# Pix-2-Pix cGAN

CS583 Computer Vision

*Mahima Modi*
*College of Computing and Informatics*
*Drexel University*
*Philadelphia, PA, USA*
*mjm838@drexel.edu*

*Kartik Vora*
*College of Computing and Informatics*
*Drexel University*
*Philadelphia, PA, USA*
*kav73@drexel.edu*

## A. Abstract

Many image processing, computer graphics, and computer vision problems can be solved by "translating" an input image into a similar output image. Despite the fact that the goal is always the same: forecast pixels from pixels, each of these tasks has traditionally been done with its own collection of special-purpose gear. This research proposes a generalized loss function for all image to image translation challenges, such as reconstructing objects from edge maps and colorizing images, among other tasks, to create a common framework for all of these problems. Conditional GAN is the inspiration for this adversarial network. We utilize a convolutional "PatchGAN" classifier as our discriminator, which penalizes structure solely at the scale of image patches. We will be translating the input image (sketched/B&W) into a colored image.

## B. Introduction

Image-to-image translation is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image. In simple words, it would mean predicting pixels from pixels.

Image-Image translation has been done by many researchers. Pix2Pix architecture has been mentioned in the paper Image-to-Image Translation with Conditional Adversarial Networks researched at UC Berkley was our basis for this project. Their implementation uses a batch norm as normalization layer. We experimented with instance norm, and we are getting slightly better results and we present the results in this paper. Pix2Pix has many applications which include – converting grayscale image to colored, sketch to colored, satellite image to map, etc. Our focus will be on the image colorization domain. The code is available at https://github.com/mahima97/pix2pix-cGAN.git

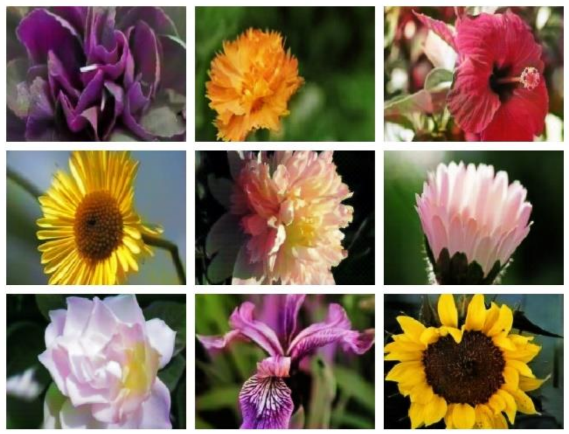


*Figure 1 Generated Flower Images*

## C. Related work

Image prediction with CNN has a drawback of predefining a loss function. Choosing a loss function has an issue that it may be inappropriate for the problem at hand. It is better if somehow the network was able to learn the loss function on its own. This is where GANs come into picture - they learn a loss that tries to classify if the output image is real or fake, while simultaneously training a generative model to minimize this loss. GANs learn a loss that adapts to the problem at hand.

In cGANs, a conditional setting is applied, meaning that both the generator and discriminator are conditioned on some sort of auxiliary information (such as class labels or data). cGAN for short, is a type of GAN that involves the conditional generation of images by a generator model. Pix2Pix is one variant of cGANs and it can serve as general-purpose solution all image-to-image translation problems. While the generator in traditional GANs produced realistic-looking images, we certainly had no control over the type or class of generated images. Hence, came CGAN allowing controlled generation of images.

## D. Contribution / Method

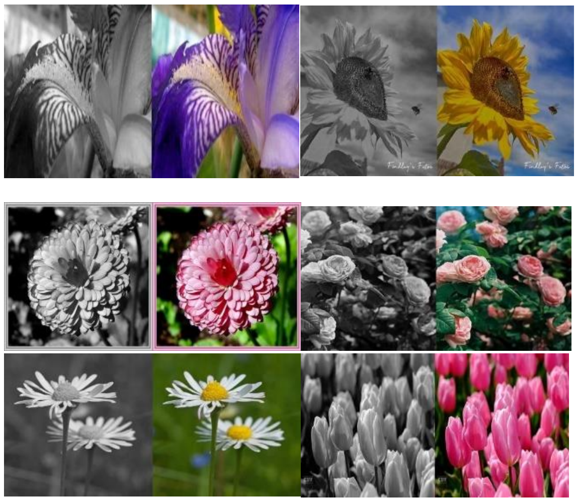
We have a discriminator and generator as in GANs, but the discriminator's job remains unchanged. Although now it has conditional inputs of observed image and the output, instead of simply the output image. The generator is tasked to not only fool the discriminator but also to be near the ground truth output. The paper on which our project is based is using Batch Normalization, we have also experiment by using Instance normalization as normalization layer. The approach to batch normalization with batch size equal to 1, is "instance normalization" and has been demonstrated to be slightly better for our implementation.

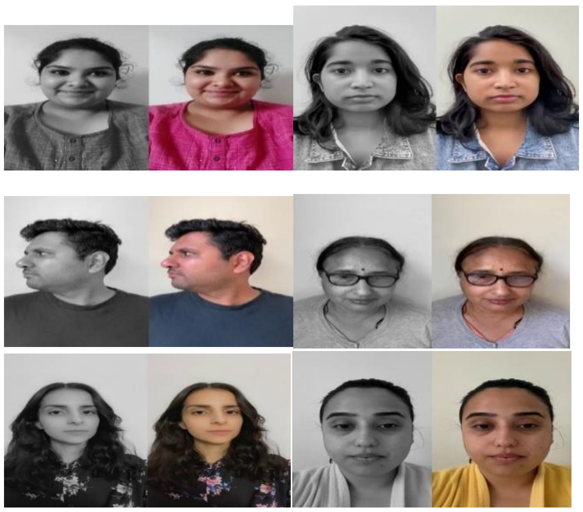
### Dataset-Details

The dataset that included of images of flowers and faces. We have used 3 datasets for training models.

1. Shortie: There are 10 classes of flowers. With total of 776 images.
2. Huge: There 17 classes with 15k+ image with high variation
3. Face: There are 21 classes with 7K+ images

Images used in training are as follows





To train the model, we attempted to gather data in the form of images/videos on our own. We collected videos of people which shows their face in different angles and expressions.

Figure 2 Collection of images per class.Figure 3 Collection of images per class.

We extracted the images from each frame of the movie using OpenCV. The faces in each frame are detected using MTCNN to remove background learning. Age, skin color, brightness, and face expression are among the dataset's variations. The dataset contains 21 unique individuals. There are 6127 photos in Training, divided into 21 categories. There are 698 photos in testing, divided into 21 categories (nearly 10 percent of dataset).

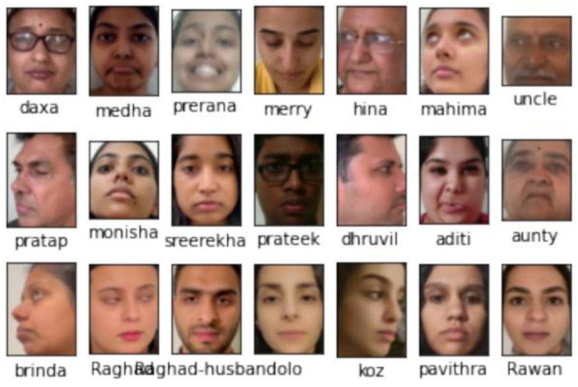


Figure 4 Collection of each class images

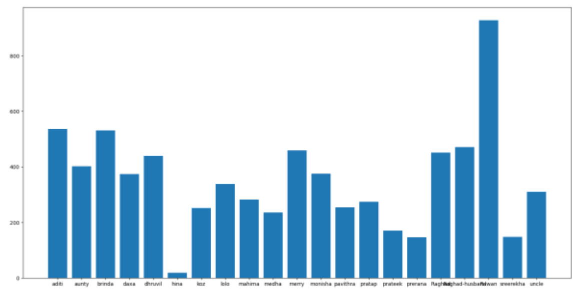


Figure 5Dataset Distribution for face-dataset

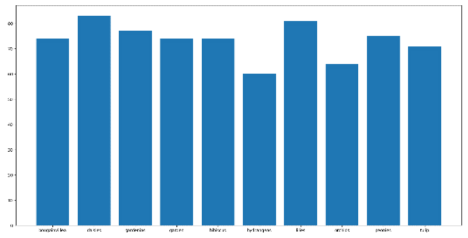


Figure 6Dataset distribution for flower classes in Shortie

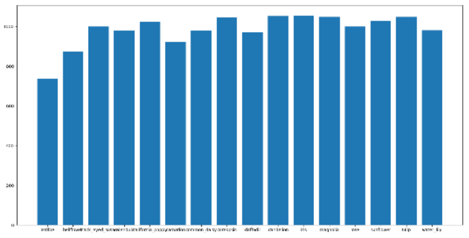


Figure 7Dataset distribution for flower classes in Huge

The challenging part of the dataset is every image has very different edges and colors which makes it difficult to for the model to generalize.

## Generator Architecture

Normally generators are supposed to be decoder. Here the Generator architecture is similar to U-Net with something called skip layers. We add skip connections between each layer i and layer n − i, where n is the total number of layers. The benefit with skip layer is we do not have to make the information travel through all the layers instead we can shuttle this information directly across the network from layer i to layer n-i.

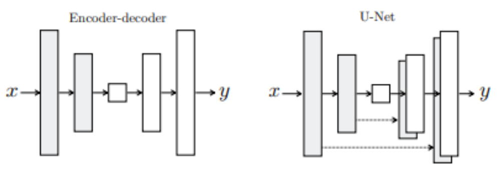
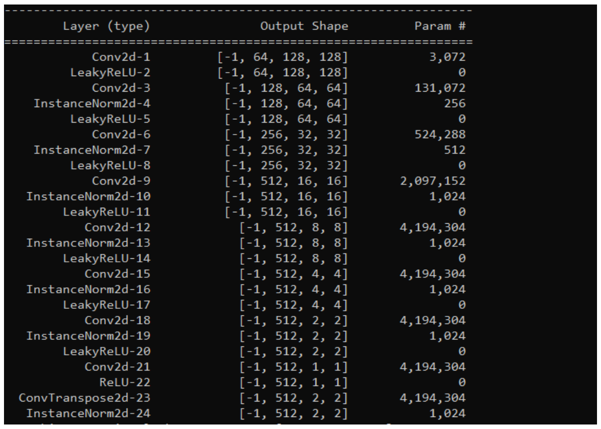


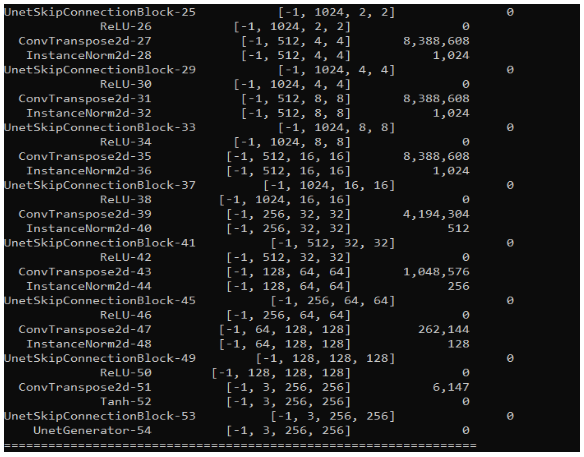
Figure 3: Two choices for the architecture of the generator. The "U-Net" [50] is an encoder-decoder with skip connections between mirrored layers in the encoder and decoder stacks.

Figure 8 UNet GAN Architecture



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 128, 128]           3,072
         LeakyReLU-2         [-1, 64, 128, 128]               0
            Conv2d-3          [-1, 128, 64, 64]         131,072
    InstanceNorm2d-4          [-1, 128, 64, 64]             256
         LeakyReLU-5          [-1, 128, 64, 64]               0
            Conv2d-6          [-1, 256, 32, 32]         524,288
    InstanceNorm2d-7          [-1, 256, 32, 32]             512
         LeakyReLU-8          [-1, 256, 32, 32]               0
            Conv2d-9          [-1, 512, 16, 16]       2,097,152
   InstanceNorm2d-10          [-1, 512, 16, 16]           1,024
        LeakyReLU-11          [-1, 512, 16, 16]               0
           Conv2d-12            [-1, 512, 8, 8]       4,194,304
   InstanceNorm2d-13            [-1, 512, 8, 8]           1,024
        LeakyReLU-14            [-1, 512, 8, 8]               0
           Conv2d-15            [-1, 512, 4, 4]       4,194,304
   InstanceNorm2d-16            [-1, 512, 4, 4]           1,024
        LeakyReLU-17            [-1, 512, 4, 4]               0
           Conv2d-18            [-1, 512, 2, 2]       4,194,304
   InstanceNorm2d-19            [-1, 512, 2, 2]           1,024
        LeakyReLU-20            [-1, 512, 2, 2]               0
           Conv2d-21            [-1, 512, 1, 1]       4,194,304
             ReLU-22            [-1, 512, 1, 1]               0
  ConvTranspose2d-23            [-1, 512, 2, 2]       4,194,304
   InstanceNorm2d-24            [-1, 512, 2, 2]           1,024
```

*Figure 9 Generative model Encoder*



```
UnetSkipConnectionBlock-25            [-1, 1024, 2, 2]               0
             ReLU-26           [-1, 1024, 2, 2]               0
  ConvTranspose2d-27            [-1, 512, 4, 4]       8,388,608
   InstanceNorm2d-28            [-1, 512, 4, 4]           1,024
UnetSkipConnectionBlock-29            [-1, 1024, 4, 4]               0
             ReLU-30           [-1, 1024, 4, 4]               0
  ConvTranspose2d-31            [-1, 512, 8, 8]       8,388,608
   InstanceNorm2d-32            [-1, 512, 8, 8]           1,024
UnetSkipConnectionBlock-33            [-1, 1024, 8, 8]               0
             ReLU-34           [-1, 1024, 8, 8]               0
  ConvTranspose2d-35          [-1, 512, 16, 16]       8,388,608
   InstanceNorm2d-36          [-1, 512, 16, 16]           1,024
UnetSkipConnectionBlock-37          [-1, 1024, 16, 16]               0
             ReLU-38         [-1, 1024, 16, 16]               0
  ConvTranspose2d-39          [-1, 256, 32, 32]       4,194,304
   InstanceNorm2d-40          [-1, 256, 32, 32]             512
UnetSkipConnectionBlock-41           [-1, 512, 32, 32]               0
             ReLU-42          [-1, 512, 32, 32]               0
  ConvTranspose2d-43          [-1, 128, 64, 64]       1,048,576
   InstanceNorm2d-44          [-1, 128, 64, 64]             256
UnetSkipConnectionBlock-45           [-1, 256, 64, 64]               0
             ReLU-46          [-1, 256, 64, 64]               0
  ConvTranspose2d-47         [-1, 64, 128, 128]         262,144
   InstanceNorm2d-48         [-1, 64, 128, 128]             128
UnetSkipConnectionBlock-49         [-1, 128, 128, 128]               0
             ReLU-50        [-1, 128, 128, 128]               0
  ConvTranspose2d-51          [-1, 3, 256, 256]           6,147
             Tanh-52          [-1, 3, 256, 256]               0
UnetSkipConnectionBlock-53           [-1, 3, 256, 256]               0
    UnetGenerator-54          [-1, 3, 256, 256]               0
================================================================
```

*Figure 10 Generative model Decoder*

## Generator Loss

GANs learn a mapping from random noise vector z to output image y, G : z → y . In contrast, conditional GANs learn a mapping from observed image x and random noise vector z, to output image y, G : {x, z} → y. image x and random noise vector z, to output image y, G : {x, z} → y. Following is the loss function of G {x,y} mapping to y.

$$\begin{aligned}\mathcal{L}_{cGAN}(G, D) =& \mathbb{E}_{x,y}[\log D(x, y)]+ \\ & \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z))],\end{aligned}$$

In the generator's context, the L1 loss is the sum of all the absolute pixel differences between the generator output and the ground truth image.

## Discriminator-Architecture

For our discriminator we use a convolutional "PatchGAN" classifier,This discriminator tries to classify if each N ×N patch in an image is real or fake. We run this discriminator convolutionally across the image, averaging all responses. This is advantageous because a smaller PatchGAN has fewer parameters, runs faster, and can be applied to arbitrarily large images.

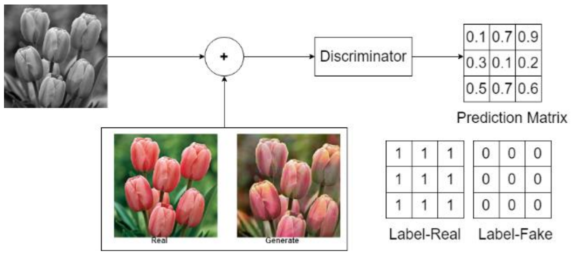
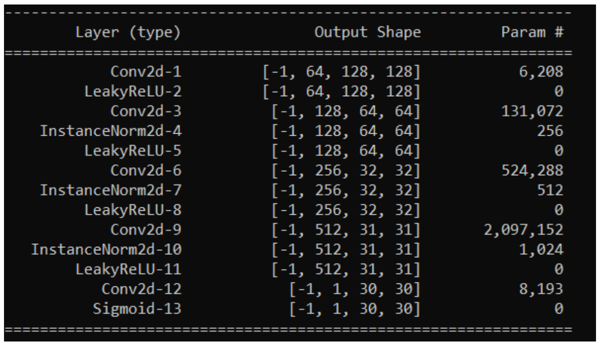


*Figure 11 Similarities between Pix2Pix GAN Discriminator and GAN Discriminator*



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 128, 128]           6,208
         LeakyReLU-2         [-1, 64, 128, 128]               0
            Conv2d-3          [-1, 128, 64, 64]         131,072
    InstanceNorm2d-4          [-1, 128, 64, 64]             256
         LeakyReLU-5          [-1, 128, 64, 64]               0
            Conv2d-6          [-1, 256, 32, 32]         524,288
    InstanceNorm2d-7          [-1, 256, 32, 32]             512
         LeakyReLU-8          [-1, 256, 32, 32]               0
            Conv2d-9          [-1, 512, 31, 31]       2,097,152
   InstanceNorm2d-10          [-1, 512, 31, 31]           1,024
        LeakyReLU-11          [-1, 512, 31, 31]               0
           Conv2d-12            [-1, 1, 30, 30]           8,193
          Sigmoid-13            [-1, 1, 30, 30]               0
================================================================
```

*Figure 12 Discriminator Architecture*

Discriminator network is trained with the same loss as the traditional GANs -the Adversarial Loss

In our code we have kept N as 30.

## E. Results

### Batch Normalization Results

- The model was trained for 1500+ epochs.
- Batch size was 32.
- Learning rate 2e-4

- Dataset used to train was Shortie.

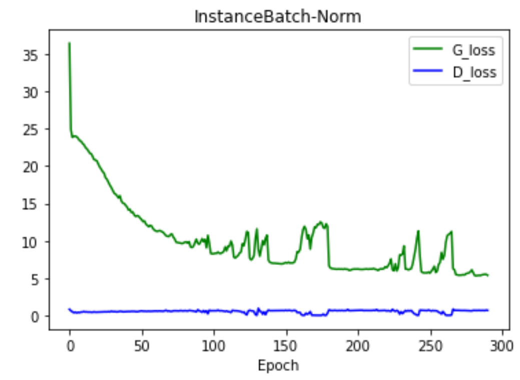


*Figure 13 Loss graph-plot*

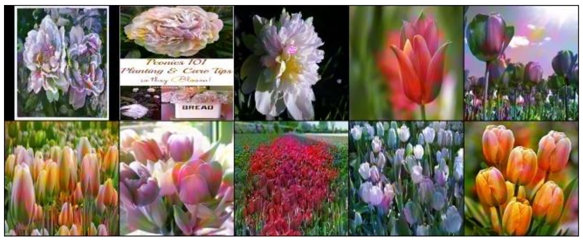
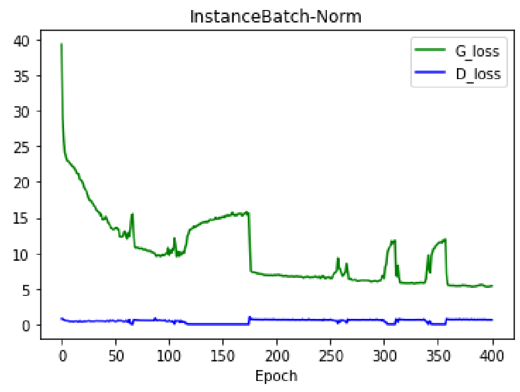


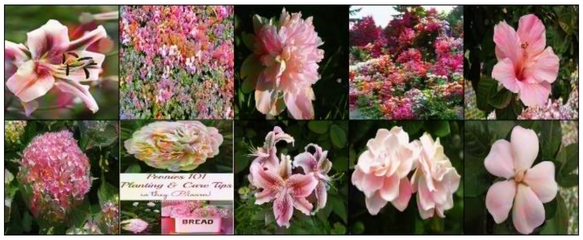
*Figure 14Dataset testset output*

## Instance Normalization Results

### *With small Dataset*

- The model was trained for 1500+ epochs.
- Batch size was 32.
- Learning rate 2e-4
- Dataset used to train was Shortie.





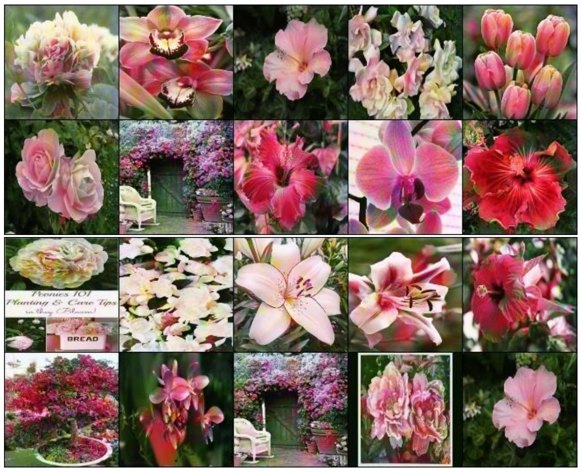


*Figure 15Dataset testset output*

### *With Large Dataset*

- The model was trained for 250 epochs.
- Batch size was 32.
- Learning rate 2e-4
- Dataset used to train was Huge.

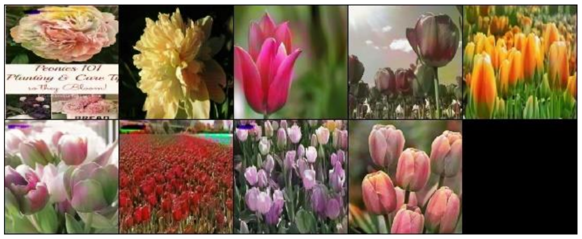


*Figure 16Dataset testset output*

### *With Face Dataset*

- The model was trained for 25 epochs.
- Batch size was 32.
- Learning rate 2e-4
- Dataset used to train was Face dataset.

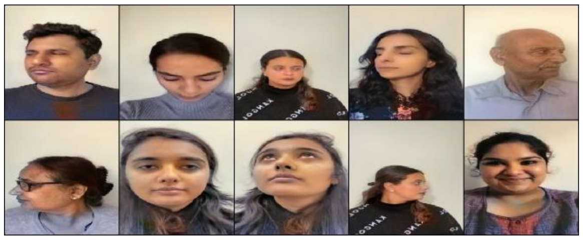


*Figure 17 Dataset testset output*

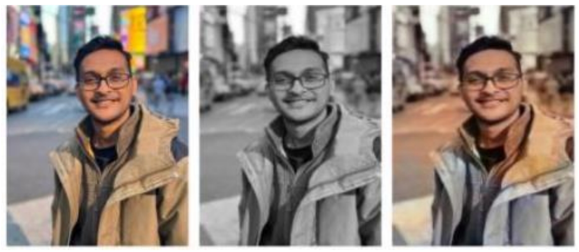


*Figure 18 left-real, center grayscale, right fake*

## Failures

While we do not expect it have the exact same colors, but this image seems pixelated and not so realistic.

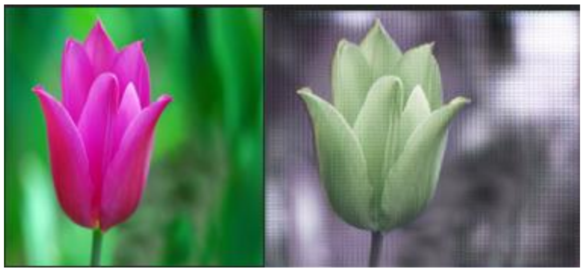
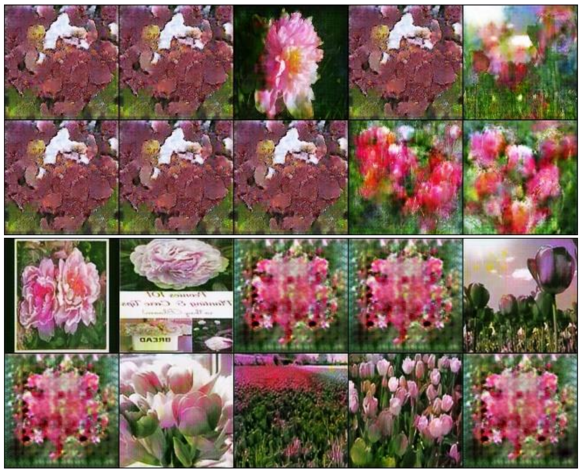
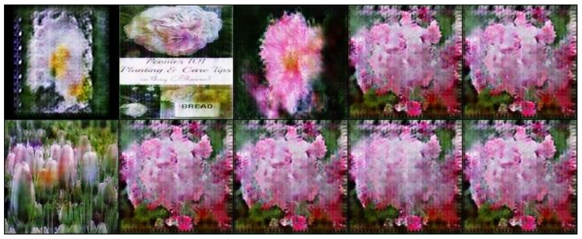


*Figure 19 Target and Generated Images*





The model generated in following images have distorted coloring because we tried to increase the model learning rate from e-5 to e-3.

## F. Conclusion

Our goals was to understand GANs a little better and experiment with lots of creative datasets. While we were limited by our GPU architectures, we still got to explore a lots of dataset and achieve decent results for some of them.

One observation was the more the number of epochs and variety in datasets, better were the results. If the dataset is not having enough variation model tends to overfit and can disperse the colors the image and spill images out of borders.

Instance norm trained better and converges faster than batch normalization. We may deduce from batch-instance normalization that models can learn to apply alternative normalizing approaches adaptively using gradient descent.

## G. Future Work

- We would like to experiment with Data Augmentations (Mosaic, Color Jitters) and check if this could optimize the results.
- So far we have only tried Leaky Relu, but experiment with Activation function like Swish/Mish is definitely in our future scope.
- We would like to train with higher resolution Images (512, 1024) and try discriminator with Patch of 70x70 for better results.
- Experiment with loss function (Changing L1 Norm to L2).
- We had set a limit to the number of epochs to about 1000, but we speculate more number of epochs would have given us better results.
- Our unfamiliarity and time constraint did not allow us the experiment with TensorFlow library and we had to stick with the pytorch library for this project. We would like to experiment with similar architecture in TensorFlow.
- CycleGANs are another architecture used for image to image translation and we would like to try it out in future.

## H. Reference

[1] Image to Image Translation with Conditional Adversarial Network:
https://arxiv.org/pdf/1611.07004.pdf

[2] Dataset :
https://www.kaggle.com/datasets/l3llff/flowers

[3] Dataset:
https://www.kaggle.com/datasets/olgabelitskaya/flower-color-images

## I. Github

https://github.com/mahima97/pix2pix-cGAN