Cyclic Generative Adversarial Networks for Photograph to Painting Translation REPORT

EE 541: A Computational Introduction to Deep Learning
Prof. Brandon Franzke
Aditya Anulekh Mantri (USC ID : 8049574464)
(**adityaan@usc.edu**)
Shoumik Nandi (USC ID : 3621442772)
(**shoumikn@usc.edu**)
Prithvi Dalal (USC ID : 1939114566)
(**prithvia@usc.edu**)

Abstract

Image to image translation is the process of discovering the mapping between the input and output images. In this project, we implement a Generative Adversarial Network GAN) that generates Monet-style images from photographs. Using adversarial loss, we aim to construct a mapping $G : X \rightarrow Y$ such that the distribution of Monet-style images from G(X) is indistinguishable from the distribution of the Monet painting. We combine this mapping with an inverse mapping $F : Y \rightarrow X$, introducing a cycle consistency loss in the form of $F(G(X)) \approx X$, which represents the conversion back to pictures. The same procedure is used when converting a Monet painting to a photograph and then back to a Monet painting. The primary purpose here is to generate as many Monet-style images as possible using the provided photographs.

Contents

1 Introduction

We aim to create images in the style of Monet from photographs. In the absence of paired examples of Monet paintings and photographs, we attempt to capture the unique qualities of the Monet paintings and determine how these characteristics might be translated into the collection of photographs, thereby transforming the photographs into Monet-style images. We make the assumption that there is some sort of relationship between the two domains (Monet paintings and Photographs), and we attempt to seek that relationship. We train a mapping $G : X \rightarrow Y$ such that the output $\hat{y} = G(x)$, $x \in X$, is indistinguishable from the paintings $y \in Y$ by an adversary trained to classify $\hat{y}$ apart from y. The translations G and F should be the inverses of each other, and both mappings should be bijections if we have a translator $G : X \rightarrow Y$ and another translator $F : Y \rightarrow X$. By training both the mapping G and the mapping F simultaneously and introducing a cycle consistency loss that encourages $F(G(x)) \approx x$ and $G(F(y)) \approx y$, we are able to test the structural assumptions. We can achieve our whole target for unpaired image-to-image translation by the combination of this loss with adversarial losses on domains X and Y.

2 Dataset

The dataset for this problem consists of two sets of unpaired images - Monet paintings and Photographs. The dataset posted on the Kaggle competition contained very few Monet styled paintings. Hence we used the dataset used by Zhu et al [4] in the original CycleGAN paper. This dataset can be found on Kaggle here.

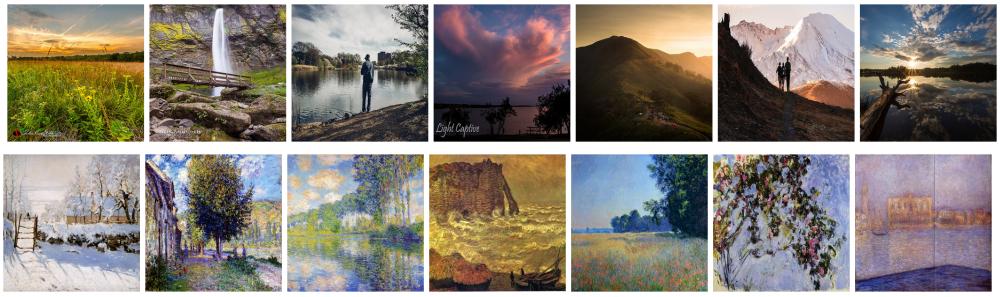

Figure 1: Sample images from the dataset. Top row: Images from the "photos" domain. Bottom row: Images from the "Monet-Paintings" domain.

3 Related Work

3.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a class of neural networks that are used to generate images that are indistinguishable from the original ones. They were first coined by I. Goodfellow in the paper titled *"Generative Adversarial Networks"* [2].

GANs have two components:

- **Generator Network** - Generates fake images based on a noise vector and
- **Discriminator Network** - Distinguishes between "real" from the dataset and "fake" images generated by generator

The generator is trained to fool the discriminator and the discriminator is trained to identify the fake samples generated by the generator. Thus they can be viewed as adversaries. The success of GANs can

be attributed to the complex adversarial loss that forces the images generated by the generator to be indistinguishable from the "real" photos. Details of loss functions are given in Section 4.2.

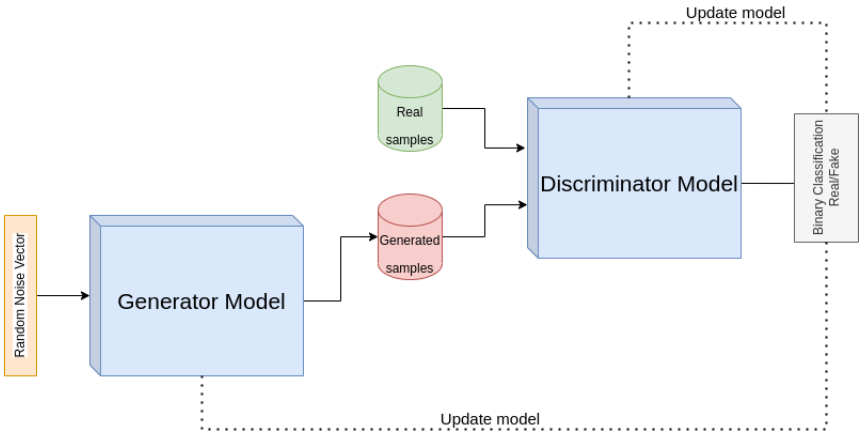

Figure 2: Simple GAN flowchart

3.2 Deep Convolution GANs

Deep Concolution GANs were first introduced by Radford et al [3] in a paper titled *Unsupervised Representation Learning With Deep Convolution Generative Adversarial Networks*. Although traditional GANs were used to generate images, they suffered from being noisy and incomprehensible. DC GANs proposed some major changes from previous architectures that are listed below:

- Replace any pooling layers with strided convolutions in the discriminator and transposed convolutions in the generator. Transposed convolutions can be viewed as normal convolutions with fractional stride. They are used for upsampling.
- Use Batch Normalization in both the generator and discriminator, except in the generator's output layer and the discriminator's input layer
- Replace fully connected layers with deeper architectures
- Use ReLU activation in the generator for all the layers except the output layer. The output layer uses tanh activation
- Activation function for all the layers in the discriminator should be leaky ReLU.

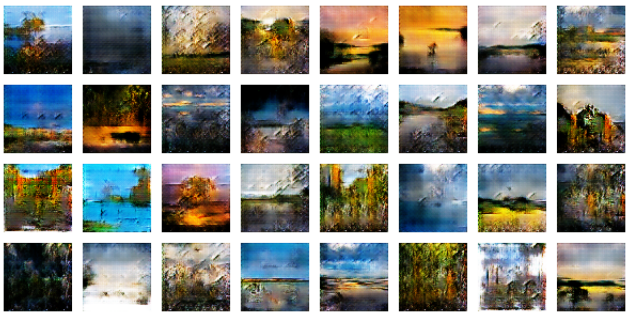

Figure 3: Sample outputs of a DC GAN trained on the monet paintings dataset. The model was trained with a random latent vector as an input. This is NOT style transfer or image translation.

4 Architecture

As it can be seen from the outputs of a simple DC GAN in figure 3 the images generated are very noisy and it is very difficult to make any sense of the images. Hence we use a architecture called CycleGAN introduced by Zhu et al in [4]. More details on the architecture of the generator, discriminator and the loss functions used in this architecture follow below.

4.1 Generator and Discriminator

The images in the Monet dataset and the photos dataset are not paired with each other. The goal of this project is to learn a mapping between the two domains given the unpaired training samples. To achieve this we need to construct two mappings $F : Photos \rightarrow Monets$ and $G : Monets \rightarrow Photos$. Hence, we use two discriminators, D_{Photo} and D_{Monet} to distinguish between real and generated images of photos and Monets respectively. We also use two generators, G_{Photo} and G_{Monet} to generated fake images of the respective class. The objective calls for multiple loss functions which are described later in the paper.

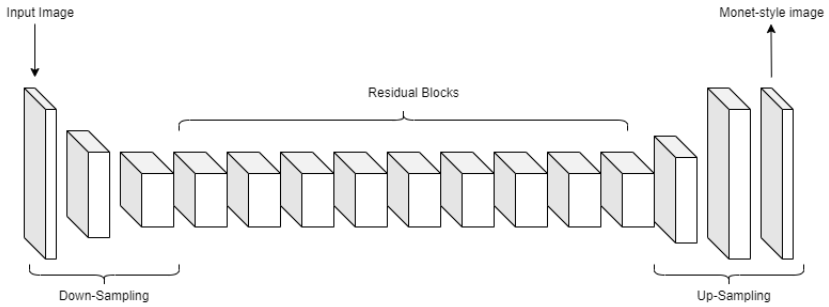

Figure 4: Architecture of the Generator

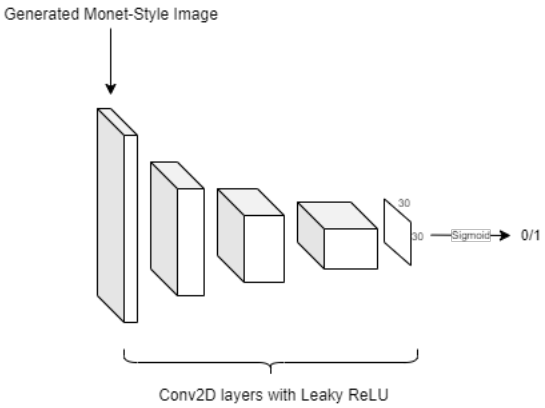

Figure 5: Architecture of the Discriminator

We adopt the architecture from Zhu et al. [4] who have shown impressive results for neural style transfer. The generator network shown in Figure 4 contains three convolutional blocks, nine residual blocks, two fractionally-strided convolutions with stride $= \frac{1}{2}$, and one last convolution that maps features to RGB. We have used 9 blocks of residuals for the high resolution 256 x 256 training images. Finally, for the discriminator network shown in Figure 5, we use 3 convolutions with Leaky ReLU essentially forming 70 x 70 PatchGAN's, which aim to classify whether 70 x 70 overlapping image patches are real or fake by passing

it via sigmoid layer in the end. Such a patch-level discriminator architecture has fewer parameters than a full-image discriminator and can work on arbitrarily sized images in a fully convolutional fashion.

Generator Architecture :
Let `c7s1-k` denote a 7 x 7 Convolution-InstanceNorm-ReLU layer with k filters and stride 1. `d-k` denotes a 3 x 3 Convolution-InstanceNorm-ReLU layer with k filters and stride 2. Reflection padding was used to reduce artifacts. `R-k` denotes a residual block containing two 3 x 3 convolutional layers with the same number of input and output filters. `u-k` denotes a 3 x 3 fractionally-strided-Convolution-InstanceNorm-ReLU layer with k filters and stride $\frac{1}{2}$.

The network consists of :
`c7s1-64` → `d-128` → `d-256` → `R-256` → `R-256` → `R-256` → `R-256` → `R-256` → `R-256` → `R-256` → `R-256` → `R-256` → `u-128` → `R-64` → `cs71-3`

Discriminator Architecture :
We use a 70 x 70 PatchGAN for this network. Let `C-k` denote a 4 x 4 Convolutional-InstanceNorm-LeakyReLU layer with k filters and stride 2. After the last layer, we apply a 1 dimensional fully-convolutional layer. We do not use InstanceNorm for the first `C-64` layer. We use Leaky ReLUs with the slope of 0.2.

The discriminator architecture consists of :
`C-64` → `C-128` → `C-256` → `C-512`

4.2 Loss Functions

GANs use very complex loss functions to train the generator and discriminator. In the case of a CycleGAN, we introduce additional loss functions to ensure robust translation from one domain to another.

4.2.1 Adversarial Loss

We define the discriminator's loss function here. Let $p_{data}(x)$ denote the distribution of real data and $p_{model}(x)$ denote the distribution of data from the generator (fake data). Then the cost function for the discriminator is:

$$\mathcal{L}^{(D)} = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{x\sim p_{model}(x)}[log(1 - D(G(y)))]$$

The loss will be the 0 if the discriminator results in 1 for all $x \sim p_{data}(x)$ and 0 otherwise. In the case of a zero-sum game, it is fairly easy to see that the loss function of the generator will be the negative of the above equation.

In the current implementation we replace the negative log likelihood by a least square loss as suggested by Zhu et al [4] to stabilize the model training procedure. This results in the following modified loss function:

$$\mathcal{L}^{(D)} = \mathbb{E}_{x\sim p_{data}(x)}[(D(x) - 1)^2] + \mathbb{E}_{x\sim p_{model}(x)}[(D(G(y)) - 1)^2]$$

4.2.2 Cycle Consistency Loss

Adversarial loss described above aids in learning the mapping from one domain to another. However it does not ensure that a fixed set of inputs produces the same set of outputs in the target domain. In order to solve this problem, we introduce a new loss function to ensure that the results are cycle consistent. This is equivalent to saying that a sentence translated from English to Spanish should be the same when translated back from Spanish to English. The cycle consistency loss function is described below:

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

Here F and G represent the generators used for domain translation. The function of cycle consistency loss can be observed in Figure 6.

4.2.3 Identity Loss

This loss is specific for photo $\leftrightarrow$ paintings translation. This loss helps preserve the color space between the input and output. Without $\mathcal{L}_{identity}$ the generators bound to change colors when there is no need to. This phenomenon can be observed in Figure 7. We used L_1 loss on x and $G(x)$ to implement identity mapping loss in the model.

Figure 6: The above figure represents photos translated to the Monet painting domain and then translated back to the photo domain. The input images x, the output images $G(x)$ and the reconstructed images $F(G(x))$.

Figure 7: Left to Right: Input Photo, Monet style painting using CycleGAN without Identity Loss, Monet style painting using CycleGAN with Identity Loss. Identity mapping loss helps preserve the color space of the input photograph.

5 Training

5.1 Parameters

Parameter	Values
Optimizer	Adam
Learning Rate Generators	10^{-5}
Learning Rate Discriminators	10^{-5}
Batch Size	1
Betas for Adam Optimizer	$\beta_0 = 0.5 \beta_1 = 0.999$
Number of GPUs	1
Discriminator steps per generator	1
Weight Initialization	$\mathcal{N}(0, 0.02)$
Normalization	Instance Normalization [4]
Cycle Consistency Loss - λ	10
Identity Loss	$0.5 * \lambda$
Input Dimensions	$3X256X256$
Output Dimensions	$3X256X256$

5.2 Training Details

We started by overfitting a small batch of data (300 Photos and 300 Monet Paintings) to make sure the architecture is able to learn the mapping from the Photos domain to the Monet domain and vice versa. We had some challenges since the dataset we were working had a very small number of Monet paintings as compared to the Photos. To tackle this challenge we used the dataset that was originally used in the reference paper.

We trained the model without identity loss for the initial 30 epochs to accelerate the learning process. Beyond 30 epochs we also incorporated identity loss above the existing mean square loss and cycle consistency loss. The training was performed on a Tesla V100 GPU. The results described in this report are based on the training results using this hardware.

The statistics of the training are described below

Parameter	Description
GPU	Tesla V100
GPU Memory	16GB
Number of CPU cores	8
System RAM	64GB
Time per epoch (W/o Identity Loss)	15 minutes
Time per epoch (W/ Identity Loss)	20 minutes
Cost per hour	\$0.8 per hour (Spot Instance pricing)
Cost for 100 epochs	\$30
Time to train	34 hours

Figure 8: Monet paintings generated through epochs in steps of 5. The first image in the grid is the input photo. As it can be seen in the image the initial suffer from a change of color space due to the lack of identity mapping loss in the training. After adding identity mapping loss, it can be observed that the color space of the input image was preserved.

6 Results

Successfully translated Monet-Paintings from Photos

Figure 9: The left photo in each pair is the input fed to the Monet painting generating network and the right image is the output of the generator. These are some of the images that could be translated to the Monet domain from the photos domain.

Failed translations of Monet-Paintings from Photos

We observed that images containing snow or ice could not be translated well as Monet paintings. This could be because such images were under represented in the dataset containing Monet paintings. As it can be seen from the example images below the generated paintings do not show properties similar to that of a Monet painting.

Figure 10: The left photo in each pair is the input fed to the Monet painting generating network and the right image is the output of the generator. These are some of the images that could NOT be translated to the Monet domain from the photos domain.

Successfully translated Photos to Monet-Paintings

Figure 11: The left photo in each pair is the input fed to the Photo generating network and the right image is the output of the generator. These are some of the images that could be translated to the photographs domain from the Monet paintings domain.

7 Challenges

- **Hardware Limitations** - CycleGAN consists of two generators and two discriminators. Although the size of the discriminators was only 55MB, the generators were close to 1.2GB in size. These models were occupying a good chunk of the GPUs memory. This meant that we had to train with a reduced batch size.
- **Imbalanced dataset** - Although we used the dataset from the original publication, it still had significantly lower number of Monet-paintings as compared to the photographs. This meant that we had to use each painting close to 6 times more than a photograph.
- It is extremely important to constantly monitor the progress of the networks as the training progresses. If the discriminator is trained very well, then it rejects all the samples that the generator generates and the system will never reach equilibrium. If the generator is trained very well as compared to the discriminator, it ends up fooling the discriminator very easily and this results in subpar images being generated.

8 Future Work

- Generate images of higher resolution
- Explore the influence of using pre-trained models as feature extractors for generator and discriminator

9 Applications

CycleGANs have applications in various domains. Some of which are listed below

- Recreating art by famous artists (Application discussed in this paper)
- Creating colored versions of legacy photographs

More applications of CycleGANs can be found on the website of Zhu et al.

10 Conclusion

In this project we developed a model using an unpaired dataset to translate a photograph to a Monet-painting and vice-versa. We studied the various methods in developing generative models and also implemented two of them - DC GAN and CycleGAN.

The code and setup instructions for this project can be found on GitHub here

11 Evaluation

Apart from the difficulties in training GANs, evaluating GANs is also an equally difficult task. The reason being that there are no fixed results that come out of GANs unlike in the case of some other deep learning tasks. As we have seen in this report, generators do not learn using a loss function, instead they learn by fooling the discriminator.

For the kaggle competition in context, the images generated are being evaluated using two metrics *Frèchet Inception Distance* and *Memorization-informed FID*. The details of both are described below

11.1 Frèchet Inception Distance

In FID, we use the Inception v3 network to extract features from an intermediate layer. These activations are calculated for collections of real and generated images and modelled as multivariate Gaussians with mean μ and covariance Σ. The FID between the real and generated distributions is computed as:

$$\text{FID} = ||\mu_r - \mu_g||^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2})$$

Where Tr represents the trace of the matrix.

11.2 Memory-informed FID

The memorization distance is defined as the minimum cosine distance of all samples in the training set averaged across all generated samples. MiFID is calculated as follows:

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

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.

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

d is the distance between the generated image across all the real images. d is thresholded as:

$$f(x) = \begin{cases} d, & \text{if } d < \epsilon \\ 1, & \text{otherwise} \end{cases}$$

Finally,

$$MiFID = FID \cdot \frac{1}{d_{thr}}$$

Evaluation of our model: We generated close to 8500 images and submitted them to Kaggle. Below are the results

Model	Score
CycleGAN without Identity Loss	70.09929
CycleGAN with Identity Loss	56.38486

References

[1] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, 2019.

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[3] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[4] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.