2.2 Applications of GANS for image applications:

There are many image applications of GANS,some are discussed:

2.2.1 Image generation with enhanced quality

The primary focus of current GAN research has been on enhancing the quality and usefulness of image generation capabilities. A notable advancement in this direction is the extension of the LAPGAN model by incorporating a CNN cascade, allowing for image generation within a Laplacian pyramid structure (Donahue et al., 2016). Another significant development is the self-attention based GAN (SAGAN) introduced by Zhang et al. (2019) for image generation tasks. SAGAN incorporates attention mechanisms, enabling the modeling of long-range dependencies. Unlike conventional convolutional GANs that rely on local points in lower-resolution feature maps to generate high-resolution details, SAGAN is intrigued by the potential information that can be extracted from a combination of features across various positions.

2.2.2 Image super resolution

The term "super resolution" encompasses a range of techniques used for upscaling videos and images. These techniques involve training models with real image data and using it to generate high-resolution images from lower-resolution ones (Wang et al., 2019). Wang et al. (2018) discovered that by combining three key aspects of SRGAN - the structural network design, adversarial and perceptual loss - they could enhance the visual effectiveness of SRGAN, resulting in an improved version called ESRGAN. The primary building block used in constructing networks without batch normalization was the residual dense block (RRDB). They also modified the relativistic GAN principle to enable the discriminator to predict relative realness instead of absolute values. Additionally, they intensified the perception loss by activating functionality before texture recovery and ensuring brightness consistency, which led to improved texture reformation and consistency monitoring. The suggested ESRGAN demonstrates consistent visual quality with more practical and realistic textures compared to SRGAN, and it secured first place in the PIRM 2018-SR Challenge with the highest perceptual index.

2.2.3 image inpainting

Visual inpainting is a technique used to reconstruct missing sections of image data in a way that conceals the fact that they have been restored. It is commonly employed to remove unwanted artifacts from images or to restore deteriorated areas in historical or artifact pictures. Nazeri et al. (2019) introduced Edge Connect, a two-stage adversarial framework consisting of an image completion network and edge generators. The edge generators generate both regular and irregular edge hallucinations, while the image completion network utilizes these hallucinated edges as guidance to fill in the missing regions.

2.2.4 Generation of anime character

The development and design of games and animations can be expensive due to the need for a large number of artists to perform repetitive tasks. To address this, Jin et al. (2017) introduced Automated Anime Characters, which utilizes a GAN to generate and color anime characters. The model consists of a generator and a discriminator system with various layers, batch normalization, ReLU activation, and skip connections.

In a similar vein, Chen et al. (2018) presented a novel approach for transforming real-world image graphs into cartoon-style visuals. Their proposed method, CartoonGAN, employs a generative adversarial network (GAN) specifically designed for generating cartoon-like images. This advancement holds promise for applications in computer vision and graphics.

2.2.5 Text to image transformation:

The task of synthesizing text into meaningful images remains a challenge despite advancements in performance. Existing techniques generate a basic outline of the image but fail to capture the true essence of the text. Fedus et al. (2018) proposed the concept of sample accuracy. By employing GANs, which are adversarial networks known for their ability to generate high-quality models, they addressed the issue of text-to-image synthesis. Their approach, called Actor Critic Conditional GAN (CGAN), bridged the gap in capturing the intended meaning. Through qualitative and quantitative analysis, they demonstrated that their method produces more natural and faithful images compared to previous models.

Chapter 3

Requirements Specifications

3.1 Requirement Specifications:

Our system enables users to provide input to the model using images.. The GAN analyzes image and gives output. This output will be super resolute enhance image. The backbone of our model is a Generative Adversarial Network that is trained on a calebA dataset containing 20000+ images.

3.1.1 Purpose

The most daring task of producing an image which have high pixels/resolutions from its counterpart image having low resolutions which is mentioned to as super resolution. From within the computer to understand and interpret the visual world super resolution received considerable attention and has a large range of applications.

3.1.2 Intended Audience

This project work is for Developers, Testers, Engineers, Project Managers

3.1.3 project scope

Converting a LR into HR image is a difficult task, but however it is possible with the use of image processing and deep learning algorithms

3.2 Overall Description

3.2.1 Overview

The function of measuring the image of high resolution (HR) creates the equivalent image of low resolution (LR). Recent work has focused on reducing the risk of square reconstruction. Emerging ratings have very high signal-to-noise ratios (PSNR), but they usually do not have high frequency data and are apparently unsatisfactory in a way that they do not match the reliability at a high resolution which is expected.

The performance correction for these SR-monitored algorithms typically mean square error (MSE) is minimize between the obtained HR image and ground reality. To this end, we are proposing a highly competitive opposition network (SRGAN).

The artistic nature of computer to understand and interpret the visual world problems is currently design by the specially sketch the architecture of CNNs. Taking pictures, assigning value (readable and non-discriminatory metrics) to the various elements in the picture is done by the CNN and be able to distinguish one from the other.

3.2.2 Machine Learning vs Deep Learning

Machine learning undergoes artificial intelligence. Model is trained in such a way that the outputs given by the machine is similar to the one with the provided data set. The machine learning model gives the output with greater accuracy. But after multiple iterations the model gives same output for same input. In Deep learning, the model is trained in such a way that after every iteration, the model trains itself and gives more accurate performance. In deep learning the model learns and trains after every input given to it. In order to enhance the ability of machine, we train it by some sample data. Machine learns the data and then predicts the new data. After some iterations the system is trained to give accurate outputs. Basic difference between both of these is that deep learning models train after every iteration. Deep learning is heavily inspired from human brain as human brain works with the help of neurons. Same as deep learning works on complex ANN.

3.2.3 Product Function

Our project aims to achieve Super-Resolution from a low-resolution counterpart. The main function is to estimate high quality image while retaining well-designed details Performance management of SR-controlled algorithms often reduce the definition.

In this work we propose a highly critical opposition network (SRGAN) in which we use a hidden residual network by disassembling connections and variations on the

MSE as a single purpose. As seen from work done before this concept, we describe the loss of understanding of the novel using high-level VGG network maps combined with prejudice that promotes solutions with complex understanding that are difficult to distinguish from HR reference images.

3.2.4 Operating Environment

It is necessary to think about the operating setting of the improved product. By doing this the event team and user will handle the merchandise in an exceedingly positive manner. a number of the wants for operational our system area unit as follows:

Operating system

- Minimum window 7 version

Hardware specifications:

- Processor minimum Core i5 (4th or 5th gen)
- Ram minimum 6 GB
- SSD

Framework and libraries

- Tensorflow
- keras
- Matplotlib
- Numpy
- pandas
- Open CV

Notebook used

- Kaggle console
- Google collab

IDE

- PyCharm

Category	Tools
System	lenovo core i5,5 gen with 8 GB RAM and graphic card (intel HD 500)
Language	python

Table 3.1: Design and implementation

3.3 Design and implementation

3.4 Assumptions and Dependencies

Complete structure of SRGAN. Next, Feature Map Input is a remnant of a non-straightforward map model. The image is then reconstructed with a sample layer and convolutional layer. Moreover, network releases restructuring effect.. And then we include non-realistic and realistic Images with high resolution in a separate discriminatory network, which is responsible for discriminating image authenticity.

We are using different layers from KERAS library to achieve our task. The main concern or which we assume is the computational power can interrupt in our work so will have to make sure it gets a high computational hardware such as a GPU.

Our algorithm performance depends on adversarial network, which includes discriminator and pre-trained VGG19 network, which we will be importing form Keras built in library.

3.5 System Features

3.5.1 Super Resolution

Currently, there are possibly two ways to enhance the image quality by editing particularly. The first is to improve hardware devices which includes image sensors and light but upgrading of hardware by this method is very difficult to promote in operating systems and expensive. Other Images Super-Resolution Reconstruction (ISRR) technology that integrates images having low resolutions and reproduces images with high resolution using digital image processing technology and machine learning algorithms.

Our program is designed to achieve Super-Resolution (SR) from a image with low resolutions counterpart, using the Generative Adversarial Network (GAN).

3.5.2 Finer Texture Details

By this framework we inferred photo realistic natural image for 4x up scaling factor. We can obtain finer texture details of image after passing through this network.

3.6 Operational Requirements

Functional requirements define the specific function of the system to be performed by the system. It explains functions of a system and its components or its components. Operational requirements are supported by non-functional requirements, which place constraints on construction or use.

3.6.1 Generator and Discriminator Network

These two networks are the basis of the Generative Adversarial Network. Next, Feature Map Input is a remnant of a non-straightforward map model. The image is then reconstructed with a basic layer and a convolutional layer. Next, the network releases the restructuring effect. Finally, we include non-realistic and realistic HR images in a separate discriminatory network, which is accountable for discriminating image originality .

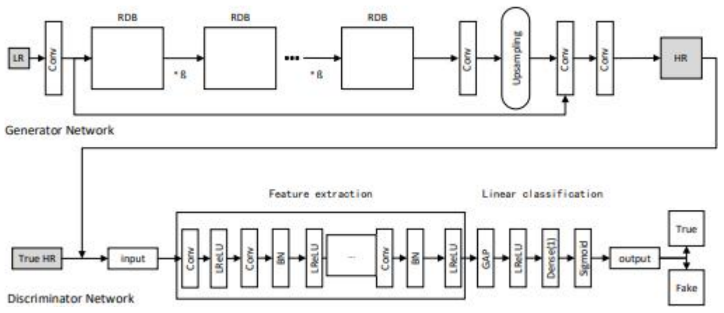


Figure 3.1: Generator and discriminator Network

3.6.2 Utility Functions

These functions are used to plot, load test data, downscale or upscale images, calculate losses and accuracy, and display progress.

3.6.3 Non-Functional Requirements

Nonfunctional requirement are requirements which shows the quality attribute of the system.

3.6.3.1 Honesty

The ability to maintain a defined performance level is what loyalty means.

3.6.3.2 Retention

Maintenance is one of the most common changes made after the program is completed. As time changes, so do needs.

3.6.3.3 Depression

Power is translated into various defined areas without the use of actions or means other than those devoted to this purpose in the product.

3.6.4 Functional Requirements

3.6.4.1 Accuracy

Good accuracy is important in our system. This is because it is meant to reduce manual work needed. It should not give too much false predictions because this can result in increased manual work. The accuracy of the system should meet certain threshold level.

3.6.4.2 Speed and efficiency

The speed of the system is important because it effects the number of images that can be processed in a certain amount of time. The system should be optimized enough to run on modest hardware as well.

3.6.4.3 Unauthorized person should not be able to use the system

The system should be secure and only available to authorized people. People who are not authorized should not be able to use the system due to the nature of its application.

3.7 Use case Model

The use case model encapsulates the necessities of the system. Usage cases square measure some way of human activity with users and alternative shareholders concerning what the system is meant to try to. The usage case diagram shows the interaction between the system and external businesses. These external structures square measure spoken as actors. Actors represent roles that will embody human users, external hardware, or alternative programs. the employment case is one unit of purposeful work. It offers a high-level read of behavior that’s visible to somebody or one thing outside the system.

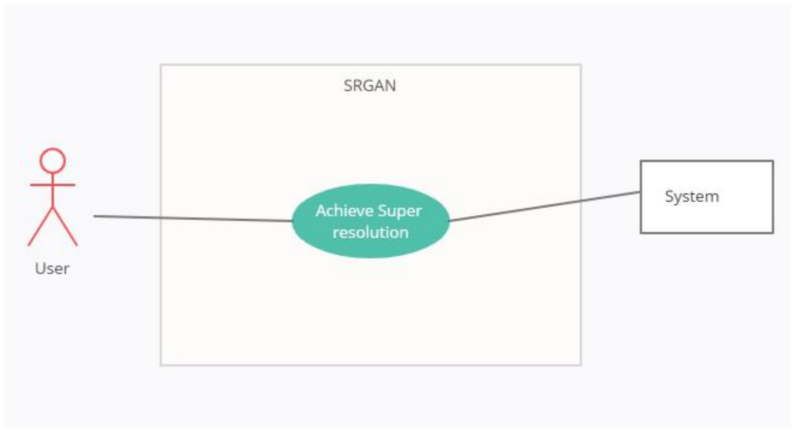
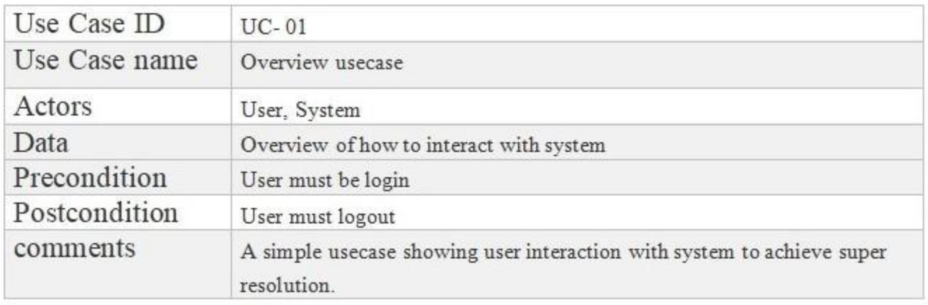


Figure 3.2: Overview usecase diagram

Use case description



Use Case ID	UC- 01
Use Case name	Overview usecase
Actors	User, System
Data	Overview of how to interact with system
Precondition	User must be login
Postcondition	User must logout
comments	A simple usecase showing user interaction with system to achieve super resolution.

Figure 3.3: overview usecase description

In below use case diagram, it is described how this application will work and highlights the interaction between the user and the application to generate high resolution image using GANs.

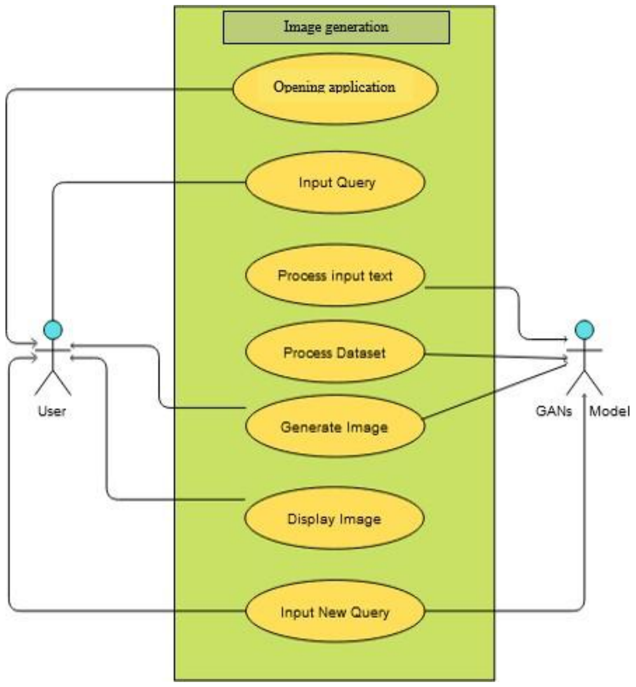
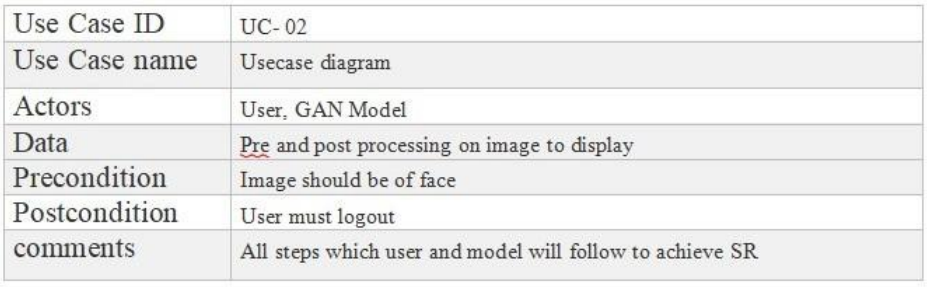


Figure 3.4: use case diagram

Use case description



Use Case ID	UC- 02
Use Case name	Usecase diagram
Actors	User, GAN Model
Data	Pre and post processing on image to display
Precondition	Image should be of face
Postcondition	User must logout
comments	All steps which user and model will follow to achieve SR

Figure 3.5: usecase diagram description

Chapter 4

System Design

In this chapter of System Design, which is all about defining component, interfaces, and data to reach to our specified requirements, we have a deeper look into the development phases of our FACE-GAN. This chapter will discuss the system architecture and provide details of the design methodology, constraints, models, interaction between the system and the user.

4.1 System Architecture

This project is basically based on artificial intelligence. Allah has blessed us with natural intelligent. Brain is main processing unit of human body. All calculations are done by brain and all work which includes intelligence are processed by the brain. So, machine trained by human being. Artificial based on computer vision.

Artificial intelligence (AI) is that the intellect displayed by machines, in distinction to the natural intelligence displayed by humans and animals, which has data and sympathy. The distinction between the previous paragraphs and therefore the last paragraphs is typically indicated by the chosen outline. ’Strong’ AI is usually noted as AGI whereas tries to mimic ’natural’ intelligence are referred to as ABI. AI’s leading manuals describe the sector as a study of "intelligent agents": any device that acknowledges its atmosphere and takes steps that increase its possibilities of with success achieving its goals. In general, the term "artificial intelligence" is usually accustomed describe intelligent devices that mimic "cognitive" tasks that folks accompany the human mind, like "problem solving" and "learning”.

4.2 Machine Learning

Machine learning relies on algorithms that square measure designed on completely different platforms like Python and MATLAB. It's an area under artificial intelligence.

TMachine learning algorithms have revolutionized the capabilities of AI, enabling it to surpass its predefined tasks. In the pre-mainstream era, AI programs were primarily utilized for fundamental functions in business settings, such as intelligent automation or rule-based decision making. These AI algorithms were confined to the specific domains they were designed for. However, with the introduction of machine learning, computers have transcended these limitations and have started to evolve with each iteration, expanding their capabilities beyond initial expectations. As mentioned, machine learning algorithms have the potential to boost themselves through coaching. Today, cc algorithms square measure trained victimization 3 completely different ways. Here square measure 3 kinds of machine learning: supervised reading, supervised reading, and advanced reading.

4.2.1 Types of Machine learning

Machine learning has usually comprises of three types.

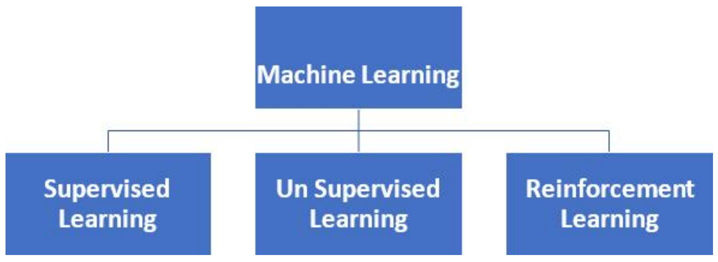


Figure 4.1: Machine Learning Types

4.2.2 Reinforcement learning

Consolidation takes inspiration from however individuals learn from the information in their lives. It introduces a self-reformation algorithmic rule and learns new things victimization trial and error methodology. Positive outcomes are inspired, or'strengthened', and negative outcomes are discouraged or 'punished'.

4.2.3 Deep Learning

Machine learning has a advanced classify as Deep learning that uses algorithms impressed by the structure and brain known as artificial neural networks. It makes computation of multilayers neural networks. It will handle high dimensional knowledge. Machine learning algorithms train deep neural networks to attain higher accuracy. Deep learning mimics the way as our brain performs i.e., it learns from expertise.

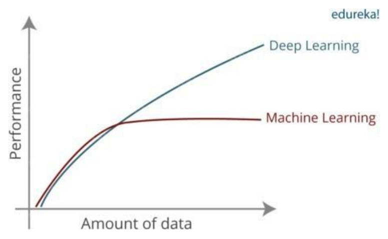


Figure 4.2: Performance Graph

4.2.4 Perceptron and Artificial Neural Networks

Perceptron and ANN studies the essential unit of brain referred to as a somatic cell or a baryon. Artificial somatic cell or a perceptron could also be a linear model used binary classification. Modeling of neurons that has sets of inputs. a specific weight is given to each input. The somatic cell calculates various functions on these inputs having some weight and offers the output.

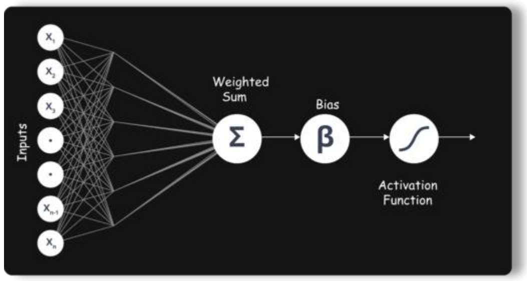


Figure 4.3: Artificial Neural Network Structure

It receives n inputs; inputs applies a transformation are then summed up and produce an output. The human being brain is made up of neuron just like that multiple perceptron build artificial neural network (ANN).

It has three important layers:

4.2.4.1 Input layer

It accepts input

4.2.4.2 Hidden layer

It is between input layer and output layer. It performs computations

4.2.4.3 Output layer

In the hidden layer of neural network, the inputs pass through the different functions. At last, obtained output is delivered. The weight shows the usefulness of a particular input, input implies a more impact on neural networks if has more weight.

4.2.4.4 Bias:

An extra parameter withinside the perceptron that is utilized to modify the output in conjunction with weighted sum of the enter to the neuron which the version in a manner that it may in shape pleasant for the given data. Transfer characteristic is giver below F(x) = w.x+b – bias

w : weighted vector. x : input vector.

4.2.5 Activation Function:

It maps the inputs into the outputs. To produce the output threshold is used by these functions.

- Identify or Linear
- Binary Step or Unit
- Logistic or sigmoid
- Tanh Function
- ReLU
- SoftMax

4.3 Generative Adverserial Networks

In generative adversarial networks there are two networks namely as generator network and discriminative networks. They work against each other that's why it is called adversarial networks. In both networks 16 residual blocks are connected in series with each other. The data is processed through these blocks of generator networks and SR image is obtained. After that this SR image with HR image fed into discriminative network and processed through the blocks and probability will be checked which is in between 0 and 1.

4.4 Design Constraints

In this project different kind of limitations are faced from staring of its implementations. They will be discussed below that what problems occurs and how these problems are solved by critically analyzing all aspects being studied in whole engineering. In life we often face hurdles in the way of progress. Now Designs constraints are now discussed.

4.5 Personal computer

Firstly, the thing which hinders is memory issue in image processing. The image is not fitting on the VRAM of GPU Secondly, for training the neural networks high processing is needed for training. Our main purpose is to train the networks, but this can be done in short time if and only on pc that has comparatively high processing power.

4.6 Errors in code

Programming the algorithm of neural networks, sometime complexity faced because of different errors. Errors occur in pre-processing the data and some occur due to language syntax because we have to follow the syntax of Python.

4.6.1 Errors while training networks

Firstly, we load the dataset in python language for processing the data and then networks algorithms are designed. So, after designing the networks we face some hinders in training of generator and discriminative network. While training code get to out of memory, and we do image reconfiguration again and again

4.7 Design Methodology

In designing of neural networks in generative adversarial network following methodology is applied. In this how the project will implement is described.

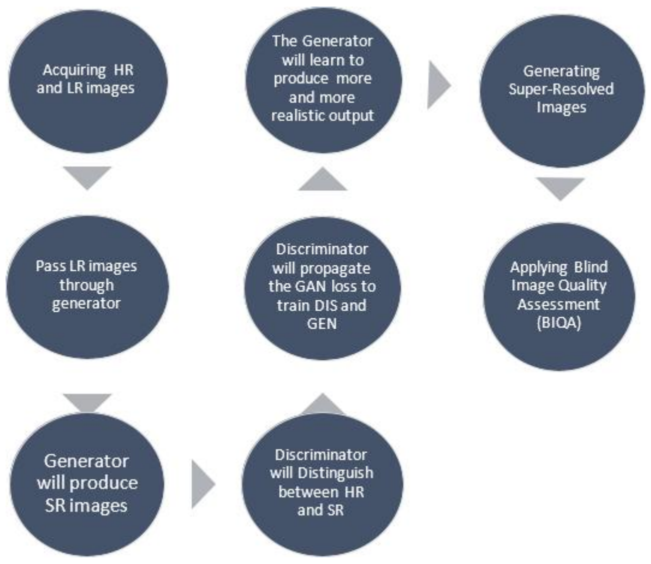

Acquiring HR and LR images
Pass LR images through generator
Generator will produce SR images
Discriminator will Distinguish between HR and SR
Discriminator will propagate the GAN loss to train DIS and GEN
The Generator will learn to produce more and more realistic output
Generating Super-Resolved Images
Applying Blind Image Quality Assessment (BIQA)

Figure 4.4: Performance Graph

4.7.1 Dataset Acquiring

In this project the dataset is consist of HR images as well as LR images. We downloaded the Caleba dataset comprises of facial images.. Dataset contains 202,599 number of face images of various celebrities, 10,177 unique identities and 40 binary attribute annotations per image.

We acquire the dataset from below directory from where it has been downloaded: "https://www.kaggle.com/datasets/jessicali9530/celeba-dataset"

4.7.2 Data Preprocessing

After acquiring the dataset, the next step is to be preprocessing of data. In this we import the different libraries like KERAS etc. Dataset is imported from directory in python by which we train our networks. In this phase missing data also be handled. Also, the dataset is split into two parts, one for testing and other for validation.

4.7.3 Generator Network

After above two explained processes, we come in a stage that now generator network is to be build. For making generator network we import activation functions like PReLU, Batch Normalization and sigmoid. After importing these functions, we give kernel size, strides for performing convolution. We do 2D convolution in generator network along with up sampling.

4.7.4 Discriminator Network

After generator network, the next turn is of discriminative network. It will discriminate between original image and fake image. It calculates the probability of two images that is HR image and SR image, these both images fed into discriminative network

4.7.5 ResNet

ResNet, short for Residual Network, is a neural network architecture specifically designed to tackle the issue of vanishing gradients in deep neural networks. It overcomes this challenge by incorporating shortcut connections between layers, enabling the gradient to flow directly from the input of one layer to the output of a subsequent layer, circumventing the intermediate layers. This effectively mitigates the problem of vanishing gradients and enhances the network's learning capability.

Comprising multiple residual blocks, each block consists of two convolutional layers with a shortcut connection linking them. The input to the block undergoes convolutional operations in the first layer, followed by the output of that layer passing through a second convolutional layer. The output of the second layer is then added to the input of the block, and the result is subjected to an activation function. This structure empowers ResNet to effectively capture and propagate information through the network, facilitating more efficient learning.

4.7.6 Training

Training of a neural network involves using a training database to update the model instruments to create a good map of the output input.

This training process is solved using an optimization algorithm that searches the space for potential weights of the neural network model of a set of weights that lead to optimal performance in the training database.

4.8 High Level Design

System architecture of project is based on generative adversarial network and then super resolution using GAN. In this ResNet **pre trained** network will also be used to get better results.

4.8.1 Generative Adverserial Network

It is primarily based totally on deep learning generative model that is used for unsupervised learning. It is a network where two networks are competing to produce contrast in the data. It has generator network and discriminator network.Generative adversarial networks (GOODFELLOW et al., 2014) framework learning production models based on the concept of the game. . Proper Background training a G (z; (G)) producing network that produces almost identical data samples distribution, p data (x), by converting the sound carrier z as x = G (z; (G)). Training the G signal is provided by a discriminating D (x) network trained to distinguish samples from the productive pG distribution from real data (from pdata distribution). A network that produces and trains in such a way that the discriminator accepts the consequences as that literally. The G and D networks are counter-intuitive using anticipated loss.

$V\ (G,D) = Ey_p data(y)[log(D(y))] + Ez_p Z(z)[log(1D(G(z))],$

Where z is a random encoding on the latent space and y is the sample from the pdata distribution. Then the optimal results for G and D will be achieved after doing

minG maxD V (G,D)

GANs, there may be a Generator network that takes a pattern and generates a pattern of information, and after this, the Discriminator network makes a decision whether or not the information is generated or taken from the actual pattern, the usage of a binary Classification problem with the assist of a sigmoid feature that offers the output within side the variety zero to 1.

4.8.2 Super Resolution Generative Adversarial Network

In this project super resolution of a single image is achieved by using GAN [5]. By this method, a low-resolution image can be super resolved up to 4x. for example if an image has pixels of 64x64, so after the passing through SRGAN [7] we reproduce an image of 256x265 pixels. We feed the 64x64 pixel image into the generator network and it regenerate the super-resolved image of 256x256 pixels. This is the great achievement by using deep learning.

4.8.3 SRGAN Architecture

Figure shows the generator construction and discrimination.. K9n64s1 means a size 9 kernel, 64 channels and step 1. Remaining blocks are used in apartheid. Two new concepts used in network design are P ReLU and Pixel Shuffler.

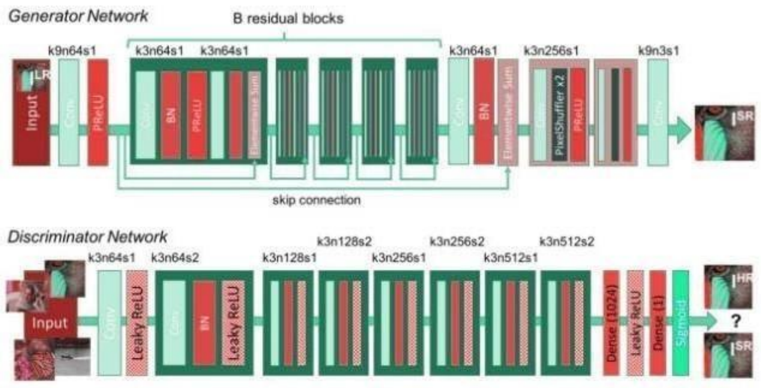


Figure 4.5: Architecture of generator and discriminator network

Both Network architectures consist of convolution layer, Parametric Rectified Linear Unit (P ReLU) layer, batch normalization and pixel shuffler. B Residual blocks consist of combination of convolution, batch normalization and PReLU. These all are types of activation function which are being used in training of neural networks. There is used a term called kernel size, channels, and steps. These define size of filter for doing the convolution with how many steps. In fig 4.5 different blocks are shown that performs the operations to obtain the super-resolved image and analyze whether the generated image is fake or real.

4.8.4 Activation Function

Parametric Rectified Linear Unit (P ReLU) Function It is an activation function called Parametric Rectified Linear Unit. From its name we came to know that it is a linear function applied in deep learning algorithm. It has slop of negative values. It is successful innovation in deep learning convolution neural networks. In a linear graph, the values with negative slope are converted into positive values. It multiplies the negation value with smallest value like 0.02 to make it positive. Like ReLU it does not eliminate the negative value in the data.

The equations for P ReLU function are given below:

$f(yi) = yi$; if $yi > 1$ $f(yi) = ai . yi$; if $yi < 1$

For back propagation, its gradient is:

$df(yi)\ 0$, if $yi>0$

dai= yi ifyi<=0

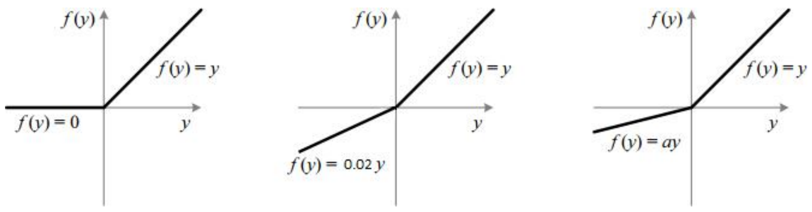


Figure 4.6: Relu,leaky Relu and PReLU Comparison

4.8.5 Batch Normalization Function

Batch normalization is a function in deep learning that normalize the values for training the data. It is batch normalization, so it normalizes each layers of input data. Training deep neural networks through dozens of layers is challenging as it can be sensitive to random initial weights and algorithm adjustment algorithm. It makes neural networks to work faster. By this technique we can save our time in training the dataset. We import batch normalization from KERAS library.

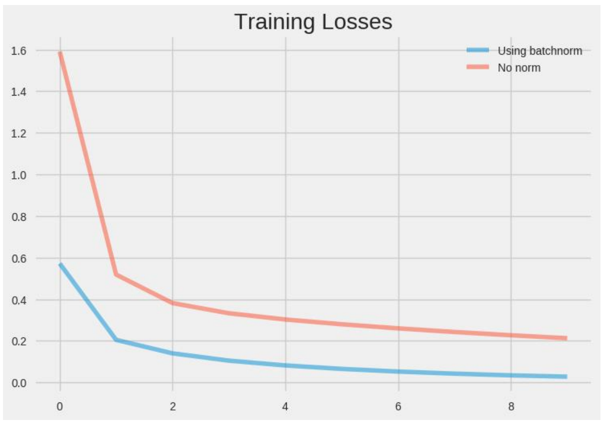


Figure 4.7: Batch Normalization vs No normalization

4.8.5.1 Leaky Rectified Linear Unit Function

It is a leaky model of ReLU. It is utilized in system gaining knowledge of for plotting the date as authentic or false. It plots the output in among the ‘0’ and ‘1’. It is a try and clear up the problem of demise ReLU Leaks assist growth the variety of ReLU work. Usually, the price of a is 0.01, however. When non is 0.01 then its miles known as Randomized ReLU. So, the scope of Leaky ReLU is (-infinity to infinity). Both profitable jobs and Randomized ReLU paintings monotonic naturally. Also, what comes out of them is also one nature.

4.8.5.2 Pixel Shuffer

Pixel Shuffling rearranges the shape (N, C, H, W) into (N, C / r * r, H * r, W * r) where the r is the push element. It basically changes the depth (channel) of space (height and width). In Generator, pixel shuffling is used to increase image size.System implementation in this project is program the algorithms of neural networks in different platform like Python and MATLAB etc. In this chapter we will discuss that how we implement the whole system. Like what technologies are adopted, what kind of platform is chosen to develop the algorithms of different networks. All these things are explained which involve in the process of developing a well-known system which is now in the field of electrical engineering. We will also discuss the system architecture thoroughly.

4.8.6 Sequence Diagram of Proposed System

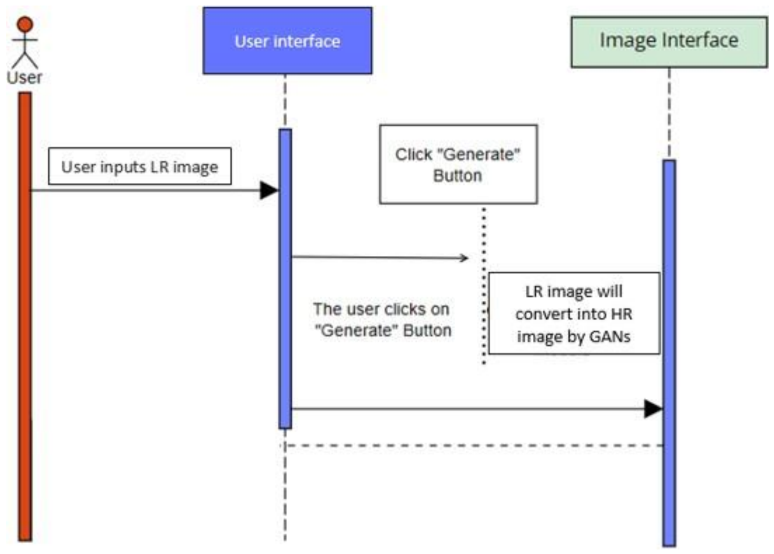

User
User interface
Image Interface
User inputs LR image
Click "Generate" Button
The user clicks on "Generate" Button
LR image will convert into HR image by GANs

Figure 4.8: Sequence diagram of proposed system

Chapter 5

System Implementation

5.1 Introduction

In system architecture we define the whole structure of super resolution generative adversarial network. Usually, GAN consist of generator network and discriminative network as discussed in previous chapters.

Sr no.	Component	Type
1	Batch Normalization	Normalization
2	Dense Connections	Feed Forward Network
3	Leaky ReLU	Activation Function
4	Pixel Shuffle	Miscellaneous Components
5	PReLU	Activation Function
6	Residual Connections	Skip Connection
7	Sigmoid Activation	Activation Function
8	SRGAN Residual Blocks	Skip Connection Loss
9	VGG Loss	Loss Functions

Table 5.1: Components of SRGAN

5.1.1 Batch Normalization

Batch Normalization is a technique designed to mitigate the impact of internal covariate shifts and facilitate faster training of deep neural networks. It accomplishes this by introducing a normalization step that adapts the statistics and characteristics of the layer input. On gradient flow through the network, Performing batch normalization implies a positive effect by decreasing the trust of the slopes on the parameter scale or its initial values. This allows for the use of very high levels of learning without the threat of variability. In addition, batch making tends to model and reduces the need for Dropout.

During training In our generative adversarial networks (GANs), training happening in both the generator and the discriminator. The generator is trained to produce data that is similar to the real data, while the discriminator is trained to distinguish between the real data and the generated data. During training, the generator tries to fool the discriminator by producing data that is indistinguishable from the real data, while the discriminator tries to correctly classify the data as real or fake. This process continues iteratively until the generator produces data that is similar enough to the real data to fool the discriminator.

5.1.2 Dense connection

A dense layer with 1 unit and the activation function ’sigmoid’ is applied to the output of the last LeakyReLU layer using the Dense() function. This output represents the predicted probability of the input image being real or fake (0 for fake and 1 for real) and is called the "real vs fake patch".

In summary, The code utilizes a sequence of tailored dense blocks, accompanied by a fully connected layer and a concluding dense layer, to determine whether a high-quality input image is real or fake in the context of a GAN.

5.1.3 Pixel shuffle

Pixel Shuffle is a function used in super-resolution models to use active sub-pixel convolutions with stride. Specifically, it rearranges objects in shape tensor to shape tensor. (*,Cxr2,H,W)

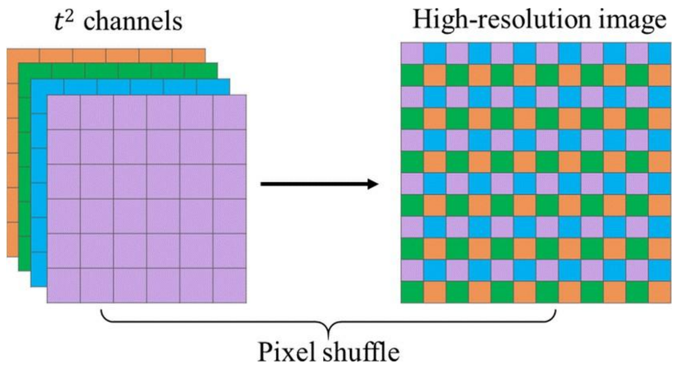


Figure 5.1: Pixel shuffle

5.1.4 Generator Architecture

Generator structure consists of a residual network in place of deep convolution networks due to the fact the final networks are clean to educate and permit them to be very deep to provide higher results. This is due to the fact the residual network makes use of a form of connection referred to as skip connections

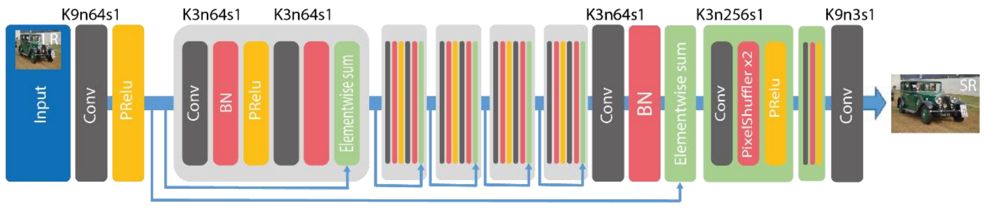


Figure 5.2: Generator Architecture(https://images.app.goo.gl/s2CFixE5hcN1EwDP7)

B (16) blocks left which are created by residual network. Inside the remaining block, two convolutional layers are used, with 64 feature and standard batch layers follows the 3 × 3 small heads maps and as the activation function we use the Parametric ReLU

Adjustment of the input image is enhanced by two trained layers of sub-pixel convolution.

5.1.5 VGG loss

VGG Loss is a type of content loss used in the SRGAN. Another way to lose a smart pixel; IVGG Loss is trying to get closer to understanding the similarities. For 19 pre-training VGG network VGG oss is based on ReLU activation layers. By presenting the feature map obtained by the solution (after operation) before the max pooling wire inside the VGG19 network, which we think is provided. We then describe the loss of VGG as the Euclidean distance between the presentation of the features of the reconstructed image and the reference image.

Instead of using a fixed amount of rectifier (alpha) parameter such as Leaky ReLU the generator structure uses the parametric ReLU as an activation function. Effectively reads editor modifiers and improves accuracy with additional calculated costs that are ignored

5.1.6 Discriminartor Architecture

The function of discrimination is to distinguish between SR images and real HR images. It uses leaky ReLU as function. The network consists of eight specification layers with 3 × 3 filter elements, which increase in size from 2 to 64 and then 64 to 512 kernels. Image correction is reduced by using the stable convolutions it doubled the number of elements every time. Two dense layers follow the appeared 512 feature maps and a leaky ReLU used during the final sigmoid opening function to determine the chance of sample separation.

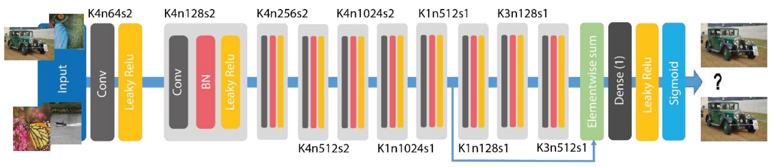
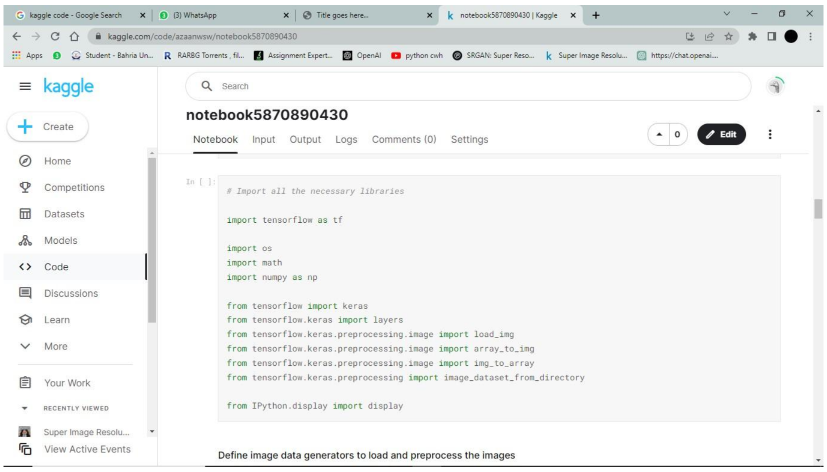


Figure 5.3: Discriminator Architecture(https://images.app.goo.gl/ceUvGpFgE7ByFFFt7)

5.2 Tools and Technology used

For programing the logarithms of neural networks in machine learning computer vision technology is used. Personal computer is used for programming. In this project personal we use kaggle code because of its powerful dual GPUs, keep in mind that kaggle code is online notebook to run your code using their cloud based resources. Because for the training of networks we need high processing power for this purpose so we used kaggle code.



notebook5870890430

```
# Import all the necessary libraries

import tensorflow as tf

import os
import math
import numpy as np

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import array_to_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing import image_dataset_from_directory

from IPython.display import display
```

Define image data generators to load and preprocess the images

Figure 5.4: Coding on kaggle code

5.2.1 RAM

In this system 8 GB ram is installed in the group of 4Gb + 4GB.It is 1600MT/s DDR3L Non ECC RAM Memory.

5.3 5.3. Development Environment/Language Used

The code is written in Python using TensorFlow and Keras libraries as a development environment. The code uses the Tensor Flow library to load and preprocess image data from a directory, and then maps a scaling function to the data to convert pixel values from the range (0, 255) to (0, 1). The code also defines functions to process the input and target images, and uses a convolutional neural network architecture to train the model.

Numpy is also used in this project because NumPy brings the computational power of languages like C and Fortran to Python, a language much easier to learn and use using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted and they both allow the user to write fast programs as long as most operations work on arrays .

Developing environment of project is like to demonstrate an application of deep learning in image processing that can be useful for tasks such as upscaling low-resolution images for better visualization or analysis.

5.3.1 Python Language

Python language is used due to its compatibility in developing the neural networks. Because for the machine leaning Python language is easier, it has built in libraries and directories.

Python is an interpreted, fine-quality, and common language of programming. The readability of the code is emphasized by the Python architecture philosophy and its amazing application of critical guidance. Its language-building and focused method intends to support editors with writing clear, logical code for small and large projects.

Python typed hard and collected garbage. It supports many planning paradigms, including programming (especially, process), object orientation and good performance. Python is often described as a "battery-powered" language because of its common library.

Python provides short and readable code. While high-tech and versatile functionality works after artificial intelligence AI and machine learning ML, simplicity of the Python allows developers to write consistent programs. The developers found that they put all their efforts into solving the ML problem instead of focusing on the nuances of language technology.

Furthermore, Python is popular with many developers as it is easy to read. Codes of Python are understandable to humans, making it easy to construct machine learning models.

Vast Selection of Frameworks Libraries Using of AI and ML algorithm requires a lot of time and can be tricky. for Finding the best coding solutions for engineers, it is essential to have a well-coordinated and well-tested environment The development time can be reduced, if the programmers turned to many Python structures and libraries then the development time can be reduced. A software library is a pre-written code used by an engineer to solve a common task of programming. Python, with its own state-of-the-art stack of technology, has a comprehensive set of implants and machine learning libraries. Here are some of them:

- Tensorflow
- Keras
- Numpy
- Matplotlib
- CV2
- math

Python allows easy and Powerful Implementation What makes Python one of the best machine learning options is that it is easy to use and powerful. Beginners or learners must be familiar with the language before they can use it for machine learning or artificial intelligence.

This is not the case with Python. If you know the Python language, you can continue to use it for machine learning due to the large number of libraries, resources, and tools available. It takes longer to write code and debug in Python than in Java or C++. Machine learning and artificial intelligence programmers generally prefer to spend time building algorithms and heuristics rather than fixing syntax errors in their code.

5.3.2 Google collab

Some of the early development work of project is also done on Google Colaboratory, popularly known as Colab, is a web IDE for python that was released by Google in 2017. Colab is an excellent tool for data scientists to execute Machine Learning and Deep Learning projects with cloud storage capabilities.

5.3.3 pycharm

Frontend is developed using flask on a localhost 5000 and i prefer pycharm for it because PyCharm is a powerful IDE specifically designed for Python development. It offers a comprehensive set of features, such as code completion, debugging tools, project management, version control integration, and more. These features make it easier to write, test, and debug **Flask** applications. PyCharm allows you to set up and configure a local development environment on your machine.PyCharm runs on your local machine, which means you don't need an internet connection to develop Flask applications.

5.4 Convolutional Layer

The Convolutional layer in CNN extracts the features from the images. The Convolution layer makes sure that the spatial relationship between pixels is intact. In this layer convolution is performed with a filter of size MxM. The filter slides over the image and dot product is taken between image and filter with respect to the size of the Convolution filter. The output from this layer is called feature map. This is then fed to other layers.

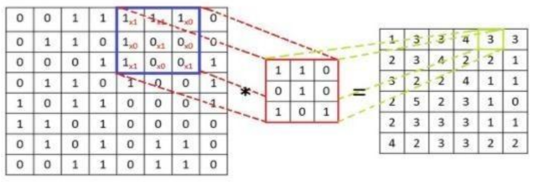


Figure 5.5: Convolution with kernel size of 3

5.4.1 Pooling layer

Pooling layer typically follows the convolution layer. . This layer is aimed at reducing the computational costs by decreasing the size of convolved feature map. This is done by decreasing the connections between layers. Convolution layer extracts the features while pooling layer summarizes them. It combines each group of the outputs of the previous layer into a single neuron. The Common variations of pooling operations: average pooling, max pooling and sum pooling. We have used max pooling in our CNN.Max pooling is used to help over-fitting by providing an abstracted form of the representation. It also reduces the number of parameters to learn thus reducing computational cost.

5.4.2 Max pooling

Takes the largest element in the feature map.

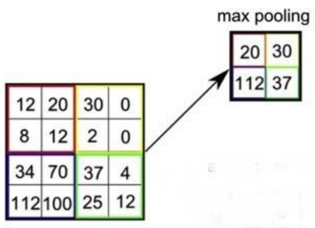


Figure 5.6: Max pooling

5.4.3 Average pooling

Takes the average element in the feature map.

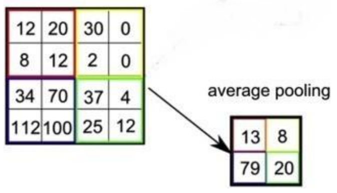


Figure 5.7: Average pooling

5.4.4 Sum pooling

The sum of all the elements in the feature map.

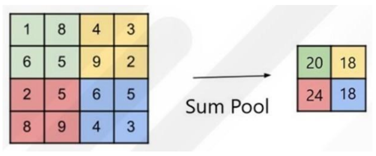


Figure 5.8: Sum pooling

5.4.5 Fully connected layer

The fully connected layer has weights,biases and the neurons. It is used to connect the neurons between two layers. The image is flattened and then fed to the Fully Connected layer.

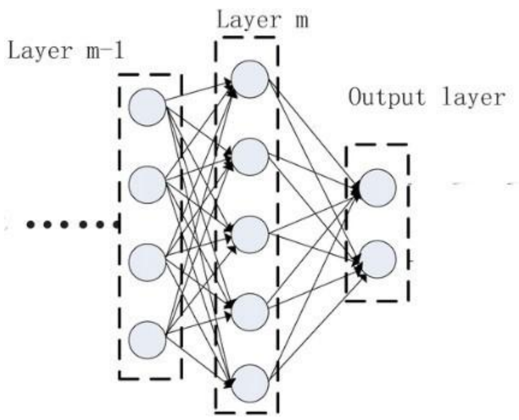


Figure 5.9: Fully connected layer

5.4.6 Dropout layer

In a dropout layer a few neurons are dropped during training of model. This is done because due to fully connected layers overfitting can occur. Dropout is a technique that operates by randomly deactivating connections between hidden units, setting their values to 0 during each update of the training phase..

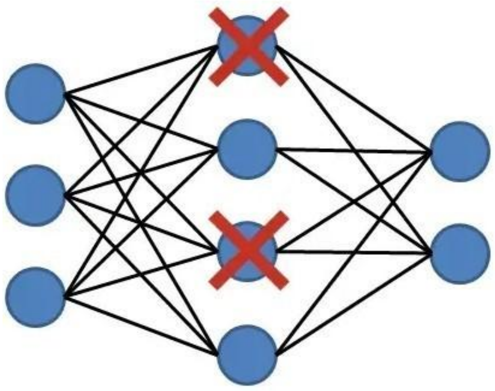


Figure 5.10: Dropout layer

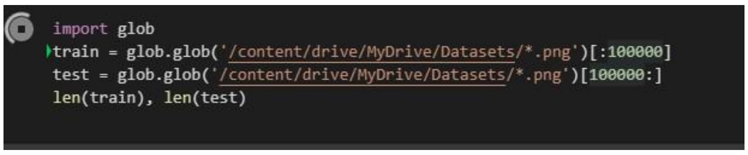
5.5 Processing logics/Algorithms

5.5.1 Dataset Acquisition

Dataset	
Name of Dataset	Caleba
year	Last updated on 2018
Number of face images	200000
Source	Kaggle

Table 5.2: Dataset

In this code, the data acquisition is being done using the ‘glob ‘ function provided by TensorFlow.



```
import glob
train = glob.glob('/content/drive/MyDrive/Datasets/*.png')[:100000]
test = glob.glob('/content/drive/MyDrive/Datasets/*.png')[100000:]
len(train), len(test)
```

Figure 5.11: Loading data into model

5.5.2 Generator Network

Generator network defines a neural network model for enhancing the quality of low-quality images. The model takes an input image of shape 64x64 pixels with 3 color channels. The output of the final residual block is added to the output of the first convolutional layer and passed through two more residual blocks with 256 filters.

Finally, the output of the last residual block is passed through a convolutional layer with 3 filters and a kernel size of 9x9. The output is then passed through a hyperbolic tangent activation function to produce the final high-quality image output.

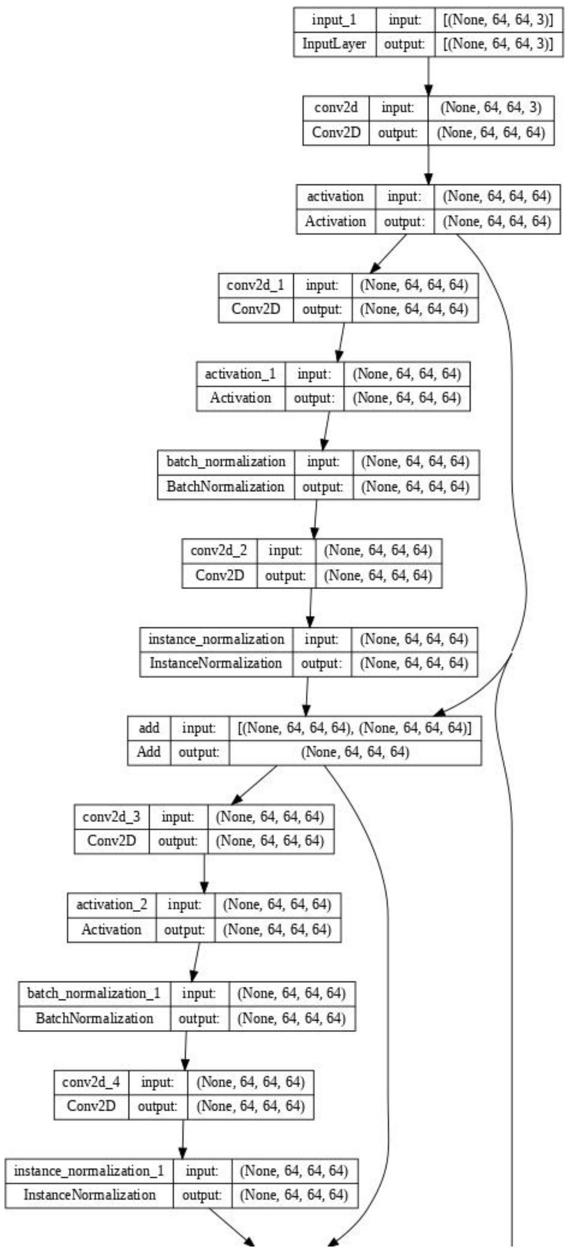

input_1 | input: | [(None, 64, 64, 3)]
InputLayer | output: | [(None, 64, 64, 3)]
conv2d | input: | (None, 64, 64, 3)
Conv2D | output: | (None, 64, 64, 64)
activation | input: | (None, 64, 64, 64)
Activation | output: | (None, 64, 64, 64)
conv2d_1 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
activation_1 | input: | (None, 64, 64, 64)
Activation | output: | (None, 64, 64, 64)
batch_normalization | input: | (None, 64, 64, 64)
BatchNormalization | output: | (None, 64, 64, 64)
conv2d_2 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
instance_normalization | input: | (None, 64, 64, 64)
InstanceNormalization | output: | (None, 64, 64, 64)
add | input: | [(None, 64, 64, 64), (None, 64, 64, 64)]
Add | output: | (None, 64, 64, 64)
conv2d_3 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
activation_2 | input: | (None, 64, 64, 64)
Activation | output: | (None, 64, 64, 64)
batch_normalization_1 | input: | (None, 64, 64, 64)
BatchNormalization | output: | (None, 64, 64, 64)
conv2d_4 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
instance_normalization_1 | input: | (None, 64, 64, 64)
InstanceNormalization | output: | (None, 64, 64, 64)

Figure 5.12: Plotting of Generator model(1)

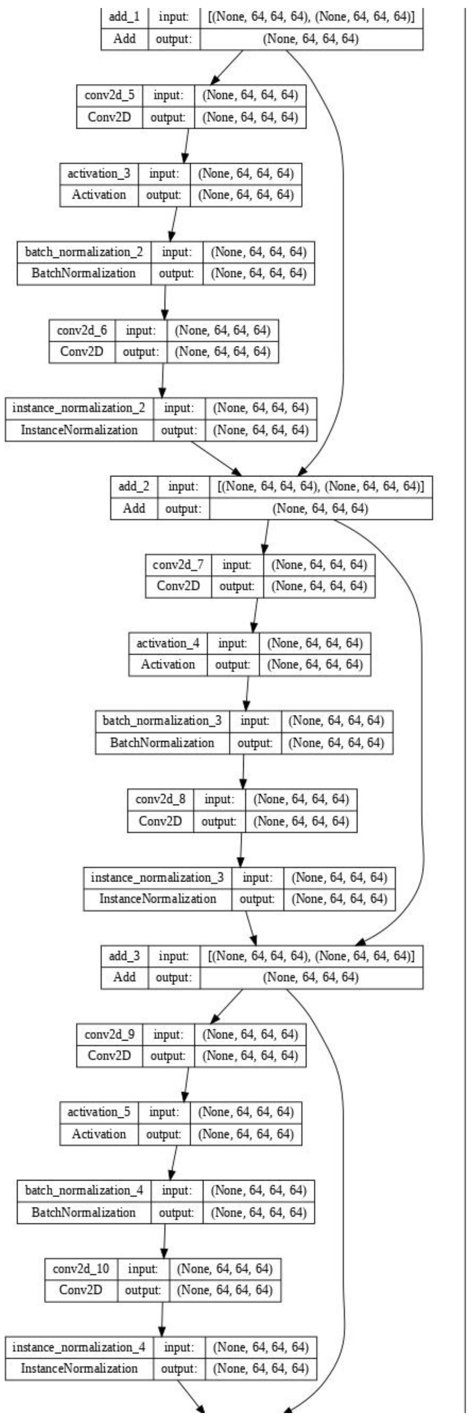

add_1 | input: | [(None, 64, 64, 64), (None, 64, 64, 64)]
Add | output: | (None, 64, 64, 64)
conv2d_5 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
activation_3 | input: | (None, 64, 64, 64)
Activation | output: | (None, 64, 64, 64)
batch_normalization_2 | input: | (None, 64, 64, 64)
BatchNormalization | output: | (None, 64, 64, 64)
conv2d_6 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
instance_normalization_2 | input: | (None, 64, 64, 64)
InstanceNormalization | output: | (None, 64, 64, 64)
add_2 | input: | [(None, 64, 64, 64), (None, 64, 64, 64)]
Add | output: | (None, 64, 64, 64)
conv2d_7 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
activation_4 | input: | (None, 64, 64, 64)
Activation | output: | (None, 64, 64, 64)
batch_normalization_3 | input: | (None, 64, 64, 64)
BatchNormalization | output: | (None, 64, 64, 64)
conv2d_8 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
instance_normalization_3 | input: | (None, 64, 64, 64)
InstanceNormalization | output: | (None, 64, 64, 64)
add_3 | input: | [(None, 64, 64, 64), (None, 64, 64, 64)]
Add | output: | (None, 64, 64, 64)
conv2d_9 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
activation_5 | input: | (None, 64, 64, 64)
Activation | output: | (None, 64, 64, 64)
batch_normalization_4 | input: | (None, 64, 64, 64)
BatchNormalization | output: | (None, 64, 64, 64)
conv2d_10 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
instance_normalization_4 | input: | (None, 64, 64, 64)
InstanceNormalization | output: | (None, 64, 64, 64)

Figure 5.13: Plotting of Generator model(2)

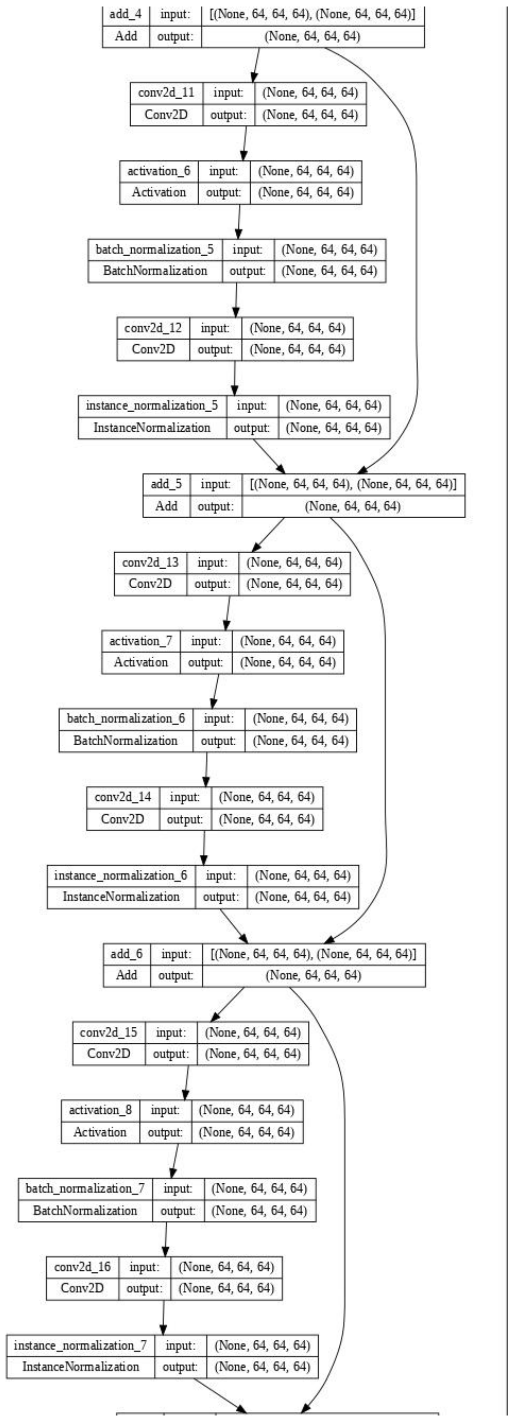

add_4 | input: | [(None, 64, 64, 64), (None, 64, 64, 64)]
Add | output: | (None, 64, 64, 64)
conv2d_11 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
activation_6 | input: | (None, 64, 64, 64)
Activation | output: | (None, 64, 64, 64)
batch_normalization_5 | input: | (None, 64, 64, 64)
BatchNormalization | output: | (None, 64, 64, 64)
conv2d_12 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
instance_normalization_5 | input: | (None, 64, 64, 64)
InstanceNormalization | output: | (None, 64, 64, 64)
add_5 | input: | [(None, 64, 64, 64), (None, 64, 64, 64)]
Add | output: | (None, 64, 64, 64)
conv2d_13 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
activation_7 | input: | (None, 64, 64, 64)
Activation | output: | (None, 64, 64, 64)
batch_normalization_6 | input: | (None, 64, 64, 64)
BatchNormalization | output: | (None, 64, 64, 64)
conv2d_14 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
instance_normalization_6 | input: | (None, 64, 64, 64)
InstanceNormalization | output: | (None, 64, 64, 64)
add_6 | input: | [(None, 64, 64, 64), (None, 64, 64, 64)]
Add | output: | (None, 64, 64, 64)
conv2d_15 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
activation_8 | input: | (None, 64, 64, 64)
Activation | output: | (None, 64, 64, 64)
batch_normalization_7 | input: | (None, 64, 64, 64)
BatchNormalization | output: | (None, 64, 64, 64)
conv2d_16 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
instance_normalization_7 | input: | (None, 64, 64, 64)
InstanceNormalization | output: | (None, 64, 64, 64)

Figure 5.14: Plotting of Generator model(3)

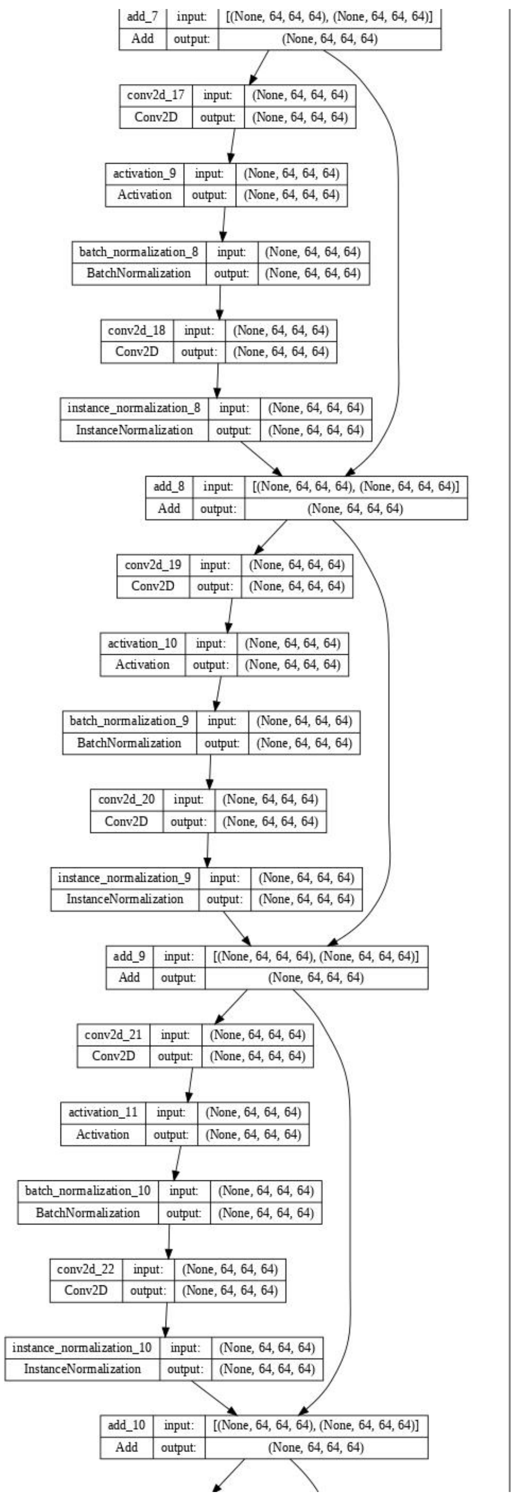

add_7 | Add | input: [(None, 64, 64, 64), (None, 64, 64, 64)] | output: (None, 64, 64, 64)
conv2d_17 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
activation_9 | Activation | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
batch_normalization_8 | BatchNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
conv2d_18 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
instance_normalization_8 | InstanceNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
add_8 | Add | input: [(None, 64, 64, 64), (None, 64, 64, 64)] | output: (None, 64, 64, 64)
conv2d_19 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
activation_10 | Activation | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
batch_normalization_9 | BatchNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
conv2d_20 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
instance_normalization_9 | InstanceNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
add_9 | Add | input: [(None, 64, 64, 64), (None, 64, 64, 64)] | output: (None, 64, 64, 64)
conv2d_21 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
activation_11 | Activation | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
batch_normalization_10 | BatchNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
conv2d_22 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
instance_normalization_10 | InstanceNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
add_10 | Add | input: [(None, 64, 64, 64), (None, 64, 64, 64)] | output: (None, 64, 64, 64)

Figure 5.15: Plotting of Generator model(4)

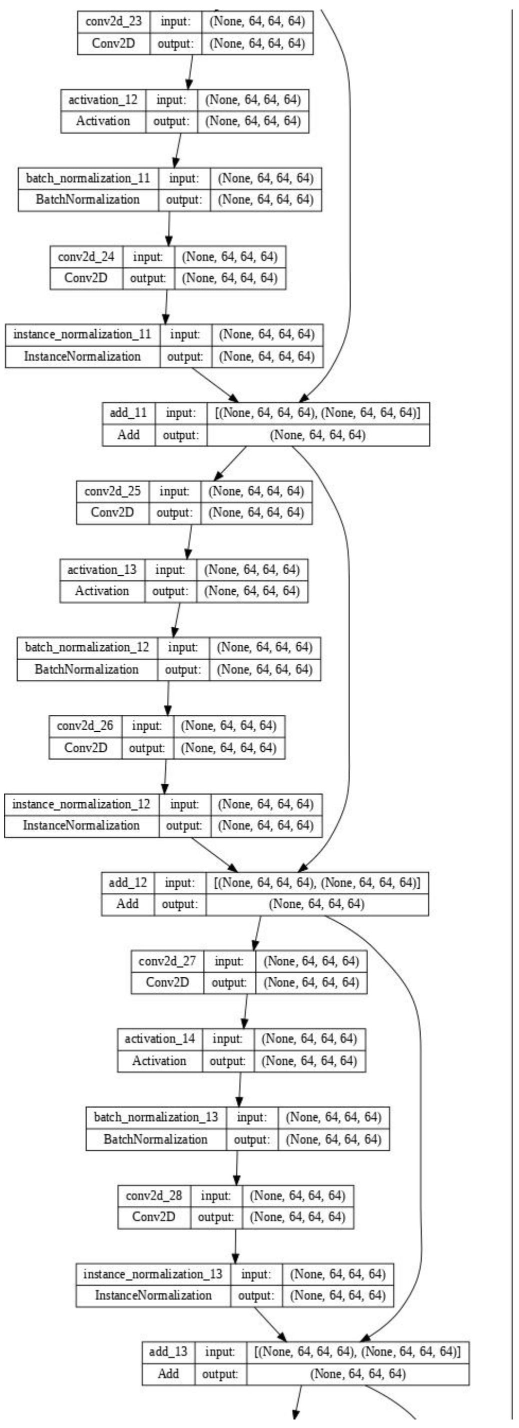

conv2d_23 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
activation_12 | Activation | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
batch_normalization_11 | BatchNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
conv2d_24 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
instance_normalization_11 | InstanceNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
add_11 | Add | input: [(None, 64, 64, 64), (None, 64, 64, 64)] | output: (None, 64, 64, 64)
conv2d_25 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
activation_13 | Activation | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
batch_normalization_12 | BatchNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
conv2d_26 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
instance_normalization_12 | InstanceNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
add_12 | Add | input: [(None, 64, 64, 64), (None, 64, 64, 64)] | output: (None, 64, 64, 64)
conv2d_27 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
activation_14 | Activation | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
batch_normalization_13 | BatchNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
conv2d_28 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
instance_normalization_13 | InstanceNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
add_13 | Add | input: [(None, 64, 64, 64), (None, 64, 64, 64)] | output: (None, 64, 64, 64)

Figure 5.16: Plotting of Generator model(5)

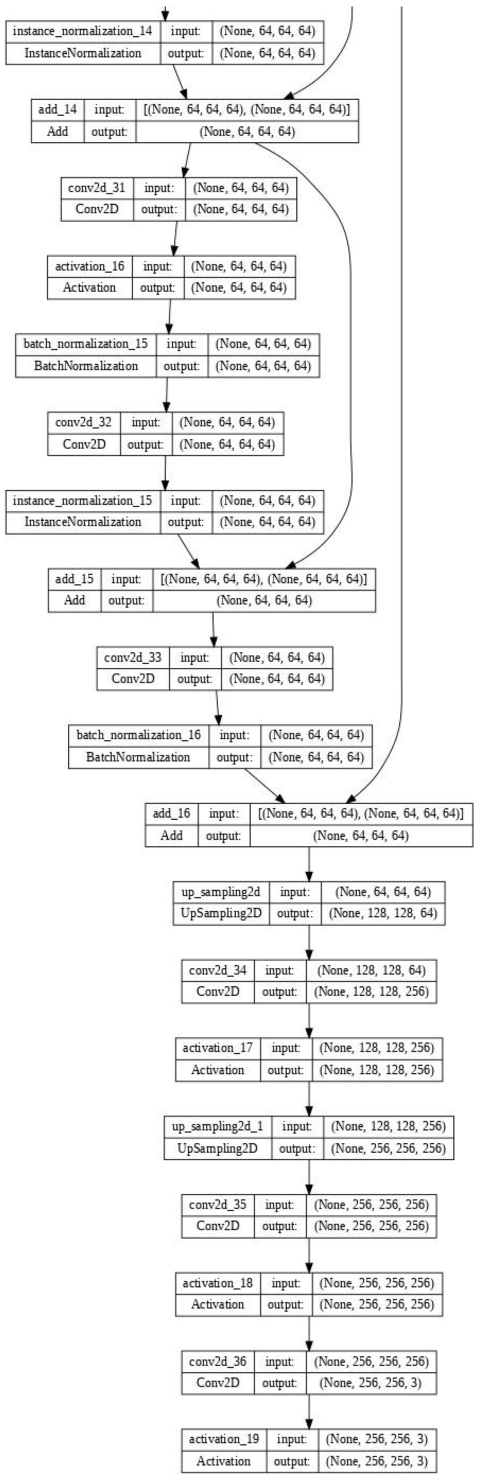

instance_normalization_14 | InstanceNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
add_14 | Add | input: [(None, 64, 64, 64), (None, 64, 64, 64)] | output: (None, 64, 64, 64)
conv2d_31 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
activation_16 | Activation | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
batch_normalization_15 | BatchNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
conv2d_32 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
instance_normalization_15 | InstanceNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
add_15 | Add | input: [(None, 64, 64, 64), (None, 64, 64, 64)] | output: (None, 64, 64, 64)
conv2d_33 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
batch_normalization_16 | BatchNormalization | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
add_16 | Add | input: [(None, 64, 64, 64), (None, 64, 64, 64)] | output: (None, 64, 64, 64)
up_sampling2d | UpSampling2D | input: (None, 64, 64, 64) | output: (None, 128, 128, 64)
conv2d_34 | Conv2D | input: (None, 128, 128, 64) | output: (None, 128, 128, 256)
activation_17 | Activation | input: (None, 128, 128, 256) | output: (None, 128, 128, 256)
up_sampling2d_1 | UpSampling2D | input: (None, 128, 128, 256) | output: (None, 256, 256, 256)
conv2d_35 | Conv2D | input: (None, 256, 256, 256) | output: (None, 256, 256, 256)
activation_18 | Activation | input: (None, 256, 256, 256) | output: (None, 256, 256, 256)
conv2d_36 | Conv2D | input: (None, 256, 256, 256) | output: (None, 256, 256, 3)
activation_19 | Activation | input: (None, 256, 256, 3) | output: (None, 256, 256, 3)

Figure 5.17: Plotting of Generator model(6)

5.5.3 Discriminator Network

Discriminator defines a discriminator neural network model for distinguishing between real and fake high-quality images. The model takes an input image of shape 256x256 pixels with 3 color channels.

The input image is passed through 8 custom convolutional layers, each with increasing numbers of filters (64, 64, 128, 128, 256, 256, 512, 512) and alternating stride values of 1 and 2. Each convolutional layer is followed by batch normalization and a LeakyReLU activation function with a negative slope of 0.2.

The output of the last convolutional layer is flattened and passed through a fully connected layer with 1024 units, followed by a LeakyReLU activation function. The output of this layer is then passed through a final fully connected layer with a single unit and a sigmoid activation function, producing a real-vs-fake patch output that represents the discriminator's prediction of the input image being real or fake.

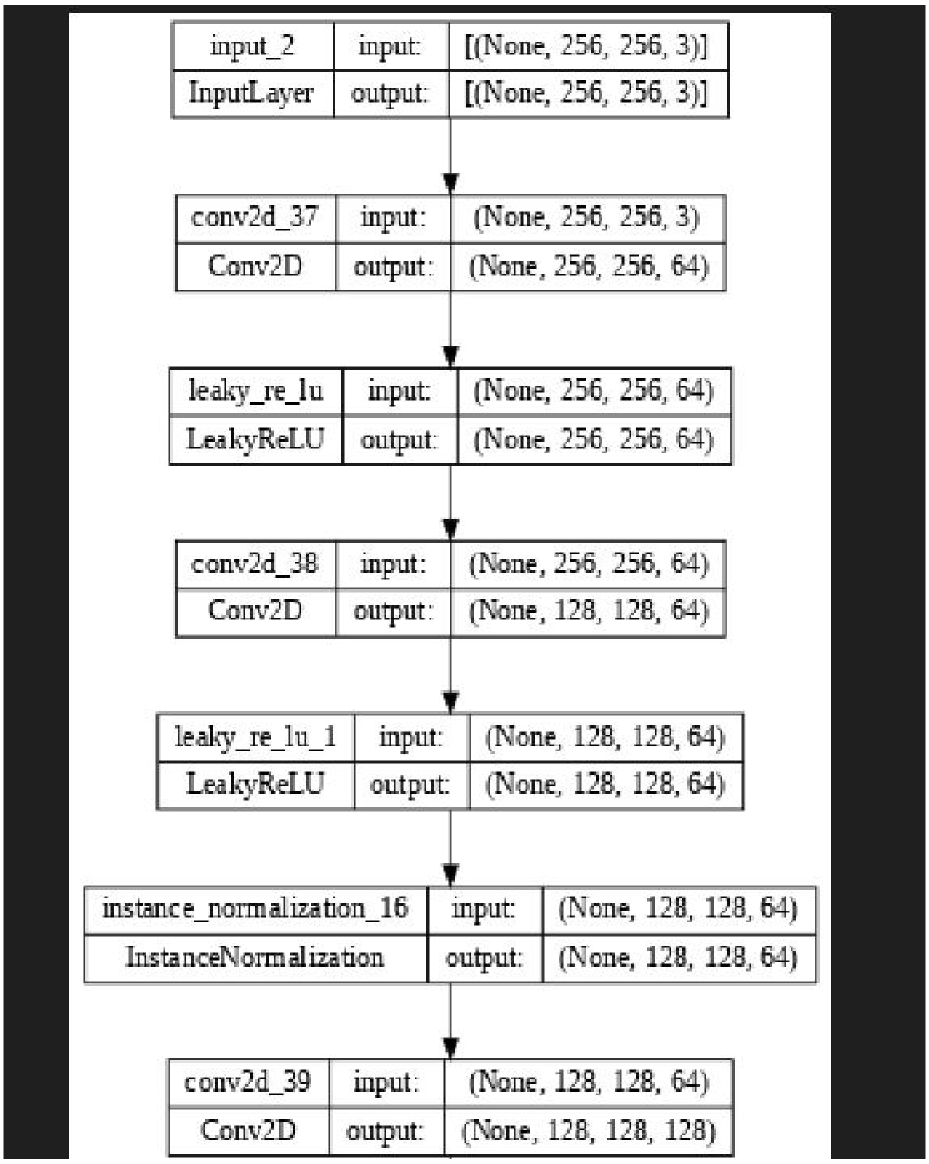


Figure 5.18: Plotting of discriminator model(1)

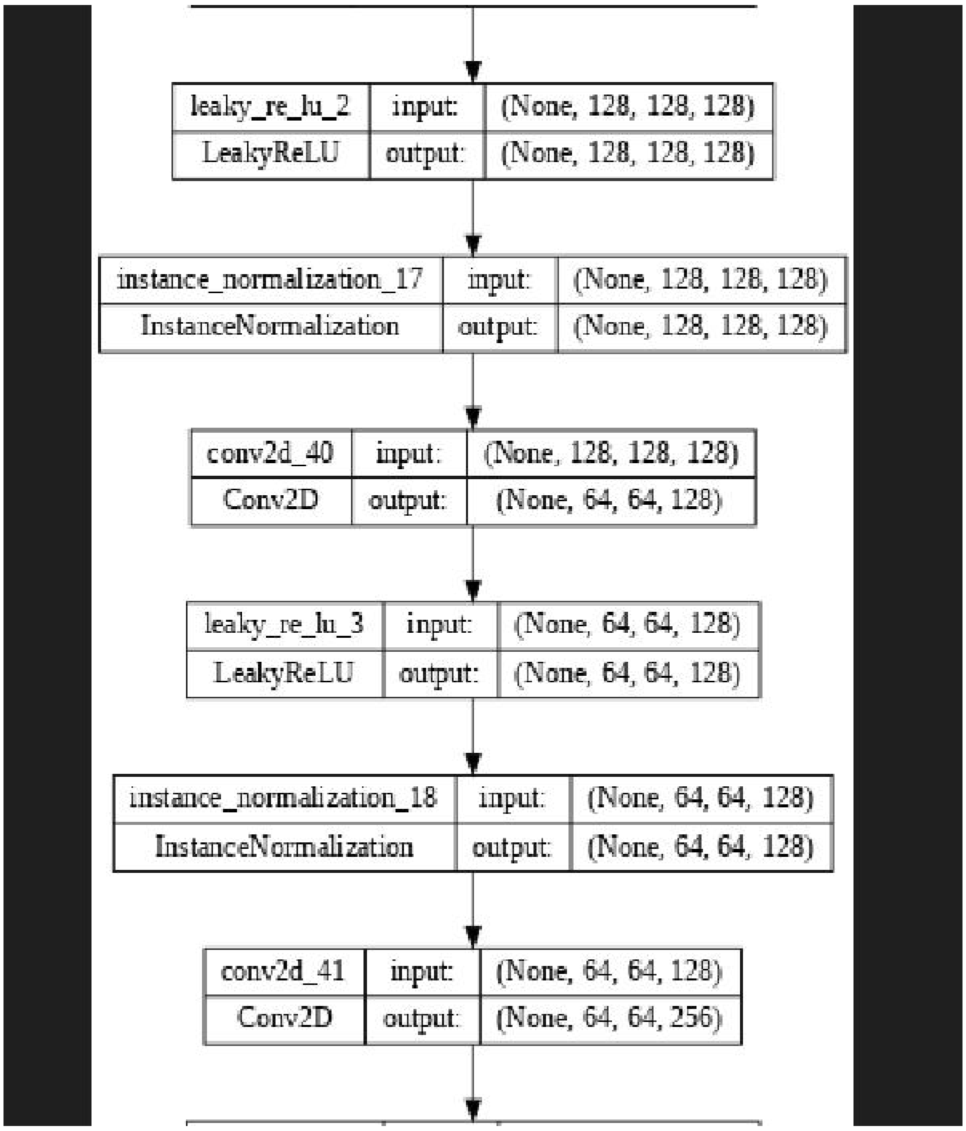
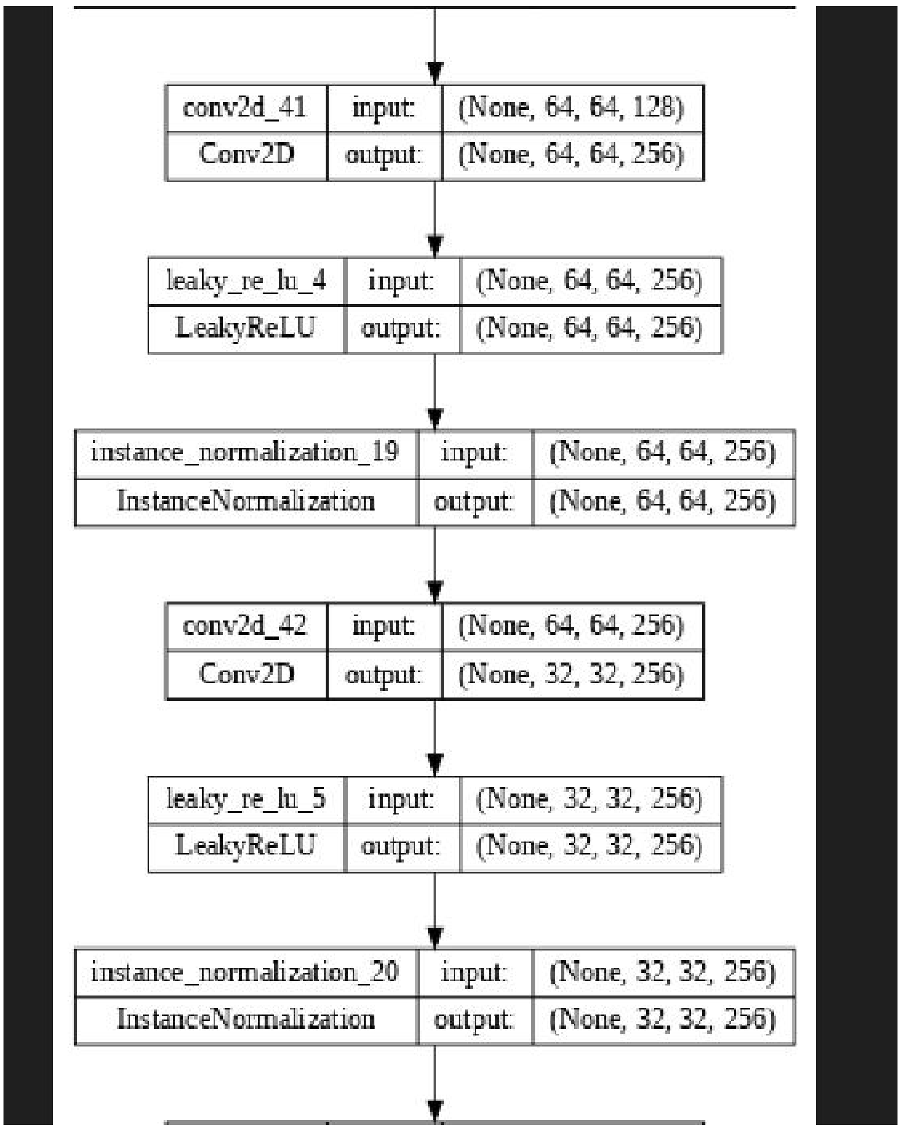


Figure 5.19: Plotting of discriminator model(2)


conv2d_41 | input: | (None, 64, 64, 128)
Conv2D | output: | (None, 64, 64, 256)
leaky_re_lu_4 | input: | (None, 64, 64, 256)
LeakyReLU | output: | (None, 64, 64, 256)
instance_normalization_19 | input: | (None, 64, 64, 256)
InstanceNormalization | output: | (None, 64, 64, 256)
conv2d_42 | input: | (None, 64, 64, 256)
Conv2D | output: | (None, 32, 32, 256)
leaky_re_lu_5 | input: | (None, 32, 32, 256)
LeakyReLU | output: | (None, 32, 32, 256)
instance_normalization_20 | input: | (None, 32, 32, 256)
InstanceNormalization | output: | (None, 32, 32, 256)
Figure 5.20: Plotting of discriminator model(3)

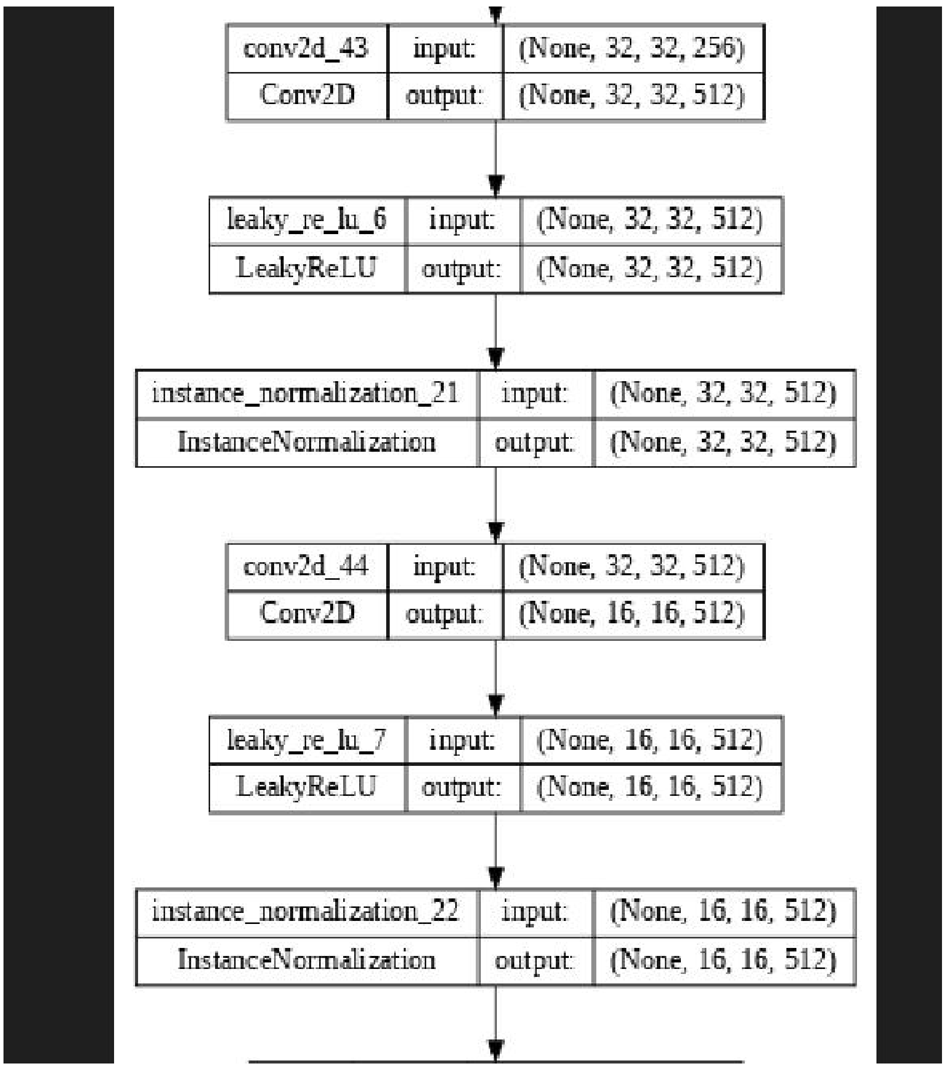

conv2d_43 | input: | (None, 32, 32, 256)
Conv2D | output: | (None, 32, 32, 512)
leaky_re_lu_6 | input: | (None, 32, 32, 512)
LeakyReLU | output: | (None, 32, 32, 512)
instance_normalization_21 | input: | (None, 32, 32, 512)
InstanceNormalization | output: | (None, 32, 32, 512)
conv2d_44 | input: | (None, 32, 32, 512)
Conv2D | output: | (None, 16, 16, 512)
leaky_re_lu_7 | input: | (None, 16, 16, 512)
LeakyReLU | output: | (None, 16, 16, 512)
instance_normalization_22 | input: | (None, 16, 16, 512)
InstanceNormalization | output: | (None, 16, 16, 512)

Figure 5.21: Plotting of discriminator model(4)

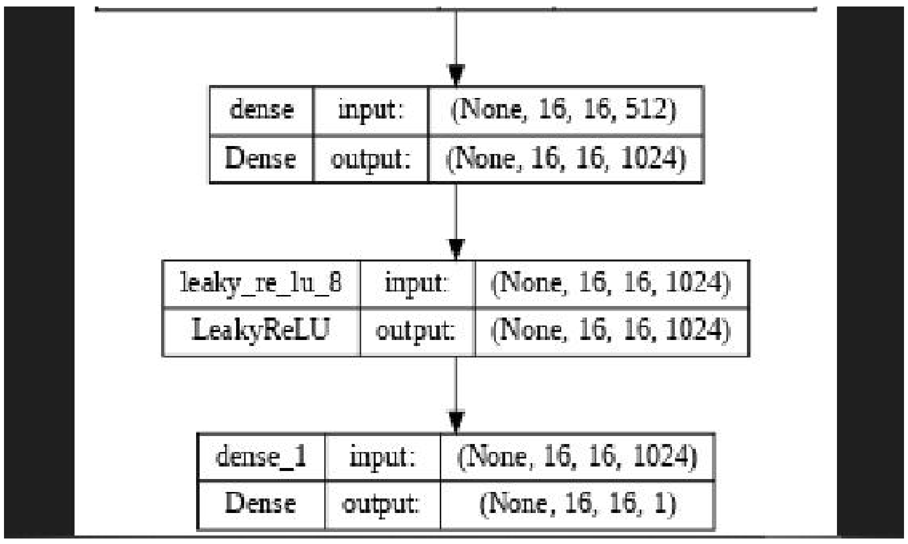

dense | input: | (None, 16, 16, 512)
Dense | output: | (None, 16, 16, 1024)
leaky_re_lu_8 | input: | (None, 16, 16, 1024)
LeakyReLU | output: | (None, 16, 16, 1024)
dense_1 | input: | (None, 16, 16, 1024)
Dense | output: | (None, 16, 16, 1)

Figure 5.22: Plotting of discriminator model(5)

Chapter 6

System Testing and Evaluation

After training of networks next step is to test the network model. For this purpose, we load the image into trained model of neural networks and then these trained networks give us output results. In our project we have to achieve super resolution, so we feed face image having low pixel values or resolution which in term Super-Resolved picture is obtained.

For the training and testing we have split the data set into two halves. One was used for training the model and the other was used for testing.

6.1 Quality Control

For Quality control, we will ensure that the data is safe and secure first. Since Facial images are private property so they should not be compromised neatly, testing at every stage and level of the system will be ensured. These tests will help in figuring out the state and performance level of the model. These tests will be used to further enhance the capability of the software to outperform.

6.2 Test cases

6.2.1 TC-01

Test case	
Test case id	TC-01
Function/need to test	Last updated on 201Training model
Initial state	Application must be installed on syste,
Input	Dataset
Expected Output	System should train on data
Actual Output	Model trained sucessfully
Status	pass

Table 6.1: TC-01

This test case refers to the training of the CNN. The dataset present in computer and goal is to train CNN on this data. The expected output of this test case is a trained CNN ready for testing and deployment.

6.2.2 TC-02

Test case	
Test case id	TC-02
Function/need to test	Load Image
Initial State	Image should be available
Input	Image
Expected Output	System should accept image
Actual Output	Image is loaded
Status	Pass

Table 6.2: TC-02

The function to be tested here is loading the image. The input will be an image. The image that the user wishes to upload should be present on the computer, The expected output of this test case is the successful uploading of the image.

6.2.3 TC-03

Test case	
Test case id	TC-03
Function/need to test	Image Preview
Initial State	Model is generating image
Input	Image
Expected Output	System should display selected image
Actual Output	Image is displayed
Status	Pass

Table 6.3: TC-03

Preview of the selected image is important because it makes sure that the right image has been selected. The expected output is preview of selected image and actual output was also preview of selected image. So this test case is deemed pass.

6.2.4 TC-04

Test case	
Test case id	TC-04
Function/need to test	Displaying ststus
Initial State	generating
Input	Image
Expected Output	System should generate and display out- put
Actual Output	HR image
Status	Pass

Table 6.4: TC-04

This use case targets whether the violator is detected and output is displayed. The expected output is prediction is displayed successfully and the actual output is according to the expected output.

6.3 Testing

The following testing was done on the system.

6.3.1 System testing

After successful integration we combined all the components and then tested the whole system. This made sure that everything was working as expected.

6.3.2 Black box testing

In black box testing we checked the output with respect to the functional specifications by giving input.We checked whether the violators were being detected.

6.3.3 White box testing

We checked the performance of application internally from the perspective of end user.

6.3.4 Performance Testing

Performance Testing We checked the overall response time of the system. Like how much time our mobile application takes to upload image and get result.

6.4 Testing Results

Some results from our trained model

low quality images given to model

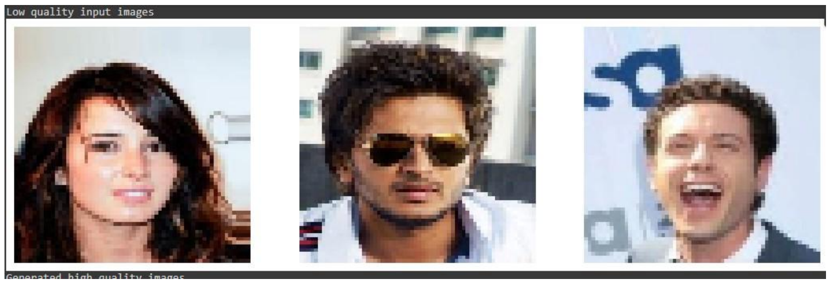


Figure 6.1: Low quality images

Generated high quality images Real high quality images

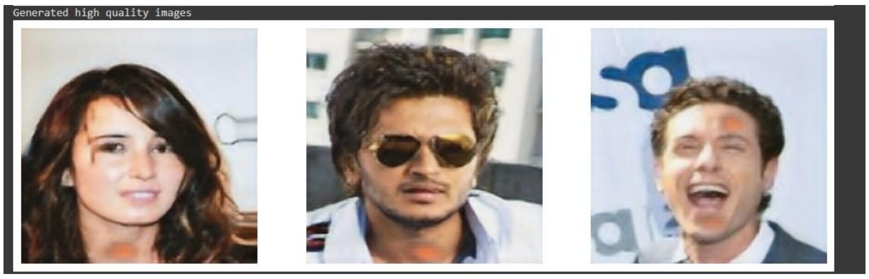


Figure 6.2: Generated high quality images

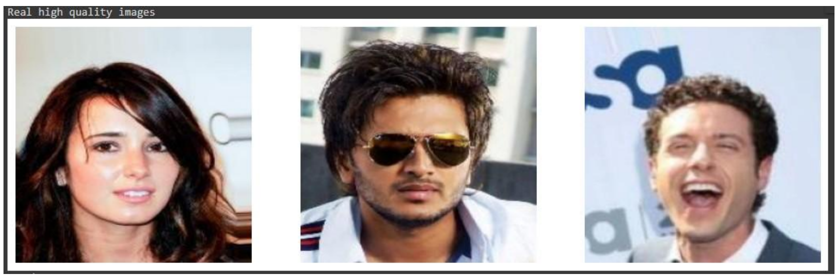


Figure 6.3: Real high quality images

Chapter 7

Conclusions

Now a days, Deep learning is playing very important role in our lives as AI is now a part of human life. The project Lower resolution to higher resolution is the project that is very helpful in our personal life as well as in professional life. Most of the editors takes too much time to enhance the image from lower resolution to higher resolution. This project is the direct solution to the problem to fill attractive colors.

7.1 Outcome

The application of GAN-based methods to convert lower resolution image to higher resolution using deep learning has shown promising results in generating realistic and visually appealing images. The use of GANs allows for the generation of high-quality images by training a generator model to produce images that are similar to the distribution of real images. Additionally, the use of deep learning techniques such as CNNs has enabled the model to learn complex features of face images and produce accurate results.

The use of super resolution GAN to convert LR to HR using deep learning in a android application has opened up new possibilities for enhancing the user experience of face image enhancement. By implementing this technology into a mobileapplication, it becomes more accessible and user-friendly, allowing users to easily upload their lower resolution face images and generate high resolution face images.

This technology has the potential to revolutionize the image enhancement process by providing a fast and efficient way to produce high-quality images, without the need for extensive manual editing. Additionally, the mobile application can also serve as a valuable tool for editors and artists to enhance their creative output.

7.2 Limitations

There are still some limitations to be addressed, such as the need for more diverse datasets to improve the generalization of the model, this approach has the potential to significantly improve the efficiency and quality of the image enhancement process.

7.3 Future work

Overall, the incorporation of SRGAN through deep learning into a web-based application is a remarkable breakthrough in the realm of computer vision, holding immense potential to transform the image enhancement industry. As the technology continues to evolve, we can expect to see even more innovative and exciting applications in the near future.

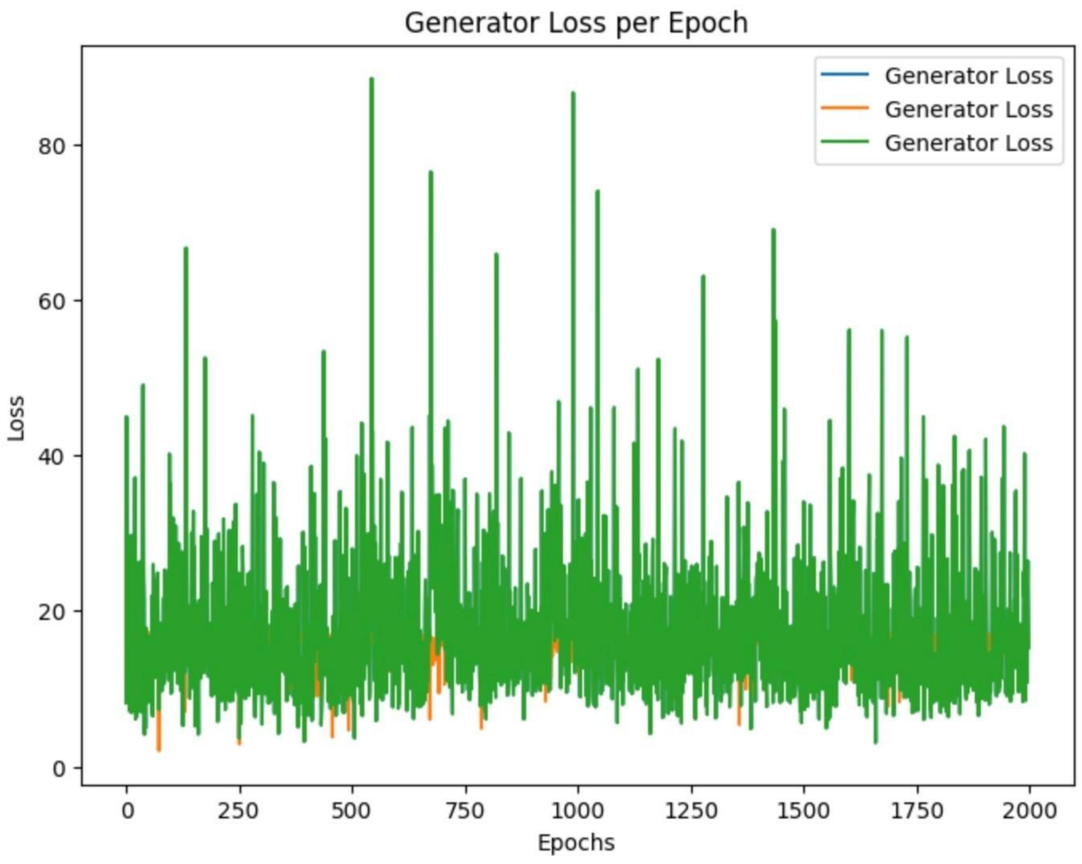


Figure 7.1: Generator loss per epoch

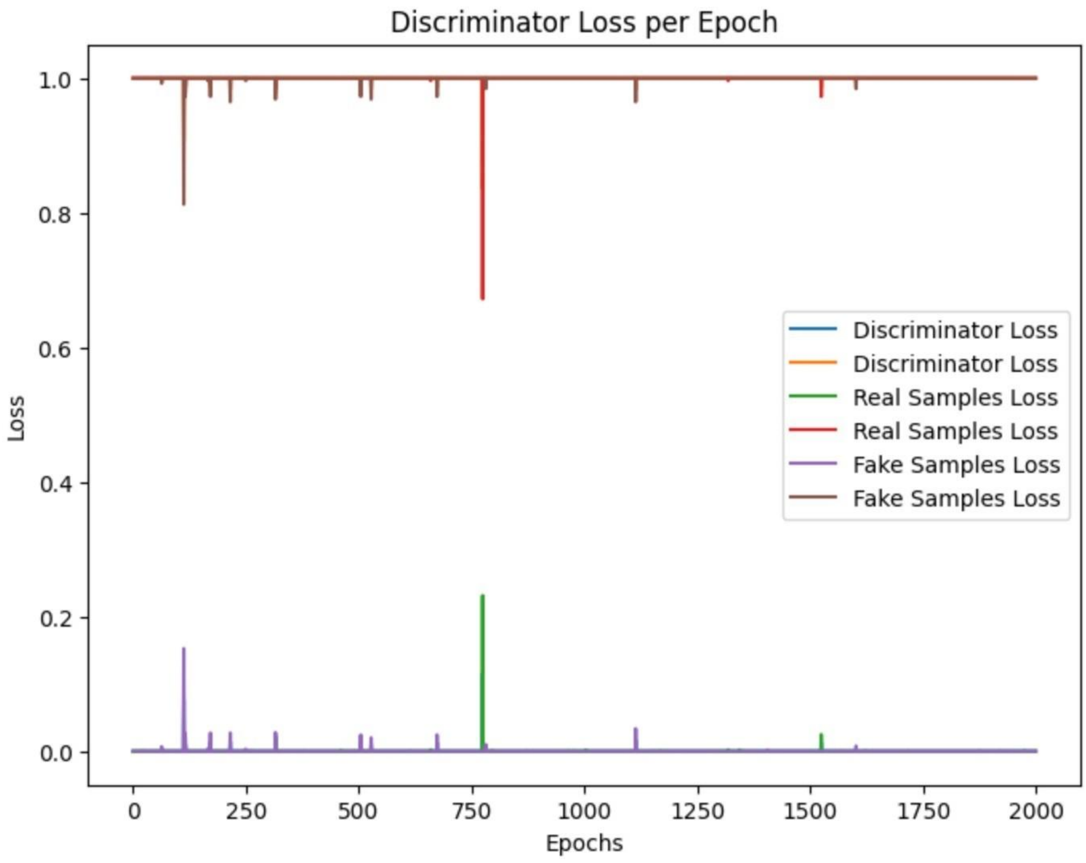


Figure 7.2: Discriminator loss per epoch

References

[1] Bulat, A., Yang, J., Tzimiropoulos, G. To learn image super-resolution, use a gan to learn how to do image degradation first. *In Proceedings of the European Conference on Computer Vision (ECCV)*, pages 185–200, 2018.

[2] Ledig, C., Theis, L., Huszar, F., Caballero, J., Aitken, A., Tejani, A., Totz, J., Wang, Z., Shi, W. Photo-realistic single image super-resolution using a generative adversarial network. In *2017 IEEE Conference on Computer Vision & Pattern Recognition* 2017.

[3] Tong, T., Li, G., Liu, X., Gao, Q. Image super-resolution using dense skip connections. In *2017 IEEE International Conference on Computer Vision,* 2017.

[4] Dong, C., Loy, C.C., Tang, X. Accelerating the super-resolution convolutional neuralnetwork. In *2016 European Conference on Computer Vision,* pages 391–407, 2016.

[5] Lim, B., Son, S., Kim, H., Nah, S., Mu Lee, K. Enhanced deep residual networks for singleimage super-resolution *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 136–144, 2017.

[6] Salimans, T., Kingma, D.P. Weight normalization: a simple reparameterization to acceleratetraining of deep neural networks. *In Advances in Neural Information Processing Systems*, pages 901–909, 2016.

[7] Ian J. Goodfellow∗ , Jean Pouget-Abadie† , Mehdi Mirza, Bing Xu, David Warde-Farley,Sherjil Ozair‡ , Aaron Courville, Yoshua Bengio§. Generative Adversarial Nets, 2014.