Generating abstract art with GANs

by Alexios Gkolovko

dit2102dsc

Abstract

In this project, we try to create and train a Generative Adversarial Network (GAN), so that it can produce abstract paintings on its own. The dataset used to train the model can be found here https://www.kaggle.com/datasets/bryanb/abstract-art-gallery. From that dataset, a subset consisting of 571 images was formed, in order to accelerate the procedure of training the GAN. The project was implemented using PyTorch. All the prerequisites are mentioned in the 'readme.txt' file, that accompanies this report and the code.

Introduction

A Generative Adversarial Network (GAN) is a class of machine learning frameworks introduced by Ian Goodfellow and his colleagues in June 2014. In GANs, two neural networks, the generator and the discriminator, contest with each other in a zero-sum game.

More specifically, the generator produces artificial data (usually images) based on a given dataset, while the discriminator tries to distinguish genuine from generated data. The following image shows the idea behind a GAN:

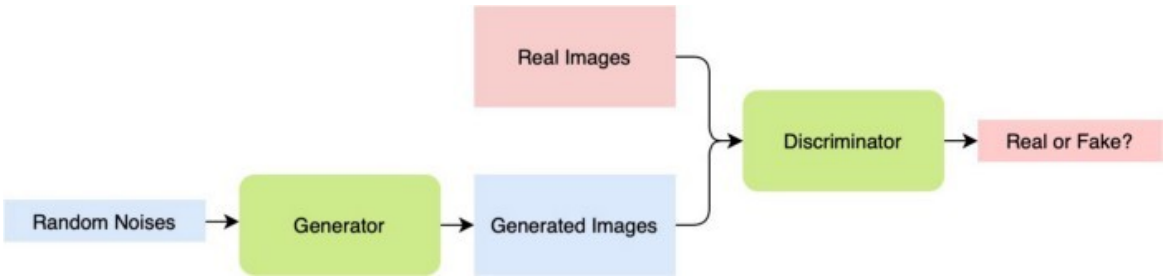

Image 1: The idea behind a Generative Adversarial Network
(source:https://towardsdatascience.com/building-a-gan-with-pytorch-237b4b07ca9a)

In this case, the task is to train the generator so that he is able to generate artificial abstract paintings, and train the discriminator so that he can distinguish genuine from artificial paintings.

Description of the training dataset

The training of the model is based on the Abstract Art Gallery Dataset, that can be found in Kaggle. This dataset consists of 2.782 .jpg images, depicting abstract paintings. Its size is about 480MB. For the purposes of this project, a sample of 571 paintings was selected (namely the first 571 paintings of the dataset, about 80MB). This reduced the amount of time needed to finish an epoch from 35’ to about 7’. Of course, this probably had a negative impact on the produced results. The sample used for the purposes of the project is located in the ‘Abstract_gallery/Abstract_Gallery/class’ folder, that is also available[1].

It is worth to notice that in the folders, the subfolder ‘class’ was created after facing problems with the implementation of the code. The ImageFolder module demanded that there should be a class folder, and took into consideration only the data in that folder.

Tools used

The present project is implemented in PyTorch along with torchvision. The plots are made using matplotlib. Also tqdm is used. For more details on the tools that are used in the project and their versions see the ‘readme.txt’ file.

Brief description of the code

The code is provided in ‘Deep_Learning_Project.ipynb’ file. The results of every part of the code can be seen under its implementation. It is based on two proposals on the GAN implementation and testing for this case[2].

After importing the needed libraries (see the previous chapter), the images pass through a process of transformation. In particular, they are resized, cropped and normalized. Resizing takes place so that the images become of the same size, while cropping so that frames and other information that could be considered as noise are excluded. Normalization and the subsequent denormalization are applied in

1 https://drive.google.com/drive/folders/1feXXbnoR7MtlIewqxwGfiR1iOhbR8AqZ?usp=sharing
2 https://www.kaggle.com/code/jerry42/abstract-rt and https://www.kaggle.com/code/algord/abstract-art-generator-dcgan

order to increase performance (see ‘denorm’ method, defined in the code). After the data insertion and transformation (a part of the transformed data can be seen in Image 2), we proceed to the implementation of the generator and the discriminator.

Image 2: Part of the transformed data.

The discriminator makes use of Convolutional Neural Networks (CNNs) in order to distinguish genuine from fake paintings. Leaky ReLU is used as it allows the pass of a small gradient signal for negative values, so the gradients from the discriminator flow stronger into the generator.

To create the generator we use the ConvTranspose2d layer from PyTorch, which performs as a transposed convolution (deconvolution). The generator has a vector of random numbers (referred to as a latent tensor) which is used as a seed to generate the images. In this case, it will convert a latent tensor of shape (128, 1, 1) into an image tensor of shape (3, 28, 28).

Then, the methods to train the discriminator and the generator are defined. And then the training of the model takes place. The measures that are taken into consideration to evaluate the model are:

- the loss of the discriminator,
- the loss of the generator,
- the correctly classified genuine images and
- the correctly classified fake images.

Due to lack of resources, and time, it was decided that the training would last for only 10 epochs. The results of each epoch can be seen below (Image 3)

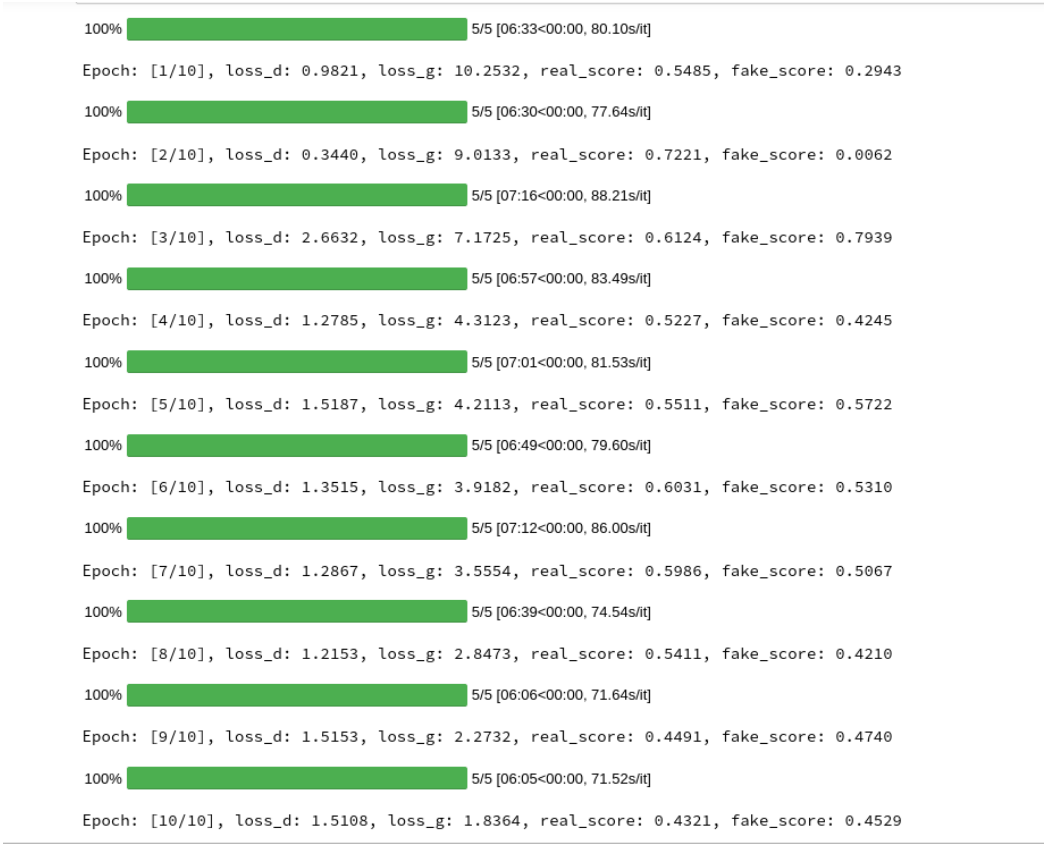

Image 3: The results by epoch.

In each epoch, the generator produces images that are saved into a folder named ‘generated’. All the results are included in the generated folder, here two representative images are shown:

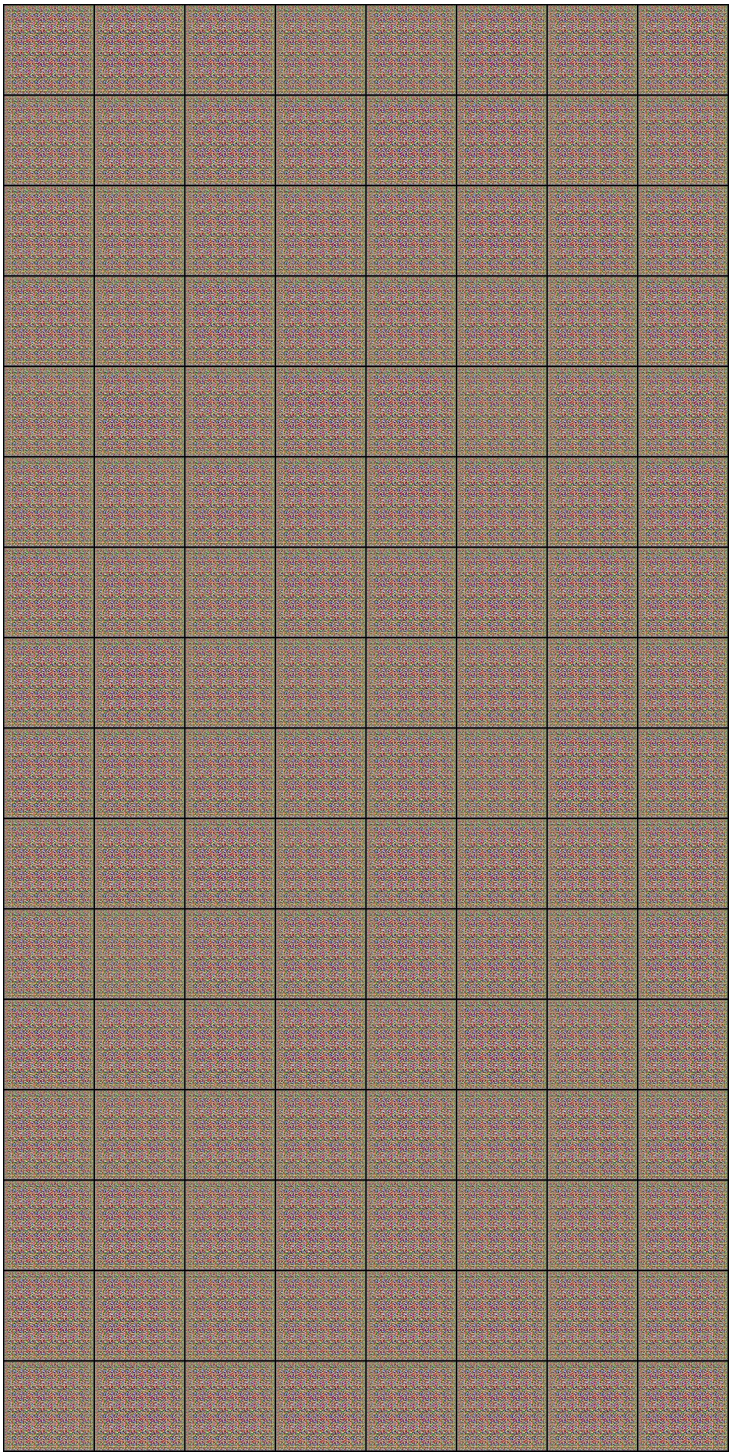

Image 4: The ‘generated-images-0001.png’

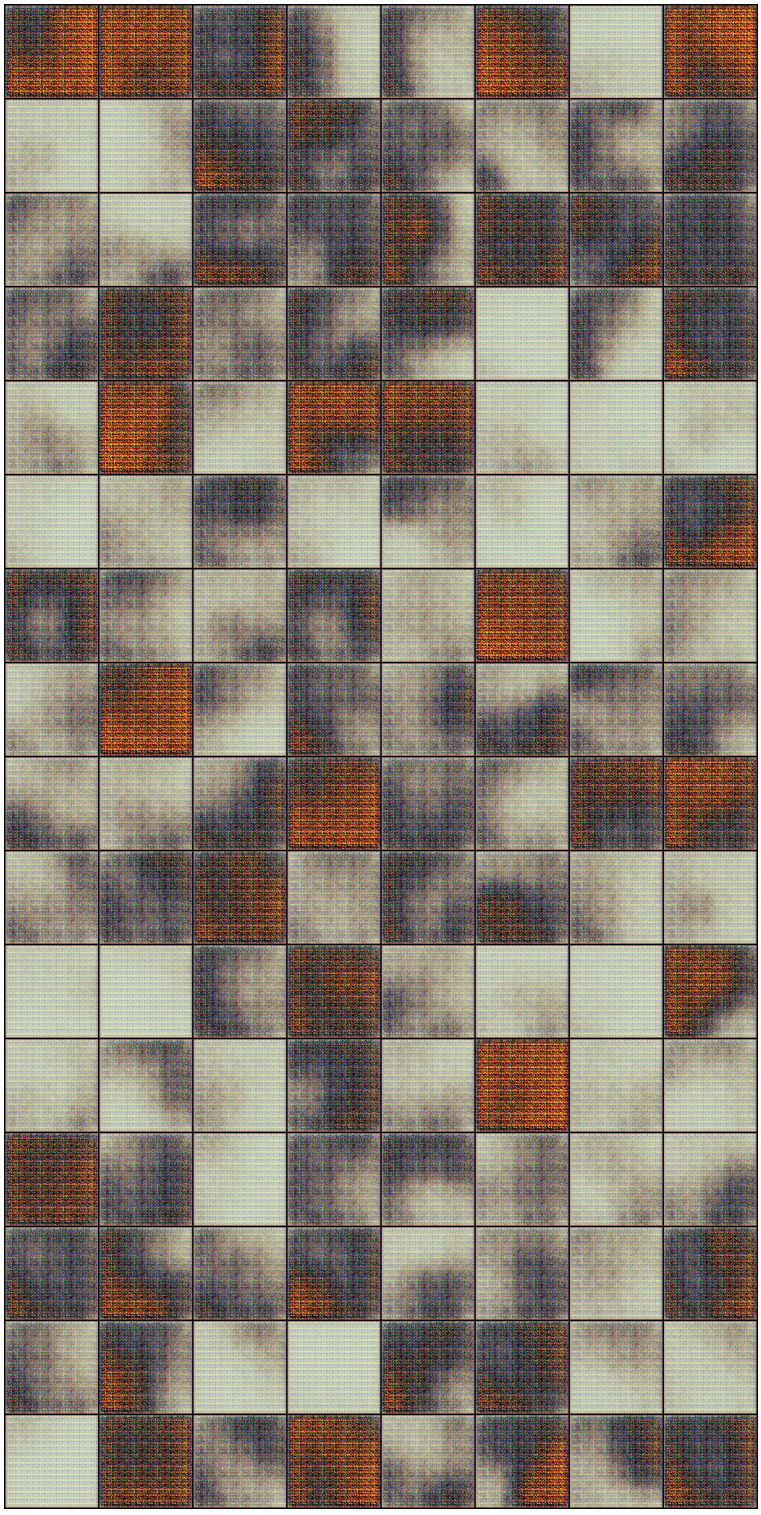

Image 5: The ‘generated-images-0010.png’

Generally speaking, the results seem to improve epoch after epoch, they seem to become closer to the genuine ones. It is certain that, if the experiment continued for more epochs, the results would be even better.

Loss and Score Diagrams

As mentioned above, the measures that are taken into consideration to evaluate the model are:

- the loss of the discriminator,
- the loss of the generator,
- the correctly classified genuine images and
- the correctly classified fake images.

Using the matplotlib library, we created the following loss-to-epoch and score-to-epoch diagrams.

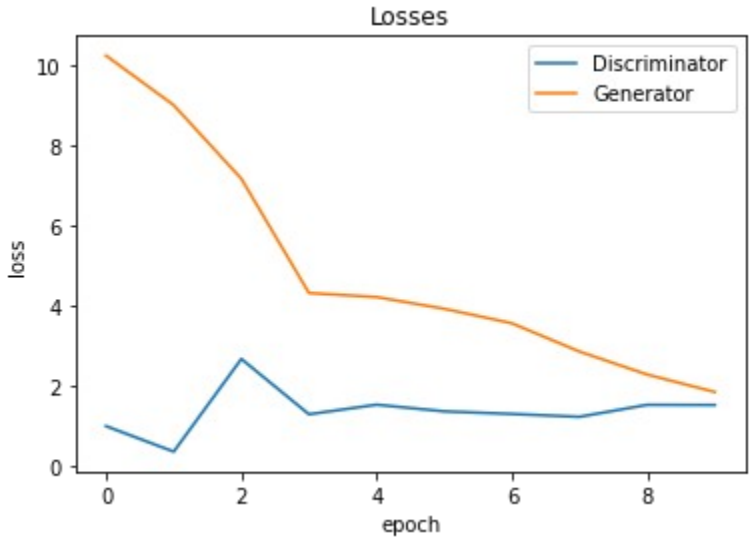

Image 6: The loss-to-epoch diagram

It seems that, with time, the generator's losses decrease rapidly and tend to reach the losses of the discriminator. On the other hand, the losses of the discriminator are somewhat stable (or maybe tend to increase slightly). The scores also seem to tend to become equal after some epochs. The real score seems to decrease slightly by epoch, while the fake score seems to improve generally. In all, a tendency to equalization of the results seems to take place, with the generator improving its performance, while the discriminator is almost stable (or exacerbates slightly).

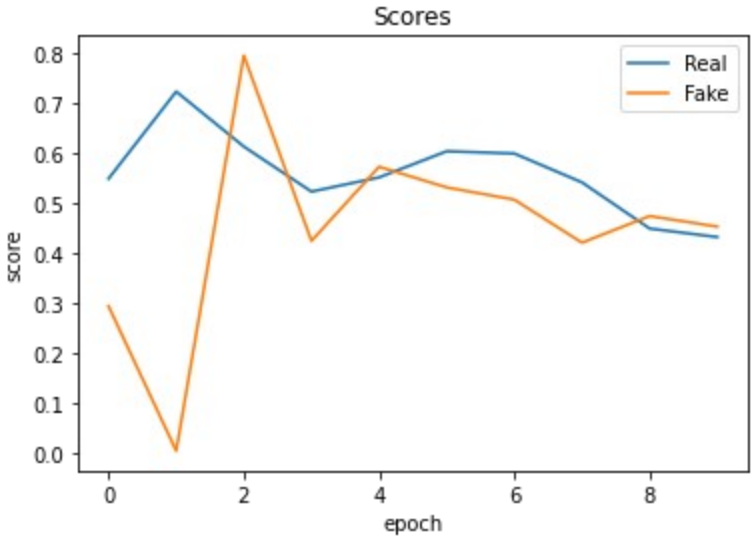

Image 7: The score-to-epoch diagram

Conclusion

This project aimed at creating a GAN that could produce artificial abstract paintings. Although the lack of resources was expected to be detrimental in this case and the results were far from perfect, it seems that the model tends to improve after a number of epochs. Other implementations show that such GAN systems can achieve impressive results when trained for some hundreds of epochs and on the whole dataset. As a result, this model would also probably improve further, if the training is to last for more epochs.