Colouring B/W Hubble Telescope Images using Pix2Pix and Cycle GAN

Contents

Abstract

Telescopic images provide a window into the vast and intricate universe, capturing the light from celestial bodies millions of light-years away. These images, obtained through advanced optical, reveal details about stars, galaxies, nebulae, and other astronomical phenomena. By analysing the data embedded in these images, scientists can infer cosmic objects' composition, distance, and motion, offering insights into the fundamental processes that govern the universe. However, these images may be distorted due to data processing, or environmental conditions. One of the methods to attain this problem is using machine learning models called Generative Adversarial Networks (Nets). This report presents the use of the Pix2Pix Generative Adversarial Network (GAN) for colorizing black-and-white images captured by the Hubble Space Telescope. Pix2Pix is a neural network designed for image- to-image translation tasks requiring paired datasets. Additionally, we explore CycleGAN, which does not need paired datasets and uses two generators and two discriminators for bidirectional translation between domains, maintaining image integrity through cycle consistency loss. This method is especially effective for enhancing unpaired telescopic images, preserving essential attributes while correcting distortions.

Keywords: Machine learning models, Generative Adversarial Networks, Image-to-Image translation, Pix2Pix GAN, CycleGAN

3 Introduction

The galaxy serves as the site of the origin of stars and encompasses a diverse array of interstellar objects and the field of astronomy provides us with the opportunity to explore essential inquiries regarding the characteristics and beginnings of the cosmos, such as the Big Bang hypothesis, the development of galaxies and stars, and the presence of black holes. Telescopic observations play a crucial role in the exploration and understanding of exoplanets, shedding light on planetary systems that exist outside our own solar system and the possibility of life beyond Earth. By capturing images through telescopes, scientists are able to delve into the intricacies of stellar evolution, the formation of galaxies, and various cosmological phenomena. These observations not only contribute valuable data for theoretical models and simulations but also offer profound insights into the vastness and diversity of the universe. It also inspires a profound sense of amazement and fascination, prompting contemplation of our position within the vast universe and sparking a desire to unravel the enigmas of our existence. One of the methods to study about the universe is optical astronomy. However, many a times the telescopic images obtained distorted due to various atmospheric disturbances. This problem can be addressed using Generative Adversarial Networks (GANs).

4 Geneartive Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are a class of machine learning frameworks designed by Ian Goodfellow and his colleagues in 2014. GANs consist of two neural networks, the generator and the discriminator, which contest with each other in a game-theoretic framework.(Goodfellow et al., 2014)

The Generator (G) is responsible for creating new data instances that resemble a given dataset. It starts with random noise and, through a multi-layered perceptron, learns to generate increasingly realistic data. Its goal is to create samples that are indistinguishable from real data. The Discriminator (D) evaluates the generated samples and real data. It is trained to distinguish between the two. The discriminator's objective is to correctly identify whether a given instance is real or generated. Thus the objective is to identify a potential arrangement in which G and D reach a state of equilibrium, meaning that G produces data that closely resembles real data to the extent that D cannot differentiate between the two and assigns equal probabilities of half to each outcome(Rath, 2020).

The adversarial nature of GANs arises from the continuous feedback loop between the generator and discriminator. The training process for G aims to increase the likelihood of D committing errors. This structure aligns with a minimax game involving two players. In simpler words, the generator's job is to create an image that the discriminator thinks came from the real data whereas the discriminator's job is to distinguish between the real and fake data. Since a game-theoretic approach is taken, our aim is to express the objective function as a minimax function. The discriminator tries to maximize the objective function, therefore we can perform gradient ascent on the objective function. The generator strives to minimize the objective function, thereby allowing us to carry out gradient descent on the objective function. Through the alternating application of gradient ascent and descent, the network can undergo training.

Let G_ϕ be the generator and D_ϑ be the discriminator (ϕ and ϑ are parameters of G and D respectively).The neural network-based generator takes a noise vector $z = N(0, 1)$ as input and produces $G_\phi = X$. Similarly, the neural network based discrimination takes real X or a generated $X = G_\phi(z)$ as input and classifies it as real or fake.

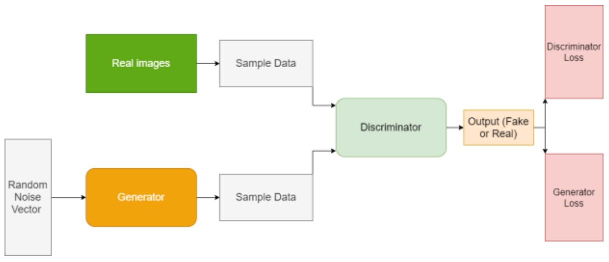


Figure 4.0.1: Working of a GAN. [1]

[1]Image credit: https://debuggercafe.com/introduction-to-generative-adversarial-networks-gans

Given an image generated by the generator as $G_\phi(z)$ the discriminator assigns a score $D_\vartheta(G_\phi(z))$ to it. This score will be between 0 and 1 and will tell us the probability of the image being real or fake. For a given z, the generator would want to maximize $log D_\vartheta(G\phi(z))$ or minimize $log(1 - D_\vartheta(G\phi(z))$, i.e the generator will want to maximize its score given by the discriminator. This is all for a single z, the generator will like to do this for all z values. If z is discrete (i.e not continuous) then generally we find the loss function for one training sample, sum it over all the training samples and divide by N

$$\min_{\phi} \sum_{i=1}^{N} \frac{1}{N} \log(1 - D_{\vartheta}(G_{\phi}(z)))$$

This however won't work in our case since z is continuous and comes from a normal distribution (non-uniform).

$$\min_{\phi} \int p(z) \log(1 - D_{\vartheta}(G_{\phi}(z)))$$

$$\min_{\phi} E_{p(z)} [\log(1 - D_{\vartheta}(G_{\phi}(z)))]$$

Therefore the summation is replaced by integration and $1/N$ which is the normal probability is replaced by $p(z)$ which is uniform probability. Similarly, for the discriminator, we get

$$\max_{\vartheta} \mathrm{E}_{x \sim \text{data}} \log D_{\vartheta}(x) + \mathrm{E}_{z \sim p(z)} [\log(1 - D_{\vartheta}(G_{\phi}(z)))]$$

Finally, by combining all the equations we get,

$$\min_{G} \max_{D} V(D; G) = \mathrm{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathrm{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

The discriminator wants to maximize the second term whereas the generator wants to minimize it, hence a two-player game (Goodfellow et al., 2014).

In the early stages of training, the generator is not so good at generating new data instances. At this stage, it is very easy for the discriminator to classify the fake data and real data. Here, $\log(1 - D(G(z)))$ saturates and the generator does not learn anything. To avoid this, instead of minimizing $\log(1 - D(G(z)))$, we try to maximize $\log D(G(z))$. This will provide better gradients in the early stages of training and will not lead to any saturation.

5 Proposed Solution

Image-to-image translation is the process of converting an image from one domain to another while maintaining its fundamental attributes. This can include transforming images from one style to another, altering the visual aspects of objects within an image, or converting images between different modalities, such as from sketches to photographs. Conditional Generative Adversarial Networks (cGANs) have emerged as a robust method for tackling this objective.

Conditional Generative Adversarial Networks (cGANs) enhance the GAN framework by incorporating conditional information, which enables a higher level of control over the image generation procedure. In image-to-image translation scenarios, cGANs utilize both the input image and a conditional label as input to produce the desired output image. By including the conditional label, the network gains supplementary information or context for the translation process, resulting in more precise and contextually appropriate outcomes (Mirza & Osindero, 2014).

Why do we need conditional GANs for image-to-image translation?

1. **Semantic Control**: cGANs allow for semantic control over the image generation process. By providing conditional labels, we can specify desired attributes or characteristics of the output image, such as the style, color, or appearance of objects within the image. This enables more targeted and meaningful translations between different image domains.

2. **Preservation of Structure**: Conditional information plays a crucial role in maintaining the structural integrity of the input image during the process of translation. To illustrate, when converting a sketch into a photograph, the conditional label serves as a guiding mechanism for the network to uphold the overall shape and structure of objects, while simultaneously incorporating authentic details and textures.

3. **Improved Quality and Realism**: By integrating conditional information, cGANs have the capability to generate output images that possess both visual appeal and contextual consistency with the input and desired output domains. Consequently, this results in image translations that are more realistic and of superior quality when compared to conventional GANs that do not incorporate conditional input.

4. **Flexible and Adaptable**: Conditional GANs exhibit a remarkable level of flexibility and adaptability when it comes to a diverse array of image translation tasks. Through the straightforward adjustment of the conditional label, the identical network architecture can be effectively employed to execute numerous forms of image transformations, including but not limited to style transfer, colorization, and image inpainting. This versatility is achieved with minimal alterations, highlighting the efficiency and versatility of Conditional GANs in addressing various image translation requirements.

6 Review of the State-of-the-Art used

6.1 Pix2Pix GAN

Pix2Pix GAN, also known as Pix2Pix, is a specialized version of conditional Generative Adversarial Networks (cGANs) that is tailored for image-to-image translation assignments. Developed by Phillip Isola and his team at UC Berkeley Isola et al., 2016. Pix2Pix enhances the fundamental GAN structure by integrating conditional data, which enables accurate manipulation of the image creation procedure. In our case, we are trying to convert input b/w images into colorized output images. The generator model takes the black and white image as its input, resulting in the generation of a pair of images consisting of the generated output and the original black and white input. This pair is considered the fake pair. On the other hand, the real pair is formed by the black and white input image and its corresponding target output, which represents the true color version of the input black and white image. The discriminator in Pix2Pix is responsible for classifying a given pair of images as either real or generated. However, the way it functions differs from what we typically anticipate from a classifier's output. Instead of providing a single classification for the entire input image pairs, the Pix2Pix discriminator produces an output classification that categorizes multiple patches within the image pairs, known as patchGAN.

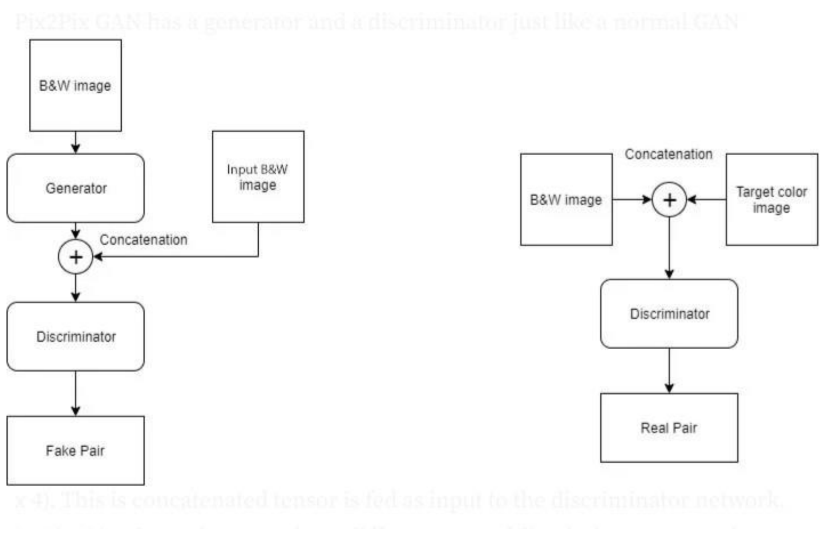

Pix2Pix GAN has a generator and a discriminator just like a normal GAN
B&W image
Generator
Input B&W image
Concatenation
Discriminator
Fake Pair
Concatenation
B&W image
Target color image
Discriminator
Real Pair
x 4). This is concatenated tensor is fed as input to the discriminator network.

Figure 6.1.1: Pix2Pix architecture.
2

The Pix2Pix GAN architecture consists of both a generator and a discriminator, mirroring the structure of a typical GAN. Specifically designed for black and white image colorization, the generator model processes the input B&W image to generate a colored version as the output. Within the Pix2Pix framework, the generator is

[2]Image credit: https://towardsdatascience.com/gan-pix2pix-generative-model-c9bf5d691bac

implemented as a convolutional network utilizing U-net architecture. The grayscale, single-channel input image is processed through a series of convolution and up-sampling layers to create an output image with three color channels that matches the size of the original input. Prior to training, the generator generates random output. After the generator, the synthetic image is concatenated with the input B&W image. Consequently, the resulting tensor will have four color channels (height x width x 4). This combined tensor is then utilized as the input for the discriminator network. In the Pix2Pix framework, a distinct discriminator network type, known as patchGAN, is utilized by the authors. The patchGAN network processes the combined input images and generates an output of size $N \times N$.

Generator and Discriminator loss function

1. **Discriminator loss**

   The discriminator loss function quantifies the accuracy of the discriminator's predictions, indicating whether they are good or bad. As the discriminator loss decreases, the discriminator's ability to correctly identify synthetic image pairs improves. A typical binary classifier utilized in GANs generates a solitary output neuron to determine authenticity. Conversely, the patchGAN's $N \times N$ output forecasts numerous overlapping patches within the input image. For instance, in Pix2Pix, the output dimensions are $30 \times 30 \times 1$, forecasting for every 70×70 patch of the input. Further insights on patchGANs will be explored in a subsequent post. The 30×30 output is inputted into a logarithmic loss function for comparison with a 30×30 zero matrix (as it is generated and not authentic). This phenomenon is referred to as generated loss. The genuine loss is computed for the combination of a black and white image and its corresponding colored image from the dataset. This pairing is considered authentic. Therefore, the term 'real loss' denotes the sigmoid cross-entropy between the $N \times N$ output and a matrix of ones with the same dimensions. The cumulative discriminator loss is calculated by adding the aforementioned two losses together. The gradients of the loss function are determined in relation to the discriminator network and are then backpropagated in order to reduce the loss. As the discriminator loss is being back-propagated, the weights of the generator network remain fixed.

2. **Generator Loss**

   The generator loss quantifies the authenticity of the artificial images. By reducing this metric, the generator has the potential to generate images that closely resemble real-life visuals. The loss incurred is similar to the loss generated, with the distinction being the utilization of the sigmoid cross-entropy between the $N \times N$ discriminator output and a matrix consisting of ones. During the back-propagation of this loss, the parameters of the discriminator network remain fixed, while solely the weights of the generator are modified.

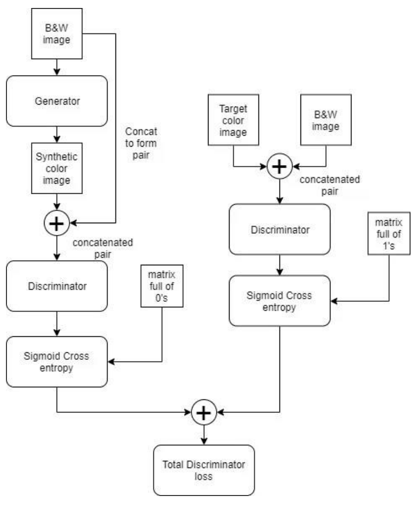


Figure 6.1.2: Generator and Discriminator.

6.2 CycleGAN

CycleGAN, short for Cycle-Consistent Generative Adversarial Network,(Zhu et al., 2020) is a type of deep learning model designed for image-to-image translation tasks without the need for paired examples. Introduced by Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros in their 2017 paper titled ”Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks,” CycleGAN has become a seminal work in the field of generative adversarial networks (GANs) and image transformation. The core idea of CycleGAN is the cycle consistency loss, which ensures that an image translated from domain X to domain Y and then back to domain X should return to the original image. This is enforced by having two sets of generators and discriminators.
The architecture of CycleGAN model consists of two generator models: one generator (Generator-A) for generating images for the first domain (Domain-A) and the second generator (Generator-B) for generating images for the second domain (Domain-B). These geenrators serves as the mapping functions while translating from one domain to another.

In this project, first domain can be refereed as original, colourized images and second domain can be referred as grey-scale versions of the colourized images. Each generator in CycleGAN has an associated discriminator model (Discriminator-A and Discriminator-B). These discriminator models evaluate real images from their respective domains and the images generated by the generators, determining whether they are genuine or artificially created.

Structure and Essentials related to CycleGAN model:

1. **Generators :**In the CycleGAN model, there are two key generators that perform image-to-image translation between two distinct domains:

    (a) **Generator1 (G)**: This generator is responsible for transforming images from domain A (grayscale images) into domain B (color images). The primary task of Generator1 is to learn the mapping $G : A \rightarrow B$ ensuring that the generated color images are realistic and visually consistent with real images from domain B.

    (b) **Generator2 (F)**: This generator is responsible for transforming images from domain B (Colorued, Original images) into domain B (greyscale images). The primary task of Generator1 is to learn the mapping $F : B \rightarrow A$ ensuring that the generated color images are realistic and visually consistent with real images from domain A.

    The structure of Gnerator is described as follows:

    (a) **Input Layer:** Accepts an input image of shape [256, 256, 3].
    (b) **Down-sampling stack:** Applies several downsampling layers to reduce the spatial dimensions.
    (c) **U[sampling Stack:** Applies several upsampling layers to restore the spatial dimensions and incorporates skip connections from the downsampling layers.
    (d) **Output Layer:** Uses a transposed convolution layer with tanh activation to generate the output image.

2. **Discriminators:** Each generator is paired with a discriminator model. Discriminator-A assesses the authenticity of images from domain A (both real and generated by Generator2), while Discriminator-B evaluates images from domain B (both real and generated by Generator1). These discriminators help the generators improve by providing feedback on the realism of the generated images. The structure of Discriminators is as follows:

    (a) **Input layer:** Accepts an input image of shape [256, 256, 3].
    (b) **Down-sampling Layers:** Applies several downsampling layers to reduce the spatial dimensions.
    (c) **Zero Padding:** Adds padding to the input before applying the final convolution.
    (d) **Conv2D Layer:** Adds a convolutional layer to process the padded input.
    (e) **Instance Normalization:** Applies instance normalization.
    (f) **Leaky ReLu:** Adds a Leaky ReLU activation function.
    (g) **Zero Padding:** Adds another padding layer.
    (h) **Final Conv2D Layer :** Adds a final convolutional layer to produce the output.

[3]Image credit: https://miro.medium.com/v2/resize:fit:640/format:webp/1*BaXc1zu6okoMIeKbZJ9rIw.jpeg

(i) **Leaky ReLu and Flatten:** Applies Leaky ReLU and flattens the output.

(j) **Dense Layer:** Adds a dense layer with sigmoid activation to produce a single output value.

3. **Cycle Consistency Loss:** To maintain the structural integrity of the original images, CycleGAN employs cycle consistency loss. This ensures that an image, when transformed to the other domain and back, remains unchanged. This loss is crucial for preserving the content and identity of the images through the translation cycles. The cycle consistency loss is defined as:
$L_{cyc}(G, F) = E_{x \backsim p_{data}(x)}[||F(G(x)) - x||_1 + E_{y \backsim p_{data}(y)}[||G(F(x)) - y||_1]$

4. **Adversarial Loss:** Adversarial Loss in CycleGAN plays a crucial role in training the generators to produce images that are indistinguishable from real images in their respective domains. It operates by creating a min-max game between the generators and their corresponding discriminators. For Generator1, which transforms grayscale images to color, Discriminator-B evaluates how realistic the generated color images are compared to actual color images. Similarly, for Generator2, which converts color images back to grayscale, Discriminator-A assesses the authenticity of the generated grayscale images. The adversarial loss penalizes the generators when the discriminators correctly identify generated images as fake, thus pushing the generators to improve their output quality. This continuous feedback loop drives the generators to produce high-quality, realistic images that can effectively fool the discriminators into believing they are real. Consequently, adversarial loss is essential for refining the performance of the generators, ensuring the translated images retain high fidelity and visual coherence with genuine images from their target domains. This loss is defined as
$LL_{GAN}(G, D_Y, X, Y) = E_{y \backsim p_{data}(y)}(\log(D_Y) + E_{x \backsim p_{data}(x)}[\log(1 - D_Y(G(x))]$

5. **Unpaired Training:** Unlike other GANs that require paired training data, CycleGAN can learn mappings using unpaired datasets. This flexibility is highly advantageous for applications where paired data is unavailable, such as in artistic style transfer or historical photo colorization. By leveraging unpaired data, CycleGAN opens up possibilities for a wider range of image-to-image translation tasks, making it a powerful tool for scenarios where collecting paired examples is difficult or impossible. This adaptability significantly broadens the scope and applicability of CycleGAN in various real-world applications.

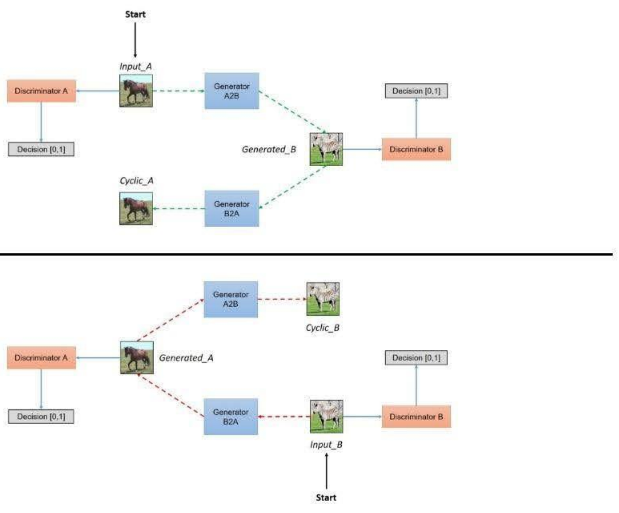


Figure 6.2.1: Generator and Discriminator of CycleGAN.
[4]

[4]Image credit: https://hardikbansal.github.io/CycleGANBlog/

7 Methodology

For our model, we used the dataset from Kaggle [5] which is the collection of the top 100 most popular images taken by the Hubble Telescope. Note before following the process below, please ensure that the versions of Tnesorflow and Keras are compatible to the environment, as well as they have same versions. For cycleGAN these references are being used CULEBRAS, 2020,HAGHJOU, 2023, DRAMSCH, 2020

The process for both Pix2Pix and CycleGAN model is as follows,

1. Firstly we imported all the necessary libraries for our model and downloaded the dataset.
2. Next process was to resize and convert images from *.tif* to *.jpegformat*. Since TensorFlow does not work well with *.tif* images hence we call a function to convert the images and resize them to prevent memory issues and maintain consistency.
3. Since we will require the real images for the training part we converted them to tensors and stored them separately.
4. Similarly, called a function to convert the real images into black and white images, convert them to tensors, and store them separately.
5. Then we visualized the data to check if the images have been correctly stored.
6. Then performed splitting of the dataset into training, testing, and validation sets (Mart´ın Abadi et al., 2015).
7. Then we defined the U-NET architecture and along with that defined the generator and discriminator (Patch GAN).
8. After that we tested the generator and discriminator, defined optimizers, and ran the model to generate images.

[5]Dataset: https://www.kaggle.com/datasets/redwankarimsony/top-100-hubble-telescope-images

8 Experimental Results & Discussion

8.1 Image outputs: Pix2Pix GAN

We ran some iterations of our model with varying values for optimizer learning rate and batch size to find the optimum PSNR value,

Steps	Learning Rate	Batch Size 1	Batch Size 10	Batch Size 25
40k	$2e^{-4}$	19.02	18.97	18.95
40k	$2e^{-3}$	18.77	18.72	18.68
40k	$2e^{-2}$	8.25	8.19	8.11
80k	$2e^{-4}$	**19.55**	19.50	19.48

Table 1: PSNR values for various learning rates and batch sizes

Below are the examples of output images obtained for batch size = 1, LR stands for learning rate.

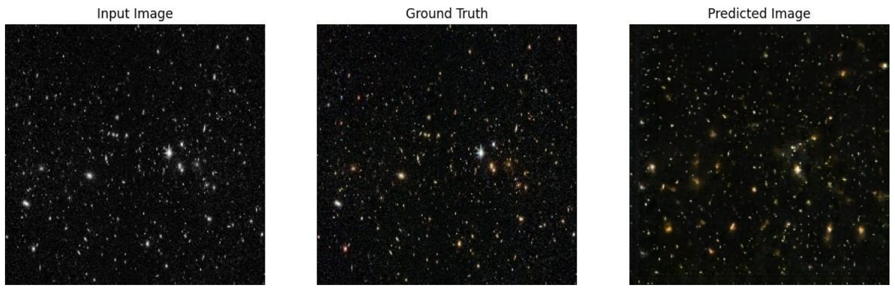


Figure 8.1.1: LR = $2e^{-4}$ and batch size = 1, steps = 40k

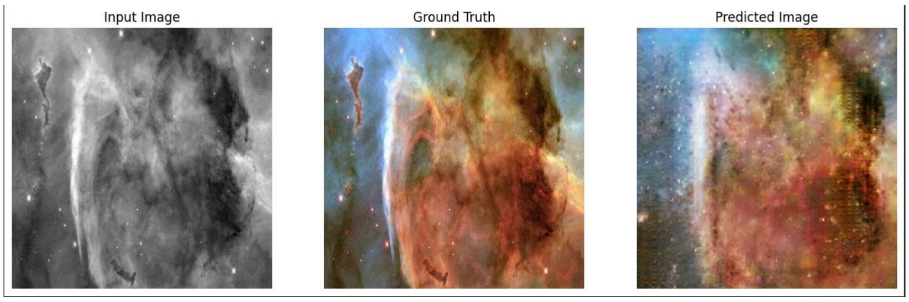


Figure 8.1.2: LR = $2e^{-3}$ and batch size = 1, steps = 40k

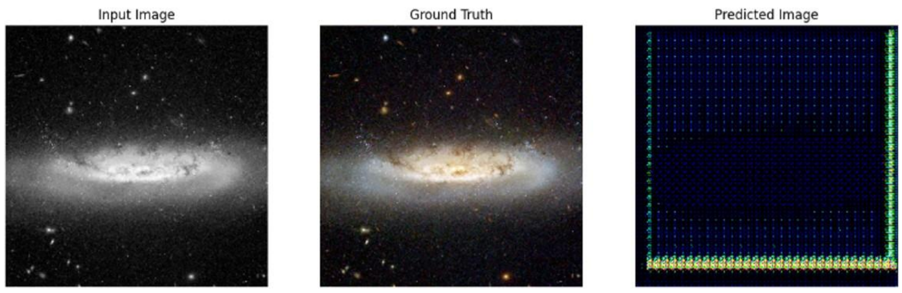


Figure 8.1.3: LR = $2e^{-2}$ and batch size = 1, steps = 40k

All the above examples were trained on 40k steps because training for more than 40k turned out to be computationally challenging. Since for LR = $2e^{-4}$ and batch size = 1 had the best PSNR value, we decided to try training for more steps. Therefore we saved the model state at 40k steps and then again trained for another 40k step to see if there was any difference in the output.

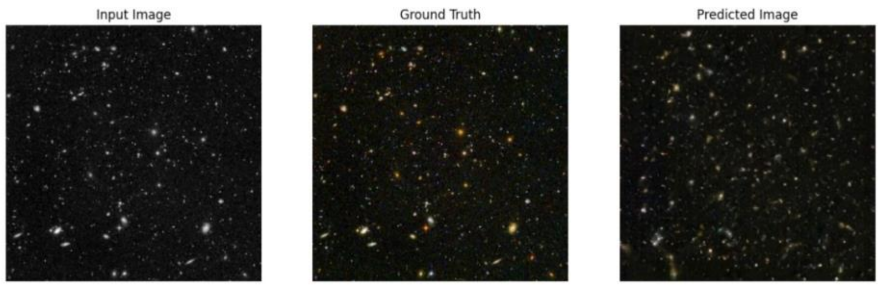


Figure 8.1.4: LR = $2e^{-4}$, batch size = 1 and steps = 80k

For 80k steps the PSNR value obtained was 19.55, 19.50, and 19.48 for batch sizes 1, 10, and 25 respectively.

8.2 Image outputs: CycleGAN

We ran some iterations of our model with varying values for optimizer learning rate and batch size to find the optimum PSNR value,

Epochs	Learning Rate	Batch Size 1	Batch Size 10	Batch Size 25
10	$2e^{-2}$	14.76	12.29	13.12
10	$2e^{-3}$	16.81	14.19	16.01
25	$2e^{-3}$	20.77	16.97	19.42
50	$2e^{-4}$	**22.56**	21.78	22.35

Table 2: PSNR values for various epochs

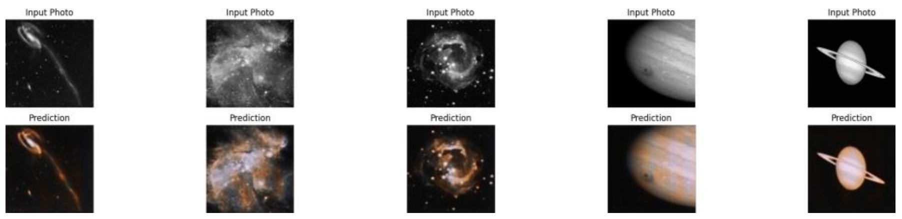


Figure 8.2.1: LR = $2e^{-4}$ and epoch = 10

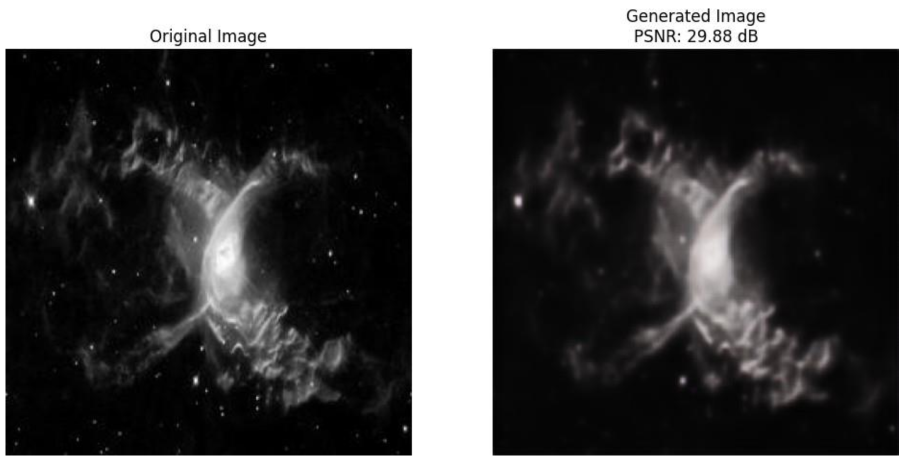


Figure 8.2.2: LR = $2e^{-4}$ and epochs = 25

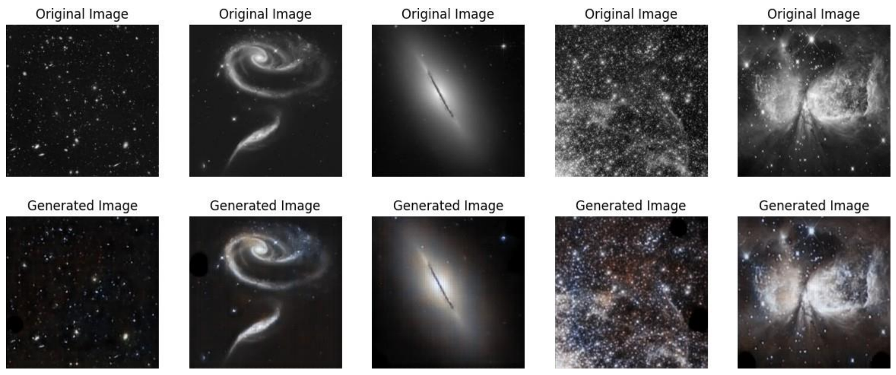


Figure 8.2.3: LR = $2e^{-4}$ and epochs = 50

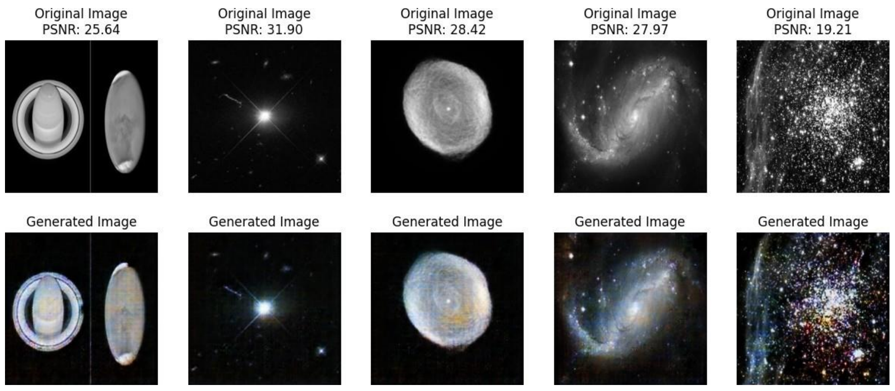


Figure 8.2.4: LR = $2e^{-4}$ and epochs = 100

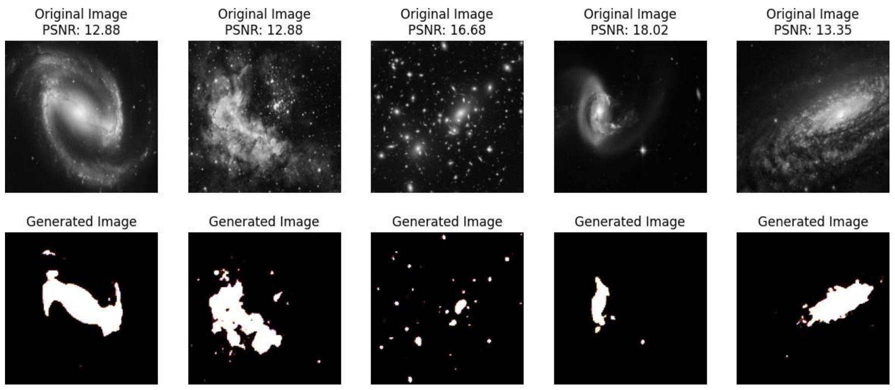


Figure 8.2.5: LR = $2e^{-2}$ and epochs = 10 and batch size =1

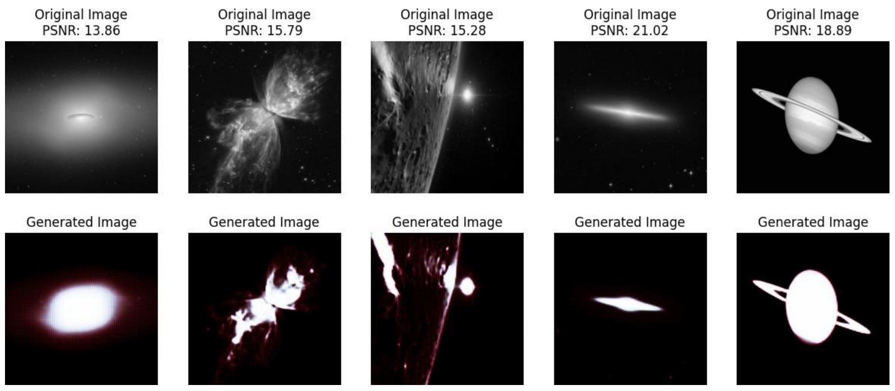


Figure 8.2.6: LR = $2e^{-3}$ and epochs = 25 and batch size = 10

9 Conclusion

1. From Table 1, it is evident that smaller batch sizes yield higher PSNR values. Consequently, the Pix2Pix model trained with a batch size of 1 delivers the best results, consistent with the findings reported by (Isola et al., 2016).

2. For the Pix2Pix model a learning rate of 2×10^{-4} produces the highest PSNR value, making it the optimal learning rate, which aligns with the results from (Isola et al., 2016).

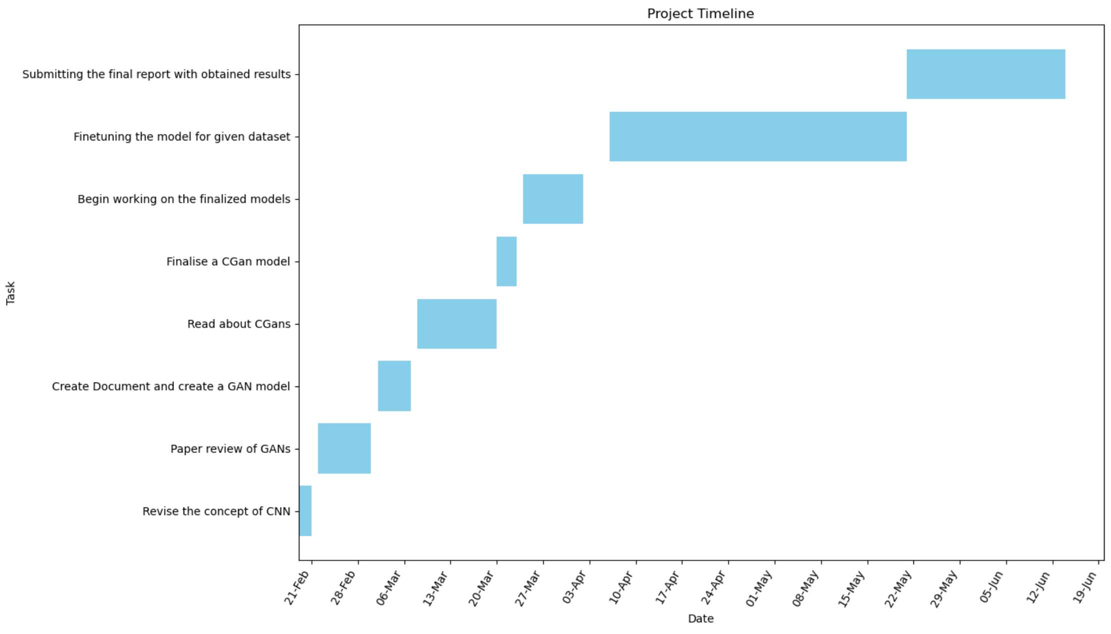
3. The PSNR value for 80,000 training steps surpasses that for 40,000 steps, indicating that extended training improves the quality of the output images for the Pix2Pix model.

4. Similarly as the number of training epochs for CycleGAN increases, the PSNR value rises, signifying an improvement in the quality of the generated images. This trend demonstrates the efficacy of adversarial and cycle consistency losses in enhancing the generator's ability to produce realistic and accurate translations between domains.

5. If we compare the best PSNR values obtained by both the models we can see that the Cycle gan model was trained on 50 epochs and had a PSNR of 22.56 whereas the Pix2Pix model trained on 80k steps or in other words 80 epochs (1 epoch = 1k steps) gives a PSNR of 19.55 which is less than the Cycle gan model. Therefore the Cycle gan model give a better output compared to the Pix2Pix model.

10 Future Scope

The obtained PSNR values can be improved if the models are trained on a more computationally powerful machine for a longer duration.


Project Timeline
Task
Submitting the final report with obtained results
Finetuning the model for given dataset
Begin working on the finalized models
Finalise a CGan model
Read about CGans
Create Document and create a GAN model
Paper review of GANs
Revise the concept of CNN
21-Feb
28-Feb
06-Mar
13-Mar
20-Mar
27-Mar
03-Apr
10-Apr
17-Apr
24-Apr
01-May
08-May
15-May
22-May
29-May
05-Jun
12-Jun
19-Jun
Date

References

CULEBRAS, J. C. S. (2020). *Step by Step CycleGAN - Photos to Monet Paintings — kaggle.com* [[Accessed 20-05-2024]].

DRAMSCH, J. S. (2020). *Understanding and Improving CycleGANs - Tutorial — kaggle.com* [[Accessed 22-05-2024]].

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks. *Advances in Neural Information Processing Systems*, *3*. https://doi.org/10.1145/3422622

HAGHJOU, R. (2023). *Use Tensorflows CycleGAN on Your Own Dataset — kaggle.com* [[Accessed 21-05-2024]].

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2016). Image-to-Image Translation with Conditional Adversarial Networks. *arXiv e-prints*, Article arXiv:1611.07004, arXiv:1611.07004. https://doi.org/10.48550/arXiv.1611.07004

Mart´ın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, . . . Xiaoqiang Zheng. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems [Software available from tensorflow.org]. https://www.tensorflow.org/

Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets.

Rath, S. R. (2020). *Introduction to generative adversarial networks (gans)*. Retrieved June 5, 2024, from https://debuggercafe.com/introduction-to-generative-adversarial-networks-gans/

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2020). Unpaired image-to-image translation using cycle-consistent adversarial networks. https://arxiv.org/abs/1703.10593