# IMAGE TO IMAGE TRANSLATION USING CYCLEGAN

Shubham Maheshwari (17EC10055)
IIT Kharagpur
shubhammaheshwari1999@gmail.com

## 1 Abstract

The primary aim of this project is to implement a system by training a CycleGAN to read an image from Domain/Set X and transform it to make it seem like an image from Domain/Set Y. In general, image to image translation involves the use of a training collection or dataset of matched image pairs to practice the conversion from an input image to a desired output image which is a problem from the class of vision and graphics. However, usable matched training data may not be available for specific tasks. The framework presented here to transform an image from a source domain dataset X to a target domain dataset Y would be beneficial for those cases where there is an absence of paired instances.

The General Adversarial Network (GAN) architecture is a method of training an image generation model that consists of a generator and a discriminator model. The discriminator inputs an image and predicts if it is true (from a dataset) or false. In contrast, the generator takes a point from a latent space as input and produces new possible domain pictures. CycleGAN is an extension to this architecture wherein the concept of cycle consistency is employed. The theory of cycle consistency is that on conversion of an image from domain X → domain Y and then back to → domain X, the net effect of this cycle should look exactly like the image we began with.

CycleGANs can do almost everything involving photo editing like altering temperature or season, changing the day's time as well as auto colouring of black and white images. I have used the dataset monet2photo, which is open source and freely available. I have used Pytorch for the implementation of the CycleGAN architecture for this project where upon training and learning, an image from Monet paintings gets translated into a lookalike of a natural photo.

## 2 Introduction and Methodology

General Adversarial Network (GANs) are part of the generative models class of algorithms. These algorithms are part of the unsupervised learning area, which is a branch of machine learning that focuses on algorithms that have the ability to learn about the underlying structure of data without defining a specific goal. They are made by unifying the generator and the discriminator models.

The generator tries to produce data from a probability distribution while maximising the chances that the discriminator would misinterpret its inputs as right. The discriminator plays the role of a judge. Its job is similar to that of a classifier as it attempts to identify the dissimilarity between real image and the image produced by the generator. In theory, throughout the training process, the generator improves upon its ability to create pictures that look closely like the original ones. On the other hand, the discriminator converges to a state where it cannot differentiate between the fake dataset and the original. A very common example for understanding this is that people weren't aware that duplicate currency was being created in the old days. There were also no detective instruments that could determine if the money was genuine or counterfeit. People used to accept currency notes simply by looking at them (i.e., fake currency also looked like real currency but they could not be verified by naked eye). In this analogy, 'generation' of duplicate currency acts as generator model and the 'detection' of whether the currency is fake or real acts as the discriminator model.

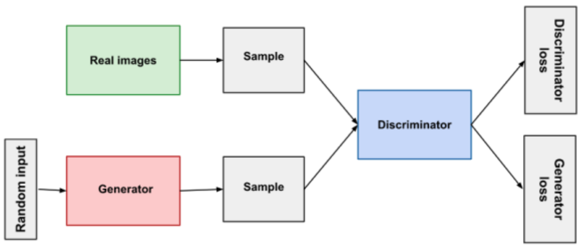


Figure 1. GAN Structure Overview

There are two generators (G and F) and two discriminators in CycleGAN (Dx, Dy). Generators may be thought of as mapping functions that convert data from domain X to domain Y and vice versa. The discriminator Dx encourages F to transform images from domain Y into output images that are distinctive and unique from those in domain X and vice versa for Dy and G. For supplemental regularization of the above-mentioned mappings, cycle consistency losses are used as those losses help in capturing the fact that we should be able to return to where we began if we convert from one domain to the other and back.

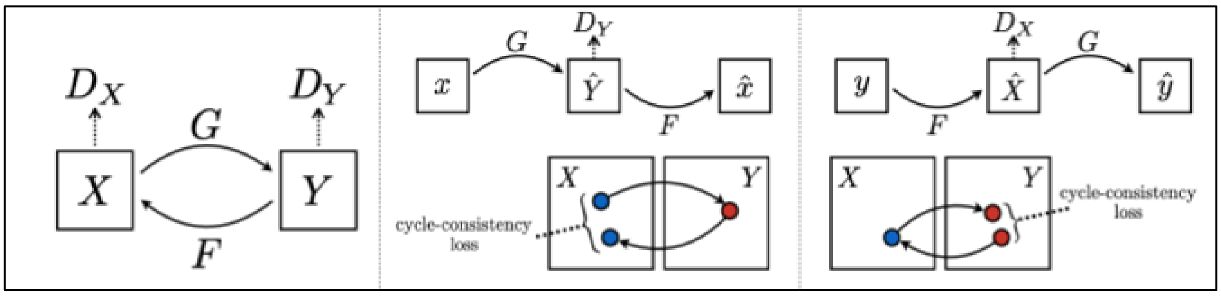


Figure 2. Original Image from the paper by Zhu et al.

The discriminator loss is calculated as the mean squared error between the discriminator's output and the target value, which is either 0 or 1 depending on whether the picture can be labelled as counterfeit or original. For instance, for a real image, x, the mean squared error (mse) can be used to train Dx by looking at how similar it is to identifying an image x as true.

```
out_x = D_X(x)
real_err = torch.mean((out_x-1)**2)
```

Calculation of the generator losses is also somewhat similar to calculating the discriminator loss. There are steps where fake images that appear to belong to the set of X images but are actually based on real images from set Y are created, and vice versa. The "true loss" on those produced images is then calculated by examining the discriminator's performance as it is applied to the false images. Hence, the generator aims to make the discriminator classify the fake images as real images. The Adversarial loss of generator G is:

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y\sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))]$$

The zero-sum game connecting the Generator and the Discriminator (G and D) is: minG maxDY LGAN(G, DY, X, Y), where G tries to reduce the below goal while DY tries to maximise it. Also with discriminator DX is a related technique used with generator F. Theoretically, Adversarial training can be used to learn mappings G and F that yield outputs that are distributed in the same way as the X and Y objective domains. With sufficient processing power, a network can map the fixed set of input images to some random permutation of images within the target domain, resulting in an output distribution that fits the target distribution.

Therefore, it can be established that adversarial losses alone cannot give us the assurance of learning a function that would map Xi to the desired Yi.

In addition to the adversarial losses, the generator loss terms also include the cycle consistency loss that is a measure of how good a reconstructed image is, when compared to an original image. Both generators are pushed to be consistent by the Cycle-consistency loss. It controls them and brings stability to a GAN's training mechanism that is otherwise very unstable.

Forward Cycle consistency means x → G(x) → F (G(x)) ≈ x, that is, for each image x ∈ domain X, the image translation cycle should be able to bring x back to the original image. Similarly for domain Y, it is called backward cycle consistency. For example, if there is a fake or generated image, x̂, and a real image, y, then a reconstructed image ŷ can be obtained by applying GXtoY(x̂) = ŷ and then checked to see if this reconstruction ŷ and the original image y match. For this, calculating the L1 loss, which is an absolute difference, between reconstructed and real images can be used. This loss is represented as the function:

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y\sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

After combining all losses, the full objective is calculated as the sum of the generator loss and the forward as well as the backward cycle consistency loss. To stress the relative importance of cyclic losses, they are multiplied by a constant lambda. The complete expression is as follows:

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) =& \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \\ &+ \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ &+ \lambda\mathcal{L}_{\text{cyc}}(G, F),\end{aligned}$$

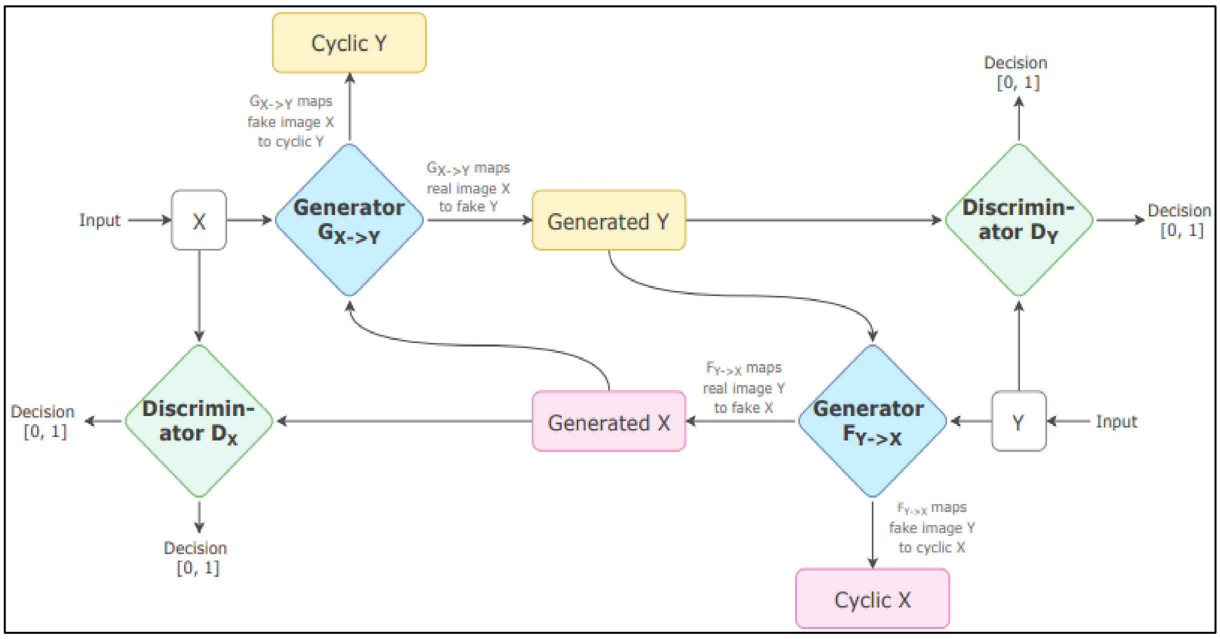


Figure 3. Overview of the working of CycleGAN for Image Translation

## 3 Contribution and Implementation

Now that the philosophy behind Cycle GANs has been covered, it's time to look at how I've implemented them.

### Dataset

The dataset "monet2photo.zip" was downloaded and for the purpose of using it in google colab the directory structure was changed a bit. It contains 1073 Monet paintings and 6288 photos as the training set. For the test set, it has 123 Monet paintings and 753 photos. As a part of preprocessing, all photos were converted to 256x256x3 as the right square size for input to the model. Apart from that, the images were converted into tensor image types. The pixel values in the images were also rescaled to -1 to 1 from the original 0 to 1 due to the fact that the tanh activated generator will output pixel values in a range from -1 to 1.

### Generator Network

G_XtoY and G_YtoX (also known as F) are generators that consist of a three layer encoder, a conv net that transforms an image into a smaller function representation, and a 3 layer decoder, a transpose conv net that transforms that representation into a transformed image. This network takes a 256x256x3 image and compresses it into a feature representation using three convolutional layers before arriving at a sequence of residual blocks. It goes through a few of these residual blocks (usually 6 or more), then three transposed convolutional layers (also known as de-convolution layers), which up sample the output of the residual network blocks and generate a new picture. With the exception of the final transpose convolutional layer, which has a tanh activation function added to the output, most of the convolutional and transpose-convolutional layers have Batch Normalization and ReLu functions applied to their outputs. The generator architecture used is depicted in table 1.

| | Layer type | Kernel size | Stride | I/O Dims | I/O Chans |
|---|---|---|---|---|---|
| conv1 | Conv2d | 4 | 2 | 256/128 | 3/64 |
| | Batch Norm2D | | | | |
| | ReLu | | | | |
| conv2 | Conv2d | 4 | 2 | 128/64 | 64/128 |
| | Batch Norm2D | | | | |
| | ReLu | | | | |
| conv3 | Conv2d | 4 | 2 | 64/32 | 128/256 |
| | Batch Norm2D | | | | |
| | ReLu | | | | |
| 9 Residual Blocks with 2 conv2D layers (kernel size 3, stride 1, 256 input-output channels) | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| deconv1 | Conv Transpose2d | 4 | 2 | 32/64 | 256/ 128 |
| | Batch Norm2D | | | | |
| | ReLu | | | | |
| deconv2 | Conv Transpose2d | 4 | 2 | 64/ 128 | 128/64 |
| | Batch Norm2D | | | | |
| | ReLu | | | | |
| deconv3 | Conv Transpose2d | 4 | 2 | 128/ 256 | 64/3 |
| | tanh | | | | |

Table 1. Generator Architecture

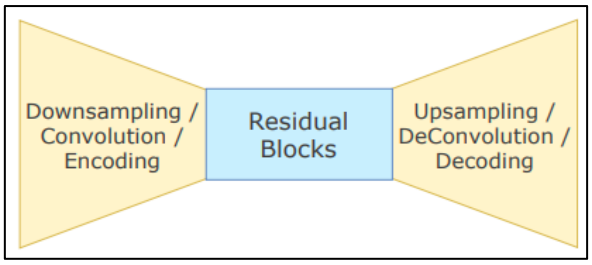

Downsampling / Convolution / Encoding
Residual Blocks
Upsampling / DeConvolution / Decoding

Figure 4. Generator Network

## Residual Blocks

Deep neural nets are harder to train because of the fact that they are most probable to have the problem of vanishing/exploding gradients. Hence, deep neural networks have problems in achieving convergence. Batch Normalization slightly assists with this problem but there occurs a training degeneration in deep networks wherein the improvement in training accuracy almost saturates or gets worse with more and more epochs of training. A solution to this is by using residual network blocks which learn certain residual functions on application to layer inputs.

Kaiming He et. al was the first to propose the idea of residual blocks. Residual blocks basically connect the output of one layer with the input of some layer before it. Two convolution layers make up each residual block. Batch normalization and ReLu activation accompany the first convolution layer. After that, the output is passed into a second

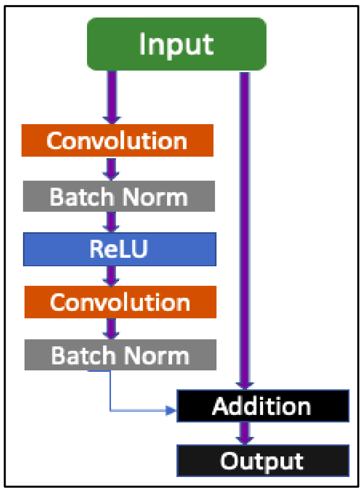


Figure 5. Residual Block

convolution layer before being batch normalized. The resultant output is then merged with the initial input. Depending on the input shape of images, the number of residual blocks in the model are changed. For example, in this model, the input image is of shape 256x256x3 and I have used 9 residual blocks but if the input image shape is changed to 128x128x3 then only 6 residual blocks would suffice. The basic architecture of the residual block used is shown in figure 5.

## Discriminator Network

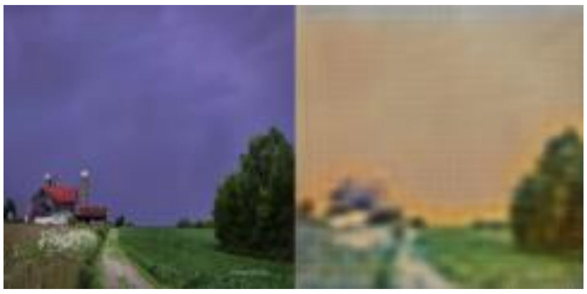
A regular GAN discriminator outputs the probability between 0-1 for an input image being fake or real whereas PatchGAN discriminator gives an array output representing probabilities of corresponding patches of the input image being real or fake. A simpler network based on PatchGAN architecture for discriminator has been used in this model due to limited computational power. I have used five convolution layers each followed by batch normalization and Leaky ReLU with the exception for no batch normalization being the first and final layer and the exception for no LeakyReLU being the final layer. LeakyReLU has been used to make sure that a small gradient exists even when the unit is inactive, that is, the function $f(x) = \alpha x$ for $x<0$.The value $\alpha = 0.2$ has been used in this case. The output image of the discriminator here is of size 15x15x1 where each pixel value represents the probability of each corresponding section being real. The discriminator architecture used is shown in the table 2.

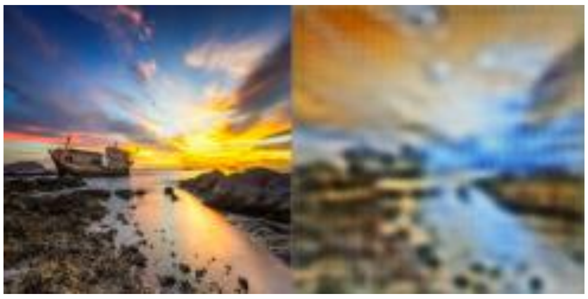
| | Layer type | Kernel size | Stride | I/O Dims | I/O Chans |
|---|---|---|---|---|---|
| conv1 | Conv2d | 4 | 2 | 256/128 | 3/64 |
| | LeakyReLU | | | | |
| conv2 | Conv2d | 4 | 2 | 128/64 | 64/128 |
| | Batch Norm2D | | | | |
| | ReLu | | | | |
| conv3 | Conv2d | 4 | 2 | 64/32 | 128/256 |
| | Batch Norm2D | | | | |
| | ReLu | | | | |
| conv4 | Conv2d | 4 | 2 | 32/16 | 256/512 |
| | Batch Norm2D | | | | |
| | ReLu | | | | |
| conv5 | Conv2d | 4 | 1 | 16/15 | 512/1 |

Table 2. Discriminator Architecture

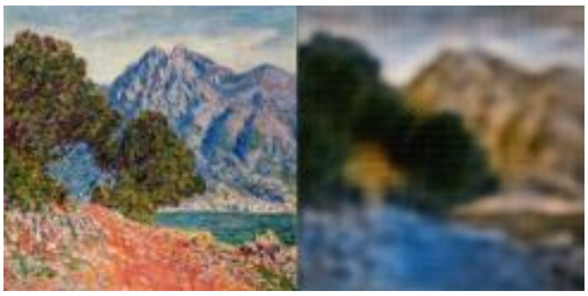
## 4 Observation and Results

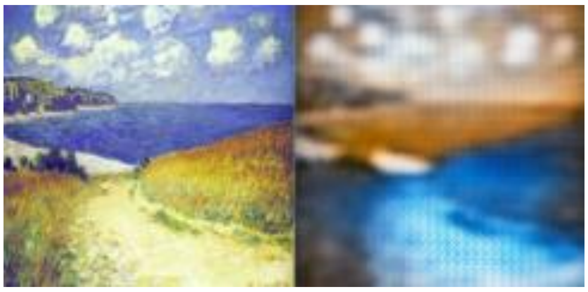
In general, transforming a Monet painting to a photo is more difficult than converting a photo to a Monet painting. Despite the non-decreasing variation, the test results were observed to be of higher quality than expected. The goal domain was correctly translated for the majority of the images. It was very important to tune the parameters and hyperparameters in a way that the generator or the discriminator did not dominate each other during training. Some of the outputs of my model for the translation from Photos to Monet paintings are as follows:





Some output results for the reverse translation, i.e., from Monet paintings to photos are given below as well:





Ideally a discriminator should not be able to differentiate between real and generated fake images but on plotting it is observed that there is always some loss. Losses for both Dx and Dy remain at almost same values throughout the training indicating that the training is not favoring any discriminator over the other. Generator’s loss started at much higher levels than the discriminator losses because it takes into account the loss of both the generators as well as weighted reconstruction errors. The generator’s loss improves a lot in the initial epochs of training because the fake generated images are too different from the original ones. The discriminator losses saturated around 0.5 and the generator’s overall loss saturated around 3-3.5 after optimization over 1000 epochs. The plots of the loss functions for the discriminators X and Y as well as the generator are shown in the figure 6.

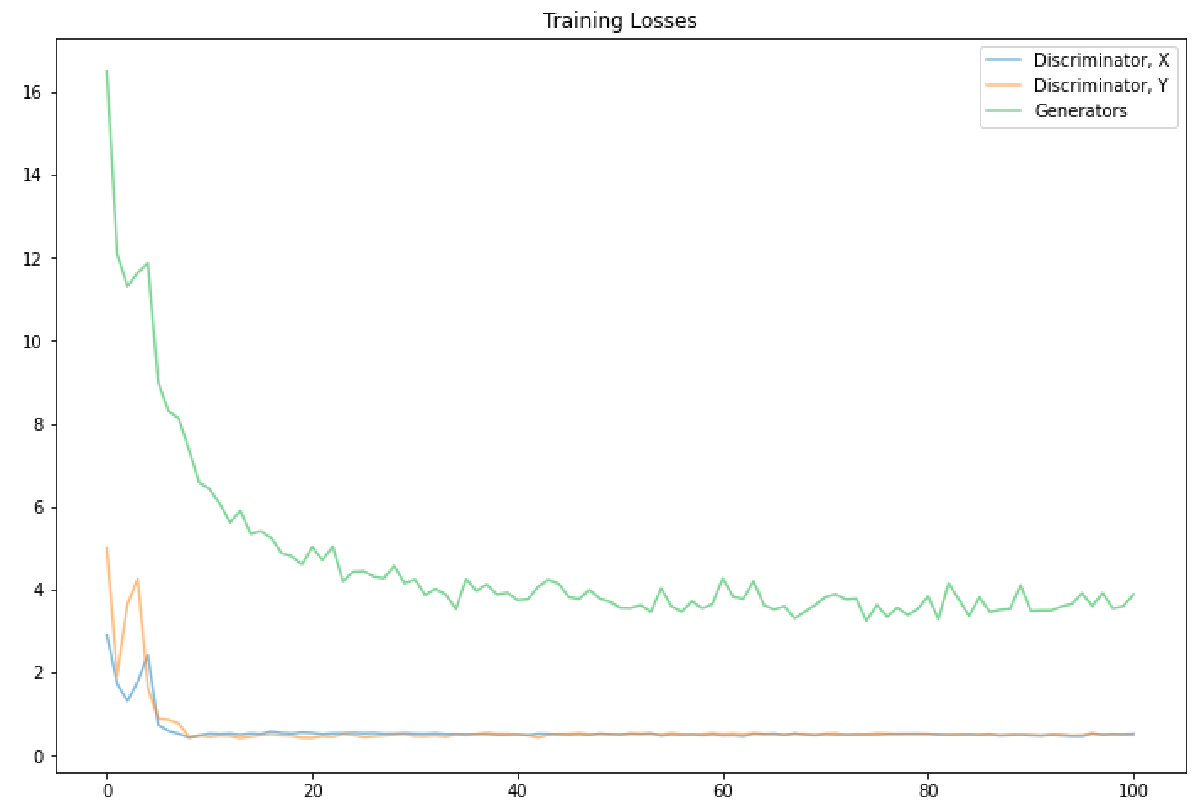


Figure 6. Training Losses over 100 epochs

## 5 Conclusion and Future Work

Learning and summarising the computational specifics of CNN, GANs, and CycleGANs, as well as the structures and architectures of its generator and discriminator networks, implementing the whole CycleGANs algorithm from scratch on PyTorch, training the model with various hyperparameters, testing the learned model with the test collection, and conducting quantitative analysis on the positive and negative results were all part of this project's work.

Most of the outputs obtained were of reasonable quality showing that the model performed well however some of the output images involving the transformation from painting to photo were not very nice. One drawback to this model is that it generates pictures with a poor resolution. This is a work in progress, and a higher-resolution formulation based on a multi-scale generator model is in the works. Another problem with the model is that it struggles in matching colours perfectly. This is because, even if GXtoY and GYtoX change the colour of an image, the cycle consistency loss is likely to be minimal. One might add a new colour based loss concept that contrasts GYtoX(y) and Y, as well as GXtoY(x) and X, but that will be a supervised learning method. This unsupervised method also struggles with geometric shifts, such as adjusting the apparent scale of individual objects in an image, so stylistic transformations or translations that involve colour and texture are the best fit. Handling of more varied extreme transfigurations involving sharp geometric changes are a must to work on in the future. In this model, fixed kernel size was used, so it may perform better if use some kind of an inception network with different kernel sizes.

There is still a visible significant difference between the outputs of paired images compared to unpaired images resolution of which might require working on semantic vision. Integrating weak or semi-supervised data could result in far more efficient translators for a fraction of the cost of fully-supervised systems.
The link to the monet2photo.zip Dataset used: https://drive.google.com/file/d/1k3dzPtZJkZxY9flBzNxtfvKrj_B80Omi/view?usp=sharing
The link to colab notebook of the code: https://colab.research.google.com/drive/1A7Hx4VmDjPksWsTU88L0P8hJ9qXKkuxJ?usp=sharing

## 6 References

[1] Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks - Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros - https://arxiv.org/pdf/1703.10593

[2] Least Squares Generative Adversarial Networks - Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, Stephen Paul Smolley - https://arxiv.org/pdf/1611.04076

[3] Taesung park cyclegan dataset monet2photo - https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets/

[4] Generative Adversarial Networks - Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio - https://arxiv.org/pdf/1406.2661

[5] Nice Introduction with good example analogy- https://machinelearningmastery.com/what-is-cyclegan/

[6] Explained in an abstract way from the research paper - https://towardsdatascience.com/cyclegan-learning-to-translate-images-without-paired-training-data-5b4e93862c8d