

A PROJECT REPORT
ON

IMPLEMENTATION OF GENERATIVE ADVERSARIAL NETWORKS FOR IMAGE INPAINTING

Submitted in partial fulfillment of the requirement for the award of Degree of
Bachelor of Technology in Information Technology

Submitted to:



Rajasthan Technical University, Kota (Raj.)

Submitted By:

Jinesha Jain (18EARIT025)

Chirag Agarwal (18EARIT015)

Hemant Harsh Rao (18EARIT023)

Ravindra Kumar (18EARIT047)

Under the supervision of

GUIDE

Dr. Vibhakar Pathak
Professor

PROJECT COORDINATOR

Dr. Vibhakar Pathak
Professor



Session: 2021-22

Department of Information Technology
ARYA COLLEGE OF ENGINEERING & I.T.
SP-42, RIICO INDUSTRIAL AREA, KUKAS, JAIPUR



Department of Information Technology

CERTIFICATE

This is to certify that the work embodies in this Project entitled **“IMPLEMENTATION OF GENERATIVE ADVERSARIAL NETWORK (GAN) FOR IMAGE INPAINTING”** being submitted by **“Jinesha Jain (18EARIT025), Chirag Agarwal (18EARIT015), Hemant Harsh Rao (18EARIT023) and Ravindra Kumar (18EARIT047)”** in partial fulfillment of the requirement for the award of **“Bachelor of Technology in Information Technology”** to Rajasthan Technical University, Kota (Raj.) during the academic year 2021-22 is a record of bonafide piece of work, carried out by them under our supervision and guidance in the **“Department of Information Technology”, Arya College of Engineering & Information Technology, Jaipur.**

Project Guide
Dr. Vibhakar Pathak
Professor

Project-coordinator
Dr. Vibhakar Pathak
Professor

Approved by

Dr. Vibhakar Pathak
Head, Department of
Information Technology



Department of Computer Science & Engineering

CERTIFICATE OF APPROVAL

The Major Project Report entitled “**IMPLEMENTATION OF GENERATIVE ADVERSARIAL NETWORK (GAN) FOR IMAGE INPAINTING**” submitted by “Jinesha Jain (18EARIT025), Chirag Agarwal (18EARIT015), Hemant Harsh Rao (18EARIT023) and Ravindra Kumar (18EARIT047)” has been examined by us and is hereby approved for carrying out the project leading to the award of degree “**Bachelor of Technology in Information Technology**”. By this approval the undersigned does not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but approve the pursuance of project only for the above mentioned purpose.

Project Guide
Dr. Vibhakar Pathak
Professor

Project Coordinator
Dr. Vibhakar Pathak
Professor



Department of Information Technology

DECLARATION

We “**Jinesha Jain, Chirag Agarwal, Hemant Harsh Rao and Ravindra Kumar**”, students of **Bachelor of Technology in Information Technology**, session **2021-22**, **Arya College of Engineering & Information Technology, Jaipur**, hereby declare that the work presented in this Project entitled “**IMPLEMENTATION OF GENERATIVE ADVERSARIAL NETWORK (GAN) FOR IMAGE INPAINTING**” is the outcome of our own work, is bonafide and correct to the best of our knowledge and this work has been carried out taking care of Engineering Ethics. The work presented does not infringe any patented work and has not been submitted to any other University or anywhere else for the award of any degree or any professional diploma.

Jinesha Jain (18E1ARITF45P025)
Chirag Agarwal (18E1ARITM40P015)
Hemant Harsh Rao (18E1ARITM40P023)
Ravindra Kumar (18E1ARITM30P047)

Date: 11-06-2022

ACKNOWLEDGEMENT

We have put our efforts to the full potential for the completion of this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

We are highly indebted to **Dr. Vibhakar Pathak** for his guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

We would like to express my gratitude towards all the members of Arya College of Engineering and I.T. for their kind cooperation and encouragement which helped me in the completion of this project.

ABSTRACT

The project discloses about detailed modules used for the implementation of project. The aim of the project is to heal the images with patches using (GAN) is justified by using mask datasets and image datasets of high-resolution. The ML and image processing techniques involve use of various python libraries and TensorFlow. The software TensorFlow supports the project training model with the help of its GPU and CPU processors. The image inpainting uses GAN and models like Wasserstein GAN, VGG and further discusses the role of Generators and Discriminators in the project. The Generator and Discriminators are part of GMCNN. The generator is trained to recreated damaged or input image. The pictures are then passed to Discriminators (which are also trained), which further passes the recreated image by Generator as “real” or “fake”. Training of the generators and discriminators would not have been possible without the image datasets, and mask dataset, comprising of 36,000 and 12,000 images respectively. The project implemented on Google Colab has been able to heal the images but not at great accuracy due to the availability of limited resources. The GPU processors needed for excellent accuracy requires better GPU for carrying out heavy training. For better understanding, the various losses are monitored using TensorFlow and represented using graphs. The losses are calculated to determine accuracy. Lesser the loss, better is the accuracy.

TABLE OF CONTENTS

ACKNOWLEDGEMENT	1
ABSTRACT.....	2
TABLE OF CONTENTS.....	3
CHAPTER – 1 OBJECTIVE & SCOPE OF THE PROJECT.....	4
CHAPTER – 2 THEORETICAL BACKGROUND.....	5
CHAPTER – 3 DEFINITION OF PROBLEM.....	12
CHAPTER – 4 SYSTEM ANALYSIS & USER REQUIREMENTS.....	13
CHAPTER – 5 SYSTEM PLANNING	15
CHAPTER – 6 METHODOLOGY ADOPTED & DETAILS OF HARDWARE & SOFTWARE USED.....	17
CHAPTER – 7 DETAILED LIFE CYCLE OF THE PROJECT	19
CHAPTER -8 ERD AND DFD	24
CHAPTER – 9 PROCESS INVOLVED, ALGORITHM, FLOWCHART, DATABASE DIAGRAM	26
CHAPTER – 10 INPUT AND OUTPUT SCREEN DESIGN	30
CHAPTER – 11 CODESHEETS.....	35
CHAPTER – 12 TESTING.....	41
CHAPTER – 13 USER/ OPERATIONAL MANUAL	42
CHAPTER – 14 CONCLUSIONS	43
CHAPTER – 15 FUTURE ENHANCEMENT	44
REFERENCES	45

CHAPTER – 1

OBJECTIVE & SCOPE OF THE PROJECT

We have found that, in the era of machines, Machine Learning (ML) has grown exponentially and resulted in performing day-to-day tasks as well as performing complex calculations. Also, with the fusion of ML with image processing techniques, there are many identifications and recognition-based projects that have been implemented. Similarly, using GAN model, we aim to fill the gaps of the pictures that are damaged or corrupted over time or because of the other reasons. The detailed explanation is supported in further sections and subsections.

CHAPTER – 2

THEORETICAL BACKGROUND

Inpainting is the way toward reproducing lost or disintegrated portions of pictures and recordings. In the historical center world, on account of a significant artwork, this assignment would be done by a talented craftsmanship conservator or workmanship restorer. In the advanced world, inpainting alludes to the use of modern calculations to supplant lost or adulterated pieces of the picture information. This official meaning of inpainting on Wikipedia as of now considers the utilization of "advanced calculations" that accomplish a similar work of physically overwriting blemishes or fixing abandons however in a small amount of the time. As profound learning advances progress further, in any case, the procedure of inpainting has gotten mechanized in so complete a way that nowadays, it requires no human mediation by any means. Basically, feed a harmed picture to a neural system and get the amended yield.

For detailed study we did a literature survey upon knowing the existing methods and tools for image inpainting. We found out about various papers: -

[1] Generative Image Inpainting with Contextual Attention

The paper proposes a coarse-to-fine generative image inpainting framework and introduces a baseline model as well as full model with a novel contextual attention module. The contextual attention module proves to significantly improve imagepainting results by learning feature representations for explicitly matching and attending to relevant background patches.

[2] New Inpainting Algorithm Based on Simplified Context Encoders and Multi-Scale Adversarial Network

This paper changes the system structure, adjusts the convolution bit and walks in the reproduce arrange, and presents a multi-scale ill-disposed system which is predictable with worldwide discriminator and a neighborhood discriminator. Subsequently, the misfortune

work is adjusted as needs be. The misfortune work comprises of two sections: remaking misfortune and multi-scale as opposed to learning misfortune. The system engineering is applied to the Paris dataset, which shows sensible fix execution. It seems to improve the Context Encoders' fixing and obscuring issues, and acquires better inpainting results. Contrasting and Context Encoders and GAN-based face picture reclamation extends, the system structure of this paper is progressively brief and quicker.

[3] A new method of Thangka image inpainting quality assessment

The paper proposes a non-reference picture inpainting quality assessment strategy for consolidating the evenness of Thangka picture with Harrison Conner point. Due to lack of Thangka image database, new method called “GAN” is also use in order to extend to the experimental data sets. It has realized better appraisal even in complex districts. It plans to complete Thangka image database for making new method more credible.

[4] Multi-scale semantic image inpainting with residual learning and GAN

The paper centers around a multi-layer convolutional model for picture inpainting. It demonstrates that skip connections built between layers are suitable for the ability of increasing semantic interferences' effectiveness while multi-convolution can smooth the discontinuous edges. The strategy utilized is superior to gauge techniques and improves the goals nature of fixed partners successfully. With Respect to the concepts of MSE, PSNR and SSIM, the results are proven to be convincingly demonstrating in its superiority and significance.

[5] Data-driven high-fidelity 2D microstructure reconstruction via non-local patch-based image inpainting

In this paper, a microstructure remaking structure dependent on the picture inpainting technique is proposed to solve the micro-structure reconstruction problem in three different

contexts. First setting centers around recreation of the impeded district. Second context focuses on outpatient problem. Its objective is to expand the original microstructure to a larger region. The third context focuses on assembly problem. Objective is to reconstruct the whole microstructure, given a collection of microstructures spatially. In the second context, this is solved by using the microstructure reconstruction framework. The UHCS datasets help in demonstrating the microstructure reconstruction in three contexts.

[6] Markov Random Field-based image inpainting with direction structure distribution analysis for maintaining structure coherence

Paper proposes another MRF-based picture inpainting calculation especially for keeping up structure rationality. To truly and consequently pick DDS, a bearing structure circulation examination methodology is proposed. After obtaining required direction edge image(s) and non-edge image, we individually calculate statistics offsets, which provides a more reliable hint for completing damaged images and the efficiency is higher than or competitive to the compared approaches.

[7] Average-face-based virtual inpainting for severely damaged statues of Dazuo Rock Carvings Paper proposes the strategy subject to a model database. The procedure ought to be practical and has different central focuses when there is insignificant earlier data in the chief picture. Ordinary picture can be genuinely observed as the amusement recommendation for the extraordinarily testing inpainting task where there are no relative priors in the main picture. Run of the mill picture can be immediate seen as the preoccupation proposal for the extraordinarily testing inpainting task where there are no relative priors in the essential picture. Paper also ensures on getting progressively possible results using the technique, when stood out from other related computations when the mischief is tremendous degree and inconsistent. The paper moreover examines the imperatives to a manual face achievement.

[8] Gradient-aware blind face inpainting for deep face verification

In this paper, another visually impaired inpainting technique is introduced to increase the efficiency of the face check between the ruined ID/life picture sets. To recuperate a reasonable ID face from the adulterated photograph, the paper proposes an improved Deep Recursive Residual Network design to demonstrate the non-straight mapping between the debased and clear ID picture sets. The proposed IDRRN takes both pixel and inclination closeness into thought during the preparation procedure. The inspiration of preparing objective is that limiting pixel or highlight level contrast, as the past works did, consistently obscure and crush the significant face edges and forms. This outcomes in a recuperated picture for example further away from genuine picture in the conservative element space.

[9] A deep learning approach to patch-based image inpainting forensics

In this paper, a novel visually impaired picture inpainting legal sciences technique dependent on profound learning has been introduced. In the technique, the creators built the class name grid for all the pixels of a picture during preparing the CNN, and planned the weighted cross-entropy as the misfortune work. The two fortifying administrative procedures were taken to control the CNN to naturally gain proficiency with the inpainting highlights as opposed to the picture content highlights. The proposed strategy has been broadly tried on different pictures, and contrasted and cutting-edge inpainting crime scene investigation techniques. Test results showed that the CNN-based strategy can naturally figure out how to distinguish picture inpainting control without considering highlight extraction and classifier structure, and utilizing any post-preparing as in regular legal sciences strategies. As indicated by the paper, the scientific techniques recommend that the CNN has essentially better criminology execution. In addition, it yields generally excellent strength against JPEG pressure and scaling controls, and offers the evident bit of leeway on running time.

COMPARISION:

S.No	Year	Paper Name	Author Name	Technique used	Drawback
1.	2018	Generative Image Inpainting with Contextual Attention	Jiahui Yu	Deep generative model-based approach	Do not give good result for high resolution inpainting applications
2.	2019	New Inpainting Algorithm Based on Simplified Context Encoders and Multi-Scale Adversarial Network	Haodi Wang	Combining the encoders and GAN	some cases where the image restoration results are poor, fuzzy, or even inconsistent with the context.
3.	2019	A new method of Thangka image inpainting quality assessment	Wenjin Hu	non-reference quality evaluation method	lack of Thangka image database

4.	2019	Multi-scale semantic image inpainting with residual learning and GAN	Libin Jiao	encoder-decoder generator and multi-layered convolutional net	Do not work for high-resolution image inpainting
5.	2019	Data-driven high-fidelity 2D microstructure reconstruction via non-local patch-based image inpainting	Anh Tran	Microstructure reconstruction method	Don't work with the images of large size and high resolution
6.	2019	Markov random field-based image inpainting with direction structure distribution analysis for maintaining structure coherence	Jixiang Cheng	Direction structure distribution analysis strategy	the performance of our method on inpainting images with curved structure needs to be further improved
7.	2019	Average-face-based virtual inpainting for severely damaged	Haiyan Wang	Average-face based method	existing excellent face detection technologies

		statues of Dazu Rock Carvings			cannot be directly utilized
8.	2019	Gradient-aware blind face inpainting for deep face verification	Fuzhang Wu	Deep Recursive Residual Network	can be further improved by incorporating with new network types such as GAN- like structure.
9.	2018	A deep learning approach to patch-based image inpainting forensics	Xinshan Zhu	CNN	Computational cost is very high

CHAPTER – 3

DEFINITION OF PROBLEM

Handling images using machine learning has always been challenging and one of the fields that gets introduced to new concepts regularly. With the rising demand of studying images in day- to-day lives such as in the field of astronomy, military or social uses, the efforts have been applied into using GAN model which supports self-learning technique and re-creating images with multiple yet probable possibilities.

The GAN features as Generator and discriminator, which after getting trained as per the data, generates the new data based on existing data and proves it real or fake.

This excites the level of performing image inpainting using GAN as it creates curiosity about testing the limits of machine to perceive world as human does. GAN is also based on trial and error concept as the results can be fake as well, yet one of the probable ones.

Image Inpainting can be of great use for data recovery:

- i) Astronomy: The better study if astronomical objects and creating more clearer images.
- ii) Military purposes for security
- iii) Recreating Legal evidences
- iv) Paleontology and archaeological excavations to revive ancient history

CHAPTER – 4

SYSTEM ANALYSIS & USER REQUIREMENTS

PRODUCT PERSPECTIVE

The product is a stand-alone project. The project can also be a part of some bigger projects as well. But, for now the project mainly focuses on image repairing by certain algorithms and adversarial networking tools. The product is a creation for helping out its customers in different ways. Some Product features focus on general use by people for socially using this product to repair any damaged image or make current image more attractive. Moreover, we expand our functional requirements to organization level to repair any image for professional purpose as well, which can be approached for military or health and safety purposes.

USER CHARACTERISTICS

The product developed so far is a demo project so it needs to be implemented online. So, the user should have a decent knowledge about Google Colab, Python language and file handling on cloud platforms like Google drive and GitHub. We will try to make it more user-friendly by linking the project with a web application if possible. By linking with a web app user characteristic will be limited to highly basic ability to operate web sites.

ASSUMPTION & DEPENDENCIES

Good Internet speed and cloud access for smooth running of the project. The project requires certain libraries of the python and tensorflow in order to perform GMCNN and Wasserstein loss calculations for certain image processing technique-based performances.

DOMAIN REQUIREMENTS

The project is a machine learning domain project which works on various techniques of Image Processing. It is implemented on an online platform. Since, the datasets are used from NVIDIA and is being updated on real-time basis we prefer to use Google Colab. The Google Colab has been a great domain for staging the required software of TensorFlow and Keras models as per the required versions.

USER REQUIREMENTS

For users we require image datasets, image inputs and masking datasets received from NVIDIA, which is updated on real-time basis and needs to be trained accordingly.

CHAPTER – 5

SYSTEM PLANNING

PRODUCT REQUIREMENTS

Product requires training of the neural network on regular intervals for accurate results. The training of Generator is done using the dataset and uses global Wasserstein loss, local Wasserstein loss, confidence reconstruction loss, id mrf loss and total loss.

EFFICIENCY

We are uploading the code online on a cloud platform so that the generator trains at regular intervals on real-time basis for accuracy. Moreover, the image datasets are a bit heavy in size, so we use Google Drive to upload the results. As far as we have come across with the project, the time taken is directly proportional to the size of datasets. We have set the count of number of times training the system as “Epoch”. More the no. of epochs better are the results. Since we needed to test the code for its working we discovered for a single epoch, the results are not as accurate as they are for multiple epochs.

RELIABILITY

Since, reliability depends on number of epochs we can say it is variable. Epochs speed may vary as per the internet speed and dataset size. In our project we used one Epoch with a step of 9200 (approx) which takes an average time of 2-3 hours.

PORTABILITY

The project is highly portable as we have uploaded the docs online in a GitHub repository and planning to put on a cloud server. Hence, this can be used anywhere online, from any system.

USABILITY

The usability is limited to a certain image files, which are retrieved from a folder. Since, this was developed to test the images are taken in a folder as sample images. In order to check for user images, we need to update the input image folder with the same.

ORGANIZATIONAL REQUIREMENTS

We plan to perform this image inpainting as a demo project. If we plan to deploy it in a more presentable manner, we need to create a proper python-based UI, so that any user can simply upload any image and create its own mask and then create a new repaired image.

ENGINEERING STANDARD REQUIREMENTS

This technique requires Flask API or Tkinter. Both, libraries of python, used for creating web- based framework or a software application for computers. Due to lack of knowledge of these API models we do not plan to create any User based website for this project.

CHAPTER – 6

METHODOLOGY ADOPTED & DETAILS OF HARDWARE & SOFTWARE USED

SOFTWARE/HARDWARE REQUIREMENTS

We needed GPU for execution of the project. this can be most vital as GPU is extremely powerful so as to coach the model, which relies on performing image processing algorithms at great extents. We used Python for this project because it is out and away very efficient in fields of Machine Learning and Image Processing. The libraries provided by python are: -

S. No.	Python Library	Version
1.	Keras	2.2.4
2.	Pydot	1.4.1
3.	Matplotlib	3.0.2
4.	H5py	2.9.0
5.	Pillow	6.2.0
6.	Opencv-python	3.4.3.18
7.	Tqdm	4.31.1
8.	Scikit-image	0.14.2

CPU REQUIREMENTS

We used TensorFlow 1.15.2 for implementation of the training model. The version we used is bit old. The rationale being the new TensorFlow models i.e., 2.x versions failed to support our project requirements for lossy conversions and step-based training epochs.

GPU REQUIREMENTS

Again, for similar reasons, we used TensorFlow-GPU version 1.15.2 for resolving compatibility issues. GPU is required to hold out model training processes due to its feature of transferring great deal of information. GPU uses its dedicated VRAM memory to transfer large chunks of memory at the time of model training. This helps in reducing load on the CPU and forestall from crashing systems.

CHAPTER – 7

DETAILED LIFE CYCLE OF THE PROJECT

STEPS TO EXECUTE THE PROJECT

As mentioned earlier, the project is implemented on Google Colab for its flexible usability and GPU required for model training. Steps involved in running the project: -

1. The necessary files i.e. NVIDIA mask datasets and image datasets are downloaded and extracted. (One-time step)
2. The pip command is used to install the necessary libraries and setting the environment. (One-time step)
3. The config.py file is executed, which creates directory named “config” and sets the configurations for image and mask datasets model training.
4. Next, we Train the generator. We train the generator in order to give it the ability to re- create the images by masking the images and recreating them. This process takes a lot of time. Since, we use 5 Epochs, consisting of 9125 steps each, the training time recorded is approximately 15 hours. Also, the risk of getting disconnected makes it a bit difficult to resume. So, the training saves at interval of 1000 steps. The configurations used for training Generator are as follows: -

```
%%writefile config/main_config.ini [TRAINING] WGAN_TRAINING_RATIO = 1
```

```
EPOCHS_COUNT = 5
```

```
BATCH_SIZE = 4
```

```
IMAGE_HEIGHT = 256
```

```
IMAGE_WIDTH = 256
```

CHANNELS_COUNT = 3

LEARNING_RATE = 0.0001

SAVE_MODEL_STEPS_PERIOD = 1000

[MODEL] ADD_MASK_AS_GENERATOR_INPUT = True
GRADIENT_PENALTY_LOSS_WEIGHT = 10

ID_MRF_LOSS_WEIGHT = 0.05

ADVERSARIAL_LOSS_WEIGHT = 0.001

NN_STRETCH_SIGMA = 0.5

VGG_16_LAYERS = 3,6,10

ID_MRF_STYLE_WEIGHT = 1.0

ID_MRF_CONTENT_WEIGHT = 1.0

NUM_GAUSSIAN_STEPS = 3

GAUSSIAN_KERNEL_SIZE = 32

GAUSSIAN_KERNEL_STD = 40.0

5. After completion of training of Generator, we train the local and global discriminators. This is termed as “Full Wasserstein GAN training mode”. The configurations required for training are: -

%%writefile config/main_config.ini [TRAINING] WGAN_TRAINING_RATIO = 5

EPOCHS_COUNT = 5

BATCH_SIZE = 4

IMAGE_HEIGHT = 256

IMAGE_WIDTH = 256

CHANNELS_COUNT = 3

LEARNING_RATE = 0.0002

SAVE_MODEL_STEPS_PERIOD = 500 [MODEL]

ADD_MASK_AS_GENERATOR_INPUT = True

GRADIENT_PENALTY_LOSS_WEIGHT = 10

ID_MRF_LOSS_WEIGHT = 0.05

ADVERSARIAL_LOSS_WEIGHT = 0.0005

NN_STRETCH_SIGMA = 0.5

VGG_16_LAYERS = 3,6,10

ID_MRF_STYLE_WEIGHT = 1.0

ID_MRF_CONTENT_WEIGHT = 1.0

NUM_GAUSSIAN_STEPS = 3

GAUSSIAN_KERNEL_SIZE = 32

GAUSSIAN_KERNEL_STD = 40.0

The input images are retrieved from the sample images folder. The image undergoes the full model and output is stored in “output” folder and compressed. Working and Explanation

For execution of the project the python files created are segregated under separate folders for easy understanding of the functioning of the project.

LAYERS

VGG16 is an improved object-recognition model that supports up to 16 layers. it's built as a Deep CNN It takes RGB image of resolution 224x224 as input. VGG has 3 fully-connected layers. the primary 2 layers have 4096 channels each while the third layer has 1000 channels, 1 for every class. To implement the layers, we use Keras python library. This segment of project is responsible from taking randomly-weighted average of two sensors, which geometrically speaking, outputs a random point on the line between each pair of input points.

The layers segment consists of reconstruction_loss function which are responsible for execution of following functions:

1.Wasserstein Loss: This is a loss function that depends on WGAN where the discriminator does not actually classify instances. For each instance it results into a number, which lies in range of (0,1). So, we cannot use 0.5 as threshold to classify image as real or fake. Discriminator training tries to make the output bigger for real instances than for fake ones.

We need Wasserstein Loss Because: -

- It provides better performance metric than misclassification rate & MSE
- Misclassification rate fails to state how wrong / correct predictions are
- Eliminates vanishing gradient in Neural Network architecture. Learning is not stalled

The loss is determined as:

Critic Loss:

$\text{Dis}(x) - \text{Dis}(\text{Gen}(z))$

The objective of Discriminator is to maximize the critic loss for higher accuracy.

Generator Loss:

$\text{Dis}(\text{Gen}(z))$

Where,

$\text{Dis}(x)$ – critic’s output for a real instance $\text{Gen}(z)$ – generator’s output when given noise z $\text{Dis}(\text{Gen}(z))$ – critic’s out for a fake instance

MODELS

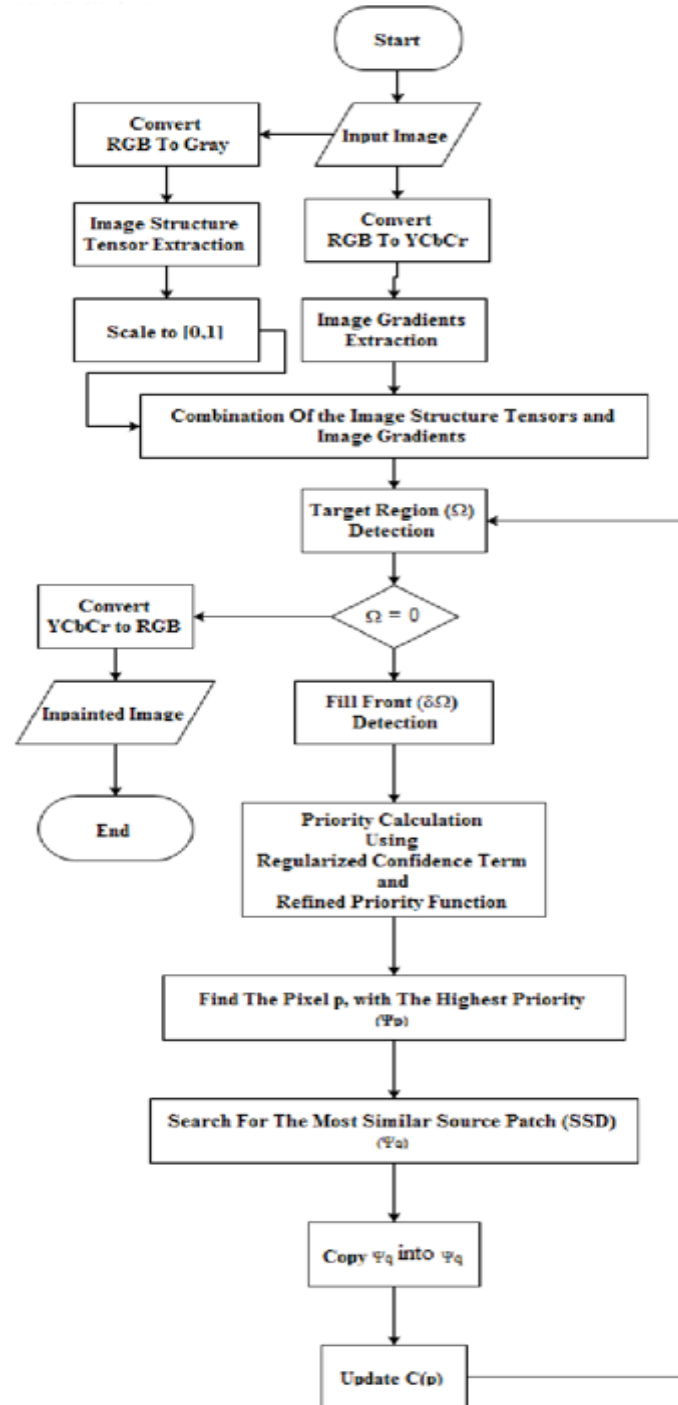
Already explained under the section “4. DESIGN APPROACH AND DETAILS”, and subsection “4.4. Codes and Standard”, the required and important sections of code i.e. “Models”, has been explained thoroughly with the code.

The architecture of the Generators and Discriminators (Local and Discriminators) has been implemented in form of code segments. The codes also show how in series the discriminators are working. The Convolutional 2D and Leaky ReLU explains the flow of image and its processing at different stages of filtering.

The Generator although uses three encoder branches and requires Convolutional 2D layering but ELU filtering for processing the input image. The figure 6.1. is perfect depiction of working of the Generator. The figure is a thorough and simpler version of Generator passing the input image through Encoder-branches and sending concatenated, decoded image to Discriminator. The image is again passed through VGG16 model (real model used) and predicts the “real” output image.

CHAPTER-8

ERD AND DFD



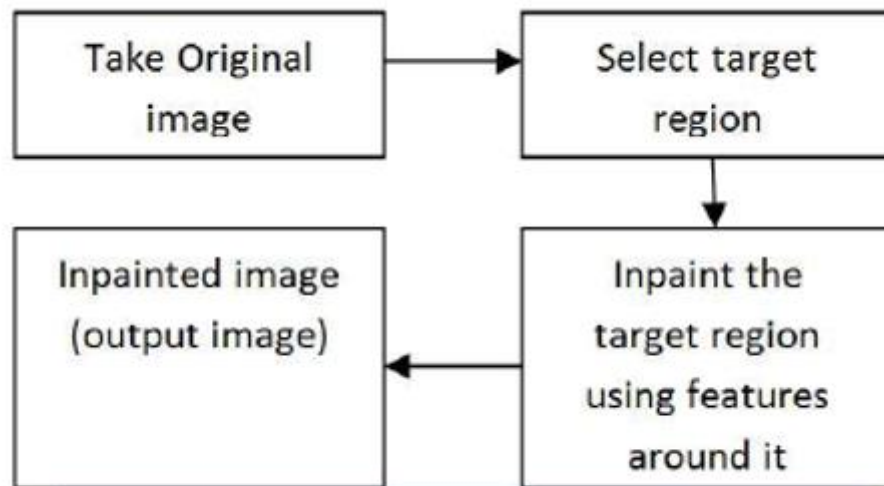


FIG. DFD

CHAPTER – 9

PROCESS INVOLVED, ALGORITHM, FLOWCHART, DATABASE DIAGRAM

INTRODUCTION & RELATED CONCEPTS

The new age alternative is to use deep learning to inpaint images by utilizing supervised image classification. the thought is that every image incorporates a specific label, and neural networks learn to acknowledge the mapping between images and their labels by repeatedly being taught or “trained”. When trained on huge datasets (size varying to range of thousands and millions of units of data), deep networks have remarkable classification performance which will often surpass human accuracy. GAN are typically used for this type of implementation, given their ability to “generate” new data, or during this case, the missing information.

The basic workflow is as follows: insert into the network an input image with irregularities (masked image) that require to be filled. These patches are considered an important parameter required by the network since the network has no way of perceiving the image and realizing what actually has to be filled in. For example, an image of someone with a missing face conveys no aspiring to the network except changing values for pixels. For the neural network, it is essential to understand what part of the image is actually damaged. We'd like a separate layer mask that contains pixel information for the missing data. The input image then passed through several convolutions and deconvolutions layers because it traverses across the network layers. The network does produce a wholly synthetic image generated from scratch. The layer mask allows us to eliminate those portions that are already presented within the incomplete image, since we don't have to fill those areas in. The new generated image is then superimposed on the unfinished one to yield the Inpainted image as an output.

AIM OF THE PROPOSED WORK

With time the pictures get damaged by getting patches because of wrinkles on them. Sometimes digital photos also get damaged because the files get corrupted. So, our aim is to repair the damaged images i.e. images with white patches and missing data (given below) using the GAN algorithm.

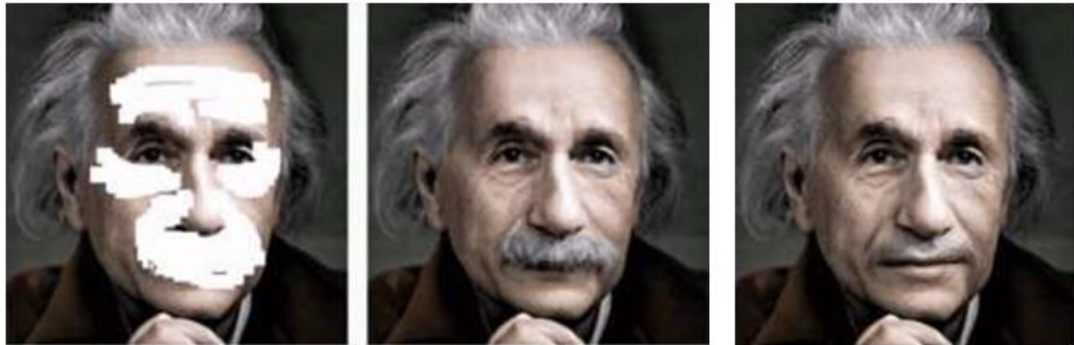


Figure 1: Damaged and Impainted Images

GENERATIVE ADVERSARIAL NETWORKS(GAN)

GANs are a strong class of neural networks that are used for unsupervised learning. It absolutely was developed and introduced by Ian J. Goodfellow in 2014. GANs are basically made of a system of two competing neural network models which compete with one another and are able to analyse, capture and replica the variations within a dataset. GANs is diminished into three parts:

- Generative: to find out a generative model, which describes how data is generated in terms of a probabilistic model.
- Adversarial: The training of a model is completed in an adversarial setting.
- Networks: Use deep neural networks because the AI (AI) algorithms for training purpose.

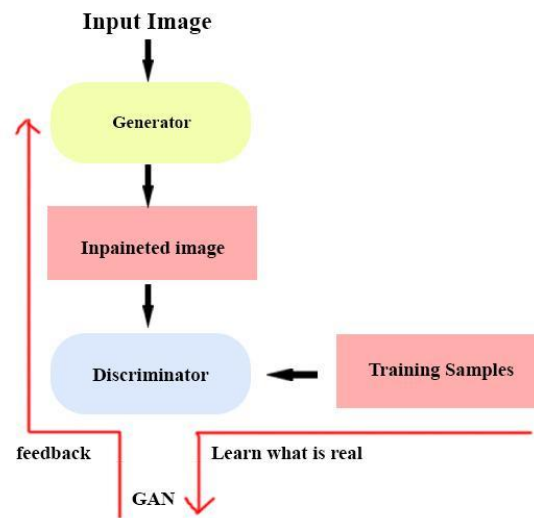


FIG. GAN ARCHITECTURE

WHAT DOES GAN DO?

The main focus for GAN is to get data from scratch, in our case, its images. GAN has also proved to be helpful in field of music. But the scope of application is much bigger than this. similar to the instance below, it generates a zebra from a horse. In reinforcement learning, it helps a robot to find out much faster.

GENERATOR AND DISCRIMINATOR

GAN composes of two deep networks, the generator, and therefore the discriminator. The generator learns to make fake data by simultaneously receiving feedback from the discriminator. It learns to create the discriminator classify its output as real. Generator training requires more strict integration between the generator and therefore the discriminator than discriminator training requires. The portion of the GAN that trains the generator includes:

- random input
- generator network, which transforms the random input into a knowledge instance

- discriminator network, which classifies the generated data
- discriminator output
- generator loss, which penalizes the generator for failing to fool the discriminator. The discriminator during a GAN is solely a classifier. It tries to differentiate real data from the information created by the generator. It could use any spec appropriate to the kind of knowledge it's classifying.

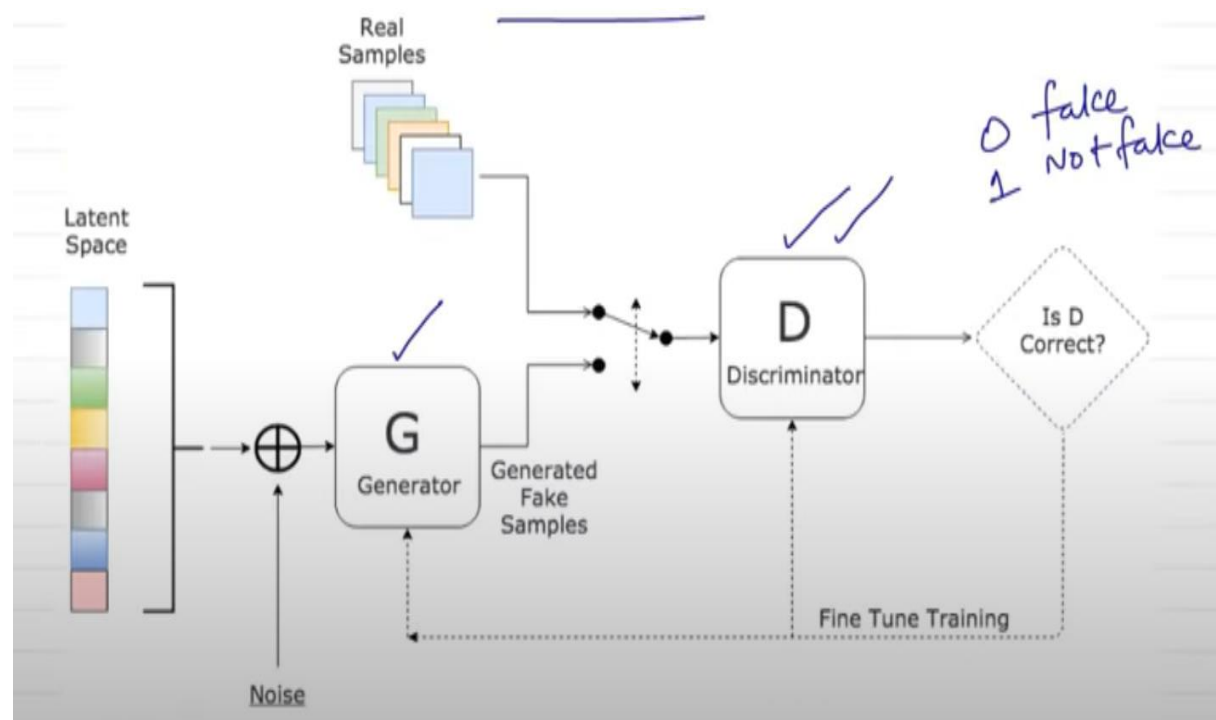


Fig. Representation of GAN based architecture

CHAPTER – 10

INPUT AND OUTPUT SCREEN DESIGN

SAMPLE IMAGE 1 AND RESULTS



FIG. 9.1

FIG.9.2

FIG.9.3

SAMPLE IMAGE 2 AND RESULTS



FIG. 9.4

FIG.9.5

FIG.9.6

Fig 9.1 and Fig 9.4 Represents an input image with white mask on it.

Fig 9.2 and Fig 9.5 Represents the blurred results of image inpainting

Fig 9.3 and Fig 9.6 The original image (sample)

elu_40 (ELU)	(None, 256, 256, 64) 0	conv2d_53[0][0]	
elu_11 (ELU)	(None, 64, 64, 128) 0	conv2d_24[0][0]	
elu_25 (ELU)	(None, 128, 128, 64) 0	conv2d_38[0][0]	
conv2d_54 (Conv2D)	(None, 256, 256, 64) 36928	elu_40[0][0]	
up_sampling2d_1 (UpSampling2D)	(None, 256, 256, 128) 0	elu_11[0][0]	
up_sampling2d_3 (UpSampling2D)	(None, 256, 256, 64) 0	elu_25[0][0]	
elu_41 (ELU)	(None, 256, 256, 64) 0	conv2d_54[0][0]	
concatenate_2 (Concatenate)	(None, 256, 256, 256) 0	up_sampling2d_1[0][0] up_sampling2d_3[0][0] elu_41[0][0]	
conv2d_55 (Conv2D)	(None, 256, 256, 16) 36880	concatenate_2[0][0]	
elu_42 (ELU)	(None, 256, 256, 16) 0	conv2d_55[0][0]	
conv2d_56 (Conv2D)	(None, 256, 256, 3) 435	elu_42[0][0]	
clip_1 (Clip)	(None, 256, 256, 3) 0	conv2d_56[0][0]	
=====			
Total params: 12,814,563			
Trainable params: 12,814,563			
Non-trainable params: 0			

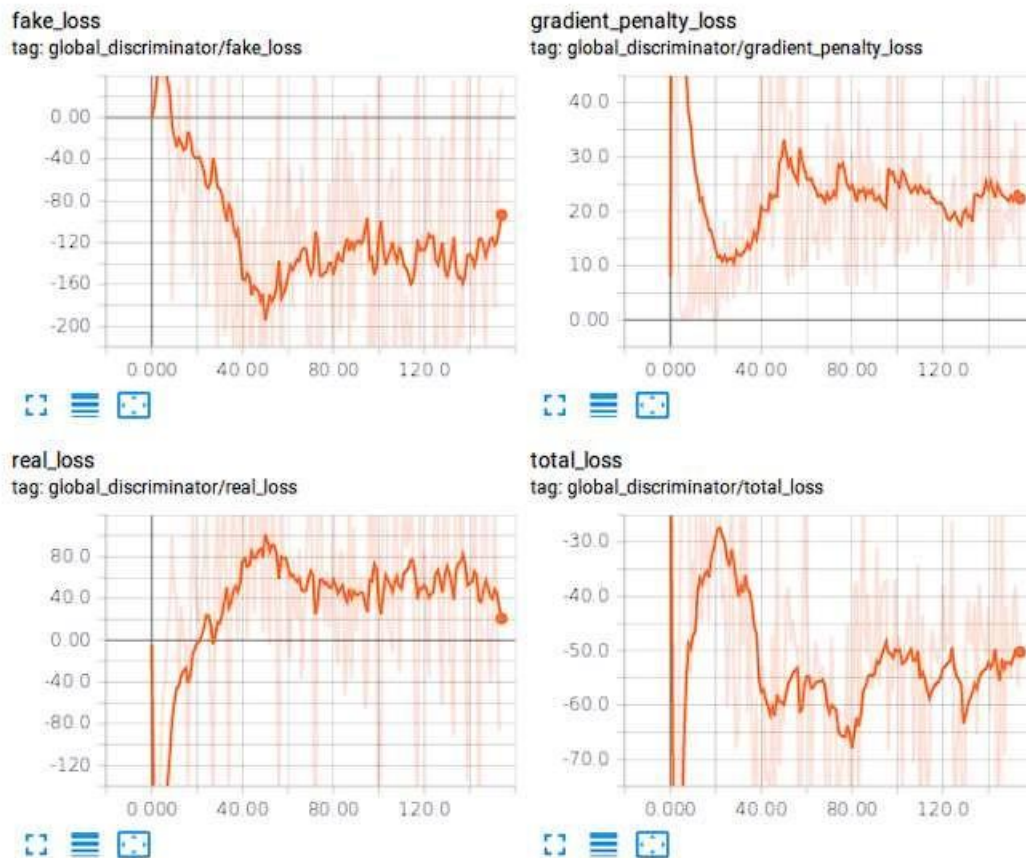
Fig 9.7 A clip of the output of Generator

The Graphical Representation of TensorFlow enables us to study the different losses generated by the generator. Below are the results of one of the tests:

1. confidence_reconstruction_loss
2. global_wasserstein_loss
3. id_mrf_loss
4. local_Wasserstein_loss

5. total_loss

global_discriminator



The graphs above show the losses being minimized for accurate results. The Loss vs steps of epochs during training has been plotted, which shows how machine is learning and trying to increase accuracy of the images.

Step_Count_Interval	Generator_loss	Global_Discriminator_loss	Local_discriminator
1	0.41	0.0	0.0
1001	0.16	0.0	0.0
2001	0.10	0.0	0.0
3001	0.10	0.0	0.0
4001	0.11	0.0	0.0
5001	0.07	0.0	0.0
6001	0.09	0.0	0.0
7001	0.09	0.0	0.0
8001	0.08	0.0	0.0
9001	0.07	0.0	0.0
9125	0.06	0.0	0.0

TABLE. WARM UP GENERATOR LOSS

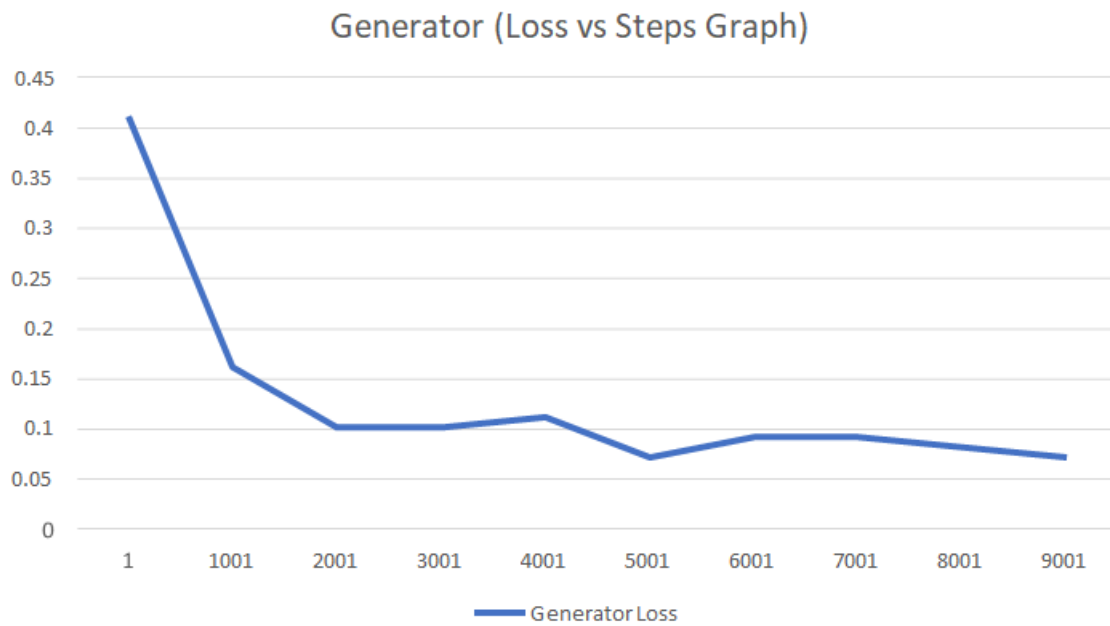


FIG. 9.8

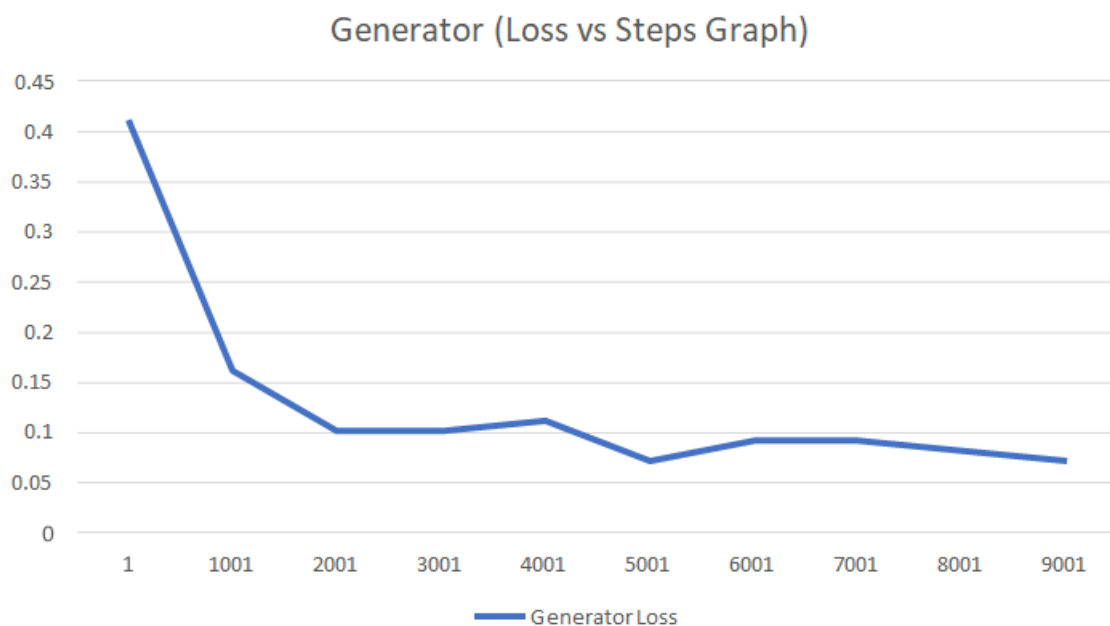


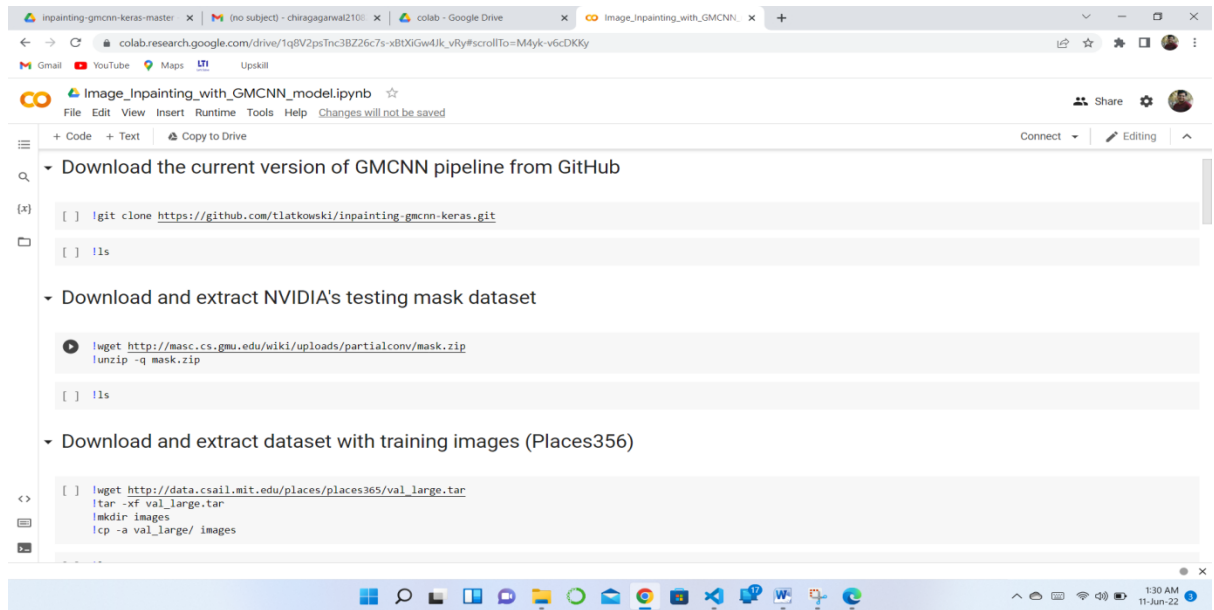
Fig. 9.9

Fig.9.8 This is a graphical representation w.r.t. Table.
Fig.9.9 Warmup Generator Loss

CHAPTER – 11

CODESHEETS

MAIN FILE



```
[ ] !git clone https://github.com/tlatkowski/inpainting_gmcnn-keras.git

[ ] !ls

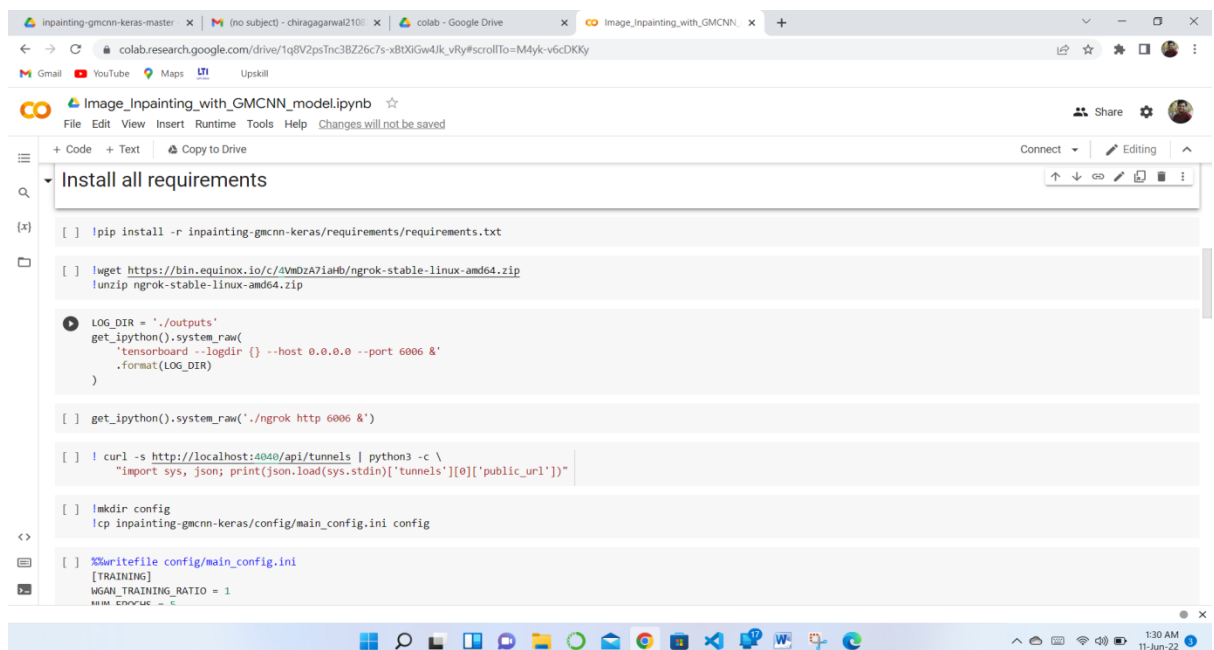
- Download and extract NVIDIA's testing mask dataset

[ ] !wget http://masc.cs.gmu.edu/wiki/uploads/partialconv/mask.zip
[ ] !unzip -q mask.zip

[ ] !ls

- Download and extract dataset with training images (Places365)

[ ] !wget http://data.csail.mit.edu/places/places365/val_large.tar
[ ] !tar -xvf val_large.tar
[ ] !mkdir images
[ ] !cp -a val_large/ images
```



```
[ ] !pip install -r inpainting_gmcnn-keras/requirements/requirements.txt

[ ] !wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
[ ] !unzip ngrok-stable-linux-amd64.zip

LOG_DIR = './outputs'
get_ipython().system_raw(
    'tensorboard --logdir {} --host 0.0.0.0 --port 6006 &'
    .format(LOG_DIR)
)

[ ] get_ipython().system_raw('ngrok http 6006 &')

[ ] ! curl -s http://localhost:4040/api/tunnels | python3 -c \
    "import sys, json; print(json.load(sys.stdin)['tunnels'][0]['public_url'])"

[ ] !mkdir config
[ ] !cp inpainting_gmcnn-keras/config/main_config.ini config

[ ] %writefile config/main_config.ini
[TRAINING]
WGAN_TRAINING_RATIO = 1
MIN_NOISE = 0.001
```

```
[ ] !mkdir config
!cp inpainting-gmcnn-keras/config/main_config.ini config

%writefile config/main_config.ini

[TRAINING]
WGAN_TRAINING_RATIO = 1
NUM_EPOCHS = 5
BATCH_SIZE = 4
IMG_HEIGHT = 256
IMG_WIDTH = 256
NUM_CHANNELS = 3
LEARNING_RATE = 0.0001
SAVE_MODEL_STEPS_PERIOD = 1000

[MODEL]
ADD_MASK_AS_GENERATOR_INPUT = True
GRADIENT_PENALTY_LOSS_WEIGHT = 10
ID_MRF_LOSS_WEIGHT = 0.05
ADVERSARIAL_LOSS_WEIGHT = 0.001
NHU_STRETCH_SIGMA = 0.5
VGG_16_LAYERS = 3,6,10
ID_MRF_STYLE_WEIGHT = 1.0
ID_MRF_CONTENT_WEIGHT = 1.0
NUM_GAUSSIAN_STEPS = 3
GAUSSIAN_KERNEL_SIZE = 32
GAUSSIAN_KERNEL_STD = 40.0

[ ] !ls
```

```
!ls outputs/gmcnn256x256/predicted_pics/wgan/

!Image('outputs/gmcnn256x256/predicted_pics/wgan/step_1000.png')

!Image('outputs/gmcnn256x256/predicted_pics/wgan/step_2000.png')

!zip -r outputs.zip outputs

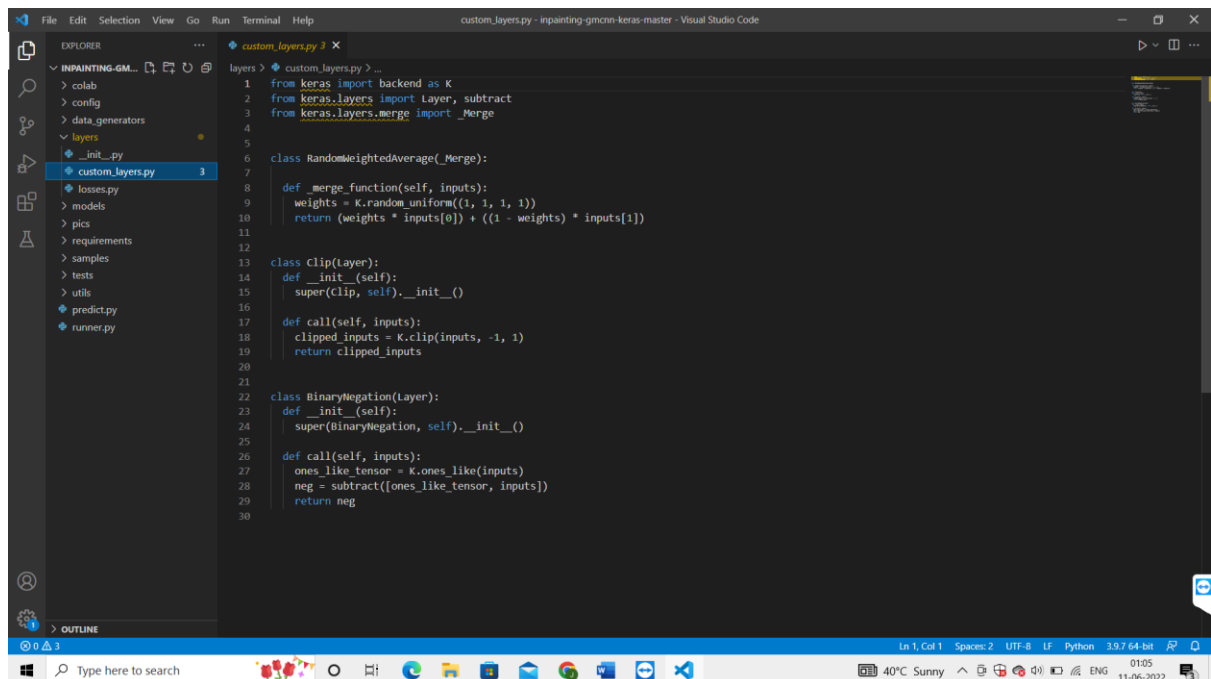
!ls

!rm -rf outputs/

!python inpainting-gmcnn-keras/runner.py --train_path images --mask_path mask --warm_up_generator --from_weights
```

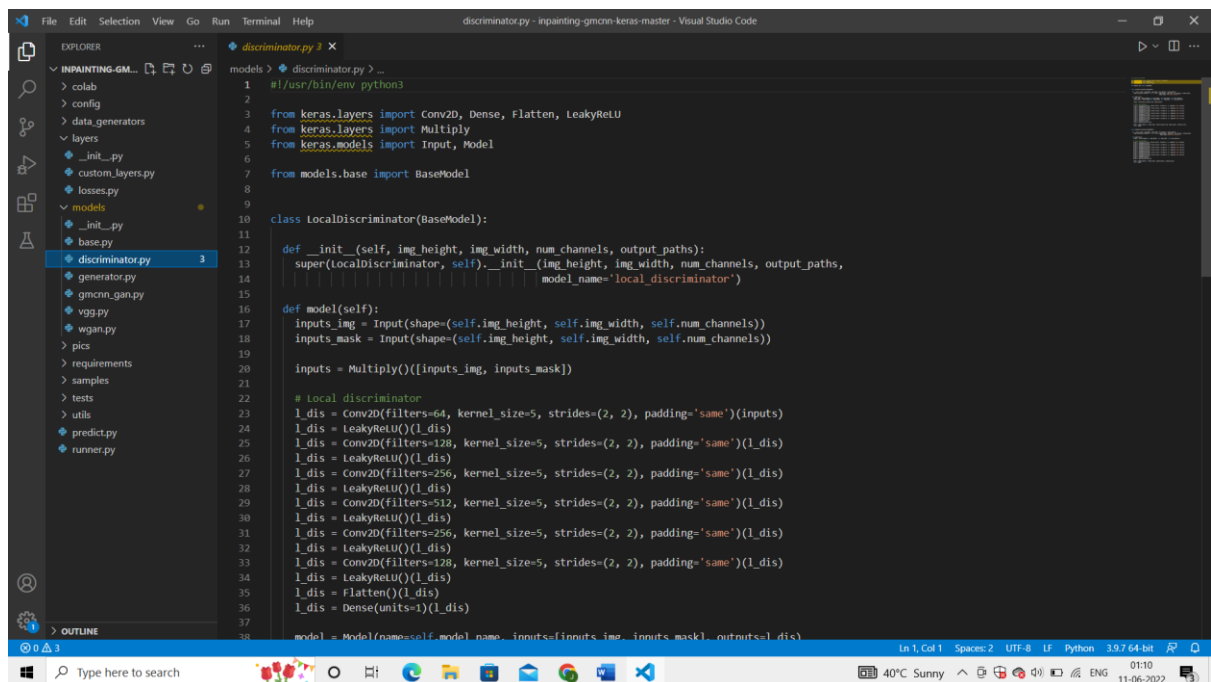
In order to download model result go to: Files -> content, right click on outputs.zip -> Download

CUSTOM_LAYERS.PY



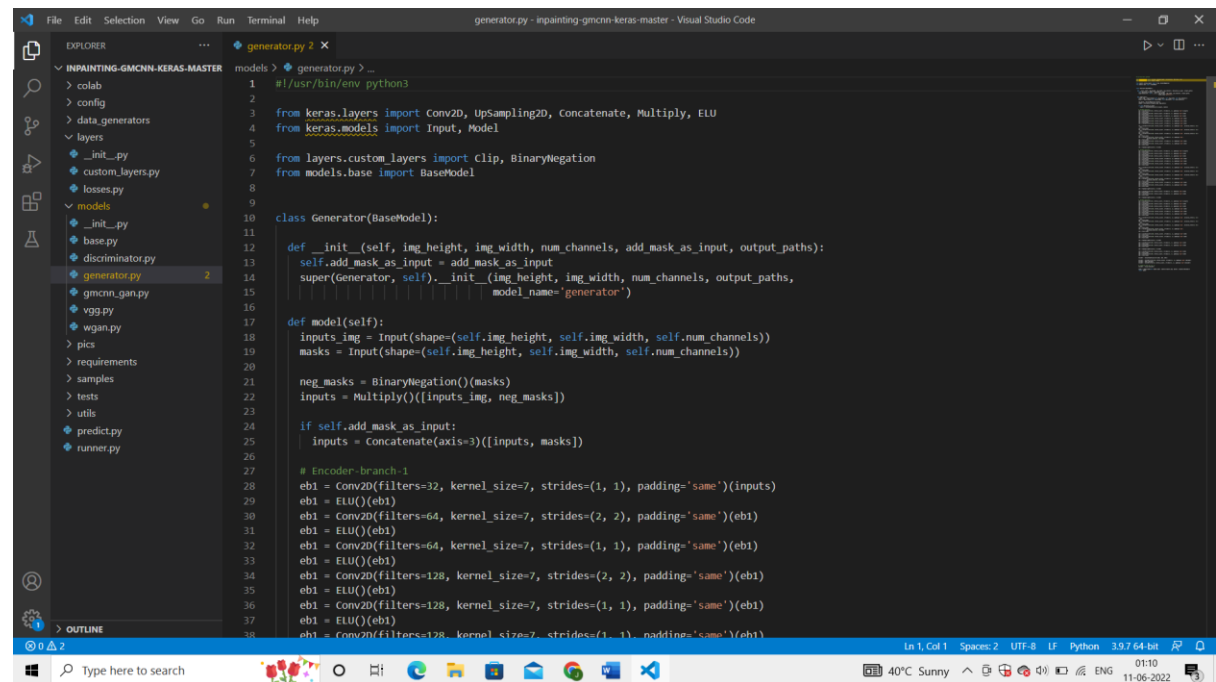
```
1 from keras import backend as K
2 from keras.layers import Layer, subtract
3 from keras.layers.merge import Merge
4
5
6 class RandomWeightedAverage(Merge):
7
8     def merge_function(self, inputs):
9         weights = K.random_uniform((1, 1, 1, 1))
10        return (weights * inputs[0]) + ((1 - weights) * inputs[1])
11
12
13 class Clip(Layer):
14     def __init__(self):
15         super(Clip, self).__init__()
16
17     def call(self, inputs):
18         clipped_inputs = K.clip(inputs, -1, 1)
19         return clipped_inputs
20
21
22 class BinaryNegation(Layer):
23     def __init__(self):
24         super(BinaryNegation, self).__init__()
25
26     def call(self, inputs):
27         ones_like_tensor = K.ones_like(inputs)
28         neg = subtract([ones_like_tensor, inputs])
29         return neg
30
```

DISCRIMINATOR.PY



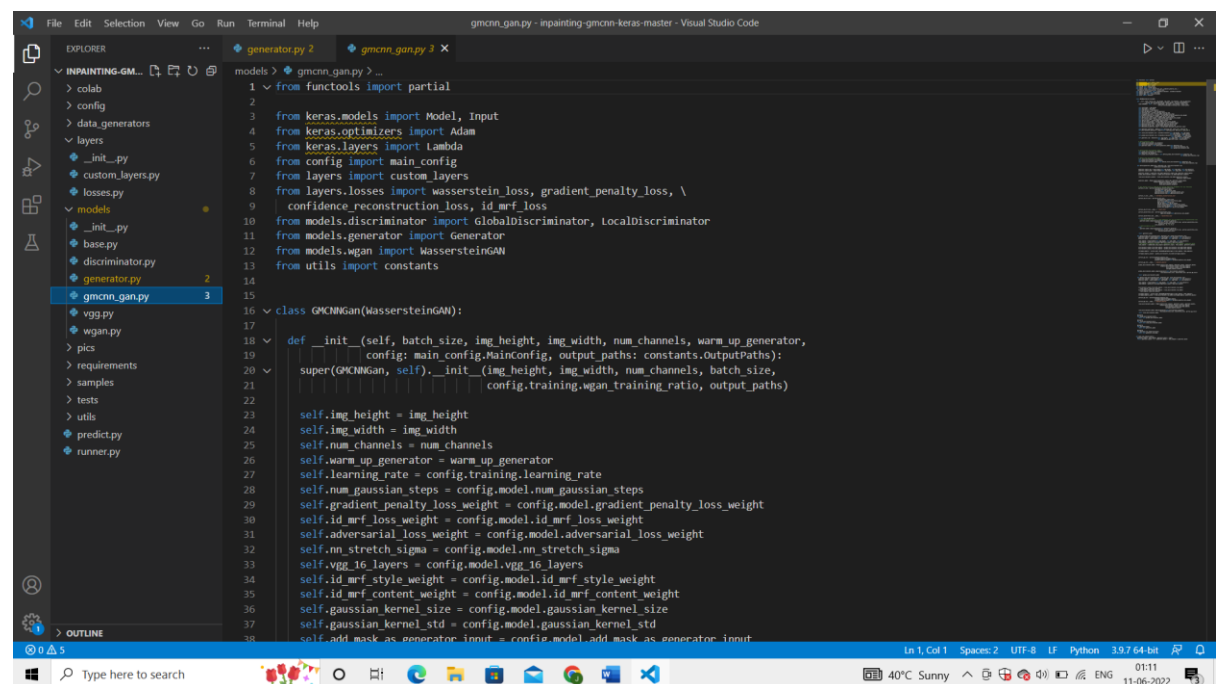
```
1 #!/usr/bin/env python3
2
3 from keras.layers import Conv2D, Dense, Flatten, LeakyReLU
4 from keras.layers import Multiply
5 from keras.models import Input, Model
6
7 from models.base import BaseModel
8
9
10 class LocalDiscriminator(BaseModel):
11
12     def __init__(self, img_height, img_width, num_channels, output_paths):
13         super(LocalDiscriminator, self).__init__(img_height, img_width, num_channels, output_paths,
14                                                  model_name='local_discriminator')
15
16     def model(self):
17         inputs_img = Input(shape=(self.img_height, self.img_width, self.num_channels))
18         inputs_mask = Input(shape=(self.img_height, self.img_width, self.num_channels))
19
20         inputs = Multiply()([inputs_img, inputs_mask])
21
22         # Local discriminator
23         l_dis = Conv2D(filters=64, kernel_size=5, strides=(2, 2), padding='same')(inputs)
24         l_dis = LeakyReLU()(l_dis)
25         l_dis = Conv2D(filters=128, kernel_size=5, strides=(2, 2), padding='same')(l_dis)
26         l_dis = LeakyReLU()(l_dis)
27         l_dis = Conv2D(filters=256, kernel_size=5, strides=(2, 2), padding='same')(l_dis)
28         l_dis = LeakyReLU()(l_dis)
29         l_dis = Conv2D(filters=512, kernel_size=5, strides=(2, 2), padding='same')(l_dis)
30         l_dis = LeakyReLU()(l_dis)
31         l_dis = Conv2D(filters=256, kernel_size=5, strides=(2, 2), padding='same')(l_dis)
32         l_dis = LeakyReLU()(l_dis)
33         l_dis = Conv2D(filters=128, kernel_size=5, strides=(2, 2), padding='same')(l_dis)
34         l_dis = LeakyReLU()(l_dis)
35         l_dis = Flatten()(l_dis)
36         l_dis = Dense(units=1)(l_dis)
37
38         model = Model(name=self.model_name, inputs=[inputs_img, inputs_mask], outputs=l_dis)
```

GENERATOR.PY



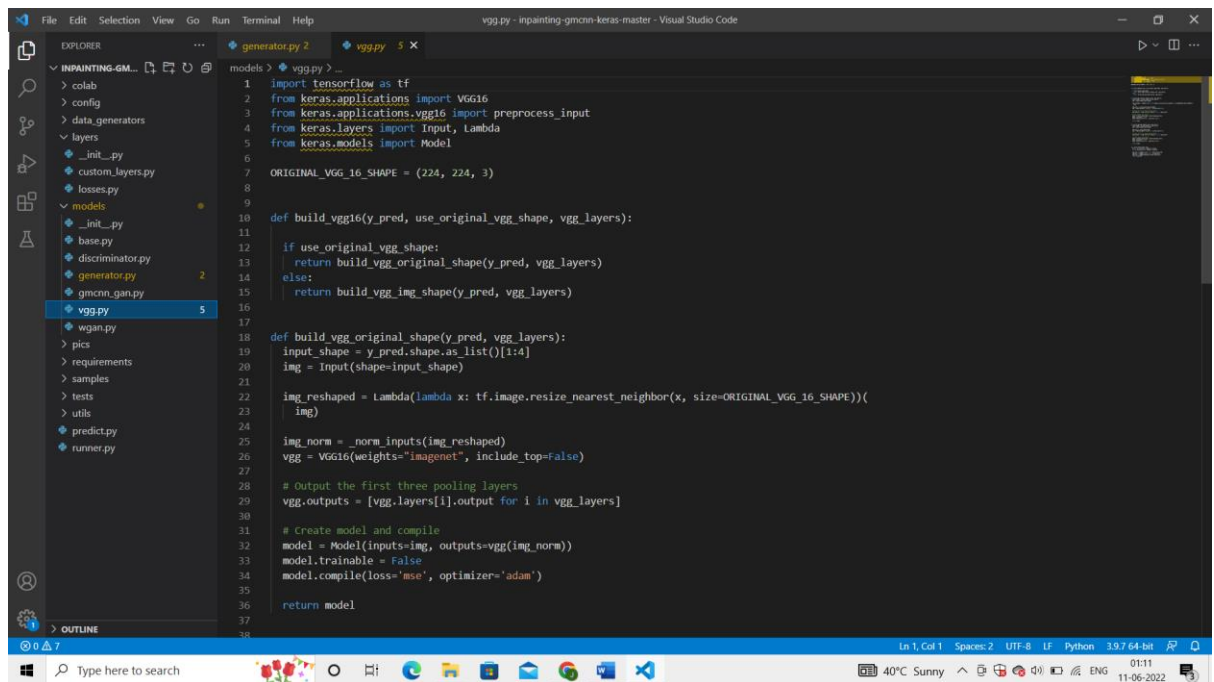
```
1 #!/usr/bin/env python3
2
3 from keras.layers import Conv2D, UpSampling2D, Concatenate, Multiply, ELU
4 from keras.models import Input, Model
5
6 from layers.custom_layers import Clip, BinaryNegation
7 from models.base import BaseModel
8
9
10 class Generator(BaseModel):
11
12     def __init__(self, img_height, img_width, num_channels, add_mask_as_input, output_paths):
13         self.add_mask_as_input = add_mask_as_input
14         super(Generator, self).__init__(img_height, img_width, num_channels, output_paths,
15                                         model_name='generator')
16
17     def model(self):
18         inputs_img = Input(shape=(self.img_height, self.img_width, self.num_channels))
19         masks = Input(shape=(self.img_height, self.img_width, self.num_channels))
20
21         neg_masks = BinaryNegation()(masks)
22         inputs = Multiply()([inputs_img, neg_masks])
23
24         if self.add_mask_as_input:
25             inputs = Concatenate(axis=3)([inputs, masks])
26
27         # Encoder-branch-1
28         eb1 = Conv2D(filters=32, kernel_size=7, strides=(1, 1), padding='same')(inputs)
29         eb1 = ELU()(eb1)
30         eb1 = Conv2D(filters=64, kernel_size=7, strides=(2, 2), padding='same')(eb1)
31         eb1 = ELU()(eb1)
32         eb1 = Conv2D(filters=64, kernel_size=7, strides=(1, 1), padding='same')(eb1)
33         eb1 = ELU()(eb1)
34         eb1 = Conv2D(filters=128, kernel_size=7, strides=(2, 2), padding='same')(eb1)
35         eb1 = ELU()(eb1)
36         eb1 = Conv2D(filters=128, kernel_size=7, strides=(1, 1), padding='same')(eb1)
37         eb1 = ELU()(eb1)
38         eb1 = Conv2D(filters=128, kernel_size=7, strides=(1, 1), padding='same')(eb1)
```

GMCNN.PY



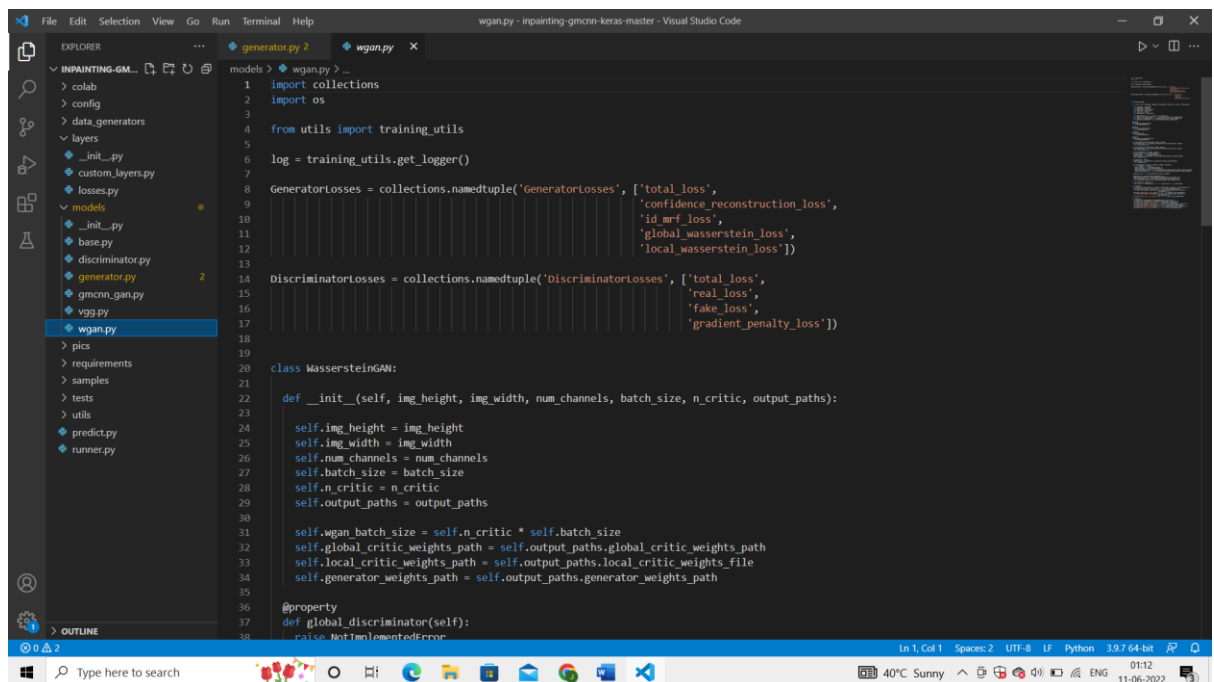
```
1 from functools import partial
2
3 from keras.models import Model, Input
4 from keras.optimizers import Adam
5 from keras.layers import Lambda
6 from config import main_config
7 from layers import custom_layers
8 from layers.losses import wasserstein_loss, gradient_penalty_loss, \
9     confidence_reconstruction_loss, id_mrf_loss
10 from models.discriminator import GlobalDiscriminator, LocalDiscriminator
11 from models.generator import Generator
12 from models.wgan import WassersteinGAN
13 from utils import constants
14
15
16 class GMCNNgan(WassersteinGAN):
17
18     def __init__(self, batch_size, img_height, img_width, num_channels, warm_up_generator,
19                 config: main_config.MainConfig, output_paths: constants.OutputPaths):
20         super(GMCNNgan, self).__init__(img_height, img_width, num_channels, batch_size,
21                                         config.training.wgan_training_ratio, output_paths)
22
23         self.img_height = img_height
24         self.img_width = img_width
25         self.num_channels = num_channels
26         self.warm_up_generator = warm_up_generator
27         self.learning_rate = config.training.learning_rate
28         self.num_gaussian_steps = config.model.num_gaussian_steps
29         self.gradient_penalty_loss_weight = config.model.gradient_penalty_loss_weight
30         self.id_mrf_loss_weight = config.model.id_mrf_loss_weight
31         self.adversarial_loss_weight = config.model.adversarial_loss_weight
32         self.nn_stretch_sigma = config.model.nn_stretch_sigma
33         self.vgg_16_layers = config.model.vgg_16_layers
34         self.id_mrf_style_weight = config.model.id_mrf_style_weight
35         self.id_mrf_content_weight = config.model.id_mrf_content_weight
36         self.gaussian_kernel_size = config.model.gaussian_kernel_size
37         self.gaussian_kernel_std = config.model.gaussian_kernel_std
38         self.add_mask_as_generator_input = config.model.add_mask_as_generator_input
```

VGG.PY



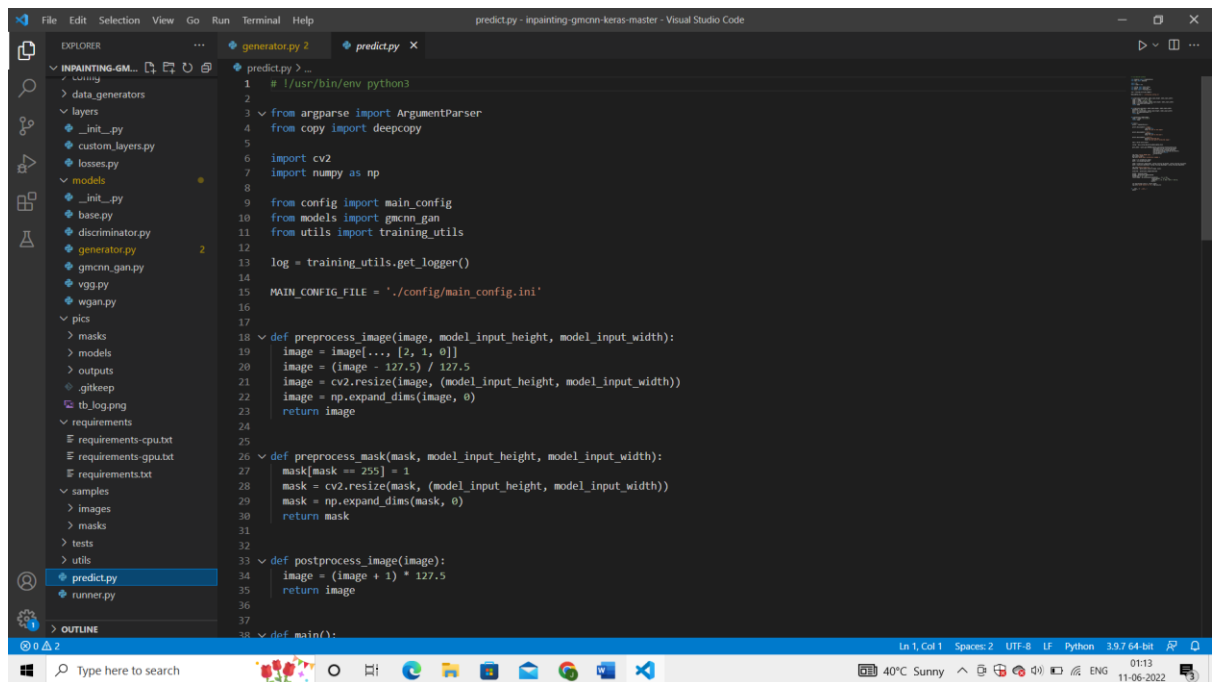
```
1 import tensorflow as tf
2 from keras.applications.vgg16 import VGG16
3 from keras.applications.vgg16 import preprocess_input
4 from keras.layers import Input, Lambda
5 from keras.models import Model
6
7 ORIGINAL_VGG_16_SHAPE = (224, 224, 3)
8
9
10 def build_vgg16(y_pred, use_original_vgg_shape, vgg_layers):
11
12     if use_original_vgg_shape:
13         return build_vgg_original_shape(y_pred, vgg_layers)
14     else:
15         return build_vgg_img_shape(y_pred, vgg_layers)
16
17
18 def build_vgg_original_shape(y_pred, vgg_layers):
19     input_shape = y_pred.shape.as_list()[1:4]
20     img = Input(shape=input_shape)
21
22     img_resized = Lambda(lambda x: tf.image.resize_nearest_neighbor(x, size=ORIGINAL_VGG_16_SHAPE))(
23         img)
24
25     img_norm = _norm_inputs(img_resized)
26     vgg = VGG16(weights='imagenet', include_top=False)
27
28     # Output the first three pooling layers
29     vgg.outputs = [vgg.layers[i].output for i in vgg_layers]
30
31     # Create model and compile
32     model = Model(inputs=img, outputs=vgg(img_norm))
33     model.trainable = False
34     model.compile(loss='mse', optimizer='adam')
35
36     return model
```

WGAN.PY



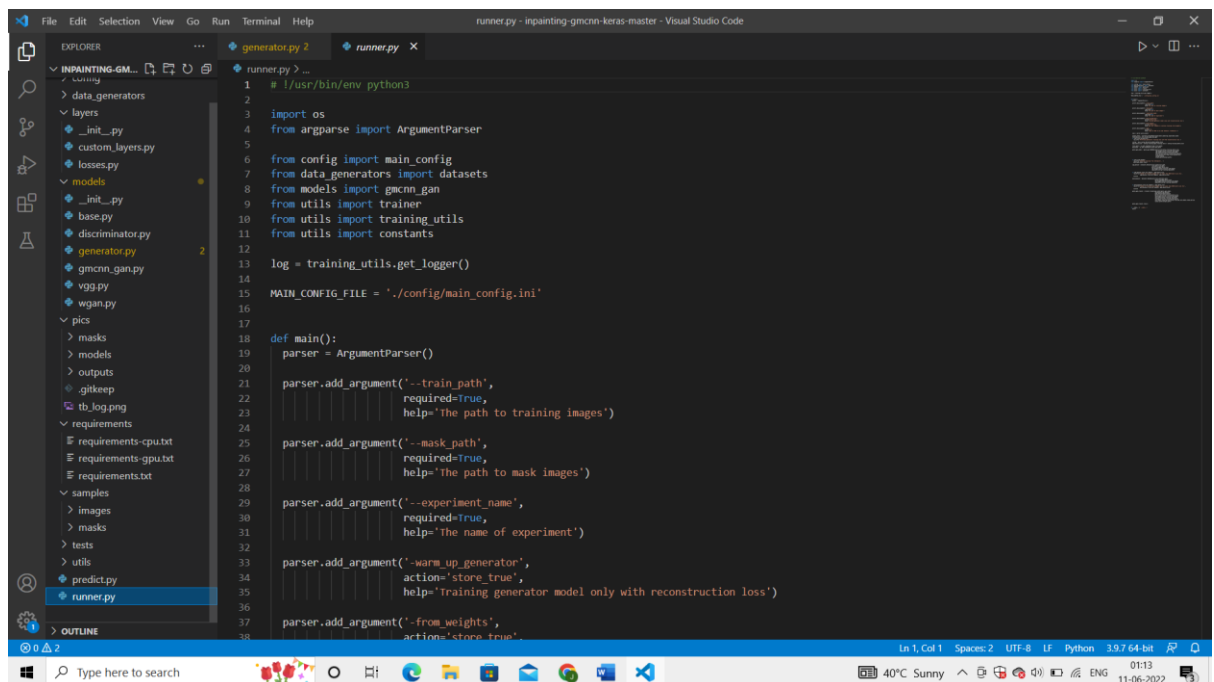
```
1 import collections
2 import os
3
4 from utils import training_utils
5
6 log = training_utils.get_logger()
7
8 GeneratorLosses = collections.namedtuple('GeneratorLosses', ['total_loss',
9                                                             'confidence_reconstruction_loss',
10                                                             'id_mrf_loss',
11                                                             'global_wasserstein_loss',
12                                                             'local_wasserstein_loss'])
13
14 DiscriminatorLosses = collections.namedtuple('DiscriminatorLosses', ['total_loss',
15                                                                     'real_loss',
16                                                                     'fake_loss',
17                                                                     'gradient_penalty_loss'])
18
19
20 class WassersteinGAN:
21
22     def __init__(self, img_height, img_width, num_channels, batch_size, n_critic, output_paths):
23
24         self.img_height = img_height
25         self.img_width = img_width
26         self.num_channels = num_channels
27         self.batch_size = batch_size
28         self.n_critic = n_critic
29         self.output_paths = output_paths
30
31         self.wgan_batch_size = self.n_critic * self.batch_size
32         self.global_critic_weights_path = self.output_paths.global_critic_weights_path
33         self.local_critic_weights_path = self.output_paths.local_critic_weights_file
34         self.generator_weights_path = self.output_paths.generator_weights_path
35
36     @property
37     def global_discriminator(self):
38         raise NotImplementedError
```

PREDICT.PY



```
1 #!/usr/bin/env python3
2
3 from argparse import ArgumentParser
4 from copy import deepcopy
5
6 import cv2
7 import numpy as np
8
9 from config import main_config
10 from models import gmcnn_gan
11 from utils import training_utils
12
13 log = training_utils.get_logger()
14
15 MAIN_CONFIG_FILE = './config/main_config.ini'
16
17
18 def preprocess_image(image, model_input_height, model_input_width):
19     image = image[...,[2,1,0]]
20     image = (image - 127.5) / 127.5
21     image = cv2.resize(image, (model_input_height, model_input_width))
22     image = np.expand_dims(image, 0)
23     return image
24
25
26 def preprocess_mask(mask, model_input_height, model_input_width):
27     mask[mask == 255] = 1
28     mask = cv2.resize(mask, (model_input_height, model_input_width))
29     mask = np.expand_dims(mask, 0)
30     return mask
31
32
33 def postprocess_image(image):
34     image = (image + 1) * 127.5
35     return image
36
37
38 def main():
39     parser = ArgumentParser()
40     parser.add_argument('--train_path',
41                         required=True,
42                         help='The path to training images')
43     parser.add_argument('--mask_path',
44                         required=True,
45                         help='The path to mask images')
46     parser.add_argument('--experiment_name',
47                         required=True,
48                         help='The name of experiment')
49     parser.add_argument('--warm_up_generator',
50                         action='store_true',
51                         help='Training generator model only with reconstruction loss')
52     parser.add_argument('--from_weights',
53                         action='store_true',
54                         help='Load weights from pre-trained model')
55
56     args = parser.parse_args()
57
58     train_path = args.train_path
59     mask_path = args.mask_path
60     experiment_name = args.experiment_name
61     warm_up_generator = args.warm_up_generator
62     from_weights = args.from_weights
63
64     log.info('Start training...')
65
66     # Load training data
67     train_data_loader = data_generators.TrainingDataLoader(train_path)
68     mask_data_loader = data_generators.MaskDataLoader(mask_path)
69
70     # Load model
71     model = gmcnn_gan.GMCNNGAN()
72
73     # Train model
74     trainer = training_utils.Trainer(model, train_data_loader, mask_data_loader)
75     trainer.train(warm_up_generator=warm_up_generator, from_weights=from_weights)
76
77     # Predict
78     predictor = training_utils.Predictor(model, mask_data_loader)
79     predictor.predict(experiment_name=experiment_name)
```

RUNNER.PY



```
1 #!/usr/bin/env python3
2
3 import os
4 from argparse import ArgumentParser
5
6 from config import main_config
7 from data_generators import datasets
8 from models import gmcnn_gan
9 from utils import trainer
10 from utils import training_utils
11 from utils import constants
12
13 log = training_utils.get_logger()
14
15 MAIN_CONFIG_FILE = './config/main_config.ini'
16
17
18 def main():
19     parser = ArgumentParser()
20     parser.add_argument('--train_path',
21                         required=True,
22                         help='The path to training images')
23     parser.add_argument('--mask_path',
24                         required=True,
25                         help='The path to mask images')
26     parser.add_argument('--experiment_name',
27                         required=True,
28                         help='The name of experiment')
29     parser.add_argument('--warm_up_generator',
30                         action='store_true',
31                         help='Training generator model only with reconstruction loss')
32     parser.add_argument('--from_weights',
33                         action='store_true',
34                         help='Load weights from pre-trained model')
35
36     args = parser.parse_args()
37
38     train_path = args.train_path
39     mask_path = args.mask_path
40     experiment_name = args.experiment_name
41     warm_up_generator = args.warm_up_generator
42     from_weights = args.from_weights
43
44     log.info('Start training...')
45
46     # Load training data
47     train_data_loader = data_generators.TrainingDataLoader(train_path)
48     mask_data_loader = data_generators.MaskDataLoader(mask_path)
49
50     # Load model
51     model = gmcnn_gan.GMCNNGAN()
52
53     # Train model
54     trainer = training_utils.Trainer(model, train_data_loader, mask_data_loader)
55     trainer.train(warm_up_generator=warm_up_generator, from_weights=from_weights)
56
57     # Predict
58     predictor = training_utils.Predictor(model, mask_data_loader)
59     predictor.predict(experiment_name=experiment_name)
```

CHAPTER – 12

TESTING

```
INFO:tensorflow:saved generator weights to: ./outputs/gmnn256x256/weights/gmnn.h5
INFO:tensorflow:saved global critic weights to: ./outputs/gmnn256x256/weights/global_critic.h5
INFO:tensorflow:saved local critic weights to: ./outputs/gmnn256x256/weights/local_critic.h5
/usr/local/lib/python3.6/dist-packages/keras/engine/training.py:490: UserWarning: Discrepancy between trainable weights and collected trainable
'Discrepancy between trainable weights and collected trainable'
Epochs: 8% 0/1 [06:44<7, 71t/s, epoch=0, generator_loss=0.22, global_discriminator_loss=0.00, local_discriminator_loss=0.00, step=163/9125]
```

Fig. This training process determines loss of 0.22 as generator loss, global_discriminator_loss as 0 and local_discriminator_loss as 0.

The damaged images are made as input and processed



CHAPTER – 13

USER/ OPERATIONAL MANUAL

Environmental: Palaeontologists who try to study about lost fauna and flora and try to recover them by creating the DNA or plant proteins.

Social: Social uses refer to people repairing any images or tampered videos to upload on social media.

Ethical: Can be used by archaeologists who try to excavate and revive ancient history. Paintings of old times can be recovered and preserved to honour great artists of the time.

Health and Safety: Images in field of medical issues like scans and Military purpose to revive any kind of tampered data can be very helpful in security services.

Legality: In order to solve any type of criminal cases or national threats, the technique can be used to retrieve the data from tampered videos of security feed or hidden camera.

CHAPTER – 14

CONCLUSIONS

The project report discusses about the project Image Inpainting using GAN. The project discloses about detailed modules used for the implementation of project. The aim of the project is to heal the images with patches using (GAN) is justified by using mask datasets and image datasets of high-resolution. The ML and image processing techniques involve use of various python libraries and TensorFlow. The software TensorFlow supports the project training model with the help of its GPU and CPU processors. The image inpainting uses the Generative Adversarial Networks and models like Wasserstein GAN, VGG and further discusses the role of Generators and Discriminators in the project. The Generator and Discriminators are part of GMCNN. The generator is trained to recreated damaged or input image. The pictures are then passed to Discriminators (which are also trained), which further passes the recreated image by Generator as “real” or “fake”. Training of the generators and discriminators would not have been possible without the image datasets, and mask dataset, comprising of 36,000 and 12,000 images respectively. The project implemented on Google Colab has been able to heal the images but not at great accuracy due to the availability of limited resources. The GPU processors needed for excellent accuracy requires better GPU for carrying out heavy training. For better understanding, the various losses are monitored using TensorFlow and represented using graphs. The losses are calculated to determine accuracy. Lesser the loss, better is the accuracy.

CHAPTER – 15

FUTURE ENHANCEMENT

With the upgrowing of digital processing of images and film archiving, the need for assisted or unsupervised restoration required the development of a series of methods and techniques. Among them, image inpainting is maybe the most impressive and useful. Based on partial derivative equations or texture synthesis, many other hybrid techniques have been proposed recently. The need for an analytical comparison, beside the visual one, urged us to perform the studies shown in the present paper. Starting with an overview of the domain, an evaluation of the five methods was performed using a common benchmark and measuring the PSNR. Conclusions regarding the performance of the investigated algorithms have been presented, categorizing them in function of the restored image structure. Based on these experiments, we have proposed an adaptation of Oliveira's and Hadhoud's algorithms, which are performing well on images with natural defects.

REFERENCES

1. Yu, Jiahui, et al. "Generative image inpainting with contextual attention." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.
2. Wang, Haodi, et al. "New inpainting algorithm based on simplified context encoders and multi-scale adversarial network." *Procedia computer science* 147 (2019): 254-263.
3. Hu, W., Ye, Y., Zeng, F., & Meng, J. (2019). A new method of Thangka image inpainting quality assessment. *Journal of Visual Communication and Image Representation*, 59, 292-299.
4. Jiao, L., Wu, H., Wang, H., & Bie, R. (2019). Multi-scale semantic image inpainting with residual learning and GAN. *Neurocomputing*, 331, 199-212.
5. Tran, A., & Tran, H. (2019). Data-driven high-fidelity 2D microstructure reconstruction via non-local patch-based image inpainting. *Acta Materialia*, 178, 207-218.
6. Cheng, J., & Li, Z. (2019). Markov random field-based image inpainting with direction structure distribution analysis for maintaining structure coherence. *Signal Processing*, 154, 182-197.
7. Wang, H., He, Z., He, Y., Chen, D., & Huang, Y. (2019). Average-face-based virtual inpainting for severely damaged statues of Dazu Rock Carvings. *Journal of Cultural Heritage*, 36, 40-50.
8. Wu, F., Kong, Y., Dong, W., & Wu, Y. (2019). Gradient-aware blind face inpainting for deep face verification. *Neurocomputing*, 331, 301-311.
9. Zhu, X., Qian, Y., Zhao, X., Sun, B., & Sun, Y. (2018). A deep learning approach to patch-based image inpainting forensics. *Signal Processing: Image Communication*, 67, 90-99.