Generating Monet's Style Images using GAN Architecture

Ibrahim Tolga Ozturk, Nika Nizharadze, Nikita Volkov,
Qureshi Asma, Temirlan Turysbek, Valari Pai
Ludwig-Maximilians-Universität (LMU)
MSc Data Science, Summer 2021

***Abstract*—A GAN model is developed to automatically generate Monet style images from photos. The generator is trained using a discriminator. Since machines have limited contextual performance, we plan to improve upon the discrimination performance by incorporating a human component for a better visual judgement. Human inputs on the similarity to Monet painting, strokes, colours, and style of the image are collected in a gamified fashion to motivate user participation.**

I. Introduction

Computer vision and image processing have significantly developed in recent years and are used in a plethora of applications. Several main directions of elaboration and application of computer vision could be highlighted, including, but not limited to, image recognition, analysis, and processing.

This work focuses on the task of implementing image processing, based on a Kaggle competition titled "I'm something of a Painter Myself". This competition poses a question if data science approaches can be used to trick art lovers, and help them to think out of the box in terms of arts of famous artists. Precisely speaking, the goal of the competition is to use a suitable machine learning pipeline to train a network that could eventually be used for generating (reprocessing) images in Monet style.

For this challenge, a generative adversarial network (GAN) is used for altering arbitrary input images to match the style of Monet paintings. Thus, it will will be challenging, at the first glance, to discover whether an output is indeed painting of the artist or not. Our model is comprised of two parts: **generator** and **discriminator**. The generator is trained using a discriminator.

A secondary goal is to add a human component for discrimination of the images, in addition to the machine discriminator. The motivation of incorporating the human inputs is threefold: (i) extending the dataset; (ii) making images visually more discernible; and (iii) collecting contextual data regarding strokes, colours, and style of the generated images to help improve the performance of the system. Thus, we let users share their inputs, regarding the accuracy of the generated images, in a subjective fashion, besides yes/no answers of whether an image is Monet or not.

As with any human competition system, users are incentivized to participate in the improvement of the training using the following:

- Giving the opportunity for users to participate in the training in a gamified way;
- Motivating users in a way that they can use the engineered application to test their knowledge of Monet's style;
- Allowing various community-based interactions and competitions;
- Providing star-ratings and store rewards for users for stimulating users' activity in the application.

II. Related Works

Firstly the term "Generative Adversarial Networks" was introduced in 2014 in the scientific paper of Ian Goodfellow et. al [1]. Since that time the field of GANs development has expanded. In this paper CycleGAN is considered as the main approach for solving the defined problem, which already showed the efficiency in other similar tasks [2]. Still there are also other implementations of GANs that can be reviewed.

Thus, PixelRNN [5] is an instance of auto-regressive Generative model. GANs learn implicit probability distribution. In comparison with GANs, auto-regressive models, in their turn, learn explicit data distribution. There is an implementation of GANs as StyleGAN [4] where the idea is a construction of a stack of layers so that base layers generate low-resolution pictures and subsequent layers consistently improve the resolution.

Indeed intriguing example of GAN performance is presented in DiscoGAN which allows to define cross-domain relations on unsupervised data [7]. As for people it is quite easily to understand for a pair of domains what is what, what is the pictures of shoes or bags, the implementation of this type of GANs is challenging. Also entertaining GAN construction is presented in a paper of S. Reed et. al [6]. The model receives as an input textual description of the image which contains various photos of birds and flowers and then generates an image with it's features that was defined in the textual description before.

To sum up, all of the mentioned GAN models have Discriminator and Generator that try to cheat each other and follow the approach of adversarial loss. Applications are diverse, starting with generation of modified photos with special effect and ending with generation of the image basing the input text.

III. Data Types and Preprocessing

In this section, we described the data types, their formats, and pre-processing pipeline.

A. Data Type and Format

Data are digital three-channel JPEG-encoded RGB images. Each channel is a $2D$ matrix, which is resized to have a uniform size of 256x256. The data are divided into two sets of images before training:

1) Set 1 comprises a total of 300 original Monet paintings, which are provided by the competition. This set is used for training the model.
2) Set 2 comprises about 7000 sample photos. The **learnt** Monet-style is applied to these images, and results are subsequently used for evaluation.

Fig. 1 shows histograms of three sample images. The histograms are obtained by adding the three color channels and then normalizing them.

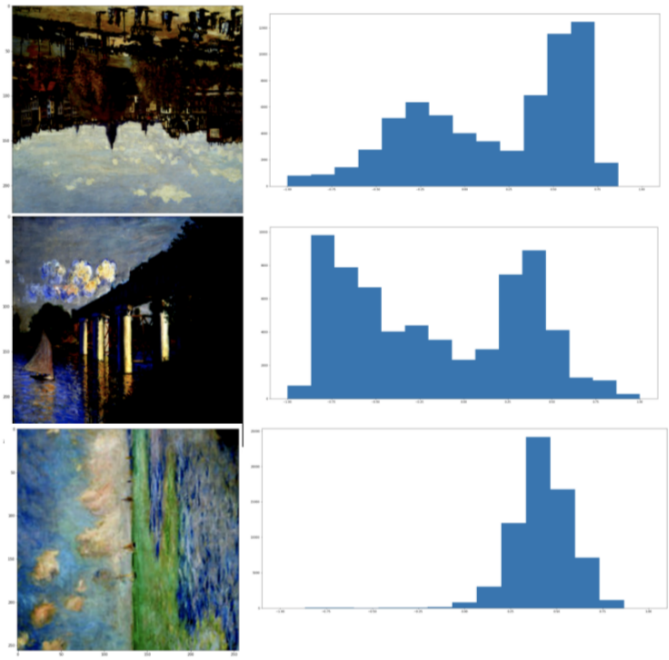

Fig. 1: Sample images and their histograms are shown. The histograms are obtained from the normalizing sum of three color (RGB) channels.

B. Software Libraries

As a base language for analysis of the data, including data preprocessing, creation, model estimation and results visualization, Python language, with its widely known libraries, is used. For example, PyTorch is utilized to implement the described CycleGAN model. Numpy is used to perform algebraic operations on matrices and standard auxiliary operations. Pandas is employed to clean and preprocess the dataset. For visualization purposes, Matplotlib and Seaborn packages are used. Lastly, the collaborative interface design tool **Figma** is used for creating UI interface demo, which will be used as a basement for developing the application to get user's inputs for subsequent model improvement.

Our preprocessing pipeline consisted of two stages, namely, data preprocessing and augmentation, each of which are described below.

C. Data Preprocessing

The data are preprocessed before training. This is to ensure that input are normalized and cropped to a uniform same size and are ready for the training. We also converted the images into TensorFlow supported TFRecord format, which is a simple format way of storing records such as the image data. TFRecord is also considered as a more efficient storage container since not only it saves data using the less space than the original data but also allows them to be partitioned into multiple files. We converted RGB images into uint8 tensors, to support processing of the image data into the TensorFlow pipeline.

D. Data Augmentation

The purpose of data augmentation is two-fold. First, we want to extend our image database to include modified versions of the provided images. Secondly, we also want to make our training more robust by using variations of the input images. We performed two of the most common data augmentation tools, namely, image rotation and random mirroring.

IV. Model Description

A. GAN Architecture

The goal of the generative adversarial network is to generate new samples of data that would, ideally, have the same distribution as the training set. In order to achieve this, the GAN architecture uses two distinct networks Generator and Discriminator. These networks can be viewed as **agents** that play game against each other.

A generative network tries to **generate** data and a discriminator network **examines** data and estimates whether the generated sample is real or fake [1]. The generator takes random noise as input and tries to generate sample images, which later, along with original images, is fed to a discriminator network. The goal of the generator is to generate images in a way that will make the discriminator network unable to distinguish fake and real images. As both players get better and better over time, eventually generator is forced to create data that is as realistic as possible. Hence, both the networks play a zero sum game.

In the beginning, the generator does not know how to create images that resemble the ones from the training set. Similarly, the discriminator does not know how to categorize the images. As a result, the discriminator receives two very different sets of images. One is composed of **true** images from the training set, and another batch contains **noisy** signals. As training progresses, the generator tries to learn the data distribution that is composed of the training set images. It then starts to output images that look closer to the images from the training set.

Meanwhile, the discriminator network also starts to get better at classifying samples as **real** or **fake**. As a consequence, the two types of mini-batches begin to look similar, in structure, to one another. That, as a result, makes the discriminator able to identify images as real or fake.

B. CycleGAN Architecture

In this project, our aim is not just generating images similar to Monet's painting but also to transfer it to non-Monet images. Thus the architecture of GAN is such that we can apply neural style transfer to any *new* input image given to the system. As discussed above, we un-pair images of two different domains (X, Y). The goal of the model is to translate an image from X to Y domain. Consequently, the model needs to learn the mapping $G : X \rightarrow Y$, such that the distribution of images from $G(X)$ is indistinguishable from the distribution of Y using an adversarial loss. However, there are infinitely many such translations that will output same distribution over y. In addition, this procedure often leads to map all input images to the same output and optimization fails.

To overcome this issue, a Cycle-GAN (Fig. 2) is proposed. The main idea of Cycle-GAN architecture is to exploit the property that translation should be "cycle consistent", in the sense that if we translate a sentence from English to German and then translate it back to English, we should arrive back at the original sentence [2]. Mathematically, if we have a mapping $G : X \rightarrow Y$ and another mapping $F : Y \rightarrow X$, then G and F should be inverses of each other. CycleGAN utilizes this assumption by training both the mappings G and F simultaneously, and adding a cycle consistency loss that encourages $F(G(x)) = x$ and $G(F(y)) = y$. Combining this loss with adversarial losses on domains X and Y yields our full objective for unpaired image-to-image translation, as shown in Eq. 1.

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \\ + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda * \mathcal{L}_{CYC}(G, F)\end{aligned} \tag{1}$$

Here, λ controls the importance of objectives, and the aim of this model it to minimize the above-described loss function.

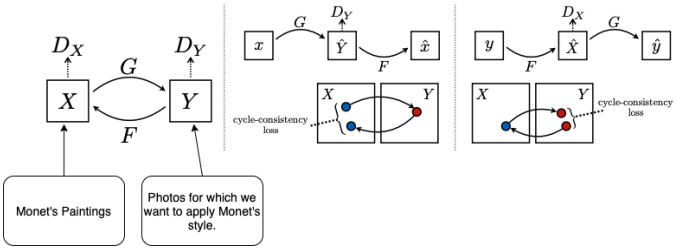

Fig. 2: The CycleGAN model with two mappings $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and the corresponding discriminators D_X and D_Y and to further regularize the translations between domains additional cycle-consistency losses are introduced.

V. Evaluation

The Kaggle competition was intended to evaluate submissions based on Memorization - informedFID **MiFID score**, which is a modification from Fréchet Inception Distance (FID). For the evaluation metric, we utilize MiFID, in addition to using human inputs in the feedback loop. FID evaluates generated fake samples based on the statistics of a collection of synthetic images compared to the statistics of real images from the training set [3]. FID is calculated using the following formula:

$$d^2 = ||\mu_1 - \mu_2||^2 + Tr(C_1 + C_2 - 2 * \sqrt{C_1 * C_2}) \tag{2}$$

μ_1, μ_2 represent feature-wise mean, and C_1, and C_2 stand for the co-variance matrices, for real and fake images, respectively,

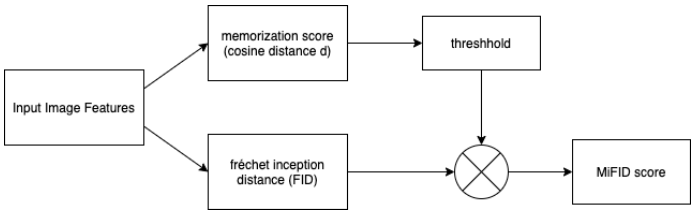

Fig. 3: MiFID evaluation metric.

It can be observed from Fig. 3, the MiFID score uses a memorization score in combination with FID. In this case, the memorization score is a *cosine distance* between generated image features and real image features. MiFID score of 0 represents that the generated and real images are the same. Thus, the lower the MiFID score, the more the similarity between two domains.

VI. Results

Due to limited time, lack of computational power, and small memory size of currently available machines, we could do training only for 7 epochs with batch size 2. In Google Colab GPU service, training of each epoch takes about two hours. However, even with the insufficient *amount* of training, we achieved satisfying results, and the generated images are mostly seen as Monet style by humans. Fig. 4 shows a sample output of our latest model.

Mathematically speaking, at the end of training, we achieved 56.84 MiFID score, which can be considered as an average score on the competition leaderboard results. However, Kaggle has its own code and method for calculation of MiFID score, so it may be slightly different in Kaggle's own evaluation.

TABLE I: Evaluation of the proposed model.

Model	MiFID Score
CycleGAN	56.84

Furthermore, we compared the initial losses which we recorded in the beginning of training and the final losses that we got at the end of training. Fig. 5 shows the loss functions' values. It can be seen that the losses are decreased significantly with during training. However, as a future work, the network can be trained for more epochs to let model weights reach their optimal values.

(a) Real images.

(b) Generated images.

Fig. 4: GAN results for sample images.

```
Loss_D: 0.6366 Loss_G: 9.2964 Loss_G_identity: 2.8456 loss_G_GAN: 0.4551 loss_G_cycle: 5.9957:
```

(a) Loss function at the beginning of training.

```
Loss_D: 0.2169 Loss_G: 6.5367 Loss_G_identity: 1.6006 loss_G_GAN: 1.1558 loss_G_cycle: 3.7802
Loss_D: 0.1937 Loss_G: 5.5022 Loss_G_identity: 1.5204 loss_G_GAN: 0.5279 loss_G_cycle: 3.4539
Loss_D: 0.0841 Loss_G: 4.1099 Loss_G_identity: 0.8514 loss_G_GAN: 1.2288 loss_G_cycle: 2.0297
```

(b) Loss functions of final 3 epochs.

Fig. 5: Loss functions over time.

VII. GUI FOR HC

Fig. 6 shows the home page of GUI that mainly focuses on the human computation part. Gamification is used to gather more data through human response to extend the data set. The GUI is currently implemented on an IOS (iPhone 10) based application. The homepage of the GUI contains 3 clickable interfaces - ”Upload Picture”, ”Play game” and ”About”, each of which are discussed below in detail.

A. Upload Picture

By using the interface ”upload picture”, users can browse through the pictures and generate the Monet-version of the picture using the model mentioned above. The users can also browse through their own galleries to select the desired picture and ”Monet-ise” it, based on the aforementioned model. Fig. 6 provides a UI to browse though the gallery.

B. Play Game

Fig. 7 shows the user interface of the mini game that users can play. This game is for the purpose of gathering data using human computation component. The user are asked different questions to extend our existing dataset. For example, the question ”Is the picture an original Monet or not” will generate a binary response. Based on this response, the user can be asked more advanced questions, in a questionnaire format, about the strokes of the brush, color, shape or texture of the painting. The users can further select the part of the painting where they think the painting looks fake or doesn’t follow the style of Monet. The game will be based on a point system. For each painting that the users correctly label as fake or original, the game will award a certain set of points to the users.

C. About

The about interface gives the information about our model and the game in general for the more curious users looking to

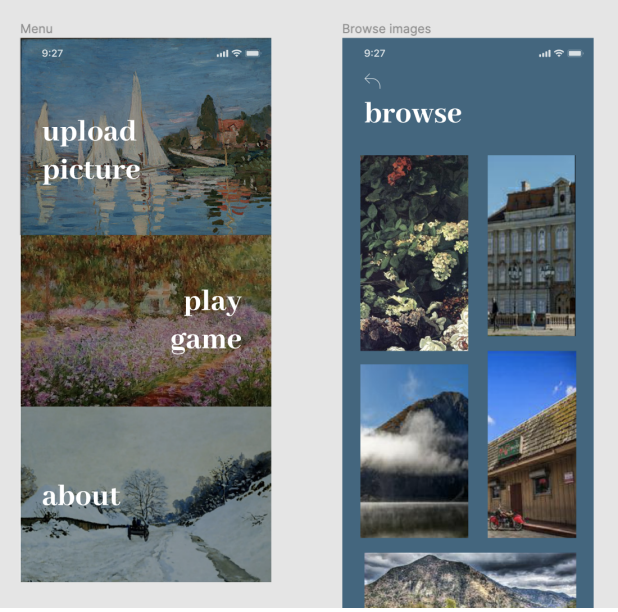

Fig. 6: App demo with main (left) and upload (right) pages are shown.

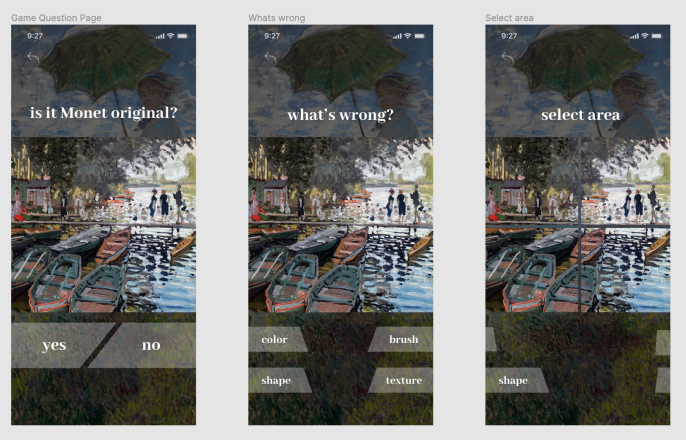

Fig. 7: Mini-game for human computation.

extend their own knowledge about GANs and the purpose of our project.

VIII. Conclusion and Future Work

As the results show, the GAN network is able to successfully transform the input photos into a Monet style painting. The user interface of the app enables us to extend our data set using the principle of human computation. Gamification is used as an incentive to gather the data which will essentially fine tune the model further once integrated. Training of the model for a longer period will generate better results and can be implemented in the near future. This model can also be applied to generate paintings in the style of other celebrated artists, which provides more depth to the model and adds multiple use cases for future research.

Acknowledgment

We like to thank our instructors, Yingding Wang, Prof. Francois Bry, Prof. Andreas Butz, and Beat Roßmy for their valuable feedback and suggestions throughout the semester.

References

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.
[2] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," 2020.
[3] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," 2018.
[4] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," 2019.
[5] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural network," 2016.
[6] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," 2016.
[7] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, "Learning to discover cross-domain relations with generative adversarial networks," 2017.