# CHAPTER 1

# INTRODUCTION

## IMAGE COLORIZATION

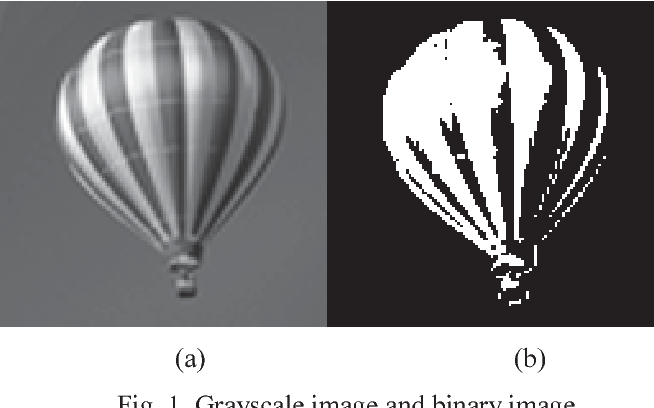
Grayscale image colorization algorithms play a crucial role in transforming monochrome images into visually appealing color representations while maintaining their natural appeal. This task presents a significant challenge due to the need to add chrominance values that suit the image without compromising its integrity. Unlike RGB color images, grayscale images lack color information and consist solely of intensity values, ranging from weak (black) to strong (white). Consequently, grayscale images are often referred to as monochrome, emphasizing their single-color nature.The concept of colorization was formally introduced by Wilson Markel in 1970, marking the beginning of efforts to add color to black and white imagery. However, achieving a perfect solution to the colorization problem remains elusive.

While manual colorization using tools like Photoshop or other photo editing software is a common approach, it is labor-intensive and reliant on user expertise. Novice users often struggle to produce high-quality results, leading to output of inferior quality. This limitation restricts the accessibility of manual colorization software to trained individuals, hindering its widespread adoption.The reliance on manual colorization not only consumes significant time and effort but also introduces variability in the quality of output, depending on the skill level of the user. This inconsistency in output quality undermines the reliability of manual colorization methods for applications requiring consistent and accurate results. Moreover, the subjective interpretation of color choices by individual users further complicates the process, potentially leading to deviations from the intended visual representation.

**1.1.1 TYPES OF IMAGES**

### 1. Binary images:

* It is the simplest type of image. It takes only two values i.e, Black and White or 0 and The binary image consists of a 1-bit image and it takes only 1 binary digit to represent a pixel. Binary images are mostly used for general shape or outline.
* Binary images are generated using threshold operation. When a pixel is above the threshold value, then it is turned white ('1') and which are below the threshold value then they are turned black('0')



**Fig 1.1 : Gray scale image to Binary Image**

### Gray-scale images:

* Grayscale images are monochrome images, means they have only one color. Grayscale images do not contain any information about color. Each pixel determines available different grey levels.
* A normal grayscale image contains 8 bits/pixel data, which has 256 different grey levels. In medical images and astronomy, 12 or 16 bits/pixel images are used.
* These intensity values typically range from 0 (black) to 255 (white) in an 8-bit grayscale image, with intermediate values representing shades of gray. Grayscale images lack color information, containing only variations in brightness or intensity.



**Fig 1.2 : Gray Scale Image**

### Color images:

* Color images are three band monochrome images in which, each band contains a different color and the actual information is stored in the digital image. The color images contain gray level information in each spectral band.
* The images are represented as red, green and blue (RGB images). And each color image has 24 bits/pixel means 8 bits for each of the three color band (RGB).

****

**Fig 1.3 : Colored Images**

**1.2 EVOLUTION OF IDEOLOGY BEHIND IMAGE COLORIZATION**

The evolution of colorization techniques has undergone a remarkable journey spanning several decades, marked by the convergence of artistic expression and technological innovation. Beginning with manual hand-coloring methods employed in the early days of photography, artists painstakingly added color to black and white images through techniques like hand painting or tinting. These methods, though labor-intensive, provided an early glimpse into the transformative power of color in visual storytelling. The formal introduction of the term "colorization" in the 1970s by Wilson Markle ignited interest in automated approaches to adding color to monochrome imagery, laying the groundwork for future advancements.

**In the early stages of the colorization era, techniques primarily fell into two categories:** manual and automated methods. Manual colorization involved hand-painting or tinting grayscale images to add color, a process that was labor-intensive and relied heavily on artistic skill. Automated methods, although nascent, began to emerge with the advent of digital technology, offering computer-assisted approaches to adding color to black and white imagery. These early automated methods laid the foundation for more sophisticated algorithms that would later revolutionize.

1. **Manual Colorization:** Manual colorization techniques involve the meticulous hand-painting or tinting of grayscale images to add color. This process dates back to the early days of photography when artists manually added color to black and white photographs using techniques such as watercolor painting, oil painting, or applying dyes directly to the print. Each color was carefully selected and applied to specific areas of the image, requiring a high level of artistic skill and attention to detail. While manual colorization provided a personalized and artistic touch, it was a time-consuming process and often resulted in variations in color accuracy and consistency across different interpretations.
2. **Automated Colorization:** Automated colorization methods leverage computational techniques, particularly in the digital realm, to automatically add color to grayscale images. Early automated approaches utilized simple algorithms and computer-assisted techniques to apply color to black and white imagery. These methods ranged from basic colorization algorithms that assigned colors based on predefined rules or user inputs to more sophisticated techniques that analyzed image features to infer plausible color information. While automated colorization offered greater efficiency and consistency compared to manual methods, early algorithms often struggled with accuracy and realism, producing colorized images that lacked fidelity to the original scene.

**1.2.2 PURPOUSE**

Colorization serves to revitalize historical photographs, making them more relatable and engaging to modern audiences. It enhances visual appeal, allowing for clearer communication and artistic expression. Colorized images find applications in education, research, and entertainment industries, offering immersive experiences and sparking creativity. Overall, colorization breathes new life into grayscale imagery, enriching storytelling and expanding the potential for visual expression.

**1.2.3 SCOPE**

Image colorization has a broad scope, encompassing diverse applications across historical preservation, artistic expression, and practical industries. Historically, colorization has been instrumental in revitalizing archival black and white photographs, making them more relatable and engaging to modern audiences. This application finds significance in museums, educational institutions, and cultural heritage organizations, where colorized imagery enhances exhibitions, educational materials, and digital archives, fostering a deeper connection to the past.Beyond its historical significance, image colorization serves as a creative outlet for artists to reinterpret existing imagery in vibrant hues. Artists leverage colorization as a medium for storytelling, evoking emotions, conveying narratives, and exploring themes through the use of color. Whether in traditional art forms or digital media, colorized images offer a captivating canvas for artistic expression and experimentation, enriching the visual landscape with imaginative interpretations of reality.

Moreover, image colorization has practical implications in industries such as advertising, marketing, and entertainment, where visually compelling content is essential for communication and engagement. Colorized images have enhanced visual appeal and effectiveness in conveying messages, making them valuable assets for brand promotion, product advertising, and multimedia content creation. In the entertainment industry, colorization contributes to the creation of immersive experiences in film, television, gaming, and animation, captivating audiences with vibrant worlds and characters.

**1.3** **OVERVIEW OF IMAGE COLORIZATION**

Image colorization is the process of imbuing grayscale or black and white images with color, enhancing their visual appeal and realism. Traditionally, colorization was a manual task, involving hand-painting or tinting, which was labor-intensive and subjective. However, modern techniques leverage deep learning algorithms, such as convolutional neural networks (CNNs), to automatically add color to grayscale images. These algorithms are trained on large datasets of paired grayscale and color images, learning to map grayscale intensity values to corresponding color values.

The applications of image colorization are vast and diverse. Historically, colorized images provide a relatable and engaging representation of the past, making historical photographs accessible to modern audiences. Additionally, colorization serves as a medium for artistic expression, allowing artists to convey narratives and evoke emotions through vibrant colors. Moreover, colorized images are valuable for visual enhancement in various fields, including advertising, marketing, and multimedia content creation, where they enhance communication and storytelling. Recent advancements in image colorization include the integration of attention mechanisms within deep learning architectures, enabling models to focus on specific regions of the image for more precise color assignments. Furthermore, there is exploration into alternative generative models beyond traditional GANs, such as Variational Autoencoders (VAEs) and Generative Adversarial Networks with Style Transfer (GANs-ST), which offer new possibilities for capturing artistic styles in colorized images. Additionally, the adoption of self-supervised learning techniques holds promise for improving colorization performance by enabling models to learn meaningful representations from unlabeled data.

* 1. **OBJECTIVE**

The objective of image colorization using both Generative Adversarial Networks (GANs) and U-Net architectures is to automate and improve the process of adding color to grayscale or black and white images. By leveraging GANs, the aim is to generate realistic colorized images by learning the complex mapping between grayscale intensity values and corresponding color distributions. GANs employ adversarial training to produce colorizations that closely resemble real color photographs, enhancing the realism and fidelity of the colorization process. On the other hand, the objective of incorporating U-Net architectures is to enhance the precision and detail of the colorization process. U-Net is a convolutional neural network architecture designed for image segmentation tasks, characterized by its symmetric encoder-decoder structure and skip connections.

By integrating U-Net into the colorization pipeline, the objective is to capture fine-grained details and preserve image structures during the colorization process. This enables the model to produce colorized images with improved clarity and accuracy, particularly in regions with intricate textures or complex features.By leveraging the strengths of both approaches, the objective is to automate and optimize the colorization process, producing colorized images that are visually appealing, faithful to the original content, and suitable for a wide range of applications across various domains and industries.

# CHAPTER-2

# LITERATURE REVIEW

**2.1 THEORETICAL BACKGROUND OF THE PROBLEM**

Before the advent of Generative Adversarial Networks (GANs) and U-Net architectures for image colorization, traditional methods relied heavily on manual intervention and handcrafted algorithms. Image colorization, the process of adding color to grayscale images, posed a significant challenge due to the inherent complexity of understanding and replicating the intricate details of real-world scenes. Classical approaches often involved laborious manual techniques where artists or experts meticulously added color to grayscale images based on their expertise and subjective interpretations. These methods were time-consuming, prone to errors, and lacked scalability, making them unsuitable for large-scale applications. Moreover, these approaches struggled to capture the nuances of color distribution and spatial relationships present in natural images, leading to colorizations that were often unrealistic or inaccurate.

The theoretical groundwork for automated image colorization using deep learning began with the emergence of convolutional neural networks (CNNs). CNNs revolutionized computer vision tasks by automatically learning hierarchical features from raw pixel data, enabling machines to interpret visual information with increasing accuracy. However, early attempts at image colorization using CNNs encountered challenges in preserving fine details and spatial coherence in the colorized outputs. The introduction of GANs further advanced the field by introducing a novel framework for generative modeling. GANs consist of two neural networks – a generator and a discriminator – engaged in a minimax game, where the generator aims to produce realistic samples while the discriminator learns to distinguish between real and fake data. This adversarial training approach proved highly effective for generating high-quality colorized images by capturing complex patterns and textures present in natural scenes

**2.2 LITERATURE STUDY**

# 1. Ivana Zeger : “Grayscale Image Colorization Methods: Overview and Evaluation” – 2021.

# This paper presents an overview and evaluation of grayscale image colorization methods and techniques applied to natural images. The paper provides a classification of existing colorization methods, explains the principles on which they are based, and highlights their advantages and disadvantages. Special attention is paid to deep learning methods. Relevant methods are compared in terms of image quality and processing time.

# p1.jfif

# Fig 2.1: Gray Scale Image Colorization

# Different metrics for color image quality assessment are used. Measuring the perceived quality of a color image is challenging due to the complexity of the human visual system. Multiple metrics used to evaluate colorization methods provide results by determining the difference between the predicted color value and the ground truth, which in several cases is not in coherence with image plausibility. The results show that user-guided neural networks are the most promising category for colorization because they successfully combine human intervention and neural network automation. Visual and quantitative comparison of several state-of-the-art colorization methods with various architectures and levels of user assistance is described in the following paragraphs.

# 2. Wujian ye : “Hybrid Scheme of Image’s Regional Colorization Using Mask R-CNN and Poisson Editing” – 2019.

# Image colorization is a creative process of reasonably adding colors on gray-scale images to generate well-pleasing colorized images. The existing colorization methods normally require user-supplied hints of color points and doodles, or handpicked color reference images for transferring colors, or diverse color images for predicting the colorized results; but the final colorized results generated by most of them may seem unnatural as a consequence of the unprofessional users' skills, inaccurately color transferring, or limited scale of color image collection. To overcome these limitations, a hybrid scheme consisting of two modules is proposed for images' region colorization by combining semantic segmentation and seamless fusion techniques in this paper.

# In the first module, the masks and category of input image's regions and background are derived from a Mask R-CNN model, and the corresponding reference images of each region are selected from a pre-classified color image database. In the second module, the background and various regions of an image are colorized by a U-Net model and a VGG model respectively. Then, the Poisson editing technique is applied for fusing all the colorized results to generate the final whole colorized image.

# p2.jfif

**Fig 2.2 : Regional Colorization Using Mask R-CNN**

# 3. Lin Cao – “Double-Channel Guided Generative Adversarial Network for Image Colorization” – 2021

# We propose a novel Double-Channel Guided Generative Adversarial Network (DCGGAN). It includes two modules: a reference component matching module and a double-channel guided colorization module. The reference component matching module is introduced to select suitable reference color components as auxiliary information of the input. The double-channel guided colorization module is designed to learn the mapping relationship from the grayscale to each color channel with the assistance of reference color components. Experimental results show that the proposed DCGGAN outperforms existing methods on different quality metrics and achieves state-of-the-art performance.

# We adopt a three-step training process as follows. First, the reference component matching module uses a pre-trained network model to match the training image’s reference component. Then, two matching pairs of the luminance component and reference component train the double-channel colorization network.

# Finally, the structure and network parameters of the model are saved.The testing process is similar to the training process. First, the model of the colorization network is loaded. Then, the a-channel and b-channel reference components of the test image are matched. Finally, the generated a-channel and b-channel color components combine with the input luminance component to form the full colorization image.

# 4. Xin jin – “A Deep Multitask Convolutional Neural Network for Remote Sensing Image Super-Resolution and Colorization” – 2022.

# Remote sensing data have become increasingly vital in target detection, disaster monitoring, and military surveillance. Abundant pan-sharpening and super-resolution (SR) methods based on deep learning have been proposed and have achieved remarkable performance. However, pan-sharpening requires paired panchromatic (PAN) and multispectral (MS) images, and SR cannot increase the spectral resolution of PAN. Thus, we introduce a computational imaging-based method to recover or produce the incomplete data of single PAN or MS. This work also explores the integration of multiple tasks by a single neural network. We start with SR and colorization, study the feasibility of simultaneously finishing SR colorization, and use a model trained in SR colorization to finish pan-sharpening without MS.

# A generic neural network, remote sensing image improvement network (RSI-Net), is designed for remote sensing image SR, colorization, simultaneous SR colorization, and pan-sharpening. To verify its performance, RSI-Net is compared with the state-of-the-art SR and colorization methods. Experiments show that RSI-Net can be competitive in visual effects and evaluation indexes, and it performs well at simultaneous SR colorization, and RSI-Net finishes pan-sharpening and only needs to input PAN. Our experiments confirm the effect of integrating multiple tasks. This work also explores the integration of multiple tasks by a single neural network. We start with SR and colorization, study the feasibility of simultaneously finishing SR colorization, and use a model trained in SR colorization to finish pan-sharpening without MS.

# p3.jfif

# Fig 2.3 : Remote Sensing Image Super-Resolution and Colorization

# 5. Xun Duan – “Adversarial Edge-Aware Image Colorization With Semantic Segmentation”- 2022

# In this paper, we propose a new framework of an edge-aware colorized deep neural network with semantic segmentation to solve the above problems. Our main focus is to make the different objects in the colored image have clear coloring boundaries, which can effectively achieve the image colorization of edge perception. The above effect can be achieved in our work mainly because of the following three reasons. First, our network is based on the architecture of a generative adversarial network, and the generator of the network has the structure of deep semantic fusion. The addition of adversarial loss can generate more vivid results. Second, image colorization is the first task of our network, and the other task is semantic segmentation. That is, our work realizes multitask output  and constrains the network output by adding a semantic segmentation task.

# Basically, the two tasks share the same goal, which is to acquire as many image features as possible, so in addition to obtaining more information about the edge of the object, the color task can also be assisted. Third, we adopted a new color difference loss LCMC , that makes the color difference calculation more in line with the characteristics of human visual observation. We trained and verified the proposed method on two public datasets, namely, the PASCAL VOC 2012 augmented dataset and the ADE20K dataset.

# Feng Xu – “Radar Image Colorization: Converting Single-Polarization to Fully Polarimetric Using Deep Neural Networks”- 2023

This paper proposes “radar image colorization” to reconstruct a full-pol image from a non full-pol image, so that existing PolSAR methods, such as model-based decomposition and unsupervised classification, can be directly applied to the reconstructed full-pol SAR images. It proposes to train a specially designed deep neural network to convert a single polarization gray-scale SAR image to full-pol. It consists of two components: a feature extractor network to extract hierarchical multi-scale spatial features of the grayscale SAR image, followed by a feature translator network to map spatial feature to polarimetric feature with which the polarimetric covariance matrix of each pixel can be reconstructed. Both qualitative and quantitative experiments with real full-pol data are conducted to show the efficacy of the proposed method. The reconstructed full-pol SAR image agrees well with the true full-pol image, not only in the sense of visual similarity but also in the sense of real PolSAR applications, such as target decomposition and terrain classification.

In conclusion, the proposed method of radar image colorization offers a promising solution to the challenge of reconstructing full-polarimetric synthetic aperture radar (PolSAR) images from non-full-polarimetric data. By leveraging a specially designed deep neural network, the approach effectively translates grayscale SAR images into full-pol representations, enabling the direct application of existing PolSAR processing methods.Through both qualitative and quantitative experiments using real full-pol data, the efficacy of the method is demonstrated. The reconstructed full-pol SAR images exhibit strong agreement with true full-pol images, both visually and in terms of their utility for PolSAR applications such as target decomposition .

# Iiya Makorov – “Robust Manga Page Colorization via Coloring Latent Space”-2022.

# Manga (Japanese comics) are commonly drawn with black ink on paper. Colorization of manga pages can enrich the visual content and provide a better reading experience. However, the existing colorization approaches are not sufficiently robust. In this paper, we propose a two-stage approach for manga page colorization that supports sampling and color modification with color hints. In the first step, we employ the Pixel2Style2Pixel architecture to map the black-and-white manga image into the latent space of StyleGAN pretrained on the highly blurred colored manga images that we call Coloring Latent Space. The latent vector is automatically or manually modified and fed into the StyleGAN synthesis network to generate a coloring draft that sets the overall color distribution for the image. In the second step, heavy Pix2Pix-like conditional GAN fuses the information from the coloring draft and user-defined color hints and generates the final high-quality coloring. Our method partially overcomes the multimodality of the considered problem and generates diverse but consistent colorings without user

# p4.jfif

# Fig 2.4 : Robust Manga Page Colorization via Coloring Latent Space

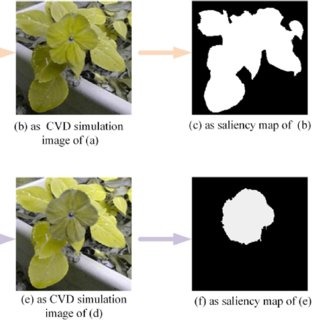
# The visual comparison, the quantitative evaluation with Frechet Inception Distance, and the qualitative evaluation via Mean Opinion Score exhibit the superiority of our approach over the existing state-of-the-art manga pages colorization method.We have explored the ability of StyleGAN to learn the distribution of color manga images and its ability to build a proper latent space for existing projection methods.We have proposed a learning-based manga colorization method that utilizes pretrained StyleGAN and exhibits high visual performance in automatic mode, as well as qualitatively supports image modification via user input. Employment of StyleGAN increases the robustness of the model.

# Hui Fan- “Saliency Consistency-Based Image Re-Colorization for Color Blindness.”-2022.

For patients with color vision defects, owing to the destruction of cone cells and loss of function, some color information is lost, hence changing the originally transmitted information by the image. The purpose of the traditional method of correction is to help patients distinguish between colors, but it does not take into account the problem of the saliency of the image. In this paper, we propose a saliency consistency-based image re-colorization for color blindness.. According to the detection results, an image with the same detection result was selected as the reference image. We grayscale the significantly changed image, recolor the grayscale image using the reference image, and the color scheme of the recolored image is similar to the reference image.

The color matching scheme of the reference image makes the significance of the image basically unchanged in standard vision and color vision defects, thereby making the color blindness patient's perception of the image close to standard vision. In addition, we use different evaluation criteria to evaluate the experimental results objectively. In the subjective evaluation and objective evaluation, the method proposed in this paper has achieved good results, which validates the effectiveness of our method.

In conclusion, the approach offers a promising solution for addressing the challenges faced by individuals with color vision deficiencies, particularly in scenarios where color plays a critical role in conveying information or distinguishing between objects. By prioritizing saliency consistency, the re-colorization process enhances the accessibility and usability of visual content for individuals with color blindness, ultimately improving their overall viewing experience.



**Fig 2.5 : Saliency Consistency-Based Image Re-Colorization**

# Bumshik Lee- “Image Colorization Using Color-Features and Adversarial Learning.”-2022.

# This is an innovative image colorization method that not only improves color accuracy and realism but also addresses common issues found in existing methods, such as desaturation and color bleeding. Our proposed method features a novel component called the Color Encoder, which extracts intrinsic color features. Moreover, the proposed Color Encoder aligns essential color features systematically, drawn from a random normal distribution, with real colors. These aligned features are fused at the bottleneck and serve as the foundation for subsequent colorization. Complementing the Color Encoder is our Color Loss mechanism, which aims to align the extracted features from the Color Encoder with the ground-truth color features, enhancing overall color representation accuracy. We also employ a Conditional Wasserstein Generative Adversarial Network (CWGAN) architecture within the framework of a Generative Adversarial Network (GAN) to improve adversarial training and colorization accuracy.

# 

# CHAPTER-3

# PROPOSED METHOD

## INTRODUCTION

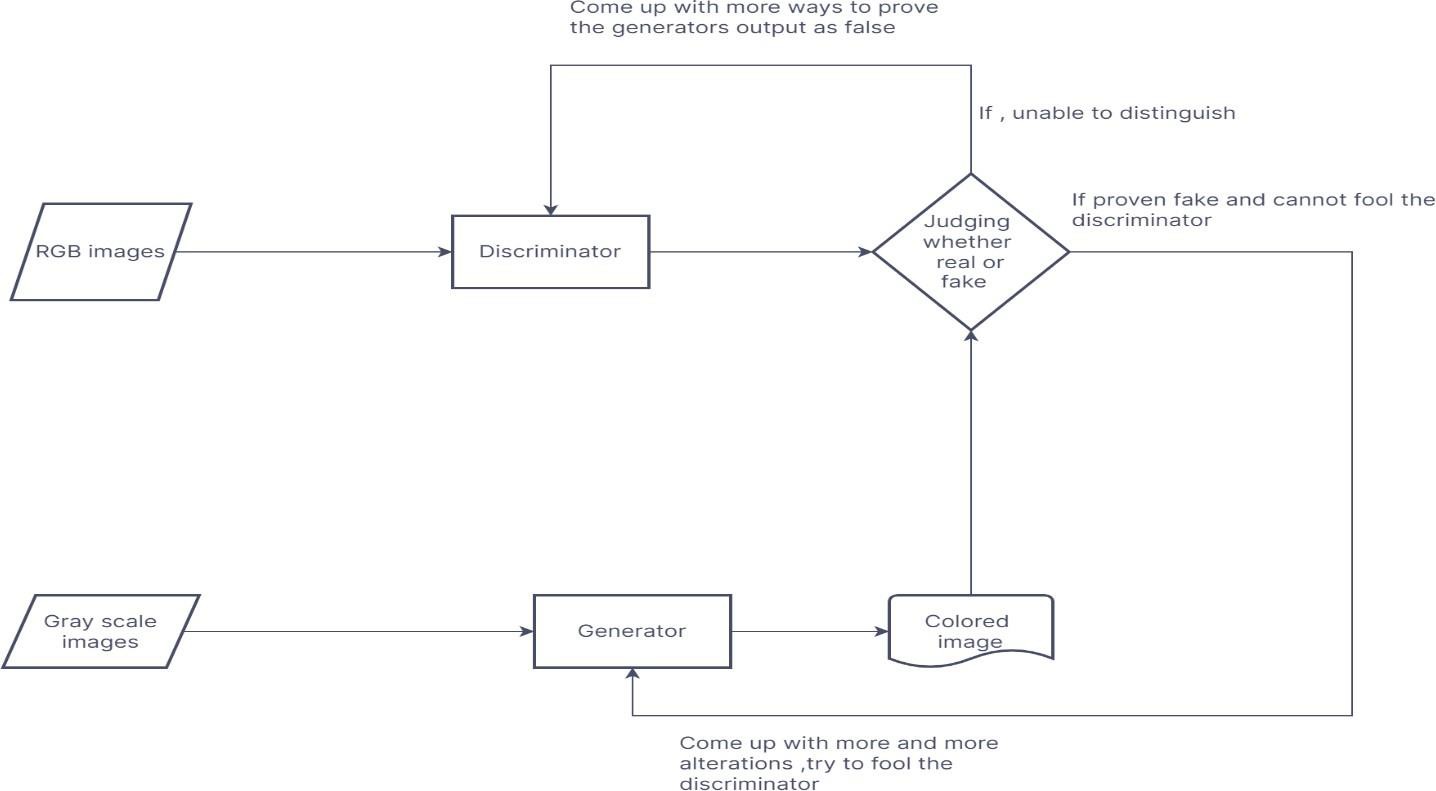
This project based on the idea of colorizing images through usage of Generative Adversarial Networks. The main aim is to color the images with no alterations in their quality. Usually, the previous methods entirely focused on coloring image and the image was considered regardless of its quality. Although some external methods are used to improve the quality of the image, which are not available for open or had some limitations. But through usage of Generative Adversarial Networks, it completely eliminates the chances of reduction in image quality.

As the Generative Adversarial Networks completely produce an entire newly colored image, rather than coloring the image itself. The GAN’s will work even upon feeding some random noise, as they produce/generate new data which is an exact copy of the desired data. Usually, the GAN’s are used for generating animated images based on their knowledge that they have.The GAN’s consists of two neural networks competing with each other.

They are Generator(G(x)) and Discriminator(D(x)). The goal of the generator is to generate the desired data which has to be exact version of the original data.Here the UNET acts as the generator for producing fake images and providing input for the discriminator in the place of generator. It actually composes of multiple convolutional layers .

## 3.2 BLOCK DIAGRAM

Now the discriminator has to judge the data generated by the generator, determine if it was an exact replica or not, simply it should determine whether the data is near to original or not. These both networks compete with each other for one to replicate the original data as it is, the other to prove the generated data isn’t original or near to original.



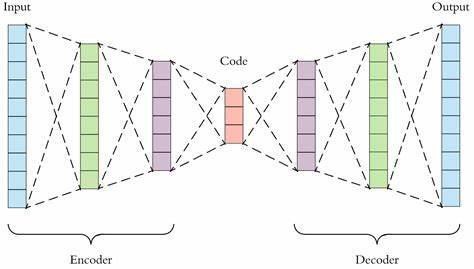
**Fig 3.1 Block Diagram**

If the either fails to achieve their goals they need to determine where they are backing (usually look for ways to determine the correct weights), which can be done with the help of loss functions and optimizers.

## PROPOSED SYSTEM

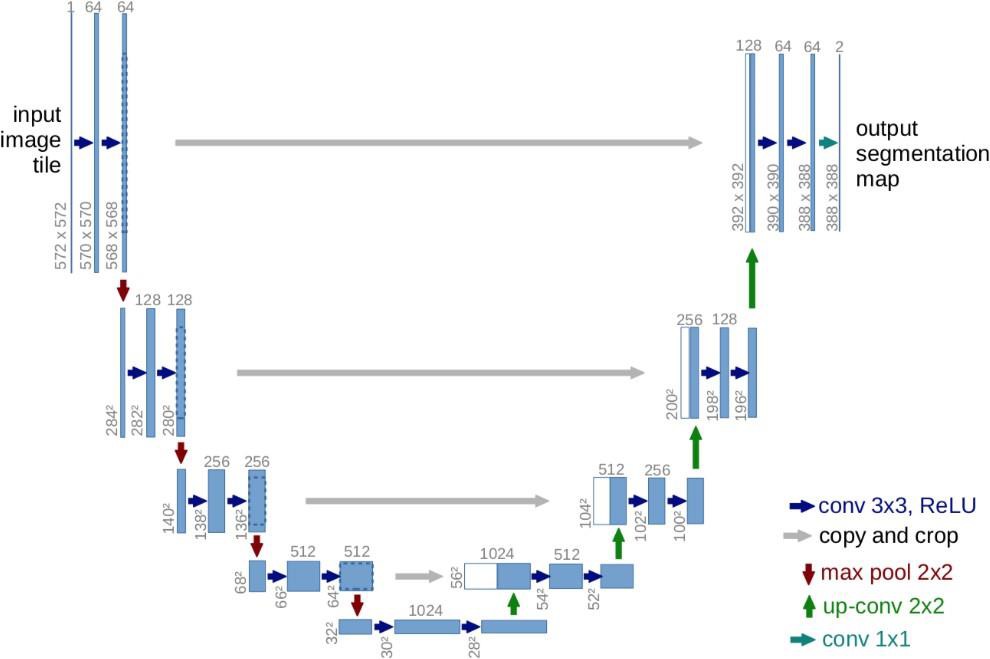
## Here we use GAN with UNET architecture for getting higher results. So as mentioned the GAN actually used to find the real and fake images more preciecly, A GAN consists of two neural networks they are Generator and a Discriminator. The Generator takes random noise or input images and produces output images. The Discriminator learns to distinguish dataset and fake images generated by the Generator.

* The Generator aims to produce images that are indistinguishable from real ones, while the Discriminator aims to accurately classify real and fake images. Through an adversarial training process, the Generator and Discriminator networks are updated iteratively to improve their performance.In a GAN with a U-Net model, the Generator often employs the U-Net architecture. U-Net's encoder-decoder structure captures high-level semantics and fine details, making it suitable for image-to-image translation tasks.



**Fig 3.2 : Encoder and Decoder**

* The UNET architecture has two paths. The first path of the UNET architecture is called contraction path which is also called as encoder that is used for capturing of context in the image. The second path is the symmetric expanding path which is also called as decoder that is used to enable the precise localization using transposed convolutions.



**Fig 3.3 UNET Architecture**

The loss functions that we used were Mean Square Error for Generator loss function and Binary Cross Entropy for Discriminator loss function.

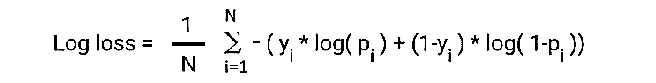
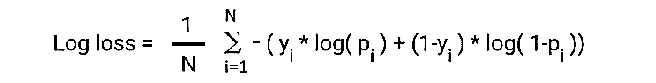
**Mean Square Error:** Mean Square Error is used to compare the compression quality of the image. The cumulative squared error between the original and the compressed image is determined by mean square error. If we consider M as number of rows and N as number of columns of input image, MSE can be calculated as follows:

### MSE = (∑ [I1(m,n)−I2(m,n)]2)/M\*N

**Loss Function:** It is a metric which gives a clear idea of up to what extent the model is good at predictions. If the model predictions are closer to the actual values then the loss function value would be minimum. Otherwise, loss function value would be maximum. Mathematically, Loss Function can be determined as:

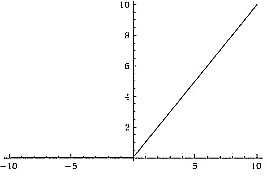
### Loss= abs(Y\_pred – Y\_actual)

**Binary Cross Entropy:** Binary cross entropy which is also called as log loss, is negative average of the log of corrected predicted probabilities. It can be calculated as follows:



**ACTIVATION FUNCTIONS :**

**RELU (Rectified Linear Unit):** Instead of sigmoid, the recent networks prefer using ReLU activation functions for the hidden layers. The function is defined as: F(x) =max (x, 0) The output of the function is X when X>0 and 0 for X<=0. The function looks like this:



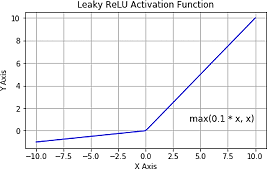
**Fig 3.4 RELU Activation Function**

The major benefit of using ReLU is that it has a constant derivative value for all inputs greater than 0. ReLU is linear (identity) for all positive values, and zero for all negative values.

**LEAKY RELU:** Leaky ReLU function is an improved version of the ReLU activation function. As for the ReLU activation function, the gradient is 0 for all the values of inputs that are less than zero, which would deactivate the neurons in that region and may cause dying ReLU problem.Leaky ReLU is defined to address this problem. Instead of defining the ReLU activation function as 0 for negative values of inputs(x), we define it as an extremely small linear component of x. Here is the formula for this activation function**:**

### f(x)=max(0.01\*x, x)

This function returns x if it receives any positive input, but for any negative value of x, it returns a really small value which is 0.01 times x. Thus it gives an output for negative values as well.

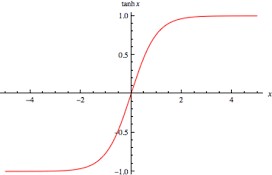


### Fig 3.5 : Leaky ReLU Activation Function

By making this small modification, the gradient of the left side of the graph comes out to be a non zero value.

**TANH:** Tanh function is very similar to the sigmoid/logistic activation function, and even has the same S-shape with the difference in output range of -1 to 1. In Tanh, the larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.

### f(x)= (ex-e-x)/(ex+e-x)



**Fig 3.6 : Tan h Activation Function**

* The output of the tanh activation function is Zero centered; hence we can easily map the output values as strongly negative, neutral, or strongly positive.Usually used in hidden layers of a neural network as its values lie between -1 to; therefore, the mean for the hidden layer comes out to be 0 or very close to it. It helps in centering the data and makes learning for the next layer much easier.

## 3.4 HARDWARE REQUIREMENTS :

* + 4GB RAM
  + INTEL 13 PROCESSOR
  + GPU OF 2GB RECOMMENDED

**3.5 SOFTWARE REQUIREMENTS :**

**PYTHON :** Python is a [high-level,](https://en.wikipedia.org/wiki/High-level_programming_language) [interpreted,](https://en.wikipedia.org/wiki/Interpreter_(computing)) [general-purpose programming](https://en.wikipedia.org/wiki/General-purpose_programming_language) [language.](https://en.wikipedia.org/wiki/General-purpose_programming_language) Its design philosophy emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability) with the use of [significant](https://en.wikipedia.org/wiki/Off-side_rule) [indentation.](https://en.wikipedia.org/wiki/Off-side_rule) Python is [dynamically-typed](https://en.wikipedia.org/wiki/Type_system#DYNAMIC) and [garbage-collected.](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)) It supports multiple [programming](https://en.wikipedia.org/wiki/Programming_paradigm) [paradigms,](https://en.wikipedia.org/wiki/Programming_paradigm) including [structured](https://en.wikipedia.org/wiki/Structured_programming) (particularly [procedural](https://en.wikipedia.org/wiki/Procedural_programming)), [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) and [functional](https://en.wikipedia.org/wiki/Functional_programming) [programming.](https://en.wikipedia.org/wiki/Functional_programming) It is often described as a "batteries included" language due to its comprehensive [standard library.Guido van Rossum](https://en.wikipedia.org/wiki/Standard_library) began working on Python in the late 1980s as a successor to the [ABC](https://en.wikipedia.org/wiki/ABC_(programming_language)) [programming language](https://en.wikipedia.org/wiki/ABC_(programming_language)) and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features such as [list comprehensions,](https://en.wikipedia.org/wiki/List_comprehension) [cycle-](https://en.wikipedia.org/wiki/Cycle_detection) [detecting](https://en.wikipedia.org/wiki/Cycle_detection) garbage collection, [reference counting,](https://en.wikipedia.org/wiki/Reference_counting) and [Unicode](https://en.wikipedia.org/wiki/Unicode) support. Python 3.0, released in 2008, was a major revision that is not completely [backward-compatible](https://en.wikipedia.org/wiki/Backward_compatibility) with earlier versions. Python 2 was discontinued with version 2.7.18 in 2020.Python consistently ranks as one of the most popular programming languages.

**KERAS :** Keras is an Open-Source Neural Network library written in Python that runs on top of Theano or TensorFlow. It is designed to be modular, fast and easy to use. It was developed by François Cholet, a Google engineer. Keras doesn't handle low-level computation. Instead, it uses another library to do it, called the "Backend. So Keras is high-level API wrapper for the low-level API, capable of running on top of Tensor Flow, NTK, or Theano. Keras doesn't handle Low-Level API such as making the computational graph, making tensors or other variables because it has been handled by the "backend" engine. Keras High-Level API handles the way we make models, defining layers, or set up multiple input-output models. In this level, Keras also compiles our model with loss and optimizer functions, training process with fit function. Keras doesn't handle Low-Level API such as making the computational graph, making tensors or other variables because it has been handled.

**TENSORFLOW :** TensorFlow is a [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source_software) [software library](https://en.wikipedia.org/wiki/Library_(computing)) for [machine](https://en.wikipedia.org/wiki/Machine_learning) [learning](https://en.wikipedia.org/wiki/Machine_learning) and [artificial](https://en.wikipedia.org/wiki/Artificial_intelligence) [intelligence.](https://en.wikipedia.org/wiki/Artificial_intelligence) It can be used across a range of tasks but has a particular focus on [training](https://en.wikipedia.org/wiki/Types_of_artificial_neural_networks#Training) and [inference](https://en.wikipedia.org/wiki/Statistical_inference) of [deep neural networks.](https://en.wikipedia.org/wiki/Deep_neural_networks)TensorFlow was developed by the [Google Brain](https://en.wikipedia.org/wiki/Google_Brain) team for internal [Google](https://en.wikipedia.org/wiki/Google) use in research and production. The initial version was released under the [Apache License 2.0](https://en.wikipedia.org/wiki/Apache_License_2.0) in 2015. Google released the updated version of TensorFlow, named TensorFlow 2.0, in September 2019.TensorFlow can be used in a wide variety of programming languages, most notably Python, as well as Javascript, C++, and Java. This flexibility lends itself to a range of applications in many different sectors.

**MATPLOTLIB :** Matplotlib is a [plotting](https://en.wikipedia.org/wiki/Plotter) [library](https://en.wikipedia.org/wiki/Library_(computer_science)) for the [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) programming language and its numerical mathematics extension [NumPy.](https://en.wikipedia.org/wiki/NumPy) It provides an [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) [API](https://en.wikipedia.org/wiki/API) for embedding plots into applications using general-purpose [GUI toolkits](https://en.wikipedia.org/wiki/GUI_toolkit) like [Tkinter](https://en.wikipedia.org/wiki/Tkinter), [wxPython,](https://en.wikipedia.org/wiki/WxPython) [Qt](https://en.wikipedia.org/wiki/Qt_(software)), or [GTK.](https://en.wikipedia.org/wiki/GTK) There is also a [procedural](https://en.wikipedia.org/wiki/Procedural_programming) "pylab" interface based on a [state](https://en.wikipedia.org/wiki/State_machine) [machine](https://en.wikipedia.org/wiki/State_machine) (like [OpenGL](https://en.wikipedia.org/wiki/OpenGL)), designed to closely resemble that of [MATLAB,](https://en.wikipedia.org/wiki/MATLAB) though its use is discouraged. [SciPy](https://en.wikipedia.org/wiki/SciPy) makes use of Matplotlib. Matplotlib was originally written by [John](https://en.wikipedia.org/wiki/John_D._Hunter)

[D. Hunter](https://en.wikipedia.org/wiki/John_D._Hunter). Since then it has an active development commun it and is distributed under a [BSD-style license.](https://en.wikipedia.org/wiki/BSD_licenses) Michael Droett boom was nominated as matplotlib's lead developer shortly before John Hunter's death in August 2012 and was further joined by Thomas Caswell. Matplotlib is a [NumFOCUS](https://en.wikipedia.org/w/index.php?title=NumFOCUS&action=edit&redlink=1) fiscally sponsored project.

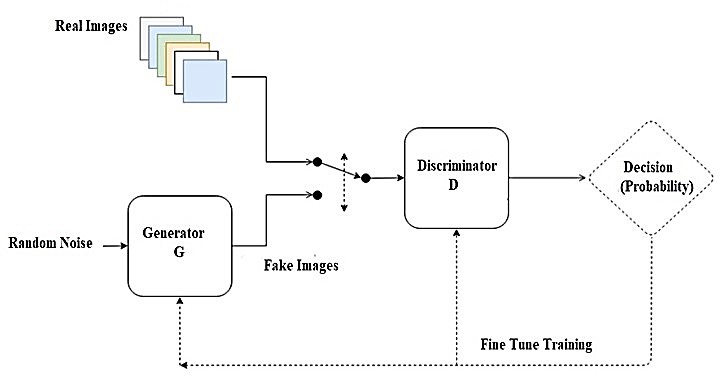
**PIL :** Python Imaging Library is a [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source_software) additional [library](https://en.wikipedia.org/wiki/Library_(computing)) for the [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) [programming language](https://en.wikipedia.org/wiki/Python_(programming_language)) that adds support for opening, [manipulating,](https://en.wikipedia.org/wiki/Image_editing) and saving many different [image file formats.](https://en.wikipedia.org/wiki/Image_file_formats) It is available for [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows), Mac OS X and [Linux.](https://en.wikipedia.org/wiki/Linux) The latest version of PIL is 1.1.7, was released in September 2009 and supports Python 1.5.2 – 2.7.Development of the original project, known for the original PIL in [Linux](https://en.wikipedia.org/wiki/Linux_distribution) [distributions](https://en.wikipedia.org/wiki/Linux_distribution) as PIL, was discontinued in 2011. Subsequently, a successor project named Pillow [forked](https://en.wikipedia.org/wiki/Fork_(software_development)) the PIL repository and added Python 3.x support. This fork has been adopted as a replacement including [Debian](https://en.wikipedia.org/wiki/Debian_GNU/Linux) and [Ubuntu](https://en.wikipedia.org/wiki/Ubuntu) (since [13.04](https://en.wikipedia.org/wiki/Ubuntu_version_history#1304)).

# CHAPTER-4

# SYSTEM DESIGN

# 4.1 DATA FLOW DIAGRAM

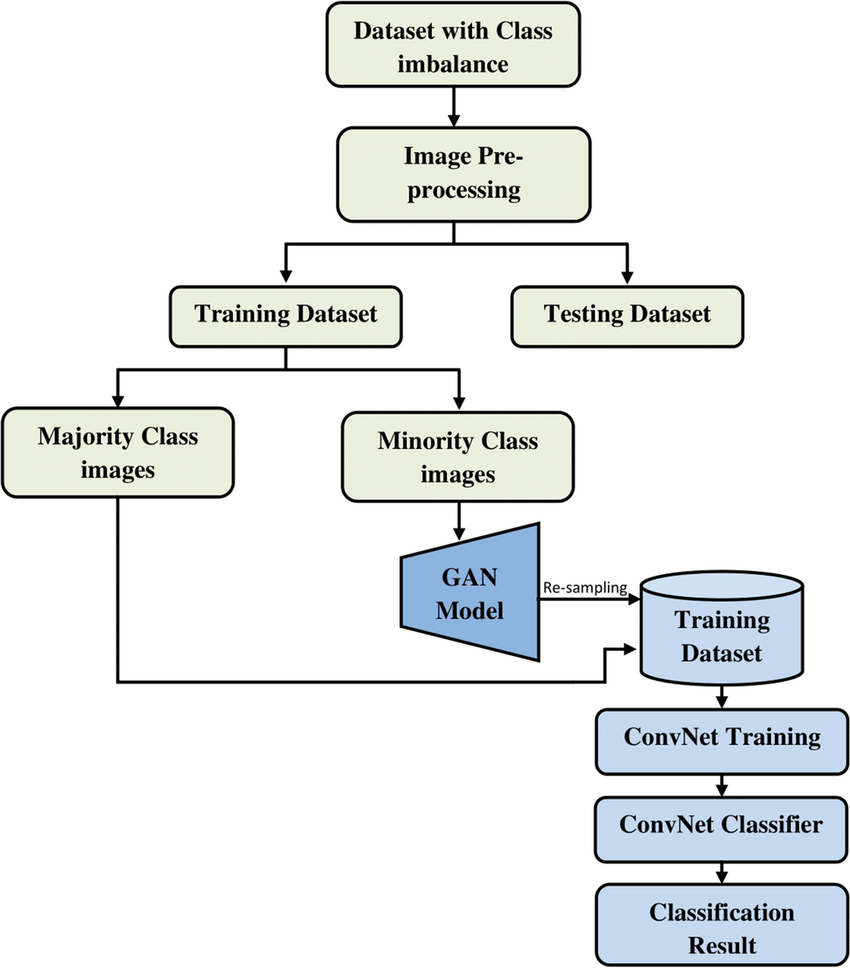
A data-flow diagram is a way of representing a flow of a data of a process or a system. The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow; there are no decision rules and no loops.



### Fig 4.1 Data Flow Diagram

In a block diagram, the grayscale input image flows into the U-Net generator, producing a colorized image. Simultaneously, the ground truth or generated colorized image is fed into the discriminator. The discriminator assesses the realism of the colorized image and provides feedback to both the generator and itself (through backpropagation). The generator learns to produce more realistic colorizations over iterat ions, guided by the adversarial feedback from the discriminator.

## UML DIAGRAMS

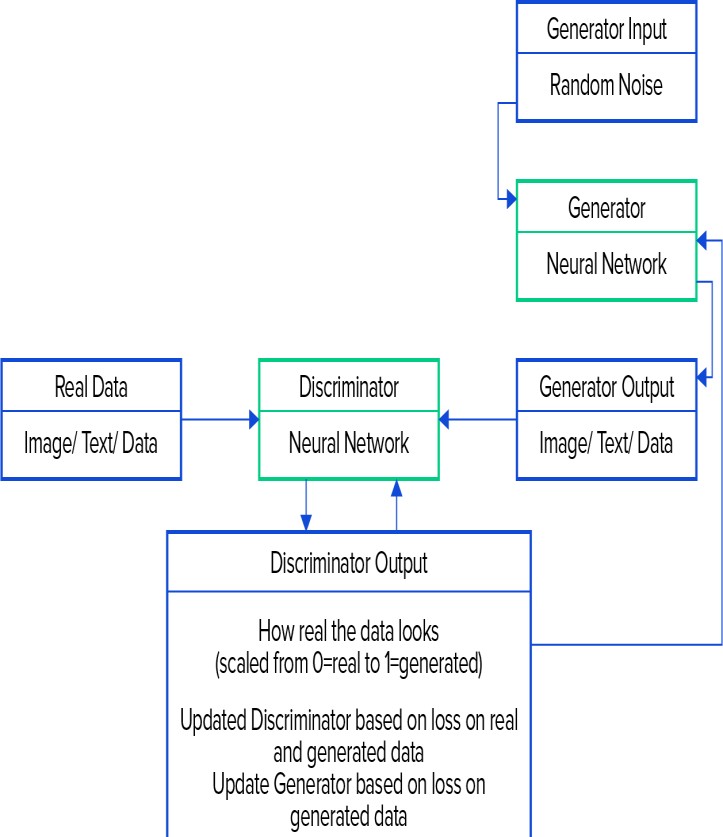


**Fig 4.2 UNET Flow Diagram**

Both the encoder and decoder of the U-Net comprise convolutional layers. Convolutional layers use learnable filters to perform convolution operations on input feature maps. In the encoder, convolutional layers extract features by convolving input feature maps with learned filters, capturing important patterns in the data. In the decoder, convolutional layers help in reconstructing the output image by combining upsampled features and skip connections from the encoder.

## 4.2.1 CLASS DIAGRAM

class diagram in the Unified Modelling Language is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations, and the relationships among objects.

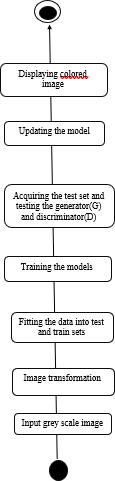


UNET OUTPUT

Higher resolution images obtained.

**Fig 4.2.1 Class Diagram**

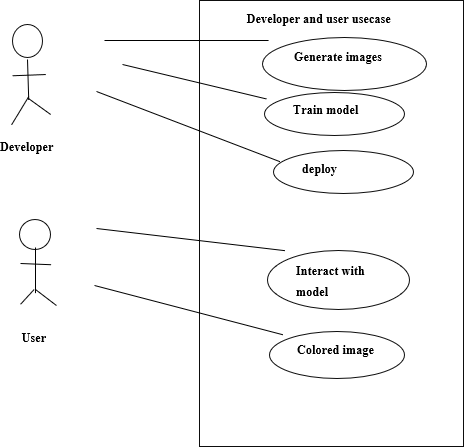
## 4.2.2 STATE CHART DIAGRAM

state diagram is a type of diagram used in computer science and related fields to describe the behaviour of systems. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction

**Fig 4.2.2 State Chart Diagram**

## 4.2.3 USE CASE DIAGRAM

Use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.



**Fig 4.2.3 Use Case Diagram**

The use case diagram for developers and users in a GAN with a UNet model outlines how developers train and optimize the model, debug and test its functionality, while users interact with the system to generate, modify, and evaluate images. Collaboration between developers and users involves feedback loops for model improvement and deployment of the trained system for user interaction. This diagram illustrates the essential activities and information flow in developing, deploying, and using the GAN- UNet system.

# CHAPTER 5

# TESTING

Software Testing is a process of executing the application with intent to find any software bugs. It is used to check whether the application met its expectations and all the functionalities of the application are working. The final goal of testing is to check whether the application is behaving in the way it is supposed to under specified conditions. All aspects of the code are examined to check the quality of the application. The primary purpose of testing is to detect software failures so that defects may be uncovered and corrected. The test cases are designed in such way that scope of finding the bugs is maximum.

## 5.1 TESTING LEVELS

There are various testing levels based on the specificity of the test.

* + - **Unit Testing: -**Unit testing refers to tests conducted on a section of code in order to verify the functionality of that piece of code. This is done at the function level.
    - **Integration Testing: -**Integration testing is any type of software testing that seeks to verify the interfaces between components of a software design. Its primary purpose is to expose the defects associated with the interfacing of modules.
    - **System Testing: -**System testing tests a completely integrated system to verify that the system meets its requirements.
    - **Acceptance Testing: -**Acceptance testing tests the readiness of application, satisfying all requirements.
    - **Performance Testing: -**Performance testing is the process of determining the speed or effectiveness of a computer, network, software program or devices such as response time or millions of instructions per second etc.

## 

## 5.2 SYSTEM TEST CASES

A test case is a set of test data, preconditions, expected results and post conditions, developed for a test scenario to verify compliance with a specific requirement. I have designed and executed a few test cases to check if the project meets the requirements.

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST CONDITION** | **INPUT SPECIFICATION** | **OUTPUT SPECIFICATION** | **PASS / FAIL** |
| Testing with the images in project dataset | Any image from the dataset | Colored version of input image | PASS |

**Table 5.1 Test case for images present in dataset**

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST CONDITION** | **INPUT SPECIFICATION** | **OUTPUT SPECIFICATION** | **PASS / FAIL** |
| Testing with the images in project dataset | Image that is not present in dataset | Colored version of input image | PASS |

### Table 5.2 Test case for images not present in dataset

The testers need to focus on for the following: -

* + - Test with new data, rather than the original training data. If necessary, split your training set into two groups: one that does training, and one that does test. Better, obtain and use fresh data if you are able.
    - Understand the architecture of the network as a part of the testing process. Testers won’t necessarily understand how the neural network was constructed but need to understand whether it meets requirements. And based on the measurements that they are testing, they may have to recommend a radically different approach, or admit the software is just not capable of doing what it was asked to do with confidence.

The data Sets are being tested with following metrics :

**SSIM (Structural Similarity Index):** SSIM is a perceptual metric used to measure the similarity between two images. It takes into account luminance, contrast, and structure, aiming to quantify how similar the structural information of two images is. A higher SSIM value indicates greater similarity between the colorized image and the ground truth color image.

**PSNR (Peak Signal-to-Noise Ratio):** PSNR is a common metric used to measure the quality of reconstructed or compressed images. It compares the peak signal strength of the original image to the noise introduced during compression or reconstruction. PSNR is expressed in decibels (dB), with higher PSNR values indicating higher image quality. While PSNR is a popular metric due to its simplicity and ease of calculation

**MSE (Mean Squared Error):** MSE is a basic measure of the average squared difference between the colorized image and the ground truth color image. It calculates the average squared difference for each pixel in the images, providing a quantitative measure of the overall difference between the images. MSE tends to prioritize reducing large errors, potentially leading to perceptually inferior results.

# CHAPTER-6

# RESULTS

The model is trained with the dataset. The dataset consists of two types of images. They are as follows:

* + - Black and White images
    - Colored images

Each type consists of hundreds of images. The dataset used is image dataset which consists of real time images of persons, places, animals, birds, sceneries etc.

## IMAGE DATASET



### Fig 6.1 Black and White Images & Colored Images

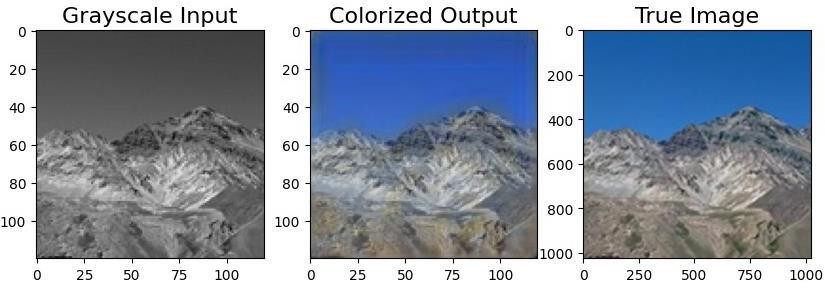
* + - The above example dataset contains black & white and the color version of the images .It includes locations, things, vehicles, humans, nature and all the scenories which we can visualise with our human eye.
    - Here we have added the real time pictures of our batch members to ensure the quality of image produced. We have drawn the results with higher Resolution and Quality.

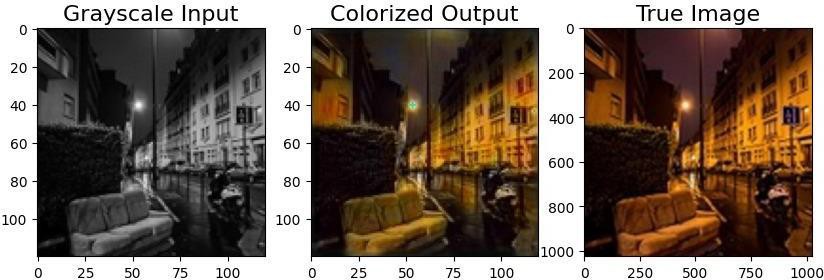


**Fig 6.2: Black & White Image and Colored Image-Test-1**



**Fig 6.3 : Black & White Image and Colored Image-Test-2**

 The results comparing to the original-colored images are show as below:



**Fig 6.4: Output Comparing with True Image**

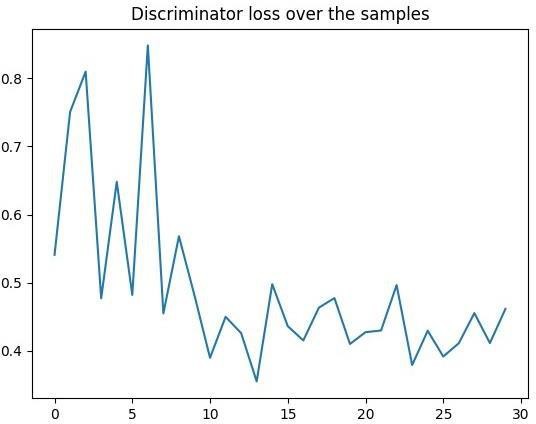
The Output of the given input image is being colored with Gan and Unet model and it is compared with the True Image

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Data**  **set No** | **PSNR (GAN)** | **PSNR (U- NET)** | **SSIM**  **(GAN)** | **SSIM (U-NET)** | **Color Accura-cy (GAN)** | **Color**  **Accura-cy**  **(UNET)** |
| 1 | 24.68dB | 26.45d B | 0.82 | 0.88 | 0.006 | 0.003 |
| 2 | 26.12dB | 27.89d B | 0.85 | 0.90 | 0.004 | 0.002 |
| 3 | 25.33dB | 27.18d B | 0.83 | 0.89 | 0.005 | 0.002 |
| 4 | 25.78dB | 28.00d B | 0.84 | 0.91 | 0.005 | 0.002 |
| 5 | 24.95dB | 26.80d B | 0.81 | 0.87 | 0.006 | 0.003 |

**Fig 6.5 : Result Table**

## ACCURACY / LOSS OF MODEL

The loss of generator and discriminator is as shown below:



**Fig 6.6 : Loss Of Discriminator**

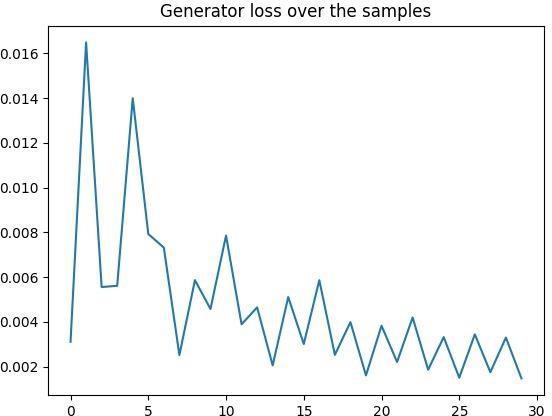
Here the X-axis Represents the training iterations or epochs during the GAN training process. And Y-axis Indicates the value of the discriminator's loss function. This loss is calculated based on the difference between the discriminator's predictions and the actual labels (real or fake).

**The graph typically shows the following trends:**

At the beginning of training, the discriminator's loss is relatively high as the discriminator struggles to differentiate between real and fake samples. As training progresses, the discriminator becomes more adept at distinguishing real from fake samples, leading to a decrease in the discriminator's loss. In ideal scenarios, the discriminator loss converges to a relatively low value, indicating that the discriminator has learned to effectively discriminate between real and generated samples.Y-axis (Loss) Represents the value of the generator's loss function. This value indicates how well the generator is performing in generating realistic data. At the beginning of training, the discriminator's loss is relatively high as the discriminator struggles to differentiate between real and fake samples. As training progresses, the discriminator becomes more adept at distinguishing real from fake samples, leading to a decrease in the discriminator's loss. In ideal scenarios, the discriminator loss converges to a relatively low value, indicating that the discriminator has learned to effectively discriminate between real and generated samples.Y-axis (Loss) Represents the value of the generator's loss function. This value indicates how well the generator is performing in generating realistic data.

As training progresses, the discriminator becomes more adept at distinguishing real from fake samples, leading to a decrease in the discriminator's loss. In ideal scenarios, the discriminator loss converges to a relatively low value, indicating that the discriminator has learned to effectively discriminate between real and generated samples.Y-axis (Loss) Represents the value of the generator's loss function.

At the beginning of training, the discriminator's loss is relatively high as the discriminator struggles to differentiate between real and fake samples. This value indicates how well the generator is performing in generating realistic data.

****

**Fig 6.7: Loss of Generator**

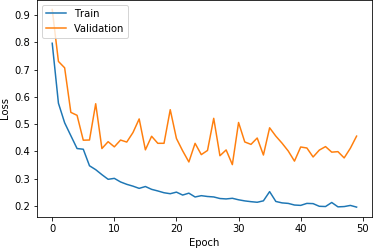
The Graph shows the following:

**Loss Trend:** Initially high and gradually decreasing loss indicates improvement in the generator's ability to generate realistic data as training progresses.

**Convergence:** The graph illustrates whether the generator loss stabilizes or converges to a certain value. Convergence suggests that the generator has effectively learned to produce realistic samples.

**Interpretation:** Fluctuations or spikes in the loss graph may signify training challenges like mode collapse or instability. These issues can inform adjustments in training strategies to enhance GAN performance.

GANs are not the only neural network architecture and training scheme where the objective function is a complex processing of the output layer. Face recognition using triplet loss also has an indirect function (this time not via a separate network). In both this case, and with GANs, you rely on being able to back-propagate through the objective function to get gradients that affect the network you are training. The take away is that objective functions do not have to be limited to direct comparisons of a neural network's output with some target value. They can be more complex than that, and this can allow invention of new types of supervised and semi-supervised training feedback.



**Fig 6.8: Training and Validation Loss**

**Training Graph:** The training graph usually shows how these losses evolve over training epochs. Initially, both the generator and discriminator losses might be high and fluctuating. As training progresses, you would ideally see the generator loss decreasing (indicating that the generator is getting better at generating realistic samples) and the discriminator loss stabilizing (indicating that the discriminator is becoming more accurate in distinguishing real from fake samples).

**Validation Graph:** The validation graph is essential for evaluating the generalization performance of the model on unseen data. It typically involves monitoring a metric such as image quality or similarity between generated and ground truth images. For a GAN with a U-Net generator, you might track metrics like Structural Similarity Index (SSI) or Mean Squared Error (MSE) between generated and target images on a validation set. These metrics should ideally improve or stabilize as training progresses, indicating that the generator is learning to produce more accurate and visually appealing outputs.

# CHAPTER 7

# CONCLUSION AND FUTURE SCOPE

Compared to the other techniques, deep learning-based methods achieve outstanding results. The following conclusions can be drawn in our experiments.

* Although GAN is much difficult to train, the generated images are much better in terms of color brightness and sharpness, whereas images generated by ConvNet have patches of dimmer color caused by averaging the colors at vicinity in L2 loss optimization. GAN is also superior to ConvNet quantitatively by having generated images closer to ground truth in both L1 and L2 distance metrics.
* The project has developed a model for realistic colorization of black and white images, but faces limitations like unexpected red spots, indicating areas for improvement. Despite its strong performance in accurately coloring specific objects and achieving satisfactory results.

## FUTURE SCOPE :

The future scope for Generative Adversarial Networks (GANs) utilizing U-Net architectures is poised for significant advancements across diverse domains. These models hold promise for generating high-fidelity images, particularly in medical imaging applications for tasks like image synthesis and segmentation. Moving forward, GANs with U-Net structures could play a pivotal role in video synthesis, enabling realistic frame interpolation and prediction. Additionally, these models offer potential in interactive content generation, allowing for conditional image synthesis and style transfer. Furthermore, exploring cross-modal applications such as text-to-image synthesis and self-supervised learning could lead to versatile AI systems with broad real-world implications in virtual reality, robotics, and beyond.

# REFERENCES

1. Bo Li, Fuchen Zhao, Zhuo Su, Xiangguo Liang, Yu-Kun Lai, Paul L. Rosin “Example- based Image Colorization using Locality Consistent Sparse Representation” Journal of latex class files, Vol. 13, NO. 9, September 2014.
2. R.C. Gonzalez, R.E. Woods, “Digital Image Processing” second ed., Addison– Wesley Publishing, Reading, MA, 1987.
3. A. Levin, D. Lischinski, and Y.Weiss, “Colorization using optimization,” ACM Transactions on Graphics, vol. 23, no. 3, pp. 689– 694, 2004.
4. Y.-C. Huang, Y.-S. Tung, J.-C. Chen, S.-W. Wang, and J.-L. Wu, “An adaptive edge detection based colorization algorithm and its applications,” in ACM Multimedia, 2005, pp. 351–354.
5. H. Noda, M. Niimi, and J. Korekuni, “Simple and efficient colorization in YCbCr color space,” in International Conference on Pattern Recognition, 2006, pp. 685–688.
6. D. Nie, Q. Ma, L. Ma, and S. Xiao, “Optimization based grayscale image colorization,” Pattern Recognition Letters, vol. 28, pp. 1445– 1451, 2007.
7. X. Ding, Y. Xu, L. Deng, and X. Yang “Colorization Using Quaternion Algebra with Automatic Scribble Generation” Springer-Verlag Berlin Heidelberg 2012. [8]https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-

Convolutional-Neural-Networks

1. Domonkos Varga, Tamas Szir´ anyi “Fully automatic image colorization based on Convolutional Neural Network” 2016 23rd International Conference on Pattern Recognition (ICPR) Cancún Center, Cancún, México, December 4-8, 2016
2. Jeff Hwang, You Zhou “Image colorization with deep convolutional neural networks”

[11] Y.-S. Chia, S. Zhuo, R. K. Gupta, Y.-W. Tai, S.-Y. Cho, P. Tan, and S. Lin, “Semantic colorization with internet images,” ACM Transactions on Graphics, pp. 156:1–7, 2011

[12] D. S`ykora, J. Buri´anek, and J.ˇZ´ara, “Colorization of black-and-white cartoons,” Image and Vision Computing, vol. 23, no. 9, pp. 767–782, 2005

[13] Z. Cheng, Q. Yang, and B. Sheng, “Deep colorization,” in IEEE International Conference on Computer Vision, 2015, pp. 415–423.

Bugeau, V.-T. Ta, and N. Papadakis, “Variational exemplar-based image colorization,” IEEE Transactions on Image Processing, vol. 23, no. 1, pp. 298–307, 2014.

1. F. Pierre, J.-F. Aujol, A. Bugeau, N. Papadakis, and V.-T. Ta, “Luminance- chrominance model for image colorization,” SIAM Journal on Imaging Sciences, vol. 8, no. 1, pp. 536–563, 2015
2. N. Anagnostopoulos, C. Iakovidou, A. Amanatiadis, Y. Boutalis, and S. Chatzichristofis, “Two-staged image colorization based on salient contours,” in IEEE International Conference on Imaging Systems and Techniques, 2014, pp. 381–385.
3. Yatziv and G. Sapiro, “Fast image and video colorization using chrominance blending,” IEEE Transactions on Image Processing, vol. 15, no. 5, pp. 1120–1129, 2006.
4. T. Welsh, M. Ashikhmin, and K. Mueller, “Transferring color to grayscale images,” ACM Transactions on Graphics, vol. 21, no. 3, pp. 277–280, 2002.
5. F. Baldassarre, D. G. Morin, and L. Rodés-Guirao, "Deep koalarization: Image colorization using CNNs and inception-resnet-v2," arXiv, no, June 2017, pp. 1-12, 2017.
6. W. Weng and X. Zhu, "UNet: Convolutional Networks for Biomedical Image Segmentation," IEEE Access, vol. 9, pp. 16591-16603, 2021, doi: 10.1109/ACCESS.2021.3053408.
7. A. Levin, D. Lischinski, and Y.Weiss, “Colorization using optimization,” ACM Transactions on Graphics, vol. 23, no. 3, pp. 689– 694, 2004.
8. Y.-C. Huang, Y.-S. Tung, J.-C. Chen, S.-W. Wang, and J.-L. Wu, “An adaptive edge detection based colorization algorithm and its applications,” in ACM Multimedia, 2005, pp. 351–354.
9. H. Noda, M. Niimi, and J. Korekuni, “Simple and efficient colorization in YCbCr color space,” in International Conference on Pattern Recognition, 2006, pp. 685–688.
10. D. Nie, Q. Ma, L. Ma, and S. Xiao, “Optimization based grayscale image colorization,” Pattern Recognition Letters, vol. 28, pp. 1445– 1451, 2007.
11. X. Ding, Y. Xu, L. Deng, and X. Yang “Colorization Using Quaternion Algebra with Automatic Scribble Generation” Springer-Verlag Berlin Heidelberg 2012. [8]https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding- Convolutional-Neural-Networks.

# 

# APPENDIX

**\*\*Import Library\*\***

import glob

import numpy as np from PIL import Image import torch

from torch import nn, optim

from torchvision import transforms

from torchvision.utils import make\_grid

from torch.utils.data import Dataset, DataLoader

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu") use\_colab = None

**#\*\*Split data to Train/Test set\*\***

from sklearn.model\_selection import train\_test\_split def split\_data(inputs,labels,test\_size=0.2):

labels=np.array(labels) inputs=np.array(inputs)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(inputs, labels, test\_size=test\_size) return x\_train, x\_test ,y\_train,y\_test

x\_train,x\_test ,y\_train, y\_test=split\_data(inputs,labels,test\_size=0.2) x\_train, x\_test=x\_train/255.0,x\_test/255.0

y\_train, y\_test=y\_train/255.0,y\_test/255.0 #\*\*Build model GAN-Unet\*\*

**##\*\*1. Build GENERATOR(Unet)\*\***

import tensorflow as tf

from tensorflow.keras.layers import BatchNormalization, Conv2D, Conv2DTranspose, Dense, Flatten, Dropout, UpSampling2D, Concatenate, ELU, Input, LeakyReLU,

MaxPooling2D, Reshape, UpSampling2D

from tensorflow.keras.models import Model

x = Conv2D(128, (3, 3), padding="same", kernel\_initializer='he\_normal', name='Conv4')(x)

x = BatchNormalization()(x) x = LeakyReLU()(x)

x = Conv2D(128, (3, 3), padding="same", kernel\_initializer='he\_normal', name='Conv5')(x)

import tensorflow.keras.layers as tfl

from tensorflow.keras.optimizers import Adam from tensorflow.keras.layers import Concatenate #GEN 128x128

def build\_generator(inputsize=(128, 128, 1)): input = Input(shape=inputsize)

x = Conv2D(64, (3, 3), padding="same", kernel\_initializer='he\_normal', name='Conv1')(input)

x = BatchNormalization()(x) x = LeakyReLU()(x)

x = Conv2D(64, (3, 3), padding="same", kernel\_initializer='he\_normal', name='Conv2')(x)

x = BatchNormalization()(x) x = LeakyReLU()(x)

x1 = Conv2D(64, (3, 3), padding="same", kernel\_initializer='he\_normal', name='Conv3')(x)

x = BatchNormalization()(x1) x = LeakyReLU()(x)

x = MaxPooling2D(pool\_size=(2, 2), strides=(2, 2), name='MaxPool1')(x)

x = Conv2D(128, (3, 3), padding="same", kernel\_initializer='he\_normal', name='Conv4')(x)

x = BatchNormalization()(x) x = LeakyReLU()(x)

x = Conv2D(128, (3, 3), padding="same", kernel\_initializer='he\_normal', name='Conv5')(x)

x = BatchNormalization()(x) x = LeakyReLU()(x)

x2 = Conv2D(128, (3, 3), padding="same", strides=(1, 1), kernel\_initializer='he\_normal', name='Conv6')(x)

x = Conv2D(128, (3, 3), padding="same", kernel\_initializer='he\_normal', name='Conv4')(x)

x = BatchNormalization()(x) x = LeakyReLU()(x)

x = Conv2D(128, (3, 3), padding="same", kernel\_initializer='he\_normal', name='Conv5')(x)

x = BatchNormalization()(x) x = LeakyReLU()(x)

x2 = Conv2D(128, (3, 3), padding="same", strides=(1, 1), kernel\_initializer='he\_normal', name='Conv6')(x)

d\_loss\_real = discriminator.train\_on\_batch(real\_images, valid) d\_loss\_fake = discriminator.train\_on\_batch(fake\_images, fake) d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

valid = np.ones((batch\_size, 1))

generator\_model.summary() return generator\_model

**##\*\*2. Build DISCRIMINATOR(CNN)\*\***

def build\_discriminator():

activation = tf.keras.layers.LeakyReLU(alpha=0.2) inputs = Input(shape=(SIZE, SIZE, 3))

conv1 = Conv2D(64, kernel\_size=(3, 3), strides=1, padding='same', kernel\_initializer='he\_normal')(inputs)

conv1 = BatchNormalization()(conv1) conv1 = activation(conv1)

conv1 = Conv2D(64, kernel\_size=(3, 3), strides=1, padding='same', kernel\_initializer='he\_normal')(conv1)

conv1 = BatchNormalization()(conv1) conv1 = activation(conv1)

conv1 = MaxPooling2D()(conv1)

conv2 = Conv2D(128, kernel\_size=(3, 3), strides=1, padding='same', kernel\_initializer='he\_normal')(conv1)

conv2 = BatchNormalization()(conv2) conv2 = activation(conv2)

conv2 = Conv2D(128, kernel\_size=(3, 3), strides=1, padding='same', kernel\_initializer='he\_normal')(conv2)

conv2 = BatchNormalization()(conv2) conv2 = activation(conv2)

conv2 = MaxPooling2D()(conv2) fully = BatchNormalization()(fully) fully = activation(fully)

fully = Dense(32)(fully)

fully = Dense(1, activation='sigmoid')(fully) model = tf.keras.models.Model(inputs, fully)

model.compile(loss='binary\_crossentropy',optimizer=tf.keras.optimizers.Adam(learning\_rate

=0.0002, beta\_1=0.5), metrics=['accuracy']) model.summary()

return model

discriminator = build\_discriminator()

**##\*\*3. Build a complete GAN model from the two Generator and Discriminator models above\*\***

def build\_gan(generator, discriminator): discriminator.trainable = False

input = tf.keras.Input(shape=(SIZE, SIZE, 1)) generated\_image = generator(input)

validity = discriminator(generated\_image)

gan = tf.keras.Model(inputs=input, outputs=[generated\_image, validity]) gan.compile(loss=['mse', 'binary\_crossentropy'], loss\_weights=[0.9999, 0.0001],

optimizer='adam') gan.summary() return gan

gan = build\_gan(generator, discriminator)

**#\*\*Train model\*\***

import time

def train\_gan(generator, discriminator, gan, x\_train, y\_train, epochs=1000, batch\_size=64): batch\_count = x\_train.shape[0] // batch\_size

d\_loss\_list = [] g\_loss\_list = []

for e in range(epochs):

start\_time = time.time() # Time start for \_ in range(batch\_count):

idx = np.random.randint(0, x\_train.shape[0], batch\_size) real\_images = y\_train[idx]

gray\_images = x\_train[idx]

fake\_images = generator.predict(gray\_images)

valid = np.ones((batch\_size, 1)) fake = np.zeros((batch\_size, 1))

d\_loss\_real = discriminator.train\_on\_batch(real\_images, valid) d\_loss\_fake = discriminator.train\_on\_batch(fake\_images, fake) d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

valid = np.ones((batch\_size, 1))

g\_loss = gan.train\_on\_batch(gray\_images, [real\_images, valid]) end\_time = time.time() # Time end

elapsed\_time = end\_time - start\_time

print(f"Epoch {e}/{epochs}, D Loss: {d\_loss[0]}, G Loss: {g\_loss[0]}, Time:

{elapsed\_time:.2f} seconds")

**# Save loss** d\_loss\_list.append(d\_loss[0]) g\_loss\_list.append(g\_loss[0])

# Compute remaining time

remaining\_time = (epochs - e - 1) \* elapsed\_time hours = int(remaining\_time // 3600)

minutes = int((remaining\_time % 3600) // 60)

print(f"Estimated time remaining: {hours} hours {minutes} minutes")

return d\_loss\_list, g\_loss\_list

print(f"Epoch {e}/{epochs}, D Loss: {d\_loss[0]}, G Loss: {g\_loss[0]}, Time:

{elapsed\_time:.2f} seconds")

def plot\_loss(d\_loss, g\_loss, num\_epochs): plt.figure(figsize=(10, 5))

plt.plot(range(num\_epochs), d\_loss, label="Discriminator Loss") plt.plot(range(num\_epochs), g\_loss, label="Generator Loss") plt.xlabel("Epochs")

plt.ylabel("Loss") plt.legend()

plt.show()

plot\_loss(d\_loss\_list, g\_loss\_list, num\_epochs)

**#\*\*Show results\*\***

def generate\_colored\_images(generator, gray\_images): colored\_images = generator.predict(gray\_images) return colored\_images

num\_examples\_to\_generate = len(x\_test)

generated\_images = generate\_colored\_images(generator, x\_test) def result\_compare(y\_test, pred, x\_test):

num\_samples = len(y\_test)

for idx in range(num\_samples): fig = plt.figure()

ax1 = fig.add\_subplot(1, 3, 1) ax1.imshow(x\_test[idx], cmap='gray') ax1.set\_title('Input')

ax2 = fig.add\_subplot(1, 3, 2) ax2.imshow(pred[idx]) ax2.set\_title('Output')

ax3 = fig.add\_subplot(1, 3, 3) ax3.imshow(y\_test[idx]) ax3.set\_title('Original image') plt.show()

result\_compare(y\_test, generated\_images, x\_test)

**#\*\*Save model to deploy to website\*\***

from tensorflow.keras.models import load\_model generator.save('generator\_model\_128.h5')

from google.colab import files

file\_path = '/content/generator\_model\_128.h5' files.download(file\_path).