# Deep Learning- Final Course Project

Nerya Aberdam (ID. 311416457),Gadi Didi ID. 208064840)

CycleGAN project report for the DL course, BIU, 2021

## 1 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 using a training set of aligned image pairs. However, for many tasks, paired training data will not be available. We present an approach for learning to translate an image from a source domain X to a target domain Y in the absence of paired examples. Our goal is to learn a mapping G: X $\rightarrow$ Y, such that the distribution of images from G(X) is indistinguishable from the distribution Y using an adversarial loss. Because this mapping is highly under-constrained, we couple it with an inverse mapping F: Y $\rightarrow$ X and introduce a cycle consistency loss to push F(G(X)) X (and vice versa). Quantitative comparisons against several prior methods demonstrate the superiority of our approach.

Our goal is to build a CGAN that generates 7,038 Monet-style images with only 30 monet's paints. Since we only had to work with thirty images, we had to use special ways to optimize the familiar model.

### 1.1 Related Works

Please provide some references to existing works, if you found any. it's an optional sub section, but can definitely help.

## 2 Solution

### 2.1 General approach

As we mentioned in the introduction, since we only need to use 30 images, we had to use more efficient ways to achieve this goal. Initially, we were looking for existing models for monet's drawings and wanted to learn the best from each one, so we came up with ideas for our solution.

The GAN architecture is an approach to training a model that is comprised of two models: a generator model and a discriminator model. The generator takes a point from a latent space as input and generates new plausible images

from the domain, and the discriminator takes an image as input and predicts whether it is real (from a data set) or fake (generated). Both models are trained in a game, such that the generator is updated to better fool the discriminator and the discriminator is updated to better detect generated images.

The CycleGAN is an extension of the GAN architecture that involves the simultaneous training of two generator models and two discriminator models.

One generator takes images from the first domain as input and outputs images for the second domain, and the other generator takes images from the second domain as input and generates images for the first domain. Discriminator models are then used to determine how plausible the generated images are and update the generator models accordingly.

## 2.2 Design

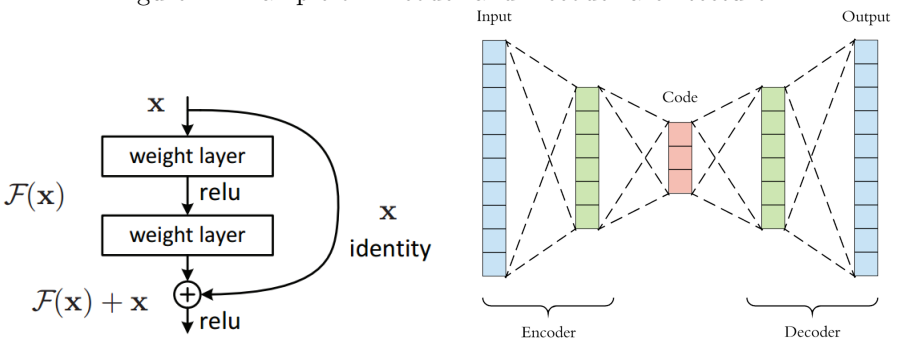
we used PyTorch framework. As we mentioned above our model consists of 2 generators. The first produces paintings by Mont (Domain A), and the second produces photographs (Domain B).

### 2.2.1 Generator

We created 2 generators, one for each domain. But they both have the same architecture. The generator starts from a noise image performing encoding for it and then decoding. We added Residual blocks. We came to see that a generator without Residual blocks does not bring good results. We set up 9 Residual blocks for the generator.

Figure 1: Example of Residual block

Figure 2: Example of Encoder and Decoder architecture




### 2.2.2 Discriminator

We created 2 Discriminators. Both with the same architecture. The first for domain A (fake photo and fake ID) and the second for domain B (fake photo

and photo ID). The networks are based on the CNN model. And then switch to the fully connected network.

#### 2.2.3 The Training Process

First, we trained the model about 10 epochs. We have seen that there is indeed an improvement in output from epoch to epoch. We tried to train about 30 epochs and saw that the improvement continued to grow. We used batches the size of 5 items per batch. Second, the learning rate is 0.2. We also used "ADAM" for optimizing (according to recommendations we saw on Internet) We trained the model as follows:

1. Training for the generators. Then calculate the loss and update the weights of the generators.
2. Training the discriminators, calculating their losses and then updating their weights.

#### 2.2.4 Loss Functions

Our loss consists of several factors:

1. **Generator Loss**:
   - **loss Identity**- Measures the mean absolute error (MAE) between each element in the input $x$ and target $y$.
     This is the loss_Identity_Photo:
     x = the output from `Generator_photo_to_monet(real_Monet)` function
     y = `real_Monet` photo
     same as loss_Identity_Monet
     to sum: identity loss = `loss_identity = (loss_Identity_Monet + loss_Identity_Photo)/2`
   - **GAN Loss** - We will mention this below
   - **Cycle Loss** - We focus on Monet Generator, but is same for the second. We get photo and generate fake Monet paint. Now we use Photo Generator with the fake paint we made. We get recovery photo. we check $G_{Photo}(G_{Monet}(x)) == x?$. We compare pixel with Measures the mean absolute error (MAE) method.

   Finally, the total loss (for both Generators) is:
   `loss_G = loss_GAN + (10.0*loss_cycle) + (5.0*loss_identity)`
2. **Discriminator Loss**:
   - **GAN Loss**- Creates a criterion that measures the mean squared error (squared L2 norm) between each element in the input $x$ and target $y$.

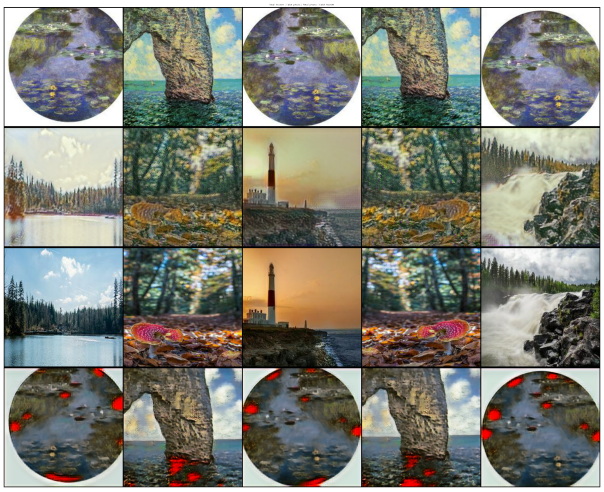
In fact we send pairs of image and label (fake/ real) to Monet Discriminator and to Photo Discriminator. We send 2 batch per Discriminator. First batch is real (Photo or paint, depended by the model) and we want to get label=1. The second batch is fake and we hope to get label=0 for all the images in this batch. we make average loss for those two batch (per model).

### 2.2.5 Data augmentation

Our data set contain only 30 Monet paints. so we use Data augmentation technique for create more data by what we have. For example, random crop, resize or add some noise to the picture.

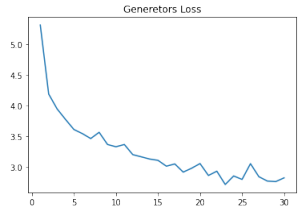
# 3 Experimental results

The result:



As you can see, the conversion from photo to Monet (our goal) is good. But the opposite, conversion from Monet to photo not so good. We tried to change the learning rate, changing the number of epochs, none of these helped.

The Generator loss:



# 4 Discussion

We found that GAN architecture can be implemented in several ways. We tried to realize it in a few different loss pancakes. The product on display is the best we have been able to achieve based on only 30 drawings. The lines that guided us are:

1. Produce a good enough amount of data from only 30 images
2. Select the most appropriate loss function

The rest of the things (such as choosing a learning rate or determining the number of epochs) we performed by trial and error, and taking the most powerful model we were able to produce.

# 5 Code

There is the colab link in the github link below:
https://github.com/gadididi/DL–BIU—2021.git

# References


[1] Jonathan Hui. *GAN — CycleGAN*. Jun 15, 2018.

[2] Erik Linder-Norén. *ML engineer at Apple*
`https://github.com/eriklindernoren`