Image Extrapolation Using Generative Adversarial Networks

A project report submitted for the partial fulfillment of the

Bachelor of Technology Degree

in

Computer Science & Engineering

under

Maulana Abul Kalam Azad University of Technology

by

Swapnil Mallick

Roll No: **10400116053**, Registration Number: **161040110194**

Ankur Paul

Roll No: **10400116223**, Registration Number: **161040110024**

Academic Session: 2016-2020

Under the Supervision of

Prof. Amit Kumar Das

Department of Computer Science and Engineering
Institute of Engineering & Management
Y-12, Salt Lake, Sector 5, Kolkata, Pin 700091, West Bengal, India

Affiliated To

Maulana Abul Kalam Azad University of Technology, West Bengal
formerly known as **West Bengal University of Technology**
In Pursuit of Knowledge and Excellence

Maulana Abul Kalam Azad University of Technology
BF 142, BF Block, Sector 1, Kolkata, West Bengal 700064

MAY 2020

INSTITUTE
OF ENGINEERING & MANAGEMENT
श्रद्धावान लभते ज्ञानम्
Salt Lake Electronics Complex, Kolkata - 700091, WB, INDIA

Phone : (033) 2357 2969/2059/2995
(033) 2357 8189/8908/5389
Fax : 91 33 2357 8302
E-mail : director@iemcal.com
Website : www.iemcal.com

CERTIFICATE

TO WHOM IT MAY CONCERN

This is to certify that the project report titled **"Image Extrapolation Using Generative Adversarial Networks"**, submitted by **Swapnil Mallick, Roll No: 10400116053, Registration Number: 161040110194**, and **Ankur Paul, Roll No: 10400116223, Registration Number: 161040110024**, students of **Institute of Engineering & Management** in partial fulfilment of requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering**, is a bona fide work carried out under the supervision of **Prof. Amit Kumar Das** during the final year of the academic session of 2016-2020. The content of this report has not been submitted to any other university or institute for the award of any other degree.

It is further certified that the work is entirely original and the performance has been found to be satisfactory.

Prof. Amit Kumar Das
Assistant Professor
Department of Computer Science and Engineering
Institute of Engineering & Management

Prof.(Dr.) Sourav Saha
H.O.D.
Department of Computer Science and Engineering
Institute of Engineering & Management

Prof.(Dr.) Amlan Kusum Nayak
Principal
Institute of Engineering & Management

INSTITUTE OF ENGINEERING & MANAGEMENT

DECLARATION
FOR NON-COMMITMENT OF PLAGIARISM

We, **Swapnil Mallick**, **Ankur Paul**, students of B.Tech. in the Department of Computer Science and Engineering, Institute of Engineering & Management have submitted the project report in partial fulfilment of the requirements to obtain the above noted degree. We declare that we have not committed plagiarism in any form or violated copyright while writing the report and have acknowledged the sources and/or the credit of other authors wherever applicable. If subsequently it is found that we have committed plagiarism or violated copyright, then the authority has full right to cancel/reject/revoke our degree.

Name of the Student: SWAPNIL MALLICK

Full Signature: Swapnil Mallick

Name of the Student: ANKUR PAUL

Full Signature: Ankur Paul

Date: 07/07/2020

Table of Contents:

List of Figures:

Acknowledgement:

We must not forget to acknowledge everyone who has provided constant support to us during our B.Tech course. The achievement that is associated with the successful completion of any task would be incomplete without mentioning the name of the person whose endless cooperation made it possible. His constant guidance and encouragement made all our efforts successful. We take this opportunity to express our deep gratitude towards our project mentor, Prof. Amit Kumar Das for giving such valuable suggestions, insightful comments, unceasing ideas and encouragement during the development of this project work. Without his guidance and persistent encouragement, this project work would not have been possible. He has been a tremendous mentor for us throughout this academic journey. Many of his academic advises about our career growth have been priceless.

We would like to convey sincere gratitude to our HOD Prof. Sourav Saha for providing us constant inspiration to stand firm against several setbacks throughout the course. We also express gratitude to all of our friends in the department for providing the friendly environment to work on the project work. We would also like to thank our Director Prof. Satyajit Chakraborti for providing us an outstanding platform in order to develop our academic career. In addition, we also preserve a very special thankful feeling about our Principal Prof. Amlan Kusum Nayak for being a constant source of inspiration. Finally, we would like to thank everybody who has provided assistance, in whatever little form, towards successful realization of this project but with an apology that we could not mention everybody's name individually.

Team Number	6
Group Members	Swapnil Mallick, Ankur Paul
Project Mentor	Prof. Amit Kumar Das
Project Title	Image Extrapolation Using Generative Adversarial Networks

Objective:

The rise of generative adversarial networks has resulted in great advances in the field of computer vision. Image generation especially image inpainting has been a hot topic of research over the years. However, the topic of image extrapolation has received comparatively less attention. Image inpainting primarily focuses on the completion or restoration of the inner portions of an image that may have been deleted.

On the other hand, image extrapolation involves the task of extrapolating the outer regions of an image which are relatively unknown. Though the topics of image inpainting and image extrapolation may seem to be closely related, the techniques used for both can be different. Hence, we will try to create a model that will be trained adversarially in order to extrapolate the boundaries of images.

Fig1: Image Extrapolation Idea

Scope:

Though the topic of image extrapolation may seem to be new and relatively unknown, rather much unexplored, it has quite a number of applications. Our main goal of this project is to explore whether a deep learning model can generate images with limited neighbouring information. Image extrapolation may find its use in:-

- panorama creation
- vertically-filmed video expansion
- texture generation

Related Works:

Deep neural networks have recently shown great performance in image completion. This section discusses previous work related to the proposed methods for image extrapolation. The problem of *image extrapolation* was first brought into light by Wang et al. in their work [1] which used a data-driven approach to the problem along with a graph representation of the source images. Their approach included representation of source images using graphs, retrieving candidate images using subgraph matching and finally extrapolating images using retrieved candidate images. Although the results obtained by the aforementioned approach are realistic, we hope to achieve better results by using an adversarial training approach to the problem.

The idea of *image extrapolation* is inspired from the problem of *image inpainting* which has been implemented by Pathak et al. [2]. They introduced the concept of a Context Encoder, a CNN

trained adversarially to reconstruct missing regions of an arbitrary image based on surrounding pixels.

The Context Encoder approach used in [2] for image inpainting was improved by Iizuka et al. [3]. Their network included a "local" discriminator along with the "global" discriminator which was already present. The "local" discriminator took only the part of the image that was being reconstructed along with the information in its surrounding regions. As usual, the entire image was fed to the "global" discriminator as input. Both the discriminators were trained to distinguish between a real and a fake image. The "local" discriminator coupled with the normal "global" discriminator helped in improving the visual quality of the reconstructed image.

The limitation of many recent methods used to solve the problem of *image inpainting* is the focus on rectangular shaped holes, often assumed to be located at the centre of the image. This limitation has been overcome by Liu et al. in their work [4] which tries to fill up irregular hole patterns within the image. The problem opposite to inpainting comprises predicting which pixels reside beyond the borders of a fully intact photo, and to our knowledge has been explored only thrice before by the academic research community - Wang et al. [1], Zhang et al. [5] and Basile Van Hoorick [6]. One approach [5] geometrically extrapolates the field of view of an image using another panoramic reference image of the same scene category using old-school computer vision, while the most recent relevant publication [6] uses a GAN to perform horizontal and vertical extrapolation. A major challenge in image extrapolation is to maintain high quality in the generated results. To solve this problem, two innovative models namely Skip Horizontal Encoder and Recurrent Content Transfer has been integrated in the encoder-decoder structure [7]. Lastly, Google's Snapseed application has a proprietary 'Expand' functionality which seems to select patches from the image and copy them to the edges, but a limited number of experiments suggest that this tool fails to capture the local structure of most scenes from region to region.

The recently introduced SinGAN framework [8] can train an unconditional generative model on a single natural photo, capturing and reproducing image statistics across various scales of the image. It allows for the creation of random samples with new object configurations by starting from a low-resolution sample at the coarsest scale, and then progressively upsampling and refining the result through the pyramid of generators. By carefully modifying the initial sample, SinGAN can be used to perform several image manipulation tasks such as editing photos, splicing foreign objects into the scene and subsequently harmonizing their style with the environment.

Dataset:

We have scraped 3505 beach images in total to create the dataset. We have used ***Beautiful Soup*** for scraping purpose. ***Beautiful Soup*** is a library that makes it easy to scrape information from web pages. It sits on top of an HTML or XML parser, providing Pythonic idioms for iterating, searching, and modifying the parse tree.

A complete guide to *web scraping* have been discussed below:

1. At first, an HTTP request is sent to the URL of the webpage that we want to access. The server responds to this request by returning the HTML content of the webpage. For this task, a third-party HTTP library will be used for python requests.
    - Import the required libraries to run the code including *BeautifulSoup*. This library is used to fetch content from a given link. *urllib.request* is another package that helps in opening and reading URLs. *argparse* allows us to parse arguments passed with the file execution. *os* provides functionalities to interact with the filesystem.
2. Once accessing the HTML content is successful, we are left with the task of parsing the data. Since most of the HTML data is nested, we will not be able to extract data by simply

using normal string processing. For this purpose, a parser is required which can create a nested or tree structure of the HTML data. There are many HTML parser libraries available but the most widely used one is html5lib.

- Initialize the argument parser and parse the filename argument. The parsers maybe either *html* or *lxml*. Initially the information stored in the web page is converted in form of plain text. After that this information is converted into a tree structure so that we can traverse the html tags in a hierarchical manner. This task is accomplished by the *BeautifulSoup()* method.

3. Now, all we need to do is tree traversal i.e. navigate and search the parse tree that we created. We have used a method called *find()/find_all()* for extracting data out of HTML and XML files.

   - Before scraping the images we must inspect the html page which we are about to scrape to find all those classes which comprise the image tags.
   - After that we shall fetch the html tag elements one by one until we reach the exact image tag we want to refer. For e.g. *find_all("div",{"class" : "wrapper"})* , this means that the image tag is inside the class “wrapper” within the “ ” tag. There might also be a possibility that the image tags are within multiple classes ; in that case we need to traverse all those sub-classes before reaching the desired image tag.
   - Then we need to use the *get()* method to extract all the images from that website.
   - We must ensure that we first list all those image links which actually can be scrapped because there might be a possibility that some image links may not be allowed to scrape directly.

The beach image dataset consists a total of 3505 images which is a relatively small number. However, this problem of shortage of training dataset have been overcome by the technique of *data augmentation*.

Fig2: Dataset

Data Preprocessing:

Data augmentation is a technique to artificially create new training data from existing training data. This can be done by applying domain-specific techniques to examples from the training data that create new and different training examples.

Image data augmentation is perhaps the most well-known type of data augmentation and involves creating transformed versions of images in the training dataset that belong to the same class as the original image.

At first, the images in the dataset have been split in the ratio of 9:1. Thus, our training set consists of 90% of the images i.e., 3154 images and the test set includes 10% of the images i.e., 351 images.

Since, each image in the dataset is of different dimension, these images cannot be fed to the model. Hence, we have chosen a uniform dimension of 256 X 256 in order to resize our images. Once resizing has been done, we move on to the augmentation step.

Primarily, we have used six data augmentation techniques namely, *Brightness Change, Height Shift Range, Width Shift Range, Rotation, Horizontal Flip and Zoom Range* [9].

- ***Brightness Change*****:** The brightness of the image can be augmented by either randomly darkening images, brightening images, or both. The intent is to allow a model to generalize across images trained on different lighting levels. This can be achieved by specifying the *brightness_range* argument to the *ImageDataGenerator()* constructor that specifies minimum and maximum range as a float representing a percentage for selecting a brightening amount. Values less than 1.0 darken the image, e.g. [0.5, 1.0], whereas values larger than 1.0 brighten the image, e.g. [1.0, 1.5], where 1.0 has no effect on brightness.
- ***Horizontal Flip*****:** An image flip means reversing the rows and columns of pixels. In horizontal flip, we swap the columns in the image matrix. The flip augmentation is specified by a Boolean *horizontal_flip* argument to the *ImageDataGenerator()* class constructor. It is preferable to use the flip operation by checking the type of image, as in whether that particular flip makes sense or not.
- ***Width Shift and Height Shift*****:** A shift to an image means moving all pixels of the image in one direction, such as horizontally or vertically, while keeping the image dimensions same. This means that some of the pixels will be clipped off the image and there will be a region of the image where new pixel values will have to be specified. The *width_shift* and *height_shift* range arguments to the *ImageDataGenerator()* constructor control the amount of horizontal and vertical shift respectively. These arguments can specify a floating point value that indicates the percentage (between 0 and 1) of the width or height of the image to shift. Alternately, a number of pixels can be specified to shift the image. The new pixel values are added according to the *fill_mode* argument. There are four types of *fill_mode*:
  - ***constant*****:** All new pixels (and subpixels if the input is a colour image) are given the constant value specified. The default value of *cval* is taken as **0**. **0** corresponds to setting all new pixels to black. This is the most common and recommended value. Similarly setting the constant value to **1** will be equivalent to filling these areas with white. E.g., kkkk|abcd|kkkk (cval=k)
  - ***nearest*****:** Adds the value of the nearest pixel of the original image. This will result in lines in the augmented image. E.g., aaaa|abcd|dddd
  - ***reflect*****:** Mirrors the original image to the new pixels. E.g., dcba|abcd|dcba

- ***wrap*:** The original image repeats in the new pixels. E.g., abcd|abcd|abcd By default, the value of *fill_mode* is taken as 'nearest'.

❖ ***Rotation*:** The rotation augmentation randomly rotates the image clockwise by a given number of degrees from 0 to 360. The rotation will likely rotate pixels out of the image frame and leave areas of the frame with no pixel value that must be filled in. The extent of rotation is specified by the *rotation_range* argument to the *ImageDataGenerator()* class constructor, with rotations to the image between 0 and 90 degrees. After rotation, there will be new areas in the image which were not included in the original image. The way we handle these areas are specified by the ***fill mode***.

❖ ***Image Zoom:*** The zoom augmentation randomly zooms the image in and either adds new pixel values around the image or interpolates pixel values respectively. Image zooming can be configured by the *zoom_range* argument to the *ImageDataGenerator()* class constructor. You can specify the percentage of the zoom as a single float or a range as an array or tuple. If a float is specified, then the range for the zoom will be [1-value, 1+value]. For example, if you specify 0.3, then the range will be [0.7, 1.3], or between 70% (zoom in) and 130% (zoom out).

We have augmented each image twice and have got three different variations of any image in the dataset (one original image, two images obtained after augmentation). Therefore, we have obtained thrice the number of images scrapped i.e., 10515 images in total.

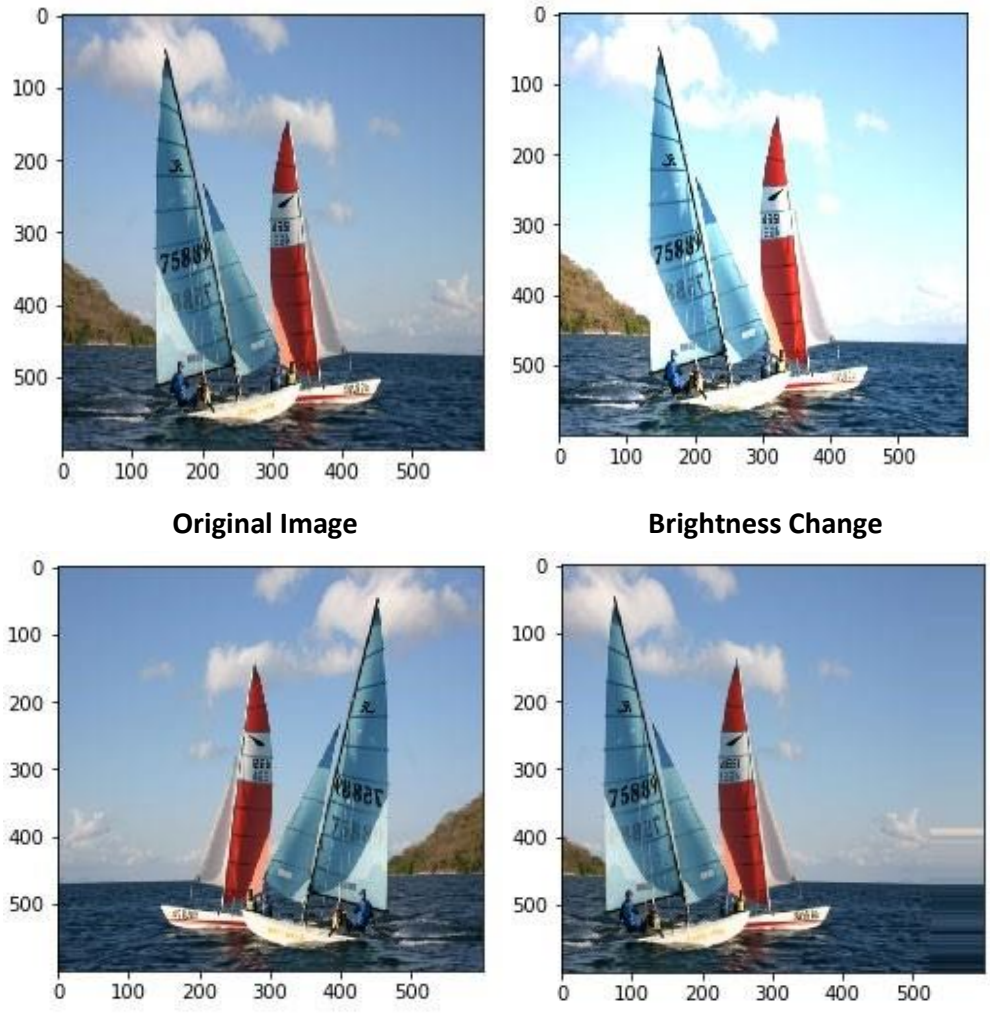


Fig3: Augmentation

We have divided the entire dataset into batches of 2000 images and stored each batch as numpy object. Finally, we have stored each numpy object into two separate folders, one for training data and the other for test data.

Once the training data is preprocessed and the training is done, we hope that the model will be able to perform image extrapolation successfully.

The Convolution Operation

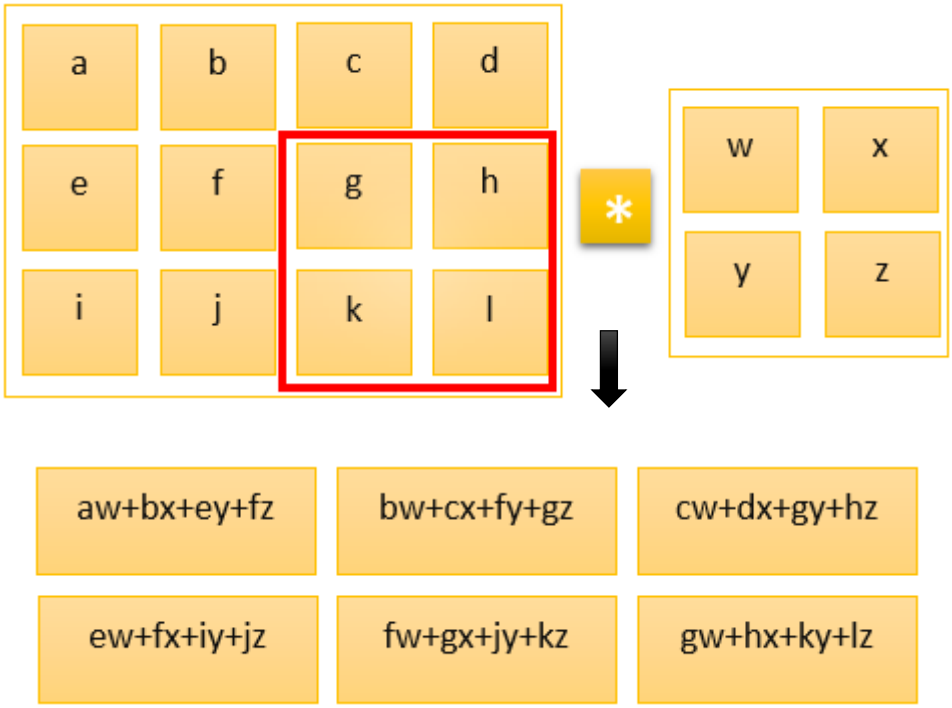


Fig4: Example of convolution operation

***Working example of a 2D convolution*:-**

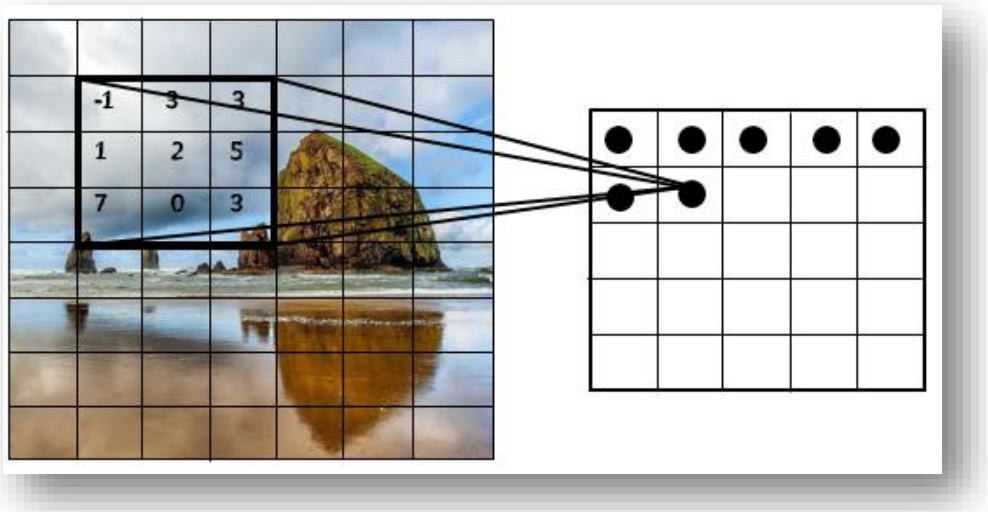


Fig5: Creating a Feature Map

- We just slide the kernel over the input.
- Each time we slide the kernel we get one value in the output; the resulting output we get is a feature map.
- Basically what gets extracted after each convolution operation is a feature and once the entire image has been traversed , what forms is a feature map.
- For colored images we basically use a 3D filter which resembles the shape of a cuboid since a colored image has 3 channels(red,blue and green), here we have just shown how a feature map is obtained after a convolution operation by using a 3X3 filter.

Generative Adversarial Networks:

Generative Adversarial Networks (GANs) are one of the most interesting ideas in computer science today. GANs are capable of learning features from the training data's distribution and create new training images from that same distribution. GANs were invented by Ian Goodfellow in 2014 and described in the paper ***Generative Adversarial Nets*** [10]**.** They are made of two distinct models, a ***generator*** and a ***discriminator*.** The ***generator*** is assigned with the task to create 'fake' images that look like the training images. The job of the ***discriminator*** is to classify whether or not an image is a real training image or a fake image created by the generator by looking at the image.

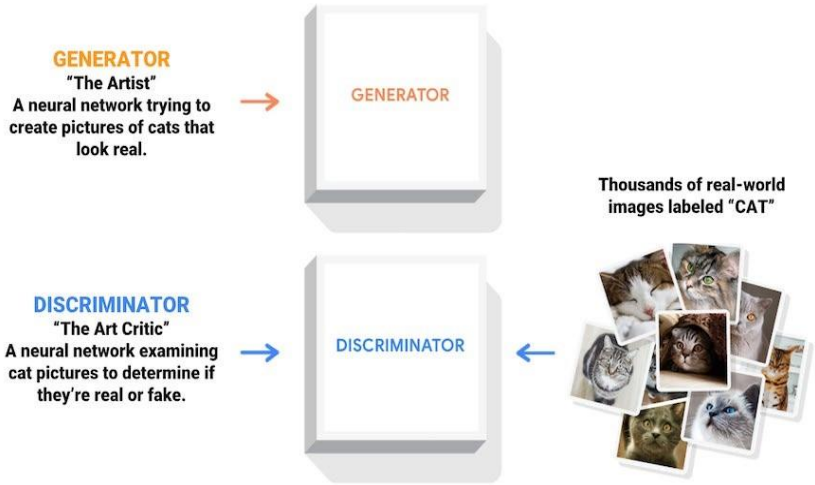


Fig6: Generator/Discriminator

During training, the generator constantly tries to outsmart the discriminator by generating better and better fake images, while the discriminator tries to improve at correctly classifying the real and fake images. The equilibrium of this process is achieved when the generator learns to generate fake images that look similar to the training data, and the discriminator is left to always guess at 50% confidence that the generator output is real or fake. GANs are very difficult to train because of the problems such as:

Mode collapse: the generator collapses which produces limited varieties of samples, and hence the model often doesn't seem to converge and thus produce pretty bad results. One of the reasons for this might be improper choosing of hyper parameters.

Diminished gradient: the discriminator gets too successful that the generator gradient vanishes and learns nothing,

Unbalance between the generator and discriminator causing overfitting

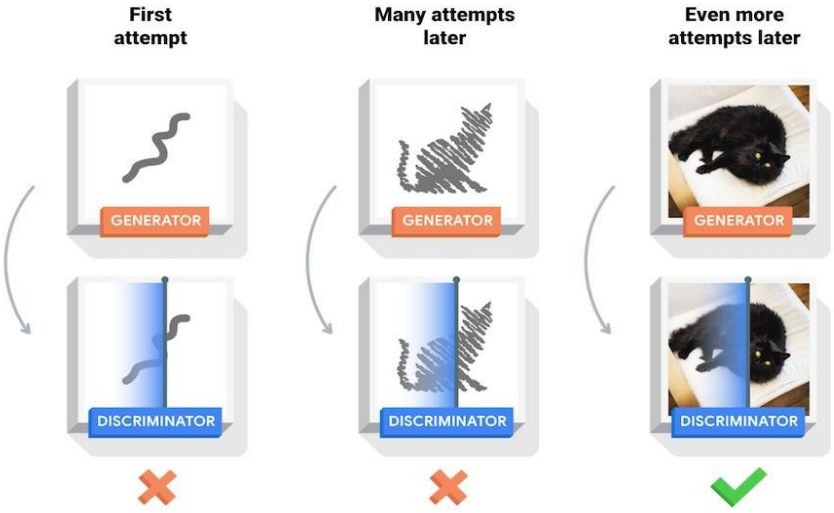


Fig7: Generation of Images by Generator

Proposed Methodology:

Once data preprocessing is complete, we have cropped each image and fed that cropped image into the DCGAN architecture (G, D) for training. Let us call each preprocessed image in the training set as I_{train} , which is also the ground truth. Each of these preprocessed images I_{train} are cropped to get images which we may call I_{crop}. After each image I_{train} has been cropped, we run the generator G on the cropped image I_{crop} to get the extrapolated image I_{out}. This extrapolated image I_{out} is the output that we are concerned with. Then, we run the discriminator D to classify the ground truth I_{train} and the extrapolated image I_{out}. Based on the output provides by discriminator D, the generator G tries to improve itself by modifying its weights. Over the course of time, we have examined that the model improves in extrapolating the boundaries of the images and recreating the missing portions at both the sides.

Training Pipeline:

At first, the generator G is fed with the cropped images I_{crop} and asked to predict the two sides of the cropped images. However due to lack of training, the generator fails to create the missing parts and only outputs a poorly extrapolated image. Then, the discriminator D is trained first with the original missing parts and then with the missing parts created by the generator G. The loss function is calculated and the classification is provided by the discriminator D.

Once the discriminator is trained, we now use the combined model to train the generator G with the cropped images I_{crop} in order to improve the generator at extrapolating images. The layers of the discriminator are frozen so that they are not updated further. So, the trained generator again predicts the missing parts and the output image provided by the generator G is again fed to the discriminator D for classification. Based on the verdict given by the discriminator D, the generator loss values and parameters are updated in order to improve the generator. This process is repeated for 50 epochs until the generator G becomes adept at creating the missing parts.

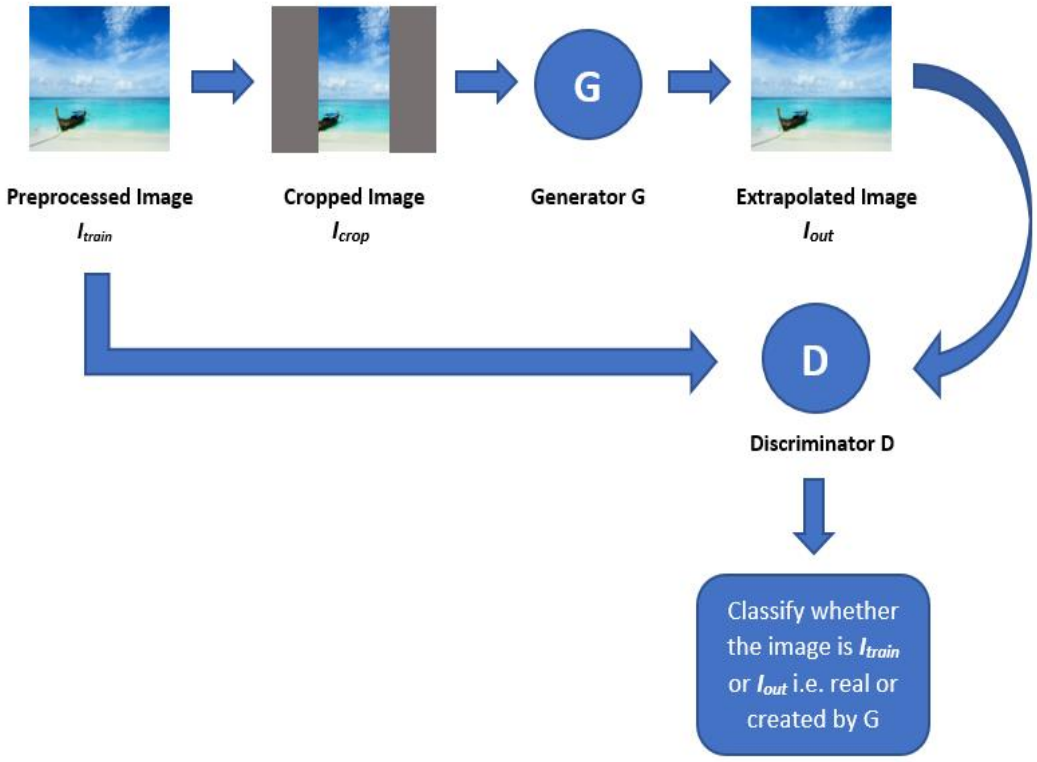


Fig8:Training Pipeline

Image Cropping:

All the preprocessed images are of the dimension 256 X 256 X 3. After data preprocessing is complete, these images are cropped from both the ends so that only the middle portion of the images remain intact. The amount of each image that needs to be cropped from both ends has been set to 25% i.e. the *crop_percentage* has been set to 0.25. Thus, regions having dimensions 256 X 64 X 3 will be cropped from both the ends of each preprocessed image in the dataset. This will result in images having dimensions of 256 X 128 X 3 (the center part of the image) which will, in turn, be fed to the generator, G as input.

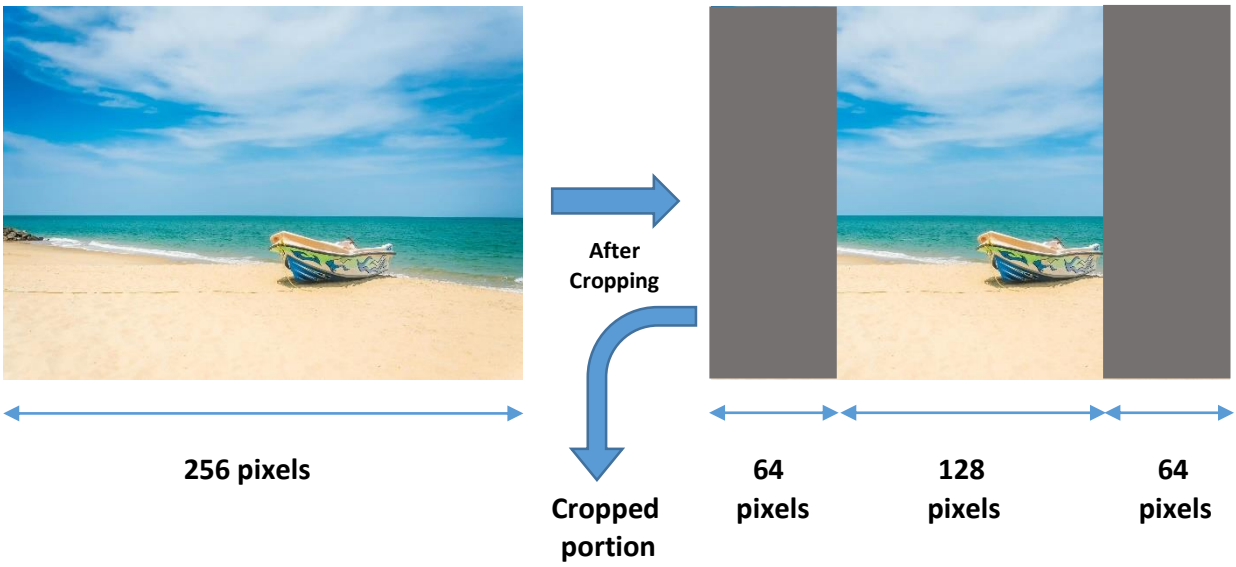
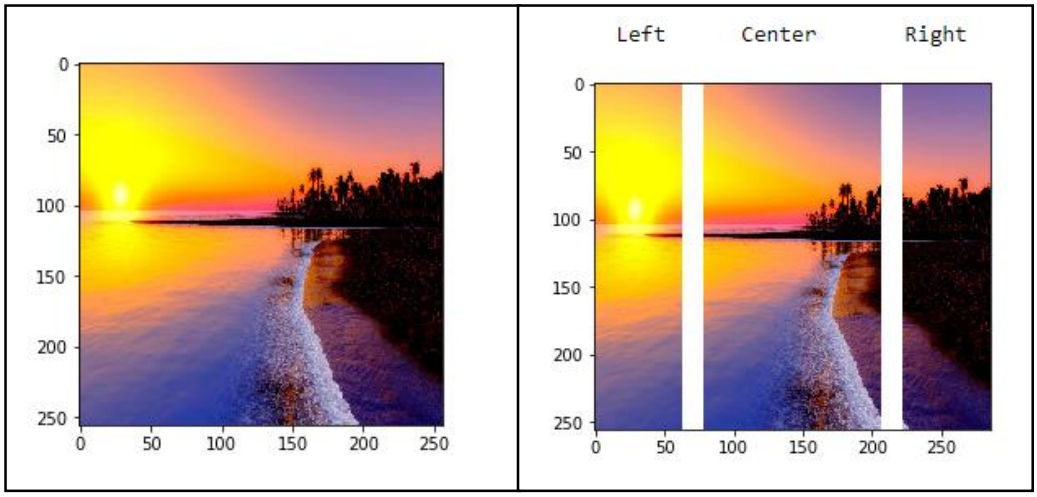


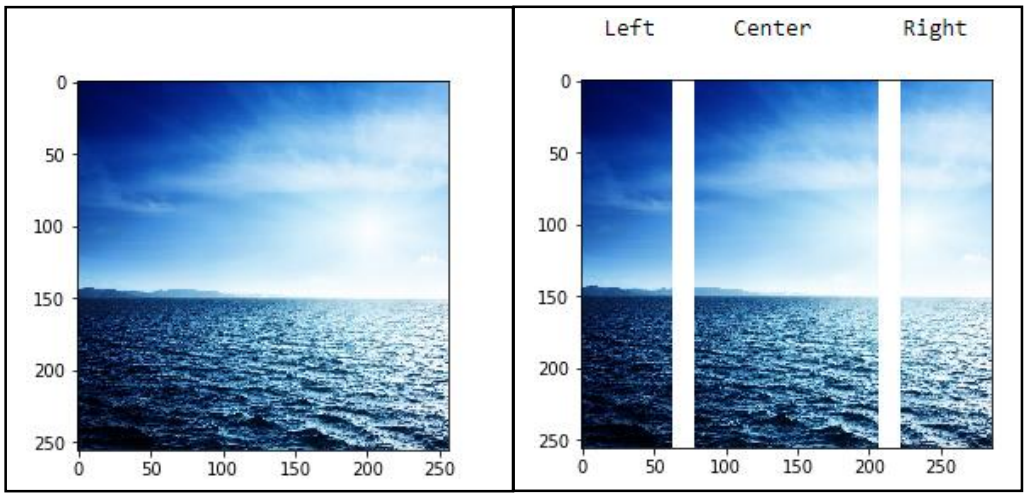
Fig9: General Idea of Cropping

Some Images from Dataset after Cropping:

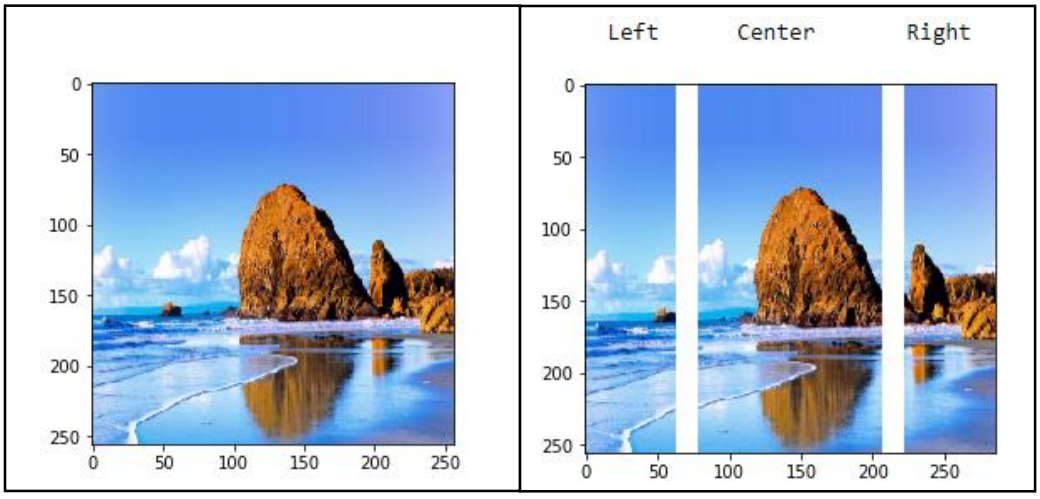
Fig10: Cropped Images



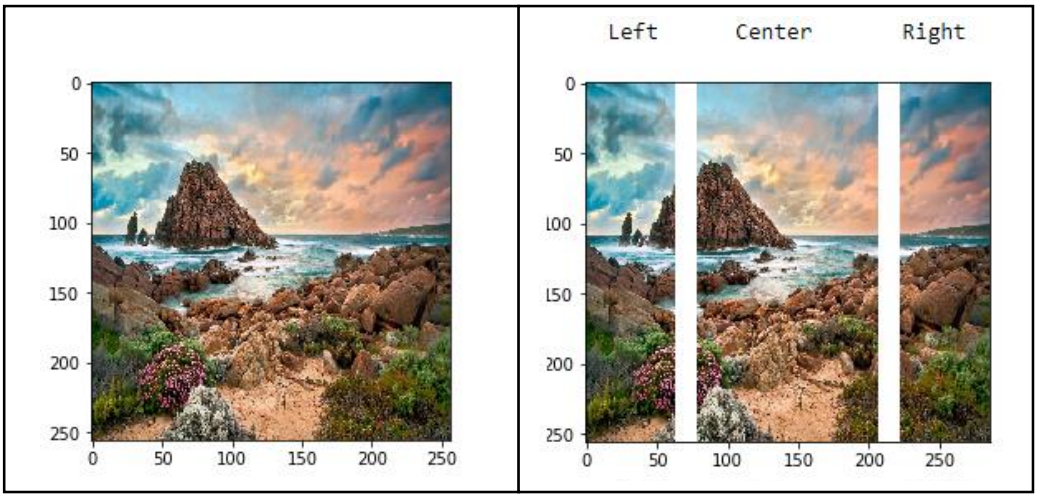
Original Image I_{train} **Cropped Image I_{crop}**



Original Image I_{train} **Cropped Image I_{crop}**



Original Image I_{train} **Cropped Image I_{crop}**



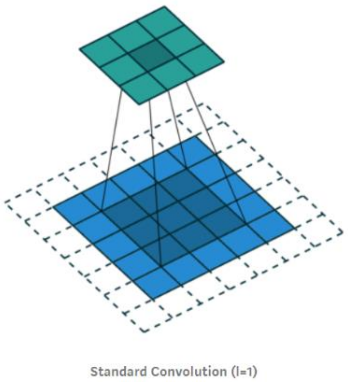
Original Image I_{train} **Cropped Image I_{crop}**

Network Architecture:

We have adopted a DCGAN architecture (G, D) which is very similar to the one used by Iizuka et al [3]. The architecture used by Iizuka et al [3] includes a generator G that takes the form of an encoder-decoder CNN, while the discriminator D uses *strided convolutions* to repeatedly downsample an image for binary classification [4].

Due to computational restrictions, we have proposed a shallower architecture for extrapolation on beach images. However, it is still conceptually similar to the architecture proposed by Iizuka et al. [3]. We still use the encoder-decoder structure for generator G. We have also used *dilated convolutions* to increase the receptive field of neurons.

- **Dilated Convolution:** Dilated convolutions introduce another parameter to convolutional layers called the **dilation rate**. This defines a spacing between the values in a kernel.



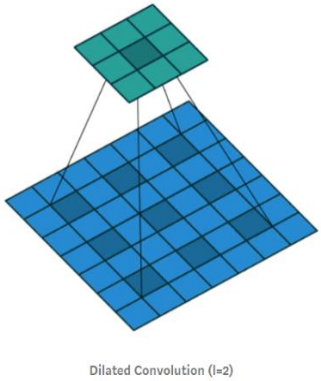


Fig11: Standard Convolution **Fig12: Dilated Convolution**

- **Deconvolution:** Deconvolution is somewhat the opposite operation of convolution. In deconvolution. For example, when we apply transposed convolution with a 3 x 3 kernel over a 2 x 2 input padded with a 2 x 2 border of zeros using unit strides, the upsampled output is of size 4 x 4.

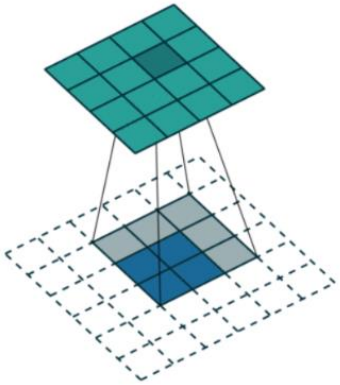
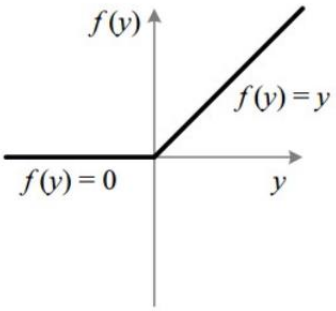


Fig13: Deconvolution Operation

- **ReLU and Leaky ReLU:** ReLU is the acronym for **Rectified Linear Unit**. ReLU is an activation function. Mathematically, it is defined as *y = max(0, x)*.
- **Leaky ReLU** is a modified version of ReLU. It has been found out that Leaky ReLU provides better results than ReLU activation function in case of generative adversarial networks.



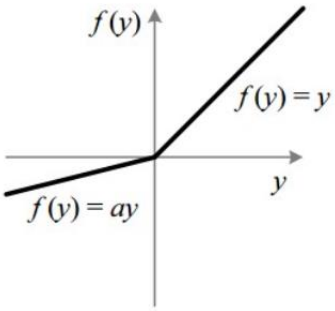


Fig14: ReLU **Fig15: Leaky ReLU**

- **Batch Normalization:** Batch normalization is a normalization method that normalizes activations in a network across the mini-batch. For each feature, batch normalization computes the mean and variance of that feature in the mini-batch.
- **Instance Normalization:** Instance Normalization computes the mean and standard deviation and normalize across each channel in each training example.

The description of the generator and discriminator has been provided below:

Generator, G

Type	*f*	*η*	*s*	*n*
CONV	5	1	1	64
CONV	4	1	2	128
CONV	4	1	2	256
CONV	4	1	1	512
CONV	4	1	1	512
CONV	4	2	1	512
CONV	4	4	1	512
CONV	4	8	1	512
CONV	4	16	1	512
CONV	4	1	1	512
CONV	4	1	1	512
DECONV	4	1	2	256
DECONV	4	1	2	128
CONV	4	1	1	128
CONV	4	1	1	64
Output	4	1	1	3

Discriminator, D

Type	*f*	*η*	*s*	*n*
CONV	5	1	2	32
CONV	5	1	2	64
CONV	5	1	2	64
CONV	5	1	2	128
CONV	5	1	2	128
FC	_____	_____	_____	1024
OUTPUT	_____	_____	_____	1

Here, *f* is the *filter size, η* is the *dilation rate, s* is the *stride, and n* is the *number of outputs*.

Every convolution layer in the generator G is followed by a *ReLU* activation layer and an *instance normalization* layer and each deconvolution layer is only followed by a *ReLU* activation layer. Only, the output layer of the generator G is followed by a *tanh* activation function. Similarly, every convolution layer in the discriminator D is followed by a *Leaky ReLU* activation layer except the fully convolutional layer which is followed by a *ReLU* activation layer and the output layer which is followed by a *sigmoid* activation layer.

Output Analysis:

We have trained the model for 50 epochs and have examined that the model gradually improves in extrapolating the boundaries of the images over the course of time. During the early stages of training, the generator G can only create a few patches in the boundaries. However, as the number of epochs go on increasing, the quality of the recreated portions gradually becomes better.

- Initially we have trained the model with 128 X 128 images for 50 epochs. Here, the generator consists of 9 layers comprising of 6 convolution layers followed by a deconvolution layer and then again a convolution layer whereas the discriminator has 5 convolution layers followed by a dense layer. In this model we have used BatchNormalisation at every layer along with Leaky Relu activation functions for both generator as well as discriminator. We observe that, with increasing number of epochs although few images tend to converge, i.e. the generator is able to hallucinate beyond the boundaries of the center image, yet we find that many images consist of colored patches. Moreover the textures doesn't seem to be quite clear for such small sized images.

- After testing for multiple models we chose 256 X 256 images and trained the model for 50 epochs. This model had 15 layers in the generator which in turn comprised of 11 convolution layers followed by 2 deconvolution layers and finally another 2 convolution layers and the discriminator has 5 convolution layers followed by a dense layer. Since the model seems too complex we also used dropout of 0.25 to prevent overfitting. Apart from that Relu activation was used for Generator and Leaky Relu activation function was used for discriminator. We also used InstanceNormalisation instead of BatchNormalisation at every layer. The former one tends to normalize for every channel in a colored image whereas the latter tends to normalize for every given batches of images. The output we obtained were quite resounding and the model was able to generate complex features like those of trees, rocks as well as humans. We have shown a comparative study between 2 of the most prominent models depicting the generator and discriminator losses. Since training of GANs is a time consuming as well as computationally expensive process we had to devote about 10 days for training 50 epochs. The GPU used for our training purpose is NVIDIA 1660 3GB as well as NVIDIA 1060 3GB.

Fig16: **Images from Initial Phase of Training**

Epoch : 2

I_{train} I_{crop} I_{out}

Epoch : 6

I_{train} I_{crop} I_{out}

Epoch : 12

I_{train} I_{crop} I_{out}

Epoch : 18

I_{train} I_{crop} I_{out}

Recursive Painting:

In our project, we have used a concept known as *recursive painting*. The idea behind recursive painting is, that whether the model is able to hallucinate beyond the original dimensions of the image. In *recursive painting*, an extrapolated image or the generated image is fed again to the network as input once the model has been trained to extrapolate the images. Let's say, the initial generated image is of factor=1, i.e. only once the generator predicts the output. Now this factor can be increased as many times as required, which means the generated output will act as the input for the next iteration until the factor reaches a value of n, where n is the times by which we want to increase the width of the image. We have repeated this process recursively three times, in turn increasing the width of the new image to be thrice as the original image. As expected, the noise tends to increase with successive iterations. Despite this, the model is found to successfully learn the general textures of the image and extrapolates the sky and landscape relatively realistically.

Comparative study between 2 distinct models

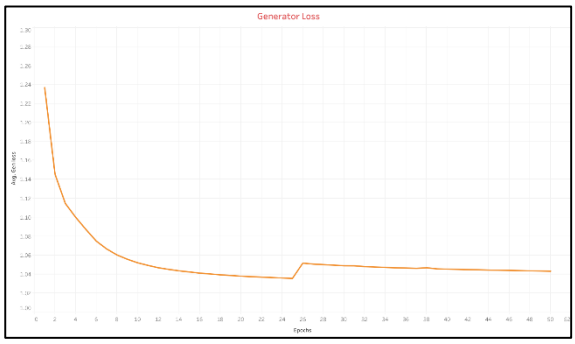
Model1:

Image Dimension: (128 X 128 x 3)

Learning Rate: 0.0003

Normalization: Batch Normalisation

Generator Activation: Leaky Relu , Discriminator: Leaky Relu



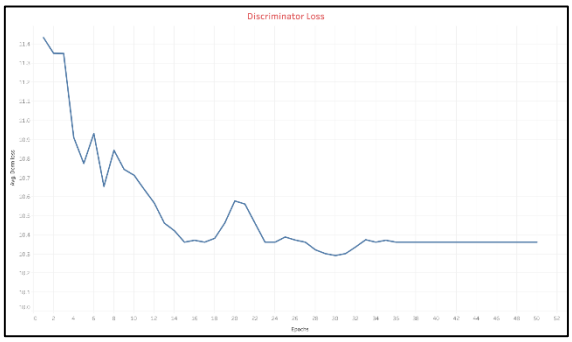


Fig17: Generator Loss **Fig18: Discriminator Loss**

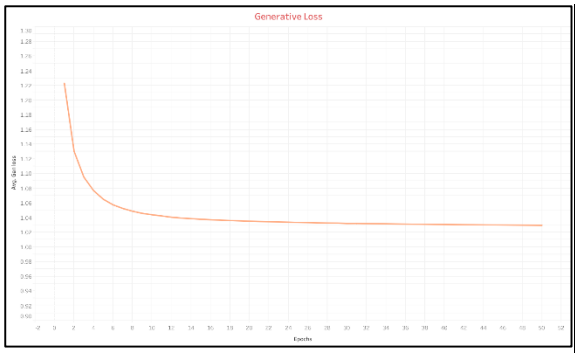
Model2:

Image Dimension: (256 X 256 x 3)

Learning Rate: 0.0004

Normalization: Instance Normalisation

Generator Activation: Relu , Discriminator: Leaky Relu



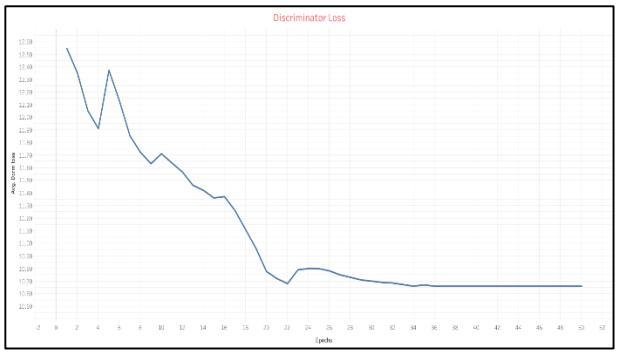


Fig19: Generator Loss **Fig20: Discriminator Loss**

Output Images:

Fig21: *Train Output*

Original Image **Generated Image**

Original Image **Generated Image**

Original Image **Generated Image**

Original Image **Generated Image**

Original Image Generated Image

Original Image Generated Image

Original Image Generated Image

Original Image Generated Image

Fig22: ***Test Output***

Original Image **Generated Image**

Original Image **Generated Image**

Original Image **Generated Image**

Original Image **Generated Image**

Original Image Generated Image

Original Image Generated Image

Original Image Generated Image

Original Image Generated Image

Fig23: ***Recursive Painting Output***

Source Code:

1. Data Preparation:-

Libraries imported:

```
import requests
import urllib.request
from bs4 import BeautifulSoup
import wpcave, wpaccess,sardinia
import get_wpaper,wp_saf,wplay,standard
```

Image Scraping:

```
file = open('links.txt','r')
links = file.readlines()

i = 1

for url in links:
    try:
        if url.startswith('http://getwallpapers.com/'):
            temp = get_wpaper.sp1(url,i)

        elif url.startswith('https://wallpaperplay.com/'):
            link_init = "https://wallpaperplay.com/"
            temp = wplay.scp(url,i,link_init)

        elif url.startswith('https://wallpaperset.com/'):
            link_init = "https://wallpaperset.com/"
            temp = wplay.scp(url,i,link_init)

        elif url.startswith('https://wallpapersafari.com/'):
            temp = wp_saf.sp2(url,i)

        elif url.startswith('https://www.standard.co.uk/'):
            temp = standard.srp(url,i)

        elif url.startswith('https://wallpapercave.com/'):
            temp = wpcave.scrape(url, i)

        elif url.startswith('https://wallpaperaccess.com/'):
            temp = wpaccess.scrape(url, i)

        elif url.startswith('https://www.sardinianplaces.co.uk/'):
            temp = sardinia.scrape(url, i)
        i = temp
        print(i)
    except Exception as e:
        print("Error :",e)
        print("Link not working : ",url)
file.close()
```

www.getwallpaper.com

```
file = open('links_not_working.txt','r')
error_links = file.readlines()
temp = [i[:-1] for i in error_links]

def sp1(url,m):
    source_code = requests.get(url)

    plain_text = source_code.text

    soup = BeautifulSoup(plain_text,"lxml")
    i = 0
    set_img = set([])
    for link in soup.find_all("div",{"class" : "wrapper"}):
        anchor_tags = link.find_all("img",{"class" : "isWiden thumb
preload ads_popup"})
        for j in anchor_tags:
            href = j.get('data-src')
            i = i+1
        if href == None:
            continue
        else:
            link_init = "http://getwallpapers.com/"
            href = link_init + href
            if href not in temp:
                set_img.add(href)
                print(href)
                print(i)
    count=0
    flag=m
    name = m
    for j in set_img:
        if j in temp:
            continue
        else:
            print(j)
            count=count+1
            print(count)
            img_name = name
            name = name + 1
            full_name = str(img_name) + ".jpg"
            urllib.request.urlretrieve(j, full_name)
            print("loop break")
            if (name-flag) == len(set_img):
                return name
```

www.wallpapersafari.com

```
file = open('links_not_working.txt','r')
error_links = file.readlines()
temp = [i[:-1] for i in error_links]

def sp2(site,m):
    url = str(site)
    url = url.rstrip()

    source_code = requests.get(url)

    plain_text = source_code.text

    soup = BeautifulSoup(plain_text,"html.parser")
```

```
    set_img = set([])
    i=0
    for link in soup.find_all("img"):
        href = link.get('src')
        if href.startswith('https://cdn.wallpapersafari.com/'):
            set_img.add(href)
            i = i + 1
            print(href)
            print(i)

    count=0
    flag = m
    name = m
    for k in set_img:
        print(k)
    print(len(set_img))
    for j in set_img:
        if j in temp:
            continue
        else:
            print(j)
            count=count+1
            print(count)
            img_name = name
            name = name + 1
            full_name = str(img_name) + ".jpg"
            urllib.request.urlretrieve(j, full_name)
            print("loop break")
            if (name - flag) == len(set_img):
                return name
```

www.wallpaperaccess.com

```
file = open('links_not_working.txt','r')
error_links = file.readlines()
temp = [i[:-1] for i in error_links]

def scrape(url, count):
    site = str(url)
    site = site.rstrip()
    source_code = requests.get(site)

    plain_text = source_code.text

    soup = BeautifulSoup(plain_text,"html.parser")

    a=0
    set_img = set([])

    for link in soup.find_all("div",{"class" : "wrapper"}):
        anchor_tags = link.find_all("img",{"class" : "thumb preload
ads_popup"})
        for j in anchor_tags:
            href = j.get('data-src')
            a = a+1
        if href == None:
            continue
        else:
            link_init = 'https://wallpaperaccess.com/'
```

```
            href = link_init + href
            set_img.add(href)
            print(href)
            print(a)

    count3 = count
    for image in set_img:
        if image in temp:
            continue
        else:
            img_name = count3
            full_name = str(img_name) + '.jpg'
            print("loop break")
            urllib.request.urlretrieve(str(image), full_name)
            count3 = count3 + 1
            print(count3)

    return count3
```

www.wallpaperplay.com

```
def scp(url,m,link_init):
    source_code = requests.get(url)

    plain_text = source_code.text

    soup = BeautifulSoup(plain_text,"lxml")

    set_img = set([])
    i=0
    for link in soup.find_all("img"):
        href = link.get('data-src')
        href = link_init + href
        set_img.add(href)
        i = i + 1
        print(href)
        print(i)

    count=0
    flag = m
    name = m
    for j in set_img:
        if count>100:
            break
        else:
            print(j)
            count=count+1
            print(count)
            img_name = name
            name = name + 1
            full_name = str(img_name) + ".jpg"
            urllib.request.urlretrieve(j, full_name)
            print("loop break")
            if (name-flag)==len(set_img):
                return name
```

www.wallpapercave.com

```
file = open('links_not_working.txt','r')
error_links = file.readlines()
temp = [i[:-1] for i in error_links]
```

```
def scrape(url, count):
    site = str(url)
    site = site.rstrip()
    source_code = requests.get(site)

    plain_text = source_code.text

    soup = BeautifulSoup(plain_text,"html.parser")

    a=0
    set_img = set([])

    for link in soup.find_all("img", {"class" : "wpimg"}):
        href = link.get('src')
        alt = link.get('alt')
        a+=1
        if href.endswith('.png') or href.endswith('.gif'):
            continue
        else:
            link_init = "https://wallpapercave.com"
            href = link_init + href
            set_img.add(href)
            print(href)
            print(a)

    count3 = count

    for image in set_img:
        if image in temp:
            continue
        else:
            img_name = count3
            full_name = str(img_name) + ".jpg"
            print("loop break")
            urllib.request.urlretrieve(str(image), full_name)
            count3 = count3 + 1
            print(count3)

    return count3
```

www.sardinia.com

```
file = open('links_not_working.txt','r')
error_links = file.readlines()
temp = [i[:-1] for i in error_links]

def scrape(url, count):
    site = str(url)
    site = site.rstrip()
    source_code = requests.get(site)

    plain_text = source_code.text

    soup = BeautifulSoup(plain_text,"html.parser")

    a=0
    set_img = set([])

    for link in soup.find_all("img"):
```

```
        href = link.get('src')
        alt = link.get('alt')
        a+=1
        if href.endswith('.png') or href.endswith('.gif'):
            continue
        else:
            link_init = "https:"
            href = link_init + href
            set_img.add(href)
            print(href)
            print(a)

    count3 = count

    for image in set_img:
        if image in temp:
            continue
        else:
            img_name = count3
            full_name = str(img_name) + ".jpg"
            print("loop break")
            urllib.request.urlretrieve(str(image), full_name)
            count3 = count3 + 1
            print(count3)

    return count3
```

www.standard.co.uk

```
file = open('links_not_working.txt','r')
error_links = file.readlines()
temp = [i[:-1] for i in error_links]

def srp(url,m):
    url = str(url)
    url = url.rstrip()

    source_code = requests.get(url)

    plain_text = source_code.text

    soup = BeautifulSoup(plain_text,"lxml")
    i=0
    set_img = set([])
    for link in soup.find_all("amp-img"):
        href = link.get('src')
        if href.startswith("https://static.standard.co.uk/"):
            if href not in temp:
                set_img.add(href)
        else:
            continue

    count=0
    flag = m
    name = m
    for j in set_img:
        if j in temp:
            continue
        else:
            print(j)
            count=count+1
            print(count)
            img_name = name
```

```
            name = name + 1
            full_name = str(img_name) + ".jpg"
            urllib.request.urlretrieve(j, full_name)
            print("loop break")
            if (name-flag) == len(set_img):
                return name
```

2.Data Preprocessing:-

Augmentation:

```
import keras
import cv2,random
from keras.preprocessing.image import
ImageDataGenerator,array_to_img,img_to_array,load_img
import numpy as np
import imageio
import matplotlib.pyplot as plt

def augment_img(original_image,flag):
    image = np.expand_dims(original_image,0)
    orginal_copy = image.copy()

    if flag == 1:
        aug_bright =  ImageDataGenerator(brightness_range =
(0.5,2),fill_mode="nearest")
        i=0
        for j in aug_bright.flow(orginal_copy):
            j = j[0].astype('uint8')
            i += 1
            if i>0:
                break

    elif flag == 2:
        aug_rotate =
ImageDataGenerator(rotation_range=5,fill_mode="nearest")
        i=0
        for j in aug_rotate.flow(orginal_copy):
            j = j[0].astype('uint8')
            i += 1
            if i>0:
                break

    elif flag == 3:
        aug_width_shift =
ImageDataGenerator(width_shift_range=0.15,fill_mode="nearest")
        i=0
        for j in aug_width_shift.flow(orginal_copy):
            j = j[0].astype('uint8')
            i += 1
            if i>0:
                break

    elif flag == 4:
        aug_height_shift =
ImageDataGenerator(height_shift_range=0.15,fill_mode="nearest")
        i=0
        for j in aug_height_shift.flow(orginal_copy):
            j = j[0].astype('uint8')
            i += 1
            if i>0:
```

```
                break

    elif flag == 5:
        aug_zoom =  ImageDataGenerator(zoom_range =
0.3,fill_mode="nearest")
        i=0
        for j in aug_zoom.flow(orginal_copy):
            j = j[0].astype('uint8')
            i += 1
            if i>0:
                break

    elif flag == 6:
        aug_hflip =  ImageDataGenerator(horizontal_flip =
True,fill_mode="nearest")
        i=0
        for j in aug_hflip.flow(orginal_copy):
            j = j[0].astype('uint8')
            i += 1
            if i>0:
                break
    return j
```

Creating Numpy Batches

```
import os
import cv2
import numpy as np
import random
import augmentation as aug

raw_data = "data/raw_data/beach_dataset"
train_save = "data/prep_data/train_save"
test_save = "data/prep_data/test_save"

files = os.listdir(raw_data)
random.shuffle(files)
train_percentage = 0.9
train_files = files[ : int(len(files) * train_percentage)]
test_files = files[int(len(files)*train_percentage):]
total_train_images = 0
total_test_images = 0
no_of_augment = 2
img_input = (256, 256)
batch_size = 2000
img_train_count = 0
img_test_count = 0

def save_img(batches,flag):
    global img_train_count,img_test_count
    if flag:
        img_train_count += 1
        save_path = os.path.join(train_save,"img_train_"+str(img_train_c
ount))
    else:
        img_test_count += 1
```

```
        save_path = os.path.join(test_save,"img_test"+str(img_test_count
))
    np.save(save_path,batches)

def make_data(img_files,flag,augments = no_of_augment):
    global total_train_images,total_test_images
    temp = []
    count_img = 0
    start = 1
    print("Processed:-")

    for src in img_files:
        img_src = os.path.join(raw_data,src)
        try:
            img = cv2.imread(img_src,cv2.IMREAD_COLOR)
            img = cv2.resize(img,img_input)
            temp.append(img)
            count_img +=1
            print("\r",count_img,end="",flush=True)
            choose = [1,2,3,4,5,6]
            for k in range(augments):
                choice = random.choice(choose)
                choose.remove(choice)
                image_mod = aug.augment_img(img,choice)
                count_img += 1
                temp.append(image_mod)
                print("\r",count_img,end="",flush=True)
        except Exception as e:
            print("Error:- ",e)
            print("File-Name:-",img_src)
        start = start + 1
        if len(temp) >= batch_size or start == len(img_files):
            print("Batch Saved :-",len(temp))
            save_img(temp,flag)
            temp = []
    if flag == True:
        total_train_images += count_img
    else:
        total_test_images += count_img
print("***TRAIN DATASET***")
make_data(train_files, flag = True)

print("***TEST DATASET***")
make_data(test_files, flag = False)

print("*-"*25)
print("Data preparation done!!")
print("*-"*25)
print("Total train images: ", total_train_images)
print("Total test images: ", total_test_images)
```

3. Image Extrapolation:

```
from keras.layers.convolutional import Conv2D, AtrousConvolution2D
from keras.layers import Activation, Dense, Input, Conv2DTranspose, Dens
e, Flatten
from keras.layers import ReLU, Dropout, Concatenate, BatchNormalization,
 Reshape
from keras.layers.advanced_activations import LeakyReLU
from keras.models import Model, model_from_json
from keras.optimizers import Adam
from keras.layers.convolutional import UpSampling2D
from IPython.display import clear_output
from datetime import datetime
from dataloader import Data, TestData
import keras.backend as K
import tensorflow as tf
import os
import numpy as np
import PIL
import cv2
import IPython.display

try:
    from keras_contrib.layers.normalization import InstanceNormalization
except Exception:
    from keras_contrib.layers.normalization.instancenormalization import
 InstanceNormalization

train_data = Data()
test_data = TestData()

IMAGE_DIMS = (256, 256, 3)
FLAG, PERCENTCROP = 1, .25
EPSILON, ALPHA = 1e-9, 0.0004
CHECKPOINT = "checkpoint_original/"
SAVED_IMAGES = "saved_images_original/"
```

Models:

(i)Discriminator:

```
def lossFunc_disc(true_val, pred_val):
    return -
tf.reduce_mean(tf.log(tf.maximum(true_val, EPSILON)) + tf.log(tf.maximum
(1. - pred_val, EPSILON)))

discm_inp_shape = (IMAGE_DIMS[0], int(IMAGE_DIMS[1] * (PERCENTCROP *2)),
 IMAGE_DIMS[2])
d_dropout = 0.25
OPTIMIZER_D = Adam(0.0001, 0.5)
```

```
def discm_convolution(inp_layers, no_of_filters, dims_kernel=4,
strides=2, activation='leakyrelu',
dropout_rate=d_dropout,normalization='inst'):
    layer_d = Conv2D(no_of_filters, kernel_size=dims_kernel,
strides=strides, padding='same')(inp_layers)
    if activation == 'leakyrelu':
        layer_d = LeakyReLU(alpha=0.2)(layer_d)
    if normalization == 'inst':
        layer_d = InstanceNormalization()(layer_d)
    if dropout_rate:
        layer_d = Dropout(dropout_rate)(layer_d)
    return layer_d

def discm_layers():
    inp_discm = Input(shape=discm_inp_shape)
    x1 = discm_convolution(inp_discm, 32, 5,strides=2, normalization
=False)

    x2 = discm_convolution(x1, 64, 5, strides=2)
    x3 = discm_convolution(x2, 64, 5, strides=2)
    x4 = discm_convolution(x3, 128, 5, strides=2)
    x5 = discm_convolution(x4, 128, 5, strides=2)

    x6 = Flatten()(x5)
    x7 = Dense(1024, activation='relu')(x6)
    out_discm = Dense(1, activation='sigmoid')(x7)

    return Model(inp_discm, out_discm)

DIS_OBJ1 = discm_layers()
DIS_OBJ1.compile(loss=lossFunc_disc, optimizer=OPTIMIZER_D)
if True:
    Dis_obj1.summary()
```

(ii)Generator:

```
def lossFunc_gen(true_val, pred_val):
    MseLoss_gen = K.mean(K.square(pred_val - true_val))
    return MseLoss_gen - ALPHA * tf.reduce_mean(tf.log(tf.maximum(pred_v
al, EPSILON)))
gen_inp_shape = (IMAGE_DIMS[0], int(IMAGE_DIMS[1] * (PERCENTCROP *2)), I
MAGE_DIMS[2])
g_dropout = 0.25
OPTIMIZER_G = Adam(0.001, 0.5)


def gen_convolution(inp_layers, no_of_filters, dims_kernel=4, strides=2,
 activation='relu', dropout_rate=g_dropout, normalization
='inst', dilation=1):
    layer_g = AtrousConvolution2D(no_of_filters, kernel_size=dims_kernel
, strides=strides,atrous_rate=(dilation,dilation), padding='same')(inp_l
ayers)
```

```
    if activation == 'relu':
        layer_g = ReLU()(layer_g)
    if dropout_rate:
        layer_g = Dropout(dropout_rate)(layer_g)
    if normalization == 'inst':
        layer_g = InstanceNormalization()(layer_g)
    return layer_g


def gen_deconvolution(inp_layers, no_of_filters, dims_kernel=3, strides=
2, activation='relu', dropout=0):
    layer_g_deconv = Conv2DTranspose(no_of_filters, kernel_size=dims_ker
nel, strides=strides, padding='same')(inp_layers)
    if activation == 'relu':
        layer_g_deconv = ReLU()(layer_g_deconv)
    return layer_g_deconv


def gen_layers():
    inp_gen = Input(shape=gen_inp_shape)

    y1 = gen_convolution(inp_gen, 64, 5, strides=1)
    y2 = gen_convolution(y1, 128, 4, strides=2)
    y3 = gen_convolution(y2, 256, 4, strides=2)
    y4 = gen_convolution(y3, 512, 4, strides=1)
    y5 = gen_convolution(y4, 512, 4, strides=1)
    y6 = gen_convolution(y5, 512, 4, strides=1, dilation=2)
    y7 = gen_convolution(y6, 512, 4, strides=1, dilation=4)
    y8 = gen_convolution(y7, 512, 4, strides=1, dilation=8)
    y9 = gen_convolution(y8, 512, 4, strides=1, dilation=16)
    y10 = gen_convolution(y9, 512, 4, strides=1)
    y11 = gen_convolution(y10, 512, 4, strides=1)

    y12 = gen_deconvolution(y11, 256, 4, strides=2)
    y13 = gen_deconvolution(y12, 128, 4, strides=2)

    y14 = gen_convolution(y13, 128, 4, strides=1)
    y15 = gen_convolution(y14, 64, 4, strides=1)

    out_gen=AtrousConvolution2D(3,kernel_size=4,strides=(1,1),activation
='tanh',padding='same',atrous_rate=(1,1))(y15)

    return Model(inp_gen, out_gen)

GEN_OBJ1 = gen_layers ()
GEN_OBJ1.compile(loss=lossFunc_gen, optimizer= OPTIMIZER_G)
if True:
    GEN_OBJ1.summary()
```

Combined Model:

```
IMAGE = Input(shape=gen_inp_shape)
Dis_obj1.trainable = False
GENERATED_IMAGE = GEN_OBJ1(IMAGE)
CONF_GENERATED_IMAGE = Dis_obj1(GENERATED_IMAGE)
COMBINED = Model(IMAGE, [CONF_GENERATED_IMAGE, GENERATED_IMAGE])
COMBINED.compile(loss=['mse', 'mse'], optimizer=OPTIMIZER_G)
```

Image Cropping:

```
def crop_percentage(image):
    orginial_image = image.copy()
    h = orginial_image.shape[0]
    w = orginial_image.shape[1]
    cropped_width = int(w * PERCENTCROP)
    left = original_image[:, :cropped_width]
    right = original_image[:, w - cropped_width:]
    l_r = np.concatenate((left, right), axis=1)
    c = orginal_image[:, 65:193]
    return c, l_r


def image_crop(images):
    center_crop = []
    lr_crop = []
    for i in images:
        center, left_right = crop_percentage(i)
        center_crop.append(center)
        lr_crop.append(left_right)
    return np.array(center_crop), np.array(lr_crop)


def join(middle_portion, generated_images):
    complete = []
    for center, gen_img in zip(middle_portion, generated_images):
        w = gen_img.shape[1] // 2
        left = gen_img[:, :w]
        right = gen_img[:, w:]
        final = np.concatenate((left,center,right), axis=1)
        complete.append(final)
    return np.asarray(complete)
```

4. Train:

```
def train():
    TIME_INTERVALS = 2
    start = datetime.now()
    sv_time = start
    for epoch in range(1, 51):
        steps = 1
```

```
        test = None
        while True:
            img = train_data.get_data(FLAG)
            if img is None:
                break
            size = img.shape[0]

            center, left_right = image_crop(img)
            center = center / 127.5 - 1
            left_right = left_right / 127.5 - 1
            gen_predict = GEN_OBJ1.predict(center)
            real = np.ones([size, 1])
            fake = np.zeros([size, 1])
            real_loss = Dis_obj1.train_on_batch(left_right, real)
            fake_loss = Dis_obj1.train_on_batch(gen_predict, fake)
            loss_disc = (np.add(real_loss, fake_loss))/2

            for i in range(2):
              loss_gen=COMBINED.train_on_batch(center,[real,left_right])

            log = "epoch: %d, steps: %d, DIS loss: %s, GEN loss: %s" \
                        %(epoch, steps, str(loss_disc), str(loss_gen[0]))
            print(log)
            save_log(log)
            steps += 1
            curr_time = datetime.now()
            difference_time = curr_time - sv_time
            if difference_time.seconds >= (TIME_INTERVALS * 60):
                save_model()
                save_image(epoch, steps)
                sv_time = curr_time
        clear_output()
```

5.Test:

```
def test(image):
    final_image = None
    gen_predict = None
    demask_image = None
    x, y = image_crop([image])
    gen_predict = GEN_OBJ1.predict(x)
    final_image = join(x, gen_predict)[0]
    return final_image

images = test_data.get_data(1050)
path = 'test/'
for i, image in enumerate(images):
    image = image / 127.5 - 1
    image = test(image)
    image = (image + 1) * 127.5
    image = image.astype(np.uint8)
    filename = str(i + 1)+'.jpg'
```

```
    cv2.imwrite(os.path.join(path , filename), image)
```

6. Recursive Painting:

```
def rec_extrapolate(image, extrapolate_times=3):
    final_image = None
    gen_predict = None
    for i in range(extrapolate_times):
        demask_image = None
        if i == 0:
            x, y = image_crop([image])
            gen_predict = GEN_OBJ1.predict(x)
            final_image = join(x, gen_predict)[0]
        else:
            gen_predict = GEN_OBJ1.predict(gen_predict)
            final_image = join([final_image], gen_predict)[0]
    return final_image
images = train_data.get_data(1)
for rec_img in images:
    rec_img = rec_img / 127.5 - 1
    rec_img = rec_extrapolate(rec_img)
    rec_img = (rec_img + 1) * 127.5
    rec_img = rec_img.astype(np.uint8)
    rec_img = cv2.cvtColor(rec_img, cv2.COLOR_BGR2RGB)
    IPython.display.display(PIL.Image.fromarray(rec_img))
```

7. Testing from url:

```
import urllib.request
path='https://media.cntraveler.com/photos/5787ae73070aa7e234ba1fa2/maste
r/w_820,c_limit/beaches-europe--Zlatni-rat-croatia-GettyImages-
537453216.jpg'
file_name = os.path.basename(path)
_ = urllib.request.urlretrieve(path, file_name)

org_img = cv2.imread(file_name)
org_img = cv2.resize(org_img, (256,256))
center = org_img [:, 65:193]
input_image = center / 127.5 - 1
input_image = np.expand_dims(input_image, axis=0)
gen_predict = GEN_OBJ1.predict(input_image)

gen_predict = join(input_image, gen_predict)[0]
gen_predict = (gen_predict + 1) * 127.5
gen_predict = gen_predict.astype(np.uint8)
org_img = cv2.cvtColor(org_img, cv2.COLOR_BGR2RGB)
gen_predict = cv2.cvtColor(gen_predict, cv2.COLOR_BGR2RGB)
print('Original Image')
```

```
IPython.display.display(PIL.Image.fromarray(org_img))
print('Generated Image')
IPython.display.display(PIL.Image.fromarray(gen_predict))
```

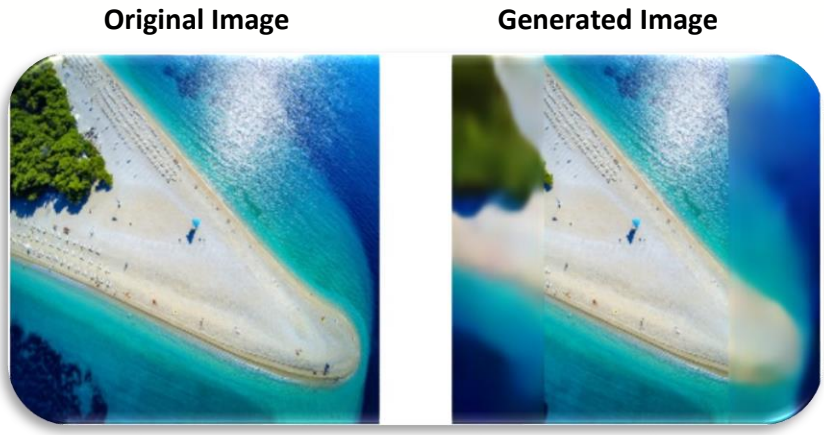


Fig24: URL Image Output

Future Works:

There are numerous improvements that can be made to our image extrapolation method to increase the output fidelity.

- One such method is improving the resolution of the extrapolated images. The process of improving the resolution of images is also known as super resolution. Thus, super resolution is increasing the resolution of an input image by a factor s. In order to overcome this problem, a special form of generative adversarial network known as SRGAN can be used. The SRGAN [11] tries to recover the finer texture details when super-resolving at large upscaling factors. SRGAN uses a perceptual loss function which consists of an adversarial loss and a content loss. The adversarial loss pushes the solution to the natural image manifold using a discriminator network that is trained to differentiate between the super-resolved images and original photo-realistic images. In addition, content loss is motivated by perceptual similarity instead of similarity in pixel space. Deep residual network is used in SRGAN which is able to recover photo-realistic textures from heavily downsampled images.

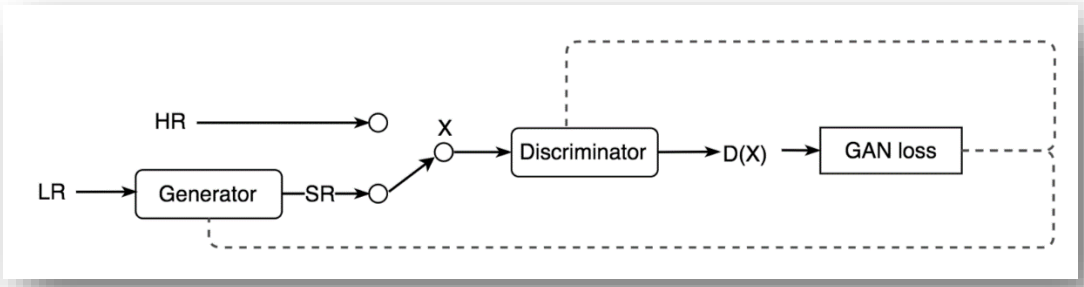


Fig25: Training Pipeline used in SRGAN

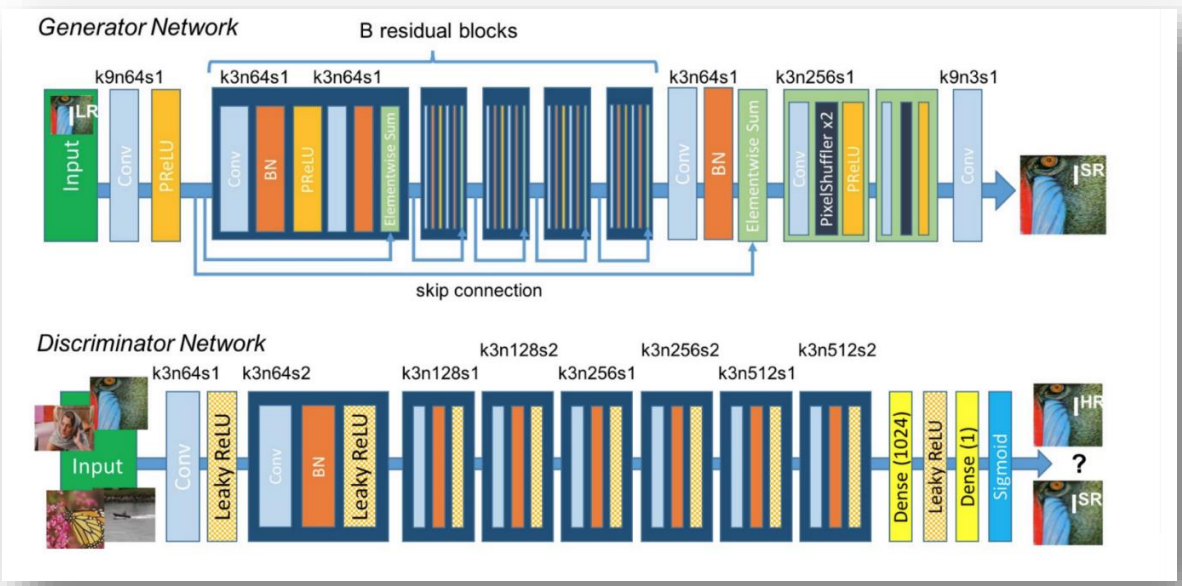


Fig26: Architecture used in SRGAN

- Another approach similar to SRGAN is using SinGAN. SinGAN [8] is an unconditional generative model that is trained on a single natural image. This task is conceptually similar to the conventional GAN setting, except that here the training samples are patches of a single image, rather than whole image samples from a dataset. SinGAN is trained to capture the internal distribution of patches within the image. After learning the features, it is then able to generate high quality, diverse samples that carry the same visual content as the image. SinGAN uses a multi-scaled training pipeline. SinGAN contains a pyramid of fully convolutional GANs, each responsible for learning the patch distribution at a different scale of the image. In other words, SinGAN consists of a pyramid of GANs, where both training and inference are done in a coarse-to-fine fashion. At each scale, Generator G_n learns to generate image samples in which all the overlapping patches cannot be distinguished from the patches in the down-sampled training image, x_n, by the discriminator D_n; the effective patch size decreases as we go up the pyramid (marked in yellow on the original image for illustration). The input to G_n is a random noise image z_n, and the generated image from the previous scale x_n, upsampled to the current resolution (except for the coarsest level which is purely generative). The generation process at level n involves all generators $\{G_N \ldots G_n\}$ and all noise maps $\{z_N, \ldots, z_n\}$ up to this level. Once trained, SinGAN can produce diverse high quality image samples which semantically resemble the training image, yet contain new object configurations and structures. However, SinGAN can only produce images with the same patch distribution as the training image.

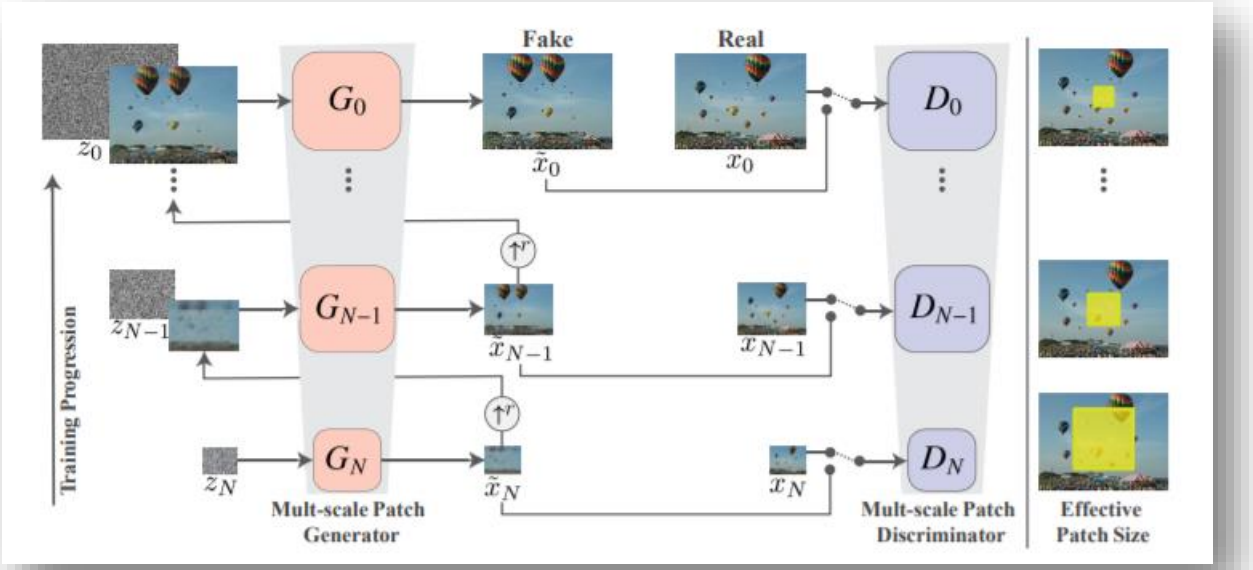


Fig27: Multi-scale pipeline used in SinGAN

Conclusion:

In this project, we scraped beach images from the multiple websites and performed image augmentation in order to increase the number of training images. Then the augmented images in the dataset were cropped from both ends which, in turn, were fed to the DCGAN architecture (G, D). After training the model for 50 epochs, the model was successfully able to extrapolate cropped images. We also used dilation in order to increase the receptive field of the neurons. Once training the model was over, we also used a technique known as recursive painting for three times. Though the noise increased with each recursive step, we observed that the model was successful in learning the general textures of the sky and the sea.

Reference:

1. M. Wang, Y. Lai, Y. Liang, R. R. Martin, and S.M. Hu. Biggerpicture: data-driven image extrapolation using graph matching. ACM Transactions on Graphics, 33(6), 2014.
2. D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2536–2544, 2016.
3. S. Iizuka, E. Simo-Serra, and H. Ishikawa. Globally and locally consistent image completion. ACM Transactions on Graphics.
4. G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro. Image inpainting for irregular holes using partial convolutions. arXiv preprint arXiv:1804.07723, 2018.
5. Y. Zhang, J. Xiao, J. Hays and P. Tan (2013) Framebreak: Dramatic image extrapolation by guided shift-maps.
6. Basile Van Hoorick: Image Extrapolation and Harmonization using Generative Adversarial Networks.
7. Zongxin Yang, Jian Dong, Ping Liu: Very Long Natural Scenery Image Prediction by Extrapolation.
8. T. R. Shaham, T. Dekel and T. Michaeli (2019-05) SinGAN: Learning a Generative Model from a Single Natural Image.
9. Documentation on Image Preprocessing using Keras.
10. Generative Adversarial Nets Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozairy, Aaron Courville, Yoshua Bengio.
11. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network - Christian Ledig • Lucas Theis • Ferenc Huszar • Jose Caballero • Andrew Cunningham • Alejandro Acosta • Andrew Aitken • Alykhan Tejani • Johannes Totz • Zehan Wang • Wenzhe Shi.
12. tensorflow.org/tutorials/generative/dcgan
13. towardsdatascience.com/dcgans-deep-convolutional-generative-adversarial-networks-c7f392c2c8f8
14. Unsupervised Representation Learning using Deep Convolutional Generative Adversarial Networks-Alec Radford & Luke Metz, Soumith Chintala
15. intellipaat.com/community/1869/instance-normalisation-vs-batch-normalisation
16. towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
17. medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b