CAP5404

FINAL PROJECT

IMAGE OUTPAINTING

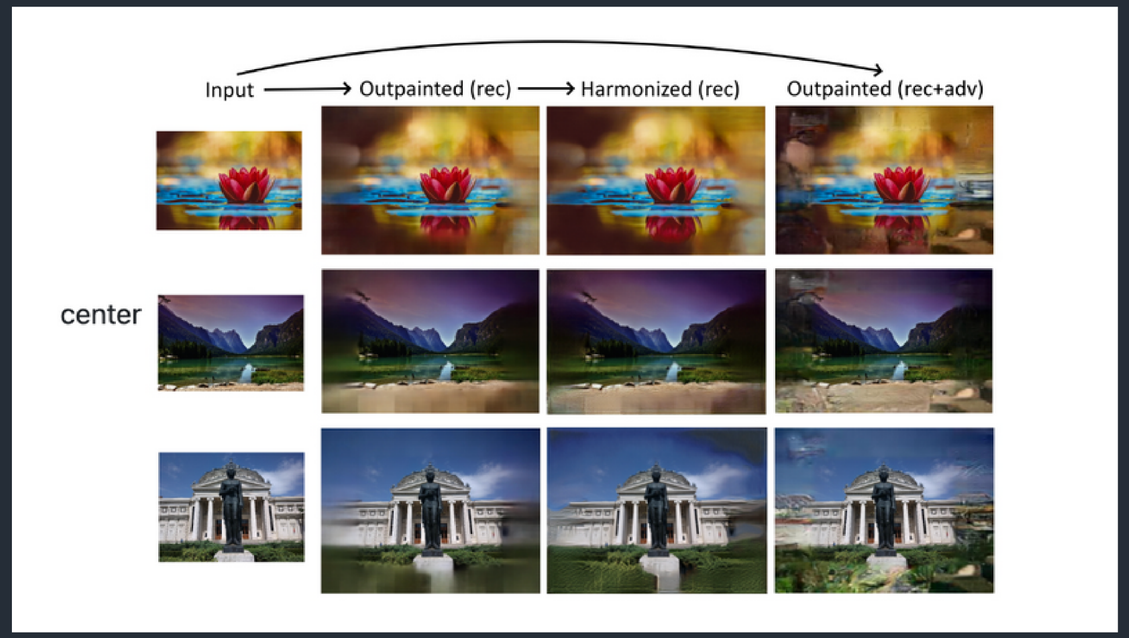
TABLE OF
CONTENT

INTRODUCTION

TO THE PROJECT

Image outpainting is extend the image outside of its boundary. This task is very easy for humans to imagine what lies beyond the image boundary. Image Inpainting has been a subject of research giving incredible results. But image outpainting is still a open problem with a huge scope of research.

In this project, we are using Deep learning to solve the problem of image outpainting.



CAP5404

Deep Learning for Computer Graphics

ABOUT OUR

PROJECT

For image outpainting using deep learning, we propose a solution using Generative Adversarial Networks. We are using MIT's Places365 dataset to train our models. We aim to develop a Image outpainting model that Inputs 128x128 image and outputs and outpainted 256x256 image. For another model we input a 192x192 image and output 256x256 image. All images are 3-channel RGB images.

PROPOSED
SOLUTIONS

DC GAN

- A simple discriminator GAN.
- Didn't give promising result

GANs with Local and Global Discrimnators

- GANs with Local discriminator for each edge
- Gave good results but too high computational requirements for training

GAN varying Loss

Use a GAN that keeps track of generative loss, and discriminator loss, and varies the weight that adversarial loss plays when training.

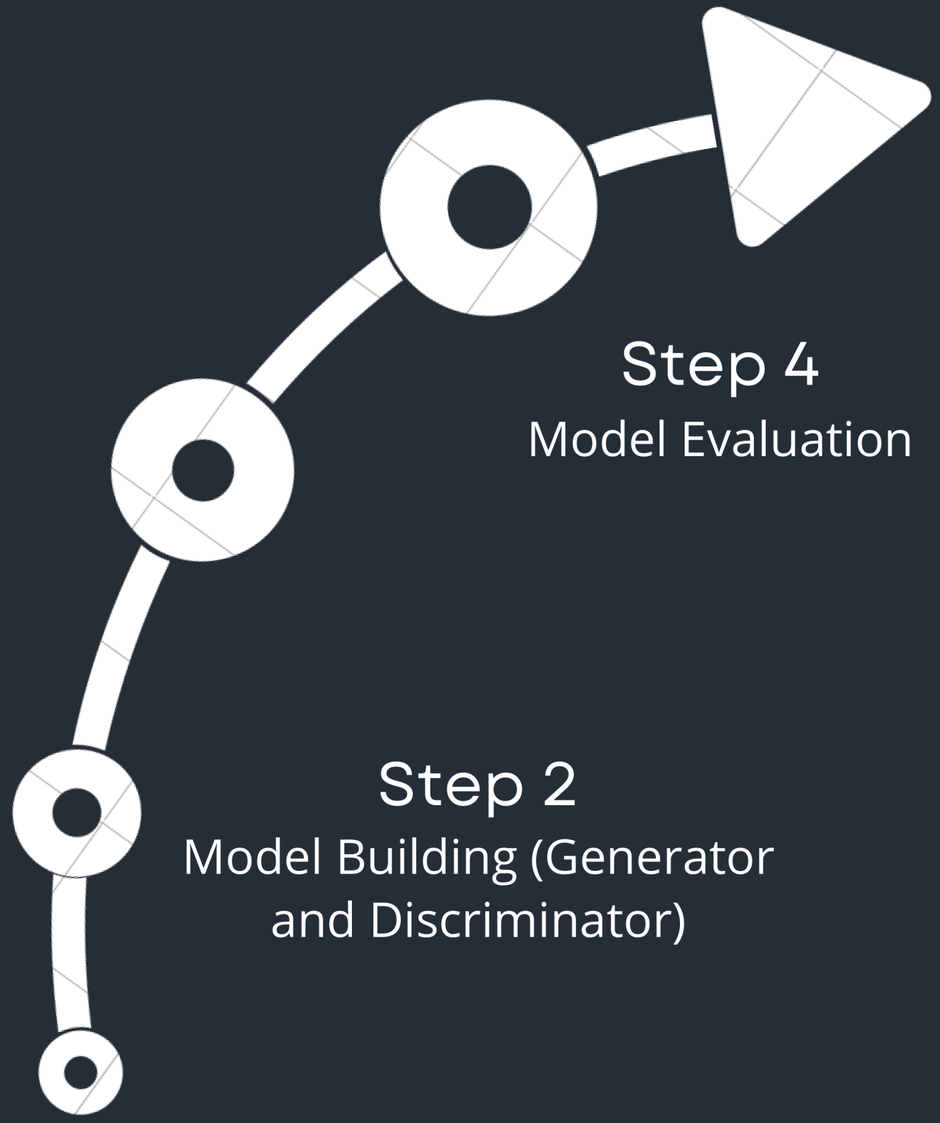
DATASET

We are using the Places365 dataset that contains about 2 Million images. But we are using the validation set of that dataset containing 36500 images due to the space constraints.

TECH STACK

1. TensorFlow for Deep Learning
2. Google Colab GPUs for faster training
3. Pillow and OpenCV to handle images.

GENERAL FLOW



Step 1
Image Preprocessing

Step 2
Model Building (Generator and Discriminator)

Step 3
Model Training

Step 4
Model Evaluation

IMPLEMENTATIONS

ABOUT MODEL

- Generator and One Discriminator
- Uses BatchNormalization
- Uses LeakyReLU as activation function
- Loss Function -
  - Generator - BinaryCross Entropy / MSE
  - Discriminator - BinaryCrossEntropy
- Optimizer: Adam Optimizer
- Batch Size - 32

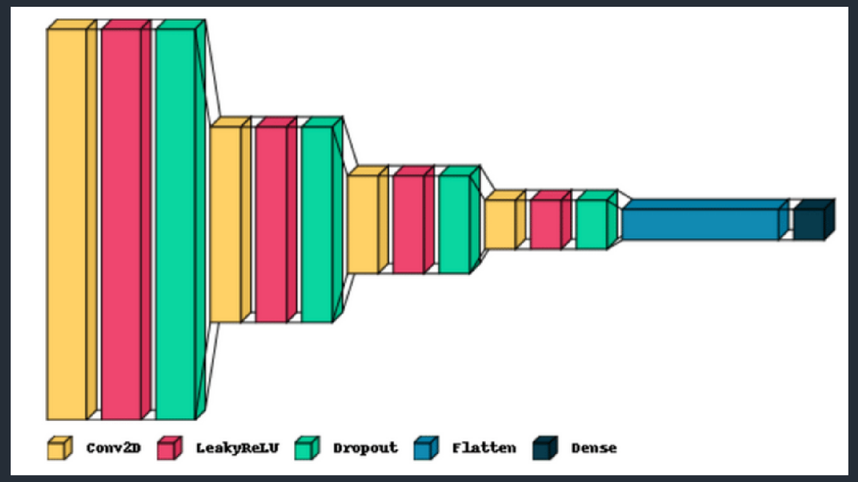
MODEL 1

DC-GAN 1

DC-GAN is an extension of regular GAN. DC-GAN uses Convolutional Layers in Generators and Transpose Convolutional Layers in Discriminators.

ARCHITECTURE

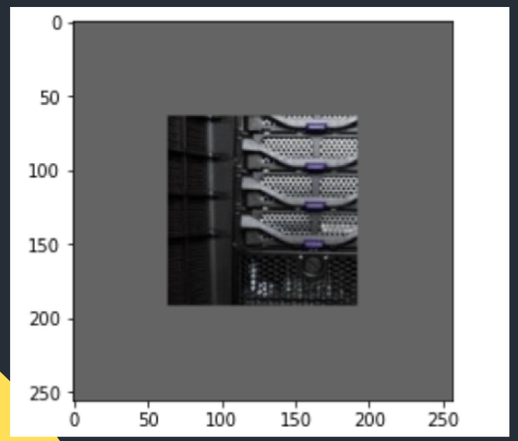
GENERATOR



DISCRIMINATOR

1. Generator model starts with 256 filter Conv2d layer then has 512 filter Conv2d layers and ending with a 256 filter conv2d layer.
2. Discriminator model start with 256 filter Conv2d layer and ends with Dense layer with single output

MODEL 1
TRAINING



PREPROCESSED IMAGE

STEP 1

- RESIZE FROM 256X256 TO 128X128
- ADDED PADDING TO CONVERT 128X128 TO 256X256 WITH A FIXED PIXEL VALUE 100
- NORMALIZED BETWEEN 0 TO 1

STEP 2

BUILDING THE MODEL ARCHITECTURE

STEP 3

- TRAINED FOR 500 EPOCHS
- 6 HRS TO TRAIN

STEP 4

- SAVED GENERATOR OUTPUT AFTER EACH EPOCH
- NOT GOOD OUTPUT
- MODEL DIDN'T LEARN ANYTHING WELL
- GAVE RANDOM PIXEL VALUE OUTPUT

ABOUT MODEL

- DEEP GENERATOR NETWORK
- 1 GLOBAL DISCRIMINATOR AND 4 LOCAL DISCRIMINATOR FOR EACH EDGE
- ADAM OPTIMIZER FOR EACH MODEL
- SIGMOID AND RELU ACTIVATION FUNCTIONS
- BATCH SIZE OF 4

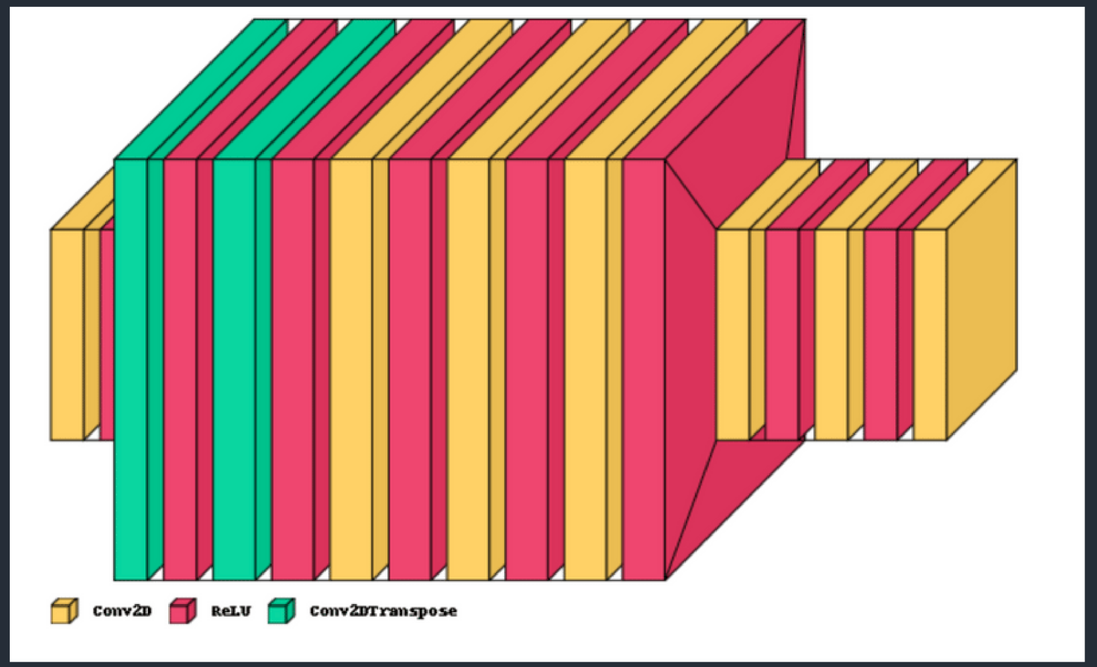
MODEL 3

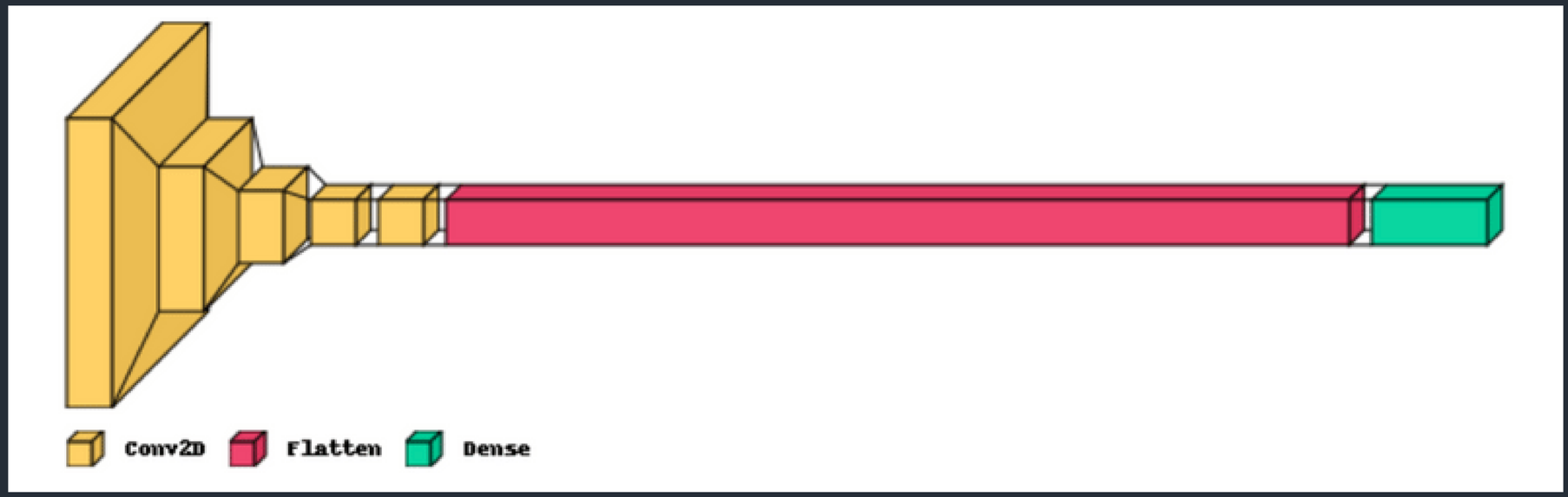
DC-GAN 2

DC-GAN is an extension of regular GAN. DC-GAN uses Convolutional Layers in Generators and Transpose Convolutional Layers in Discriminators.

ARCHITECTURE

GENERATOR





GLOBAL DISCRIMINATOR

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 64)	6464
re_lu (ReLU)	(None, 256, 256, 64)	0
conv2d_transpose (Conv2DTranspose)	(None, 512, 512, 128)	73856
re_lu_1 (ReLU)	(None, 512, 512, 128)	0
conv2d_transpose_1 (Conv2DTranspose)	(None, 512, 512, 256)	295168
re_lu_2 (ReLU)	(None, 512, 512, 256)	0
conv2d_1 (Conv2D)	(None, 512, 512, 256)	590080
re_lu_3 (ReLU)	(None, 512, 512, 256)	0
conv2d_2 (Conv2D)	(None, 512, 512, 256)	590080
re_lu_4 (ReLU)	(None, 512, 512, 256)	0
conv2d_3 (Conv2D)	(None, 512, 512, 256)	590080
re_lu_5 (ReLU)	(None, 512, 512, 256)	0
conv2d_4 (Conv2D)	(None, 256, 256, 128)	524416
re_lu_6 (ReLU)	(None, 256, 256, 128)	0
conv2d_5 (Conv2D)	(None, 256, 256, 64)	73792
re_lu_7 (ReLU)	(None, 256, 256, 64)	0
conv2d_6 (Conv2D)	(None, 256, 256, 3)	1731

GENERATOR

Layer (type)	Output Shape	Param #
conv2d_39 (Conv2D)	(None, 128, 32, 32)	2432
conv2d_40 (Conv2D)	(None, 64, 16, 64)	51264
conv2d_41 (Conv2D)	(None, 32, 8, 64)	102464
flatten_3 (Flatten)	(None, 16384)	0
dense_3 (Dense)	(None, 512)	8389120

Layer (type)	Output Shape	Param #
conv2d_36 (Conv2D)	(None, 32, 128, 32)	2432
conv2d_37 (Conv2D)	(None, 16, 64, 64)	51264
conv2d_38 (Conv2D)	(None, 8, 32, 64)	102464
flatten_2 (Flatten)	(None, 16384)	0
dense_2 (Dense)	(None, 512)	8389120

LOCAL DISCRIMINATOR

STEP 1
PRE-PROCESSING

- Convert 256x256 image to 192x192 and pad to make it 256x256 again. Pad with the mean pixel value.
- Normalize between 0 and 1
- Add a channel to the image i.e. 4th channel as the mask channel with pixel value 1 for the middle part and rest as 0.

STEP 2
MODEL BUILDING

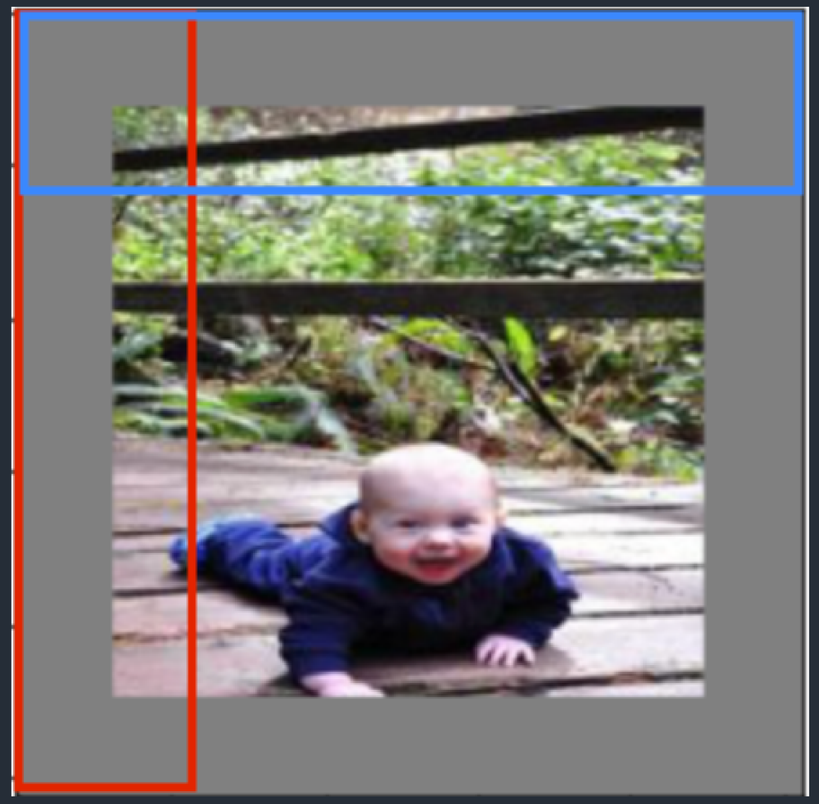
1. Build generator and global discriminator
2. Build 4 local discriminators for each side with corresponding input size
3. Details about local discriminators in next slide
4. Loss Function Generator: MSE
5. Loss Function Discriminators: Binary cross entropy over all discriminators.

STEP 3
TRAINING

- Train generator for 30000 iterations
- Train discriminators for 6000 iterations
- Train both generator and discriminators for 44000 more iterations.
- One Iterations is generating a single batch of images, calculating the loss and updating the weights.

STEP 4
MODEL EVALUATION

1. Save the losses of all iterations after every 1000. Generate test images after each 1000 iteration
2. Plot the losses vs iterations.



TRAINING THE
DISCRIMINATORS

- Train global discriminator on the whole image
- Train local discriminator on red and blue box
  - Red Box - vertical edge discriminator takes input 256x64
  - Blue box - horizontal egde discriminator takes input 64x256
- Concatenate the outputs of all these discriminators, calculate loss
- Adjust the weights.

ABOUT MODEL

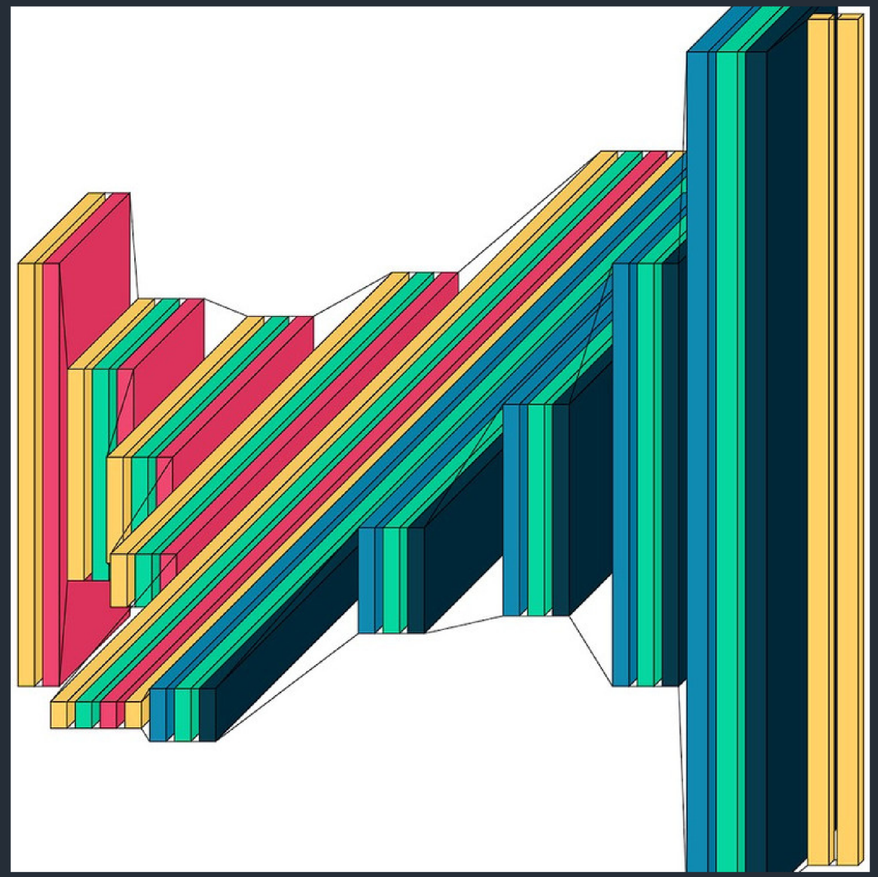
- DEEP GENERATOR NETWORK
- 1 GENERATOR 1 DISCRIMINATOR
- ADAM OPTIMIZER FOR EACH MODEL
- LOSS FUNCTION: GENERATOR WEIGHTED MAE, DISCRIMINATOR MSE
- Leaky Relu AND RELU ACTIVATION FUNCTIONS
- BATCH SIZE OF 4

MODEL 3

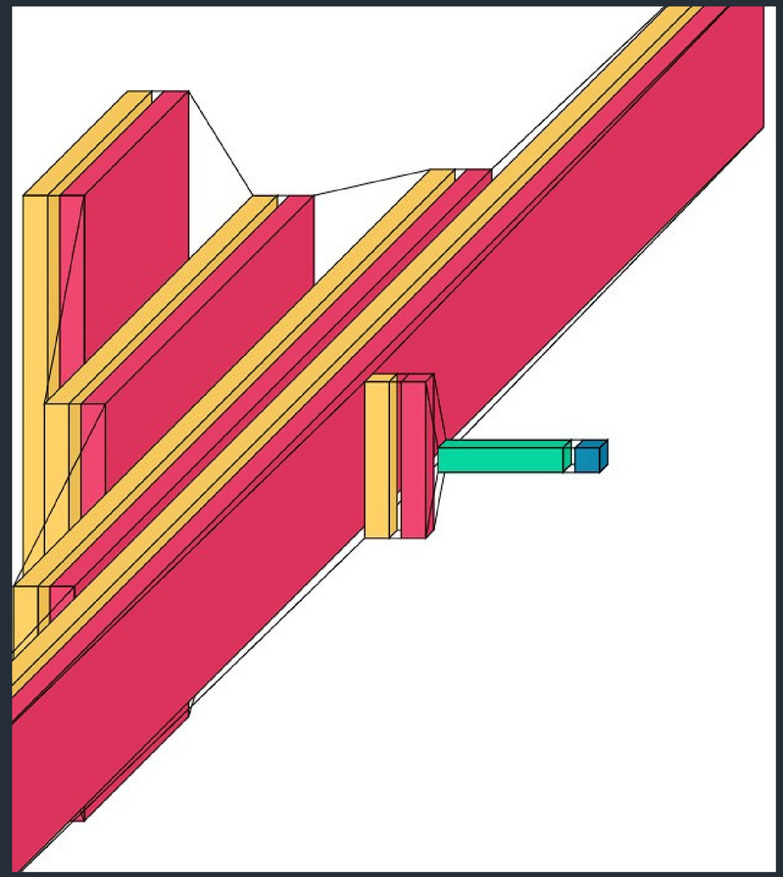
GAN 3

GAN THAT USES MIRRORED INPAINTING ARCHITECTURE FOR GENERATOR AND HAS A WEIGHTED GENERATOR LOSS THAT DEPENDS ON ADVERERIAL LOSS

ARCHITECTURE



GENERATOR



DISCRIMINATOR

STEP 1
PRE-PROCESSING

- Convert images from size 256x256 then mask out the middle portion from 64 - 192
- Resize the masked image to 256x256 with 0 padding along the outside.
- Normalize all RGB channels between 0 - 1

STEP 2
MODEL BUILDING

1. Build Generator and Discriminator
2. Generator consists of 6 down convolution layers a linear layer and 6 up convolution layers.
3. The down convolution layers consist of a conv2d that halves the image resolution, a batch normalization layer, and a leaky relu activation function.
4. The Up convolution layers consist of conv2d transpose that doubles the image resolution of each layer along with a relu activation function at each layer.
5. Discriminator consists of 5 down convolution layers followed by a dense layer. Each layer consists of a conv2d layer and a leaky relu activation function.

STEP 3
TRAINING

- Training is done together in 200 epochs with batch sizes of 4.
- After epochs 10 30 and 60, the weight of how adversarial loss affects backpropagation is changed to prevent generator loss from not decreasing.

STEP 4
MODEL EVALUATION

- Save 10 sample image of each epoch to show training progress over time.
- Track Genrator and discriminator loss
- Generator loss is MeanAbsoluteError between real and fake outputs that is added to an adversarial loss(multiplied by a varying weight)
- Discriminator loss is a sum of real loss(MSE between real outputs and 1s) and fake loss(MSE between fake outputs and 0s)

MODEL
RESULTS

1 DG GAN 1

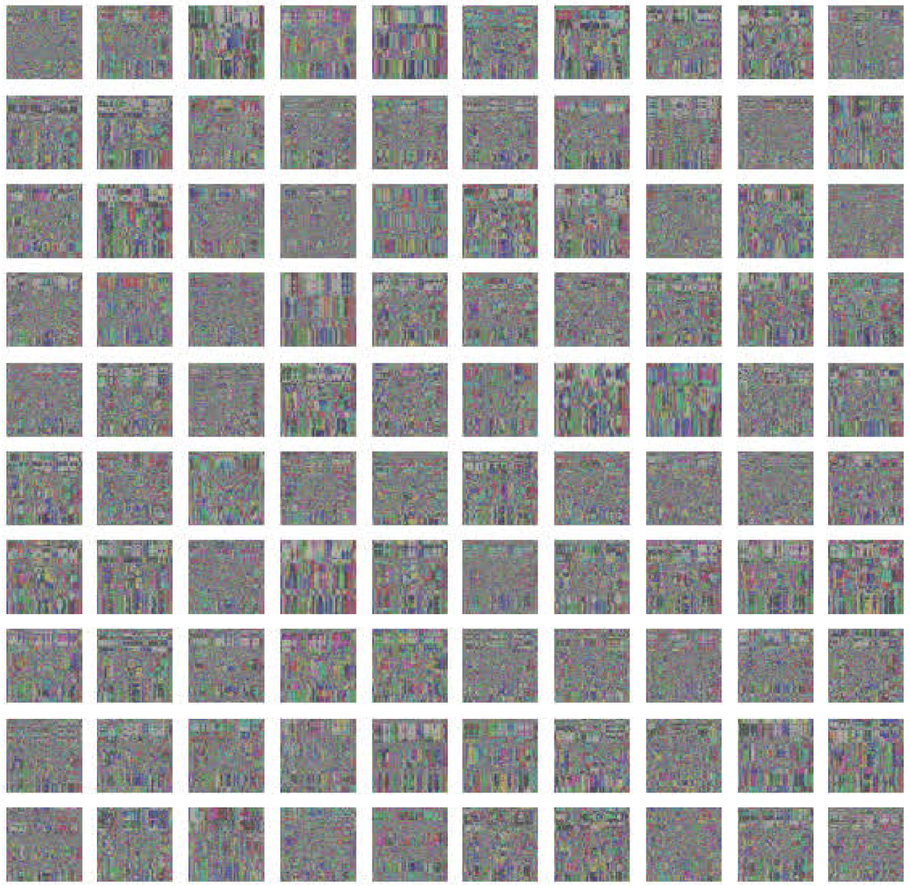
One Generator And One discriminator

2 GAN 2

One Generator 5 Discriminators

3 GAN 3

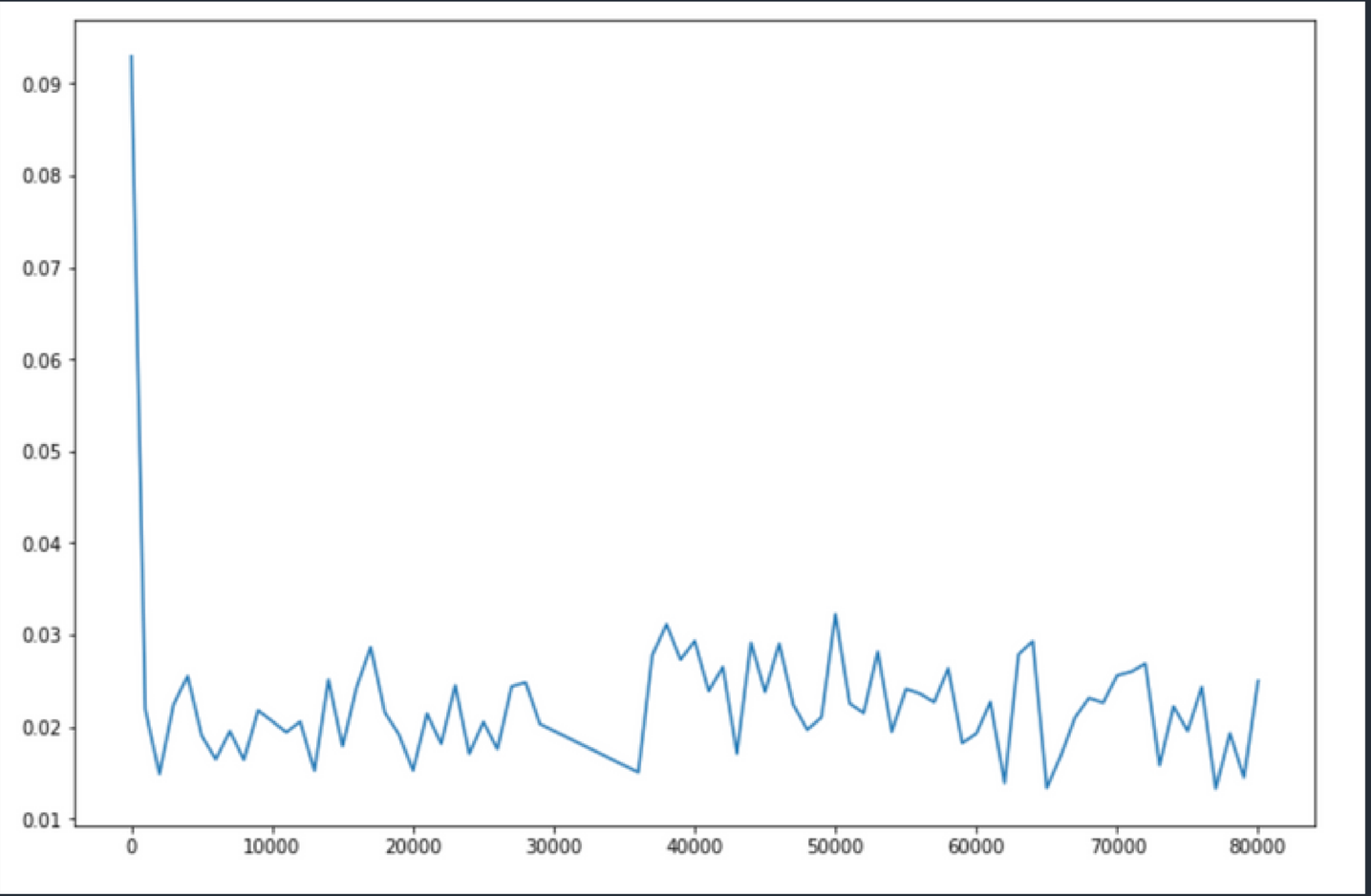
One generator and one discriminator but adversarial loss affect back propagation



TRAINING RESULT

DC GAN 1

- Stopped the training after 350 epochs as it showed no improvement at all
- Generalizing over the whole image so no beneficial learning



PLOTS

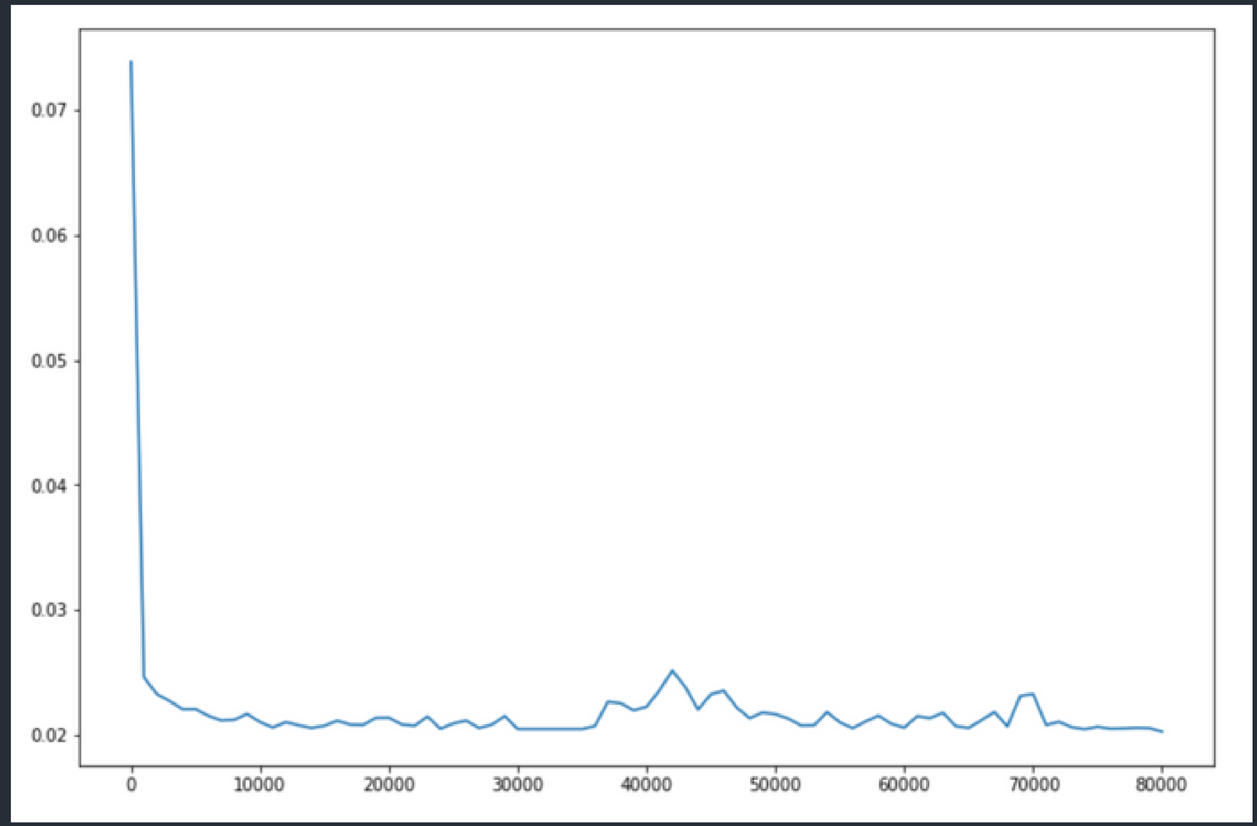
GAN 2

GENERATOR TRAINING LOSS PLOT. The loss decreases over iterations t

<0.02

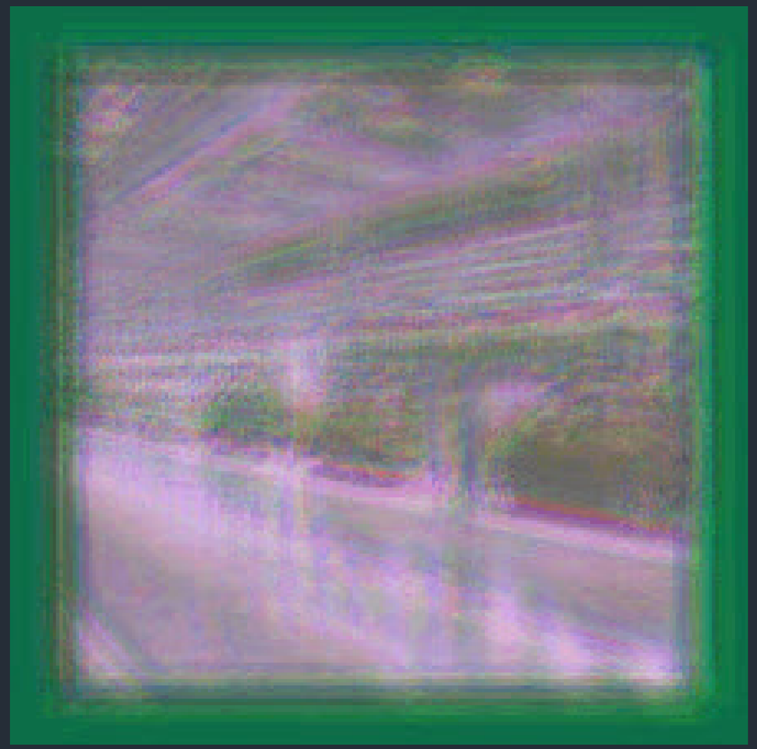
VAL LOSS

GAN 2



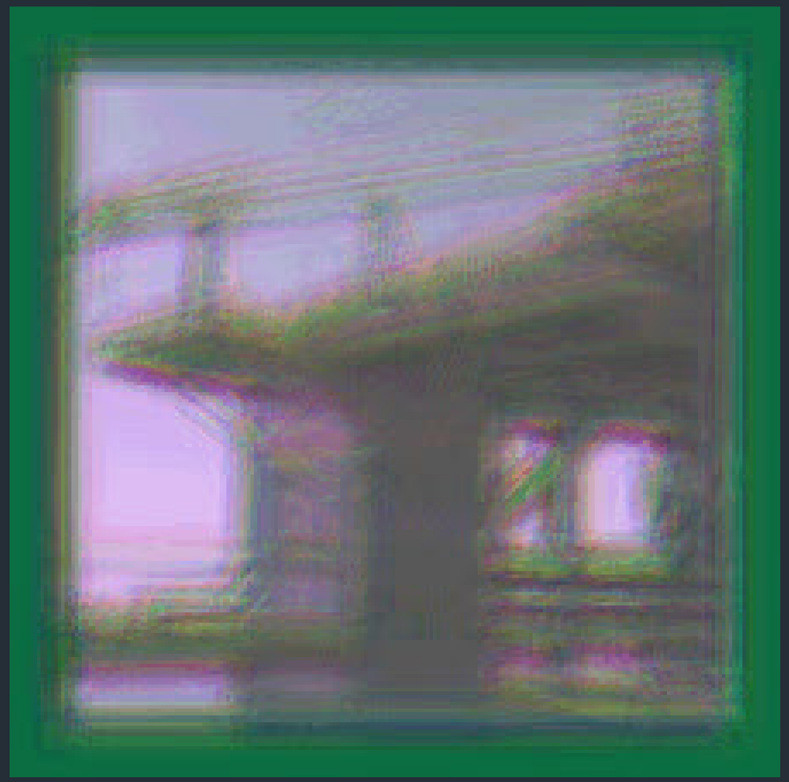
GAN 2
RESULTS

IMAGE 1



GAN 2
RESULTS

IMAGE 2

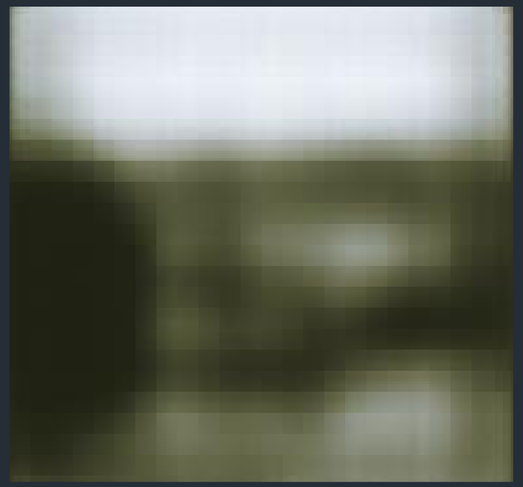


GAN 3
RESULTS

IMAGE 1

GAN 3
RESULTS

IMAGE 2



MEET

OUR TEAM

Chintan Acharya

UFID 64354143

Lixandross Gernier

UFID 53038781

CAP5404

THANK YOU