I'M SOMETHING OF A PAINTER MYSELF

Kavya Sudha Kollu [redacted], Ishana Shinde [redacted]

December 12, 2021

Abstract

Image-to-image translation is a type of vision and graphics problem in which the goal is to learn the mapping between an input image and an output image from a training set of image pairs. However, paired training data will not be available for many tasks. In the absence of paired examples, we present a method for learning to translate an image from a source domain X to a target domain Y. Using an adversarial loss, we want to learn a mapping G: X→Y such that the distribution of images from G(X) is indistinguishable from the distribution Y. Because this mapping is highly constrained, we combine it with an inverse mapping F: Y→X and apply a cycle consistency loss to push F(G(X))≈X(and vice versa). Qualitative results are presented for several tasks that do not have paired training data, such as collection style transfer, object transfiguration, season transfer, photo enhancement, and so on. A GAN is made up of at least two neural networks: a generator and a discriminator model. The images are generated by the generator and a discriminator is used to train this generator. The two models will compete, with the generator attempting to fool the discriminator and the discriminator attempting to accurately classify the real vs. generated images. The model would take in photo images and then generate monet style images in a zip format, which would be evaluated by MiFid score to tell us how good the model's monets are.

1. Introduction

Computer Vision (CV) has without a doubt experienced a renaissance during the past decade, and perhaps the Generative Adversarial Network (GAN) is one of the most captivating examples of modern deep learning. The idea is quite intuitive: if we want to generate images that match our dataset, then we can have a generator network make an image, and a discriminator attempts to distinguish between the generated image and the real image. The magic of backpropagation ensures that, in theory, the generator produces better images to the point that the generated and real images are indistinguishable.

We recognize the works of artists through their unique style, such as color choices or brush strokes. The "je ne sais quoi" of artists like Claude Monet can now be imitated with algorithms like Generative adversarial networks (GANs). So can (data) science, in the form of GANs, trick classifiers into believing you've created a true Monet?GAN's help us create more realistic images which will be used to generate new datasets for machine learning models where we have less data, creating high resolution images from blurry low resolution images and also image manipulation is possible with GAN's (where an image of summer can be transformed into a winter image).

2. Problem Statement

To develop a GAN architecture that captures special characteristics of one image collection, in our case Monet paintings, and figuring out how these characteristics could be translated into the other image collection all in the absence of any paired training examples.

We aim to generate 7,000 to 10,000 Monet-style images which would trick the classifiers into believing that we have created a true monet using CycleGAN.

2.1 Notations

GAN - Generative adversarial networks.
DCGAN - Deep Convolutional Generative Adversarial Network
MiFID - Memorization-informed Fréchet Inception Distance

3. Literature Review

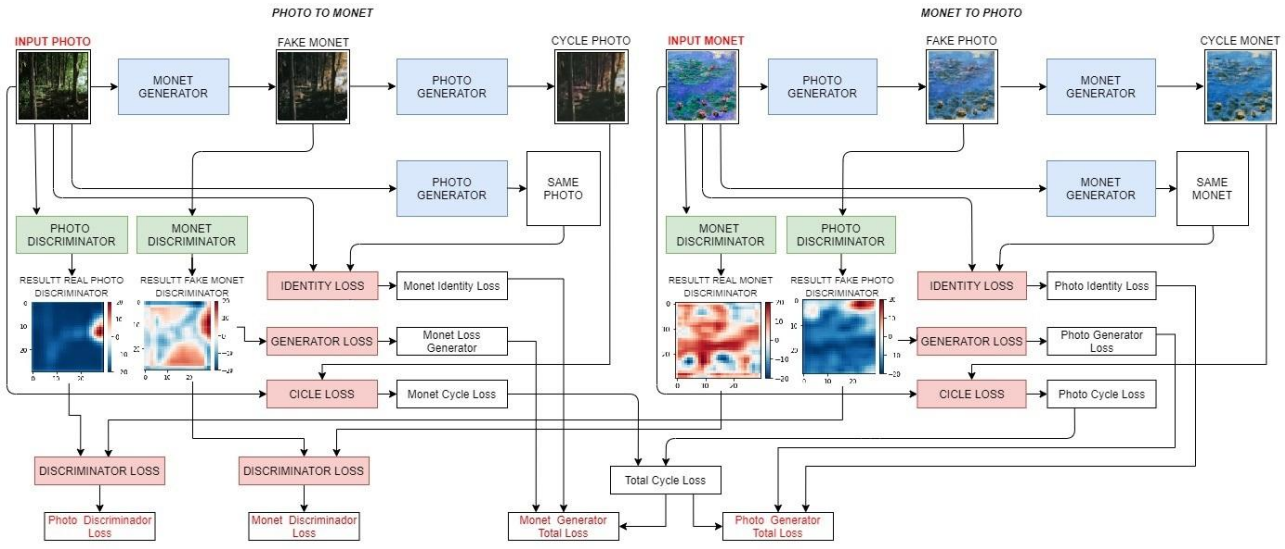
We use CycleGAN in this project in order to generate monet-style images for the photos.
While researching on the GAN architectures have come across other GAN's which can be used for image translation from one domain to another.

1. **Pix2Pix-** Pix2pix is a conditional generative adversarial network (cGAN) that learns a mapping from input images to output images. Pix2pix is not application specific, that is it can be applied to a wide range of tasks, including synthesizing photos from label maps, generating colorized photos from black and white images, transforming sketches into photos. In the pix2pix cGAN, you condition on input images and generate corresponding output images. It consists of a generator with U-net architecture and a discriminator that is represented by a convolutional PatchGAN classifier.
2. **DCGAN-** DCGAN uses convolutional and convolutional-transpose layers in the generator and discriminator, respectively. The discriminator consists of strided convolution layers, batch normalization layers, and LeakyRelu as activation function. It takes a 3x64x64 input image. The generator consists of convolutional-transpose layers, batch normalization layers, and ReLU activations. The output will be a 3x64x64 RGB image.

Apart from the published studies, there were few other concepts published on Kaggle for the data set, among which most of them have generated monet style images for the data using Pytorch and also using other architectures like DCGAN and Pix2Pix. As part of our project, we are using CycleGAN to develop monet style images. We developed our project using Python language.

4. Methods and Techniques

Generative Adversarial Networks (GAN) uses a generator and a discriminator. The generator has to generate images that are accepted by the discriminator. The discriminator tries to discover the images that are not real and reject the images generated by the generator. CycleGAN uses a **loss of cycle consistency** to allow training without the need for paired data. In unpaired dataset, there is no predefined meaningful transformation that the generator can learn, so we will create it. The model taking an input image from domain DA which is fed to our first generator $GeneratorA \rightarrow B$ whose job is to transform a given image from domain DA to an image in target domain DB. This new generated image is then fed to another generator $GeneratorB \rightarrow A$ which converts it back into an image, $CyclicA$, from our original domain DA. This output image must be close to the original input image to define a meaningful mapping that is absent in unpaired dataset.



4.1 Data Cleaning (Image Preprocessing)

Resizing image (In this case it would not be necessary to do it, because the images are already in the necessary size. But with this step if we wanted to add new images it would not be necessary to scale them previously

Normalizing the images to [-1, 1]

Random jittering and mirroring to the training dataset. These are some of the image augmentation techniques that avoids overfitting, Random jittering performs:Resize an image to bigger height and width Randomly crop to the target size Randomly flip the image horizontally.

4. BUILDING THE MODEL

To build the model we follow the following steps:

1. Building the Generator -

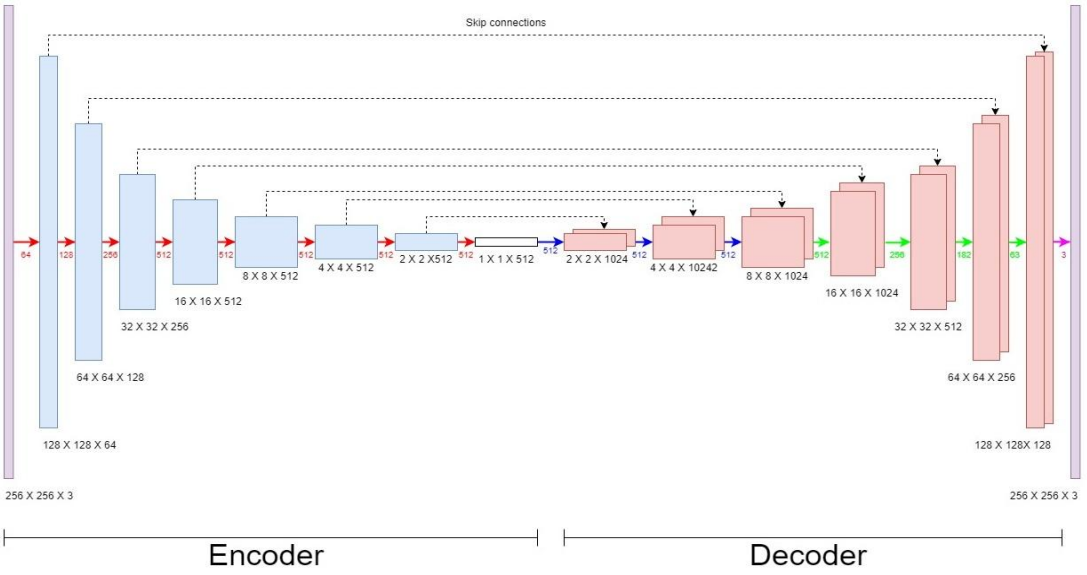
The architecture of the generator is a modified U-Net, consisting of an encoder block and a decoder block, each of them is made up of simpler blocks of layers:

Each block of the encoder, we call it downsample-k where k denotes the number of filters, this block performs an image compression operation (downsample). It consists of the following layers

- Convolution
- Instance Normalization
- Leaky ReLU

In the decoder we can find two types of blocks, depending on whether a dropout operation is performed or not, we will call each one of them upsampleD-k and upsample-k, since they are performing a decompression.The Skip connections exist between encoder and decoder. Each of these blocks is made up of:

- Transposed Convolution
- Instance Normalization
- Dropout
- ReLU



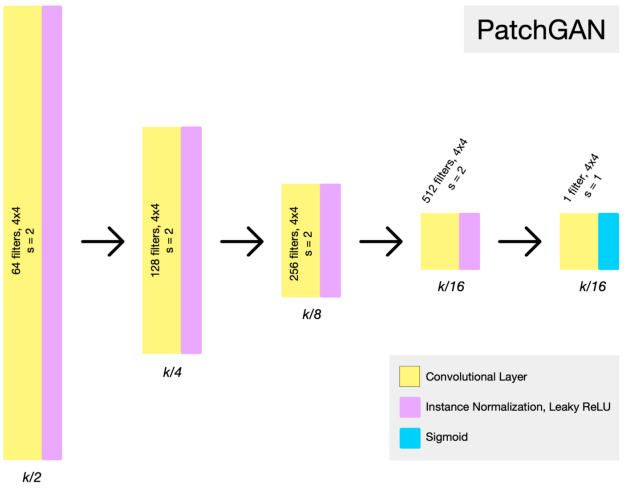
In the above diagram of the Generator, in which the inputs and outputs of the different blocks can be seen.

- The red arrows represent the execution of a downsample-k block
- The blue arrows represent the execution of an upsampleD-k block (with Dropout)
- The green arrows represent the execution of an upsample-k block (without Dropout)
- Lastly, an output layer, represented by a pink arrow is executed, which is simply Transposed Convolution of 3 filters to convert the output into a 256X256 image by 3 channels

All convolutional layers of downsample have the parameter strides = 2, which causes the dimensions to be reduced by half, likewise the Transposed Convolution layers of upsample also have the parameter strides = 2 so the dimensions are doubled. In the first two dimensions, not counting the Batch size dimension. The size of the 3 dimensions will coincide with the number of filters applied.

2. Building the Discriminator -

The task of the discriminator is whether an input image is original or fake (the output of a generator). The architecture of the discriminator is a convolution network of the PatchGAN type, instead of returning whether the image is real or not, this architecture returns whether pieces of the image can be considered real or false. consisting of an encoder block made up of simpler blocks of layers:



As in the generator, the encoder is made up of downsample-k blocks; this block performs an image compression operation (downsample). It consists of the following layers

- Convolution
- Instance Normalization (not apply to the first block)
- Leaky ReLU

So the already implemented function is used as a "downsample", all convolutional layers of downsample have the parameter strides = 2, which causes the dimensions to be reduced by half. The shape of the discriminator output layer is (batch_size, 30, 30, 1), each 30x30 patch of the output sorts a 70x70 portion of the input image.

3. Definition of Loss Function -

Sigmoid cross entropy is used to calculate the adversary losses in the discriminator and generator. **Discriminator loss** -

    - For the discriminator_photo will take as input:
        - The output of the discriminator_photo whose input is the real photo of the training set
        - The output of the discriminator_photo whose input is the fake photo generated by the generator_photo
    - For the discriminator_monet will take as input:
        - The output of the discriminator_monet whose input is the real Monet of the training set
        - The output of the discriminator_monet whose input is the fake Monet generated by the generator_monet
- The calculation of the loss has two components:
    - real_loss compares the real image with a matrix of 1. (All Ok)
    - generate_loss compare the fake image with a matrix of 0 (All Fake)
- So the total_loss is 1/2 of the sum of the real_loss and the generate_loss

Generator loss

1. **Generator adversary loss -** It Takes as input the output of the discriminator

- For the loss of the generator_monet the function will take the output of the discriminator_monet executed with fake monet
- For the loss of the generator_photo the function will take the output of the discriminator_photo executed with fake photo

The perfect generator will have the discriminator output only ones. Therefore, compare the generated image with a matrix of 1 to find the loss.

2. **Generator cycle consistency loss -** As the training data is not paired, to make the network learn the correct mapping and the result is similar to the original input. The cycle consists of:

- Generate a monet style image from a photo, this generated image is passed as input to the second generator, which should generate a photo from a fake monet style image.
  - Input Photo -> G_MONET -> Fake Monet -> G_PHOTO -> Cycle Photo
- On the other hand, an image is generated that aims to imitate a real photo from a Monet painting, this generated image is passed as an input to the second generator, which should generate the Monet painting again from the fake photo.
  - Input Monet -> G_PHOTO -> Fake Photo -> G_MONET -> Cycle Monet

To calculate the cycle consistency loss, two terms are calculated

- Average absolute error for photo, is calculated between Input Photo and Cycle Photo
- Mean absolute error for monet, is calculated between Input Monet and Cycle Monet

The cycle error will be the sum of both terms. This term is applied to the total loss calculation for both generators (generator_monet and generator_photo)

Identity loss - It is desired that if the generator_monet is executed with the image of the original monet painting, the result should be a very similar image, and also with the generator_photo and the original photo. Therefore, the loss of identity forces what the generator generates to resemble the input. It takes as inputs, 2 images, the original image and the output of running the generator with the original image.

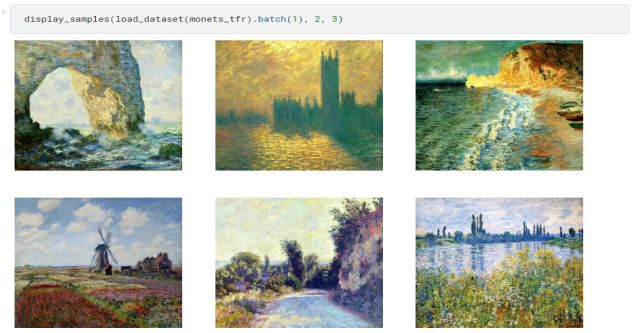
- For the loss of the generator_monet the function will take the training monet image and the output of the generator_monet with the same input (same_monet)
- For the loss of the generator_photo the function will take the training photo image and the output of the generator_photo with the same input (same_photo)

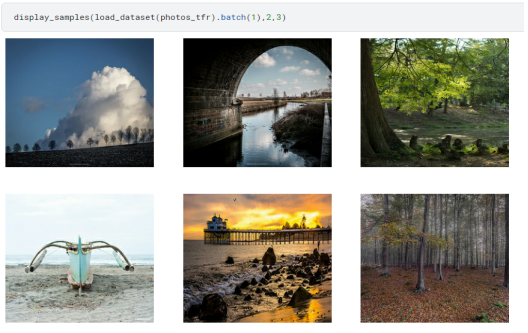
The loss will be the mean absolute error between the real image and the generated one.

5. Discussion and Results

5.1 Datasets

- The dataset contains four directories: `monet_tfrec`, `photo_tfrec`, `monet_jpg`, and `photo_jpg`. The `monet_tfrec` and `monet_jpg` directories contain the same painting images, and the `photo_tfrec` and `photo_jpg` directories contain the same photos.
- The Monet directories contain Monet paintings. Use these images to train your model.
- The `photo` directories contain photos. Add Monet-style to these images and submit your generated jpeg images as a zip file. Other photos outside of this dataset can be transformed but keep your submission file limited to 10,000 images.

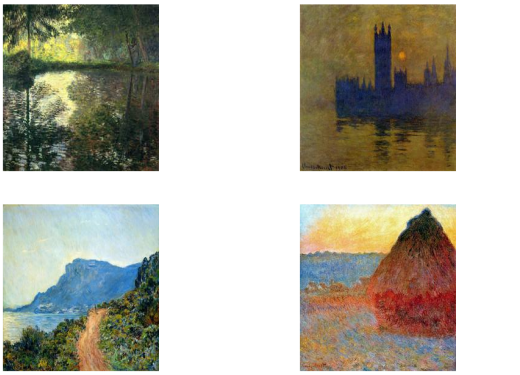




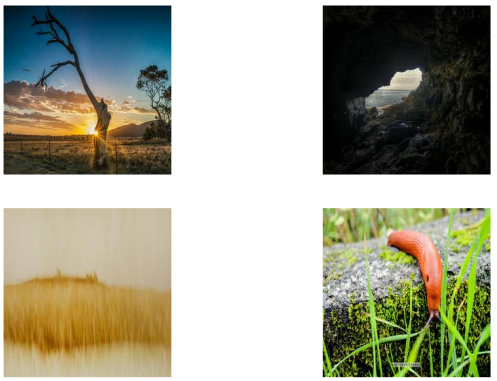
5.3 Visualization of data to understand images better

- **Batch Visualization -**

```
batch_visualization(MONET_PATH, 6, is_random=True, figsize=(16, 16))
```

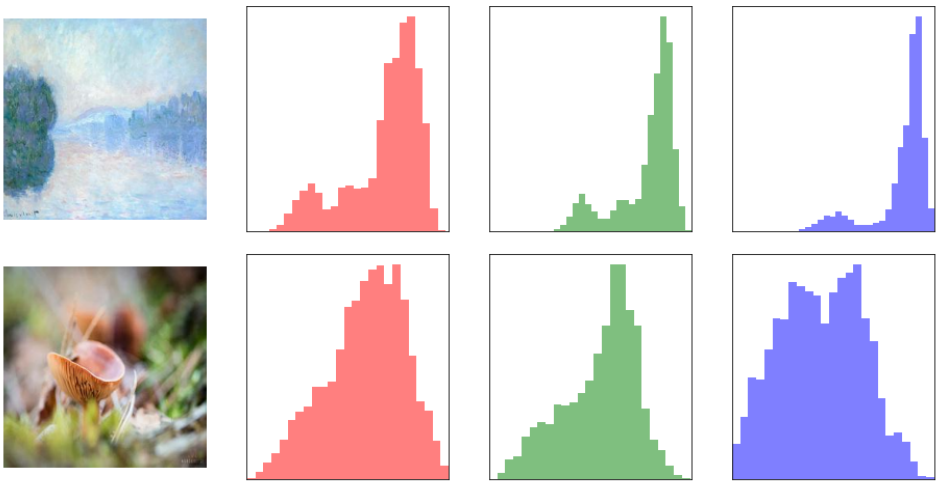


```
batch_visualization(PHOTO_PATH, 6, is_random=True, figsize=(16, 16))
```

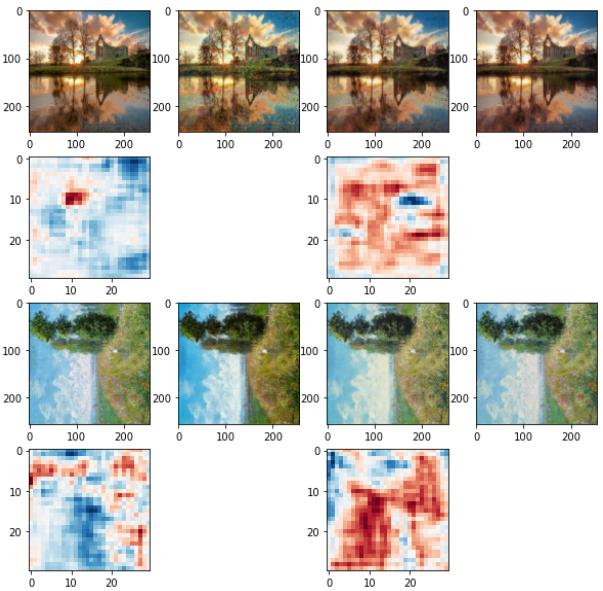


- **Colour histograms -**

```
image_1 = '../input/gan-getting-started/monet_jpg/0260d15306.jpg'
image_2 = '../input/gan-getting-started/photo_jpg/0033c5f971.jpg'
color_hist_visualization(image_1)
color_hist_visualization(image_2)
```



- **Displaying Discriminator Photo and Monet -**



5.4 Evaluation Metrics

- The evaluation metric used in order to determine the accuracy of the model is MiFID (Memorization-informed FID)
- The memorization distance is defined as the minimum cosine distance of all training samples in the feature space, averaged across all user generated image samples. This distance is thresholded, and it's assigned to 1.0 if the distance exceeds a pre-defined epsilon

In mathematical form:

$$d_{ij} = 1 - cos(f_{gi}, f_{rj}) = 1 - \frac{f_{gi} \cdot f_{rj}}{|f_{gi}||f_{rj}|}$$

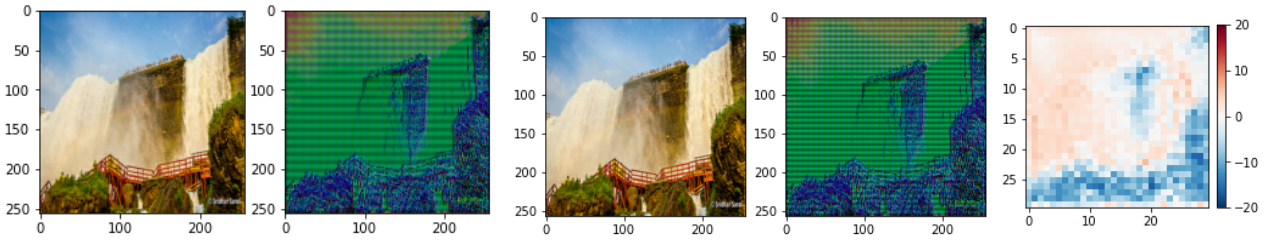
where f_g and f_r represent the generated/real images in feature space (defined in pre-trained networks); and f_{gi} and f_{rj} represent the i^{th} and j^{th} vectors of f_g and f_r, respectively.

$$d = \frac{1}{N} \sum_i \min_j d_{ij}$$

.

5.5 Experimental Results

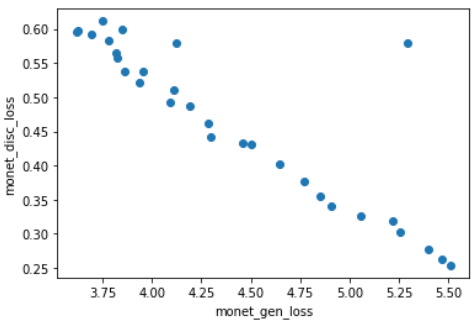
Generator Initial Test - **Discriminator Initial Test -**

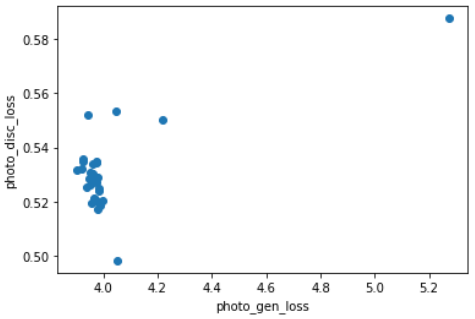


- Initially the model generated a MiFid score of 51.30010 this was when epochs were set to 30 and the batch_size was set to 32 with the tensorflow random normal initializer set to (0.0,0.02)
- For improving the generated images and to reduce the MiFID score we started **tuning in different parameters** and for our next submission we changed the random normal initializer from (0.,0.02) to (0.,0.04) which gave us an MifiD score of 46.00195 which was an improvement to the earlier version
- For the next version of CycleGAN we reduced the batch_size which was set to 32 to 4 and also steps_per_epoch which was set to the (max(monet_jpg, photo_jpg)//BATCH_SIZE) to 1407 which decreased the MiFid score to 42.30138.
- For our final submission we tuned in few more parameters where we set the BATCH_SIZE to 4 , changed the random normal initializer from (0.,0.02) to (0.,0.04) for upsampling and downsampling and leaving it to (0.,0.02) for discriminator and generator and increased the lambda_cycle to 20 which gives us an MiFid score of 40.34.

5.5.1 Cross Validation

Since CycleGANs are unsupervised, we are using the generator loss for photos and monets as well as the discriminator loss for photos and monet for cross validation.The epoch value for the model is set to 30 and each epoch generates monet_gen_loss, photo_gen_loss, monet_disc_loss, photo_dis_loss. These metrics determine how the model should learn in order to generate better monet images. As the model keeps learning the generator loss as well as the discriminator loss decreases. Below are a few visualizations for the cross





validation.

Final Output Images Generated:

6. Conclusion

GANs are exciting and a rapidly changing field delivering promise on generative models using deep learning methods. Overall, the results produced by CycleGAN are very good, especially the image quality approaches that of paired image-to-image translation on many tasks. This is impressive, because paired translation tasks are a form of fully supervised learning, and this is not.

We implemented CycleGAN using tensorflow keras and achieved a **MiFID score of 40.34** with a **rank of 17** on the public score standings on Kaggle. Along with implementing CycleGAN, we have tried DCGAN and Pix2pix but both models gave us a high value of MiFID 210 and 53.4 respectively. As it was yielding a very high Mifid score it was discarded from our Implementation.

6.1 Directions for Future Work

- Enhancing the model by integrating Reinforcement learning along with the use of Generative Adversarial Networks (GANs) and modifying the Q-learning function to generate a more efficient model.[9]
- Using Autoencoders and Variational Autoencoders[2] instead of GANs to convert images to Monet.
- Incorporating different styles like Van Goh along with Monet style.
- Along with the above suggestions, we would like to try and build a User Interface for our project where the user will be able to upload any photo and its monet style would be generated and vice versa.

7. References

[1] https://ucladatares.medium.com/make-a-monet-image-style-transfer-with-cycle-gans-5475dcb525b8
[2]https://openaccess.thecvf.com/content_CVPR_2019/papers/Sarmad_RL-GAN-Net_A_Reinforcement_Learning_Agent_Controlled_GAN_Network_for_Real-Time_CVPR_2019_paper.pdf
[3] https://towardsdatascience.com/must-read-papers-on-gans-b665bbae3317
[4] https://arxiv.org/pdf/1703.10593.pdff
[5] https://towardsdatascience.com/cyclegans-to-create-computer-generated-art-161082601709
[6] https://hardikbansal.github.io/CycleGANBlog/
[7]https://medium.com/analytics-vidhya/transforming-the-world-into-paintings-with-cyclegan-6748c0b852
[8] https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b
[9]https://openaccess.thecvf.com/content_CVPR_2020/papers/Rao_RL-CycleGAN_Reinforcement_Learning_Aware_Simulation-to-Real_CVPR_2020_paper.pdf