Deep Learning- Final Course Project

Ido Natan, Doron Nevo

Submitted as final project report for the DL course, BIU, 2021

1 Introduction

Image-to-image translation involves generating a new synthetic version of a given image with a specific modification, such as translating a summer landscape to winter. Training a model for image-to-image translation typically requires a large dataset of paired examples. These datasets can be difficult and expensive to prepare, and in some cases impossible, such as photographs of paintings by an artist who lived more than a 100 years ago. The CycleGAN is a technique that involves the automatic training of image-to-image translation models without paired examples. The models are trained in an unsupervised manner using a collection of images from the source and target domain that do not need to be related in any way. It consists of at least two neural networks: a generator model and a discriminator model. The generator is a neural network that creates " fake" images. The two models will work against each other, with the generator trying to trick the discriminator, and the discriminator trying to accurately classify the real vs. generated images. This simple technique is powerful, achieving visually impressive results on a range of application domains.

In this report we will discuss our modifications to this original CycleGAN implementation to achieve visually convincing Monet style drawings which were never painted by him and consequentially as low MiFID score as possible with only 30 original Monet paintings.

1.1 Related Works

- CycleGAN original Image-to-image translation paper the original CycleGAN paper – served as a baseline for our project
- CycleGAN and pix2pix repository the GitHub repository with implementation by the authors of the original paper.
- CycleGAN Face-of, X. Jin and all, 2018 article that compares CycleGAN networks architecture options (ResNet and U-NET).
- Super Resolution Convolutional Neural Network article that we used for image resolution increase as part of data augmentation.
- DL Lecture 9 Generation - PixelRNN PixelCNN VAE.pdf tips to make GANs work that were provided in lecture 9 of the course (slides 235 – 239).
- CycleGAN Kaggle Notebook a Kaggle notebook that provided Pytorch implementation ideas.

2 Solution

2.1 General approach

Our approach to solve the problem:

1. Model architecture selection: we identified on 2 alternatives for Generator network architecture:
    a. ResNet – as described in original CycleGAN paper.
    b. U-NET – an architecture used mostly for image segmentation and Pix2Pix.

    We decided to focus on the ResNet option, as research literatures (ex. CycleGAN Face Off and original paper author) are indicating the U_NET poor results due to mode collapse.

2. DNNs, and especially GANs are challenging to debug, therefore we decided to implement in a phased approach:
    a. Implementation of plain vanilla CycleGAN as described in the original paper with the full 300 Monet picture. First, we visually inspected some samples to confirm seen Monetstyle results, and afterwards verifying Kaggle score (target: <60).
    b. Implementation of the model with baseline of 30 Monet pictures while iterating with different30 pictures subsets and augmentation techniques and reviewing achieved scores. First, by randomly selecting the pictures and using naïve augmentation techniques (random crop, rotation, flip). We ended up with a specialized subset selection and augmentation method (described in next session).
    c. Iterating on model architecture and hyperparameters tweaking – this included:
        i. Increasing the number of residual blocks – resulted in improvement
        ii. Adding soft labels – improvement noticed
        iii. Leaky ReLU instead of ReLU iv. Adding skip-connects between encoder and decoder blocks of the Generator
        v. Reducing the weight (λ) of the Identity loss. vi. Increasing number of epochs.

2.2 Monet subset selection and data augmentation

Monet subset (30 pictures) selection: after several trials with random selection, we decided to proceed with a manual selection. We eliminated and selected representative 30 pictures following information retrieved from Wikipedia regarding Monet:

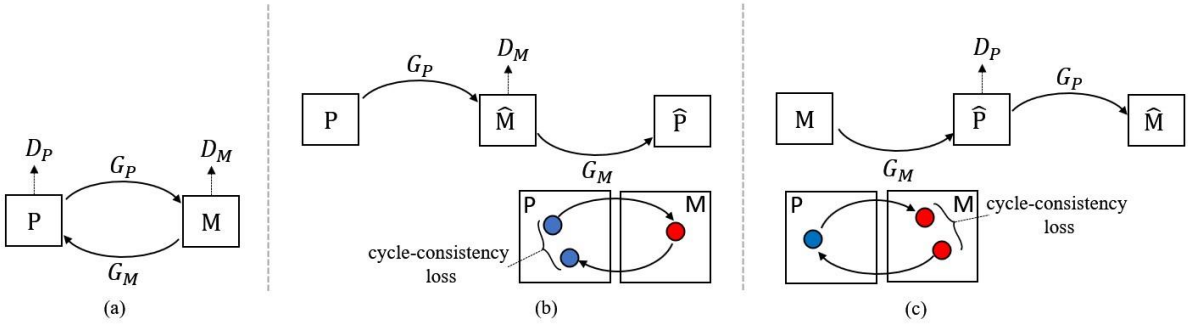
1. Monet is mostly famous for his landscape's paintings. I.e., indoors, and urban painting should be screened out.
2. Water lilies were his signature pictures (he made about 250) – they should be well represented in the subset.
3. He used to paint same scene several times, with different lightning conditions – meaning augmentation can be used to imitate it.
4. He was the founder of impressionism, characterized by relatively small, thin, yet visible brush strokes – meaning painting without brush strokes can be screened out.

Monet subset augmentation from 30 to 300 pictures:

1. As described above, we have manually selected 30 images highly variated one from the other: various painting styles, scenes, and coloring.
2. We created 30 new images out by applying a super-resolution algorithm (256x256 ➐ 512x512) based on "C. Dong and others "Image Super-Resolution Using Deep Convolutional Networks".
3. We wrote a preprocessing program using **`skimage match_histograms`** that enable color palette transfer:

2.3 Design

2.3.1. GAN Architecture



(a) Our model contains 2 Generators: G_P: $P(\text{Photo}) \rightarrow M(\text{Monet})$ and G_M: $M(\text{Monet}) \rightarrow P(\text{Photo})$, and associated adversarial discriminators $D_P$and D_M. D_M encourages G_P to translate P into outputs indistinguishable from domain M , and vice versa.

(b) The forward cycle-consistency loss: $P \rightarrow G_P(P) \rightarrow G_M(G_P(P)) \approx P$.

(c) Backward cycle-consistency loss: $M \rightarrow G_M(M) \rightarrow G_P(G_M(M)) \approx M$.

<u>The loss functions to be used:</u>

1. Identity loss: $\mathcal{L}_{identity}(G_P, G_M) = \mathbb{E}_{P\sim p_{data}(P)}[\|G_M(P) - P\|_1] + \mathbb{E}_{M\sim p_{data}(M)}[\|G_P(M) - M\|_1]$

   As mentioned in the paper: without Identity loss, the generator G and F are free to change the tint of input images when there is no need to. For example, when learning the mapping between Monet's paintings and Flickr photographs, the generator often maps paintings of daytime to photographs taken during sunset, because such a mapping may be equally valid under the adversarial loss and cycle consistency loss.

2. Cycle consistency loss: $\mathcal{L}_{cyc}(G_P, G_M) = \mathbb{E}_{P\sim p_{data}(P)}[\|G_M(G_P(P)) - P\|_1] +$

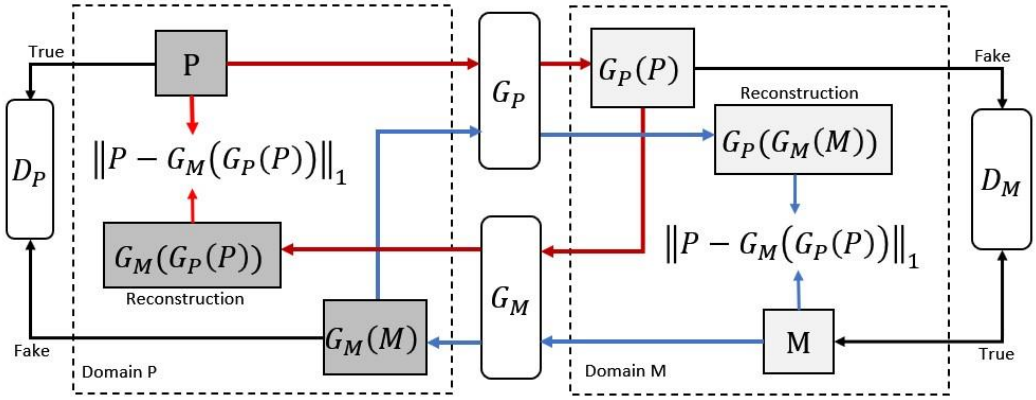
   $\mathbb{E}_{M\sim p_{data}(M)}[\|G_P(G_M(M)) - M\|_1]$

   Cycle consistency loss compares an input photo to the Cycle GAN, to the generated photo and calculates the difference between the two: e.g., using the L_1norm or summed absolute difference in pixel values.

3. Adversarial loss: $\mathcal{L}_{GAN}(G_P, D_M, P, M) = \mathbb{E}_{M\sim p_{data}(M)}[\log D_M(M)] + \mathbb{E}_{P\sim p_{data}(P)}[\log(1 - D_M(G_P(P))]$

4. Total loss: $\mathcal{L}(G_P, G_M, D_P, D_M) = \mathcal{L}_{GAN}(G_P, D_M, P, M) + \mathcal{L}_{GAN}(G_M, D_P, P, M) + \lambda\mathcal{L}_{cyc}(G_P, G_M)$

The process flow is described in the following figure:



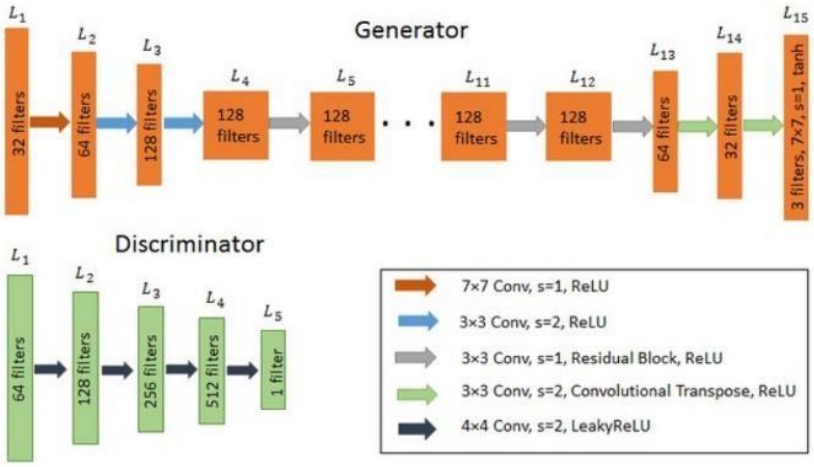
2.3.2. Network Architecture

Generators: we adopted the architecture of the CycleGAN paper:

- We define a residual block class with reflection pad and dropout of 0.5 probability.
- We first tried using 6 residual blocks but found 9 blocks providing better visual results and better MiFID scores.
- The 9 residual blocks are in the middle of 3 layers of convolution starting from 3 channels to 64 then 128 finally to 256, and 2 layers of 2d convolution transpose starting from 256 channels down to 64 channels which then, to get 3 channels for RGB we apply a final 2d convolution.
- We use ReLU as the activation between each convolution layer same between convolution transpose layers. Finally, we apply Tanh.

Discriminators:

- we have implemented 5 layers of 2d convolution.
- Initially we used ReLU as activation function, and later found Leaky ReLU with slope=0.2 providing better results.



2.3.3. Training and experiments details

- We train the Generators and Discriminators from scratch using learning rate of 0.0002.
- We selected the Adam optimizer as it was mentioned in the papers, we read to be the most effective for GANs
- The weight of set cycle consistency loss: $\lambda = 10$. Identity mapping loss: we tried several values between 0 to 0.9. Without identity loss (0.0λ), the MiFID was close to two times higher than with the loss. We found the optimal value to be 0.5λ.
- We have tried additional online augmentations (random crops, rotations, and flips) – the added value was marginal.
- We tried using soft labels: 0.0 ..0.3 for fake and 0.7..1.0 for real. We saw improvement with best results when using 0.1 for fake and 0.9 for the real.
- We tried adding skip connections in the Generators $L_2 \rightarrow L_{14}$, $L_3 \rightarrow L_{13}$. It did not provide better results
- Number of epochs: we tried several combinations and found 40 epochs to provide good results with reasonable training time (below 2 hours).

2.3.4. Code and implementation platforms

- We selected Pytorch infrastructure as we are familiar with it.
- Coding principles: minimal clean coding with high attention on naming convention to increase readability and maintainability.
- Google Colab Pro – used for development as it provides unlimited session times and upgraded computational resources (GPU and RAM).
- Kaggle – used for submission and assessment of scores (MiFID).

2.3.5. Technical challenges

- Kaggle is not the most user-friendly platform, to say the least - we had some "CUDA OUT OF MEMORY" days and we could not find the problem. So, we had to downgrade our code to previous version and progress from there. To this day we don't know what caused the problem.

3 Experimental results

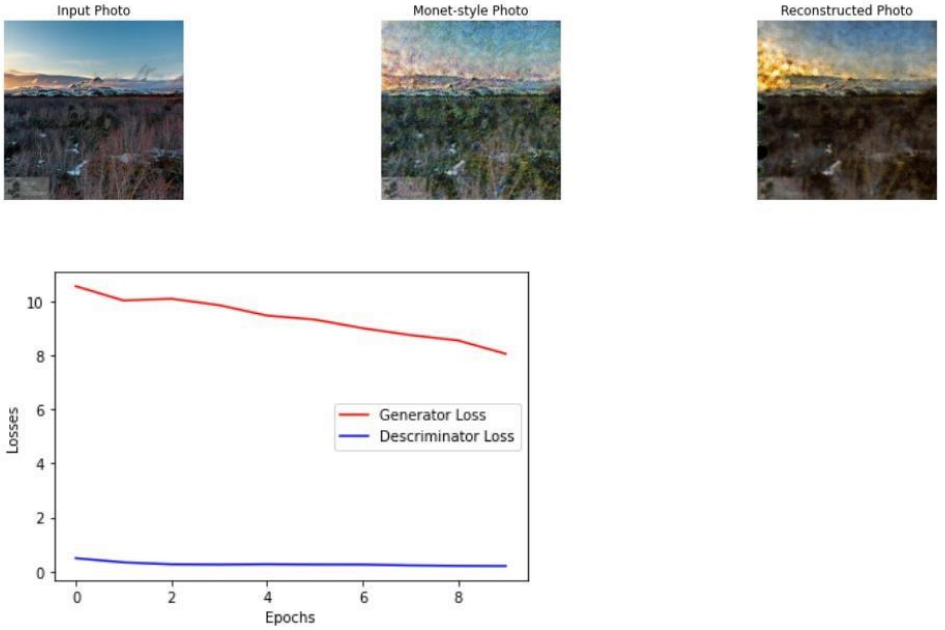
- This section contains two parts: The structure of the model and its hyper parameters, and the data augmentations. That said, we did add comparison between ReLU vs LeakyReLU in data augmentations stage.
- Due to lack of time, we had first tested model structure with no augmentation, once we had found the optimal solution with data that has no augmentation we progressed to the second stage – augmentations.

3.1 First stage: model structure

Train Generator together with Discriminator	Include Identity Loss	ReLU or Leaky ReLU	Epochs	Generator Loss	Discriminator Loss	Kaggle's MiFID

No	Yes	ReLU	10	7.158	0.086	63.86
No	No	ReLU	10	6.213	0.199	124.85
No	Yes	ReLU	10	6.206	0.014	73.62
Yes	Yes	ReLU	10	6.173	0.013	74.23
No	Yes	Leaky ReLU	10	8.047	0.202	57.23

Best graph in terms of MiFID of all combinations:

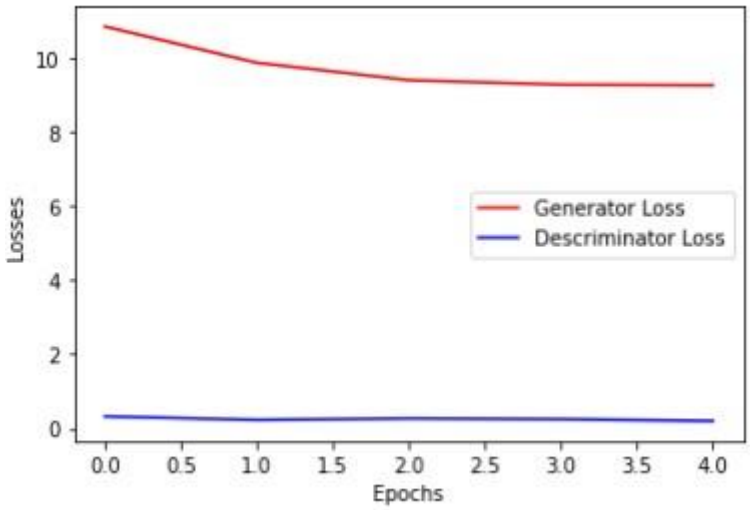


3.2 Second stage: data augmentation

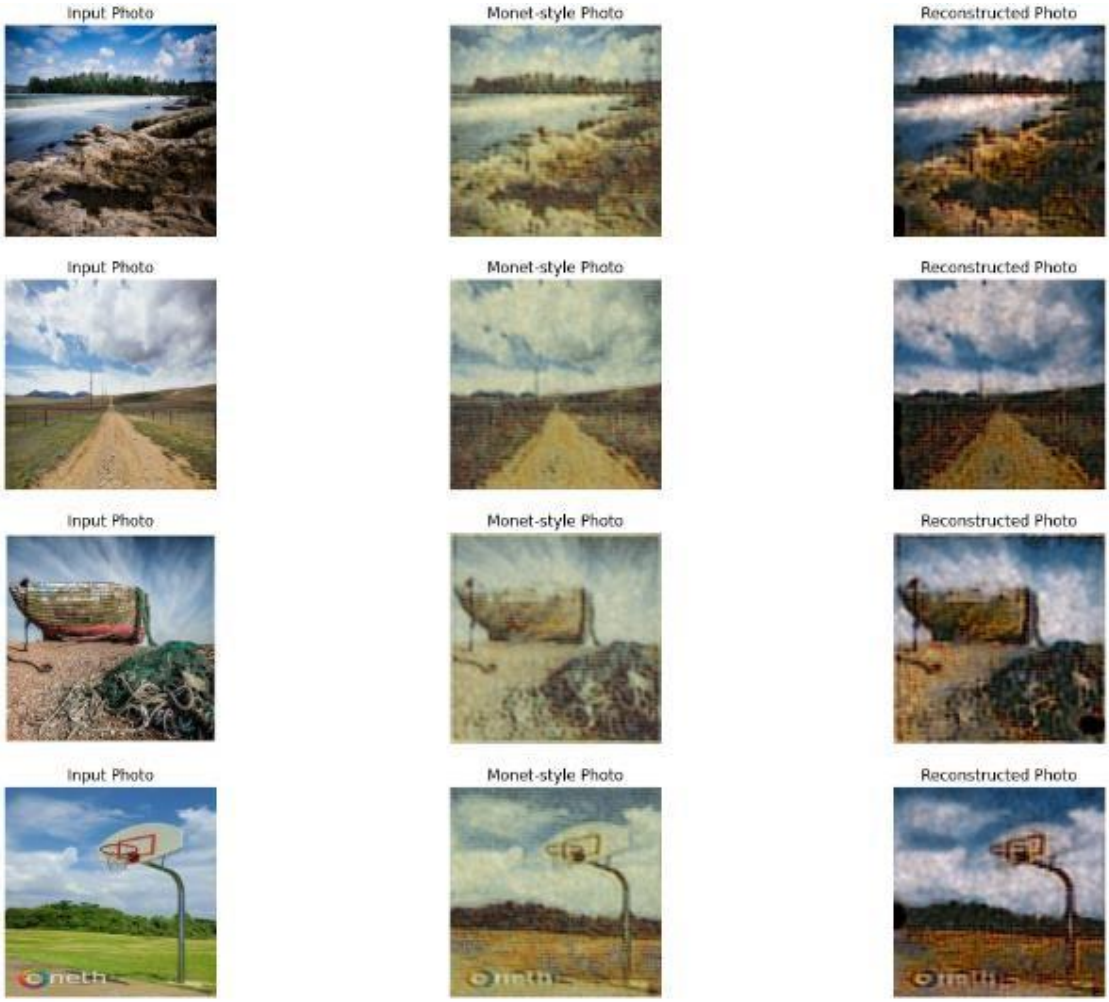
Monet random Augmentations	ReLU or LeakyReLU	Soft labels	IDT	Skip connect	Epochs	Gen. Loss	Disc. Loss	Kaggle's MiFID
No	ReLU	No	0.5	No	30	7.158	0.086	57.55
No	LeakyReLU	No	0.5	No	30	7.088	0.081	61.71
Rotate & crop	LeakyReLU	No	0.5	No	30	7.123	0.097	57.74
No	LeakyReLU	No	0.7	No	30	7.750	0.110	64.66
No	LeakyReLU	No	0.3	No	30	8.050	0.120	65.50
No	LeakyReLU	0.0/0.9	0.5	No	30	6.974	0.138	65.80
No	LeakyReLU	0.1/0.9	0.5	No	30	7.165	0.063	58.20
No	**LeakyReLU**	**0.1/0.9**	**0.5**	**No**	**40**	**6.300**	**0.049**	**55.30 (*)**
No	LeakyReLU	0.1/0.9	0.5	No	60	6.250	0.044	55.52
No	LeakyReLU	0.1/0.9	0.5	Yes	40	6.560	0.028	56.94

(*) Best score

Best graph of all combinations:



Best couple of generated Monet paintings:



4 Discussion

- In this project we implemented CycleGAN with small number of example limitation. We gained practical experience with model training and GANs hyperparameters tuning.
- The implementation involved different measures and techniques to improve results.
- We established that GANs model "babysitting" is a challenging task, because GANs training process is somewhat unstable and unpredictable and prone to bugs difficult to isolated. Therefore, it requires a phased approach and consultation with relevant literature is highly recommended.
- We have learned to avoid sparse gradients – to use LeakyRelu instead of the hard strict ReLU, do soft labeling and label smoothing to improve generated paintings, visually and MIFD.
- We have learned to add noise to the inputs – randomly we give the discriminator previously generated Monet images (from a generator weaker than it's current state) thus we confuse the generator as to why the discriminator managed to expose the fraud. This makes the generator work and learn even harder.
- We learned the importance of domain expertise as described about the process of training subset selection.
- The Deep Learning course has proved itself as very helpful to accomplish the project.

Future steps:

- Include in the code an FID / MiFID evaluation (not included to lack of time) – this may enable usage of early stopping in the training cycle.
- Explore possibility to add transfer learning to the process.
- Migration to TPU and using minibatches – this will enable us to explore additional possibilities that required more training power (additional epochs).
- Modify the loss function to incorporate better correlation to visual appearance results and to FID metric.

5 Code

- The inference notebook ○ Note: during inference, you will be requested to download a ZIP file containing 2-5 JPG images (256x256 pixels). We will provide an example ZIP file as part of our submission.
- The training notebook