ART GENERATING WITH DCGANS

[Belge alt konu başlığı]

BERK COŞAR 69557
FURKAN TUNA 69730
ALP AKKANLAR 76022
YİĞİT MERİÇ 69170

1. Introduction

1.1 Concept

Artistic expression has exceeded traditional mediums and transferred to the digital frontier, with the advancements in deep learning. In this context, the utilization of Deep Convolutional Generative Adversarial Networks (DCGAN) became a powerful tool for the creation of digital art. This report includes the exploration of DCGANs to learning an art movement and producing images that seem like they to belong that art era.

1.2 Objectives

This project addresses the challenge of diversity and novelty within generative art, aiming to create images that exceed just replication and become a real artistic image. Our objective is to utilize DCGAN as an art creator, not just an art replicator.

1.3 Background

Generative art is an interdisciplinary field that is an intersection of computer science and visual arts. Historically, generative art has been driven by various techniques, from rule-based systems to stochastic processes. However, the improvement of deep learning, has marked a significant leap forward.

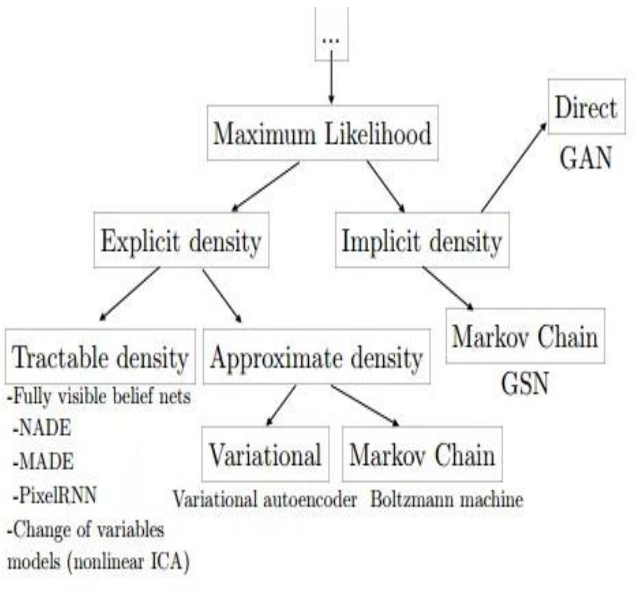

Figure 1. Taxonomy of Generative models. [1]

GANs were designed to overcome the problems of the other generative models. GANs were first proposed by Goodfellow et al. in 2014, involving 2 Deep Neural networks, called generator and discriminator. In 2016, DCGANs introduced by Alec et al. DCGANs. There is a phenomenon called mode collapse with DCGANs, so in 2018, ProGAN introduced by Karras et al. from NDIVIA. But again, there is a phenomenon called latent space entanglement. In 2019, Karras came out with StyleGAN, which minimizes the latent space entanglement. Even though StyleGAN solved most of the problems of older models, the generator was still prone to visual artefacts. In 2020, Karras et al. came up with StyleGAN2, which is an attempt to improve the previous StyleGAN architecture, and this architecture solved all problems.
Even though models such as ProGAN, StyleGAN, and StyleGAN2 showed unprecedented image generation qualities in an unsupervised environment, they lacked control over the type of generated image. In 2014, Mirza et al. proposed Conditional GAN that allowed greater control over image features by including class labels. In 2018, Pix2Pix GAN introduced by Isola et al. from UC Berkeley, which is an Image-to-Image translation model that can learn interesting mappings from one image to another. [2]

2. Methodology & Tools

In the first phase of the project, our initial focus is finding the best dataset for the task we will work on. We decided to use the WikiArt Art Movements/Styles dataset, which is a dataset consisting of 13 different art movement folders, with a total of 42500 images. We decided to utilize the Romanticism folder from that dataset, which has 6183 unique images.

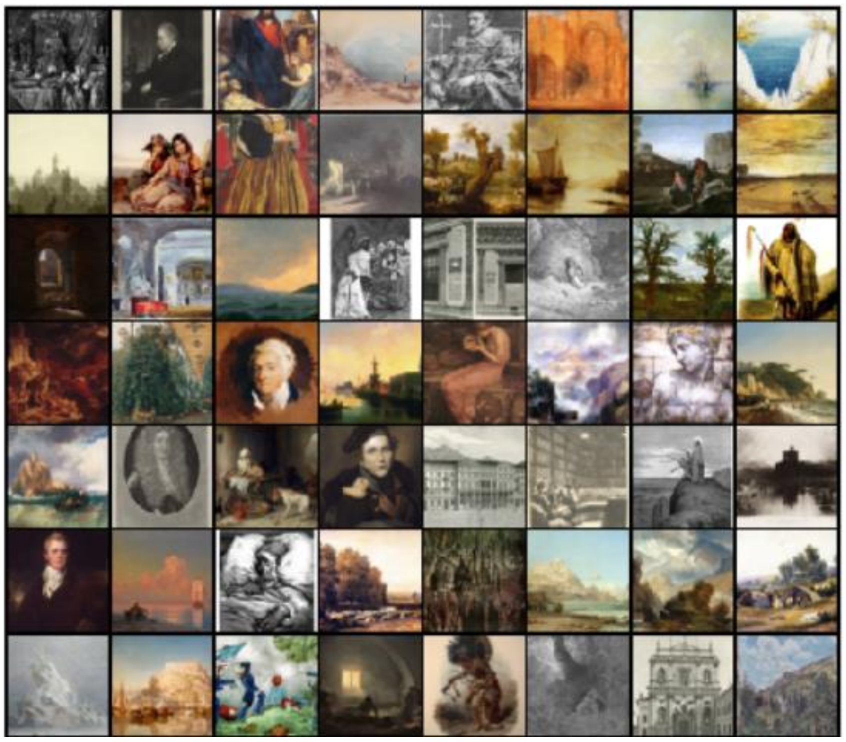

Figure 2. Example images from the dataset.

As a deep learning framework, TensorFlow and PyTorch are the most popular choices. We decided to use PyTorch for this project. One of the main problems with a DCGANs is getting a stable DCGAN architecture, so we decided to utilize Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Network paper introduced by Radford et al. [3]

3. Model Architecture

3.1. Weight Initialization

Weight initialization is a crucial step for training a DCGAN, because without the proper weight initialization, problems like vanishing gradients and exploding gradients can be occur. These problems can damage the convergence and stability of the training process. There are different types of initializations for DCGANs, like Xavier/Glorot initialization, Kaiming initialization and Normal distribution. We choose a normal distribution with mean of 0, and standard deviation of 0.02.

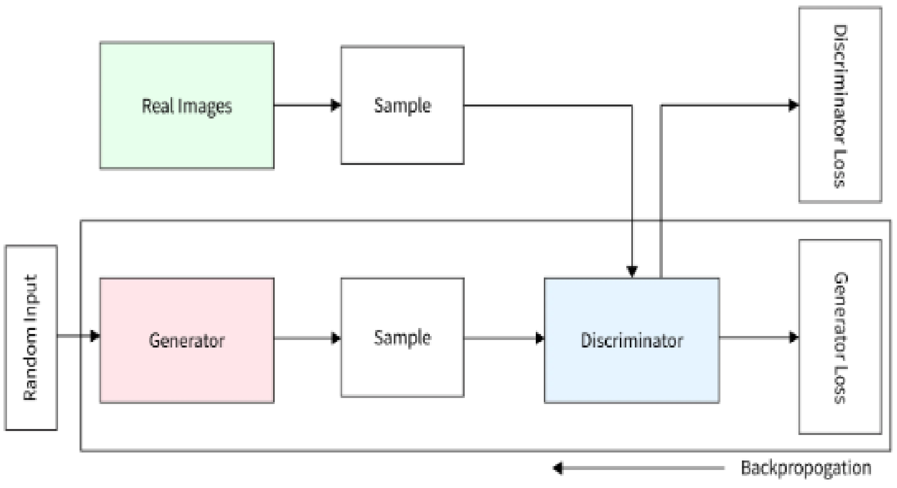

Figure 3. Scheme of a DCGAN architecture [4]

3.2. Generator

The generator is crucial for mapping the latent space vector to data space in DCGAN. To reach that, we used fractional-strided convolutions in the generator. For the normalization, we used batchnorm, and as activation, we used ReLU for all layers, except from the last layer, which we used tanh. The usage of tanh scales the training images between the range of [-1,1].

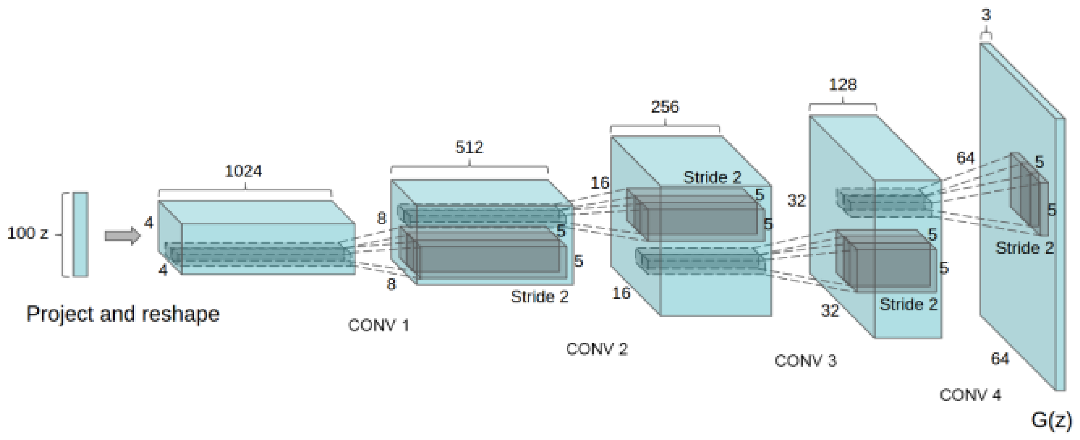

Figure 4. Generator [3]

3.3. Discriminator

The discriminator is a binary classification network that calculates the probability that a given image is fake or not. To deal with probabilities, we used sigmoid activation in the final layer. We used strided convolutions, batchnorm as normalization, and leaky ReLU activation for the rest of all layers, so we get a good gradient flow. For leaky ReLU, the slope of the leak is set to 0.2.

3.4. Loss Function

Choosing an appropriate loss function is important to create a stable, converging DCGAN architecture. We choose Binary Cross Entropy (BCE) loss for the implementation.

$$\ell(x,y) = L = \{l_1,\dots,l_N\}^\top, \quad l_n = -w_n\left[y_n \cdot \log x_n + (1-y_n)\cdot \log(1-x_n)\right],$$

where N is the batch size. If `reduction` is not `'none'` (default `'mean'`), then

$$\ell(x,y) = \begin{cases} \operatorname{mean}(L), & \text{if reduction} = \text{`mean';} \\ \operatorname{sum}(L), & \text{if reduction} = \text{`sum'.} \end{cases}$$

Figure 5. Binary Cross Entropy Loss [5]

3.5. Optimizers

Optimizer is also an important point of the model. We decided to use the Adam optimizer, which handles both saddle points problem, noisy gradients problem, and high condition number problem. We decided to use the suggested parameters for Adam from the paper. In the paper, it stated that a learning rate of 0.001 is too high, so we used 0.002 instead. Again in the paper, beta1 of 0.9 resulted in training oscillation, so we used 0.5 to improve the stability of the model. As beta2, we used 0.999.

4. Results & Conclusion

We trained the model with batch size of 128, generator input size of 100, and feature map size of 64. Started from 10 epoch, we tried 10 epoch, 30 epoch, and 40 epoch in final.

4.1. Loss & Images

We plotted the generator and discriminator losses to understand whether there is a problem with them or not. First, we trained with 10 epochs, and the losses are bad than we expected. Especially, generator loss seems high.

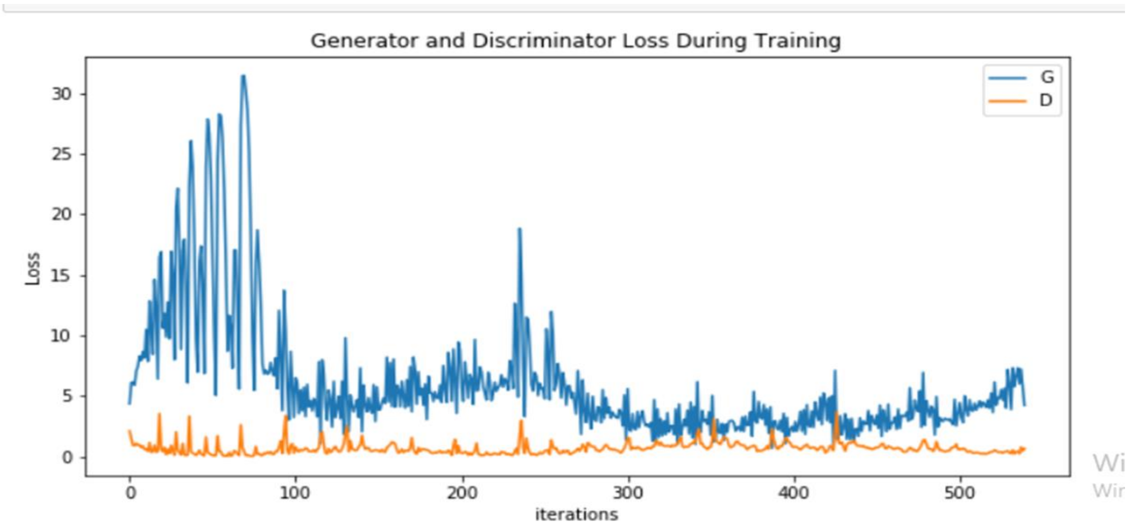

Figure 6. Loss at epoch 10

We decided to increase the number of epochs we are training, because our model may not trained long enough to minimize the loss.

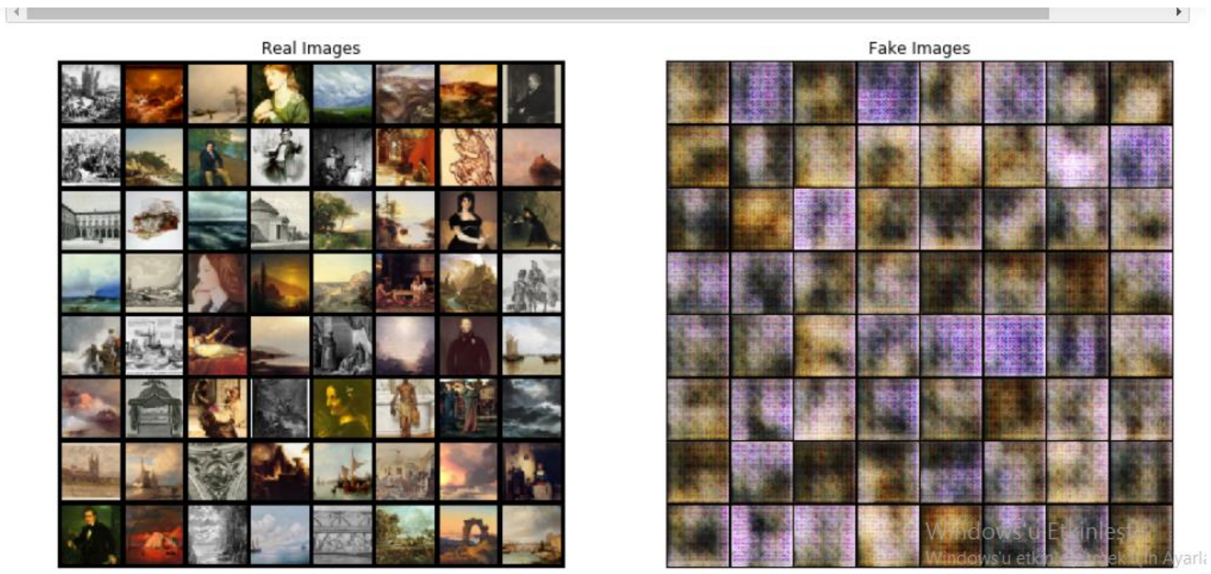

Figure 7. Results of 10 epochs

As it seems from Figure 7, images we generated is not meaningful with 10 epochs.

The model could not eliminate the noise properly, and it is hard to say there is a figure in the images.

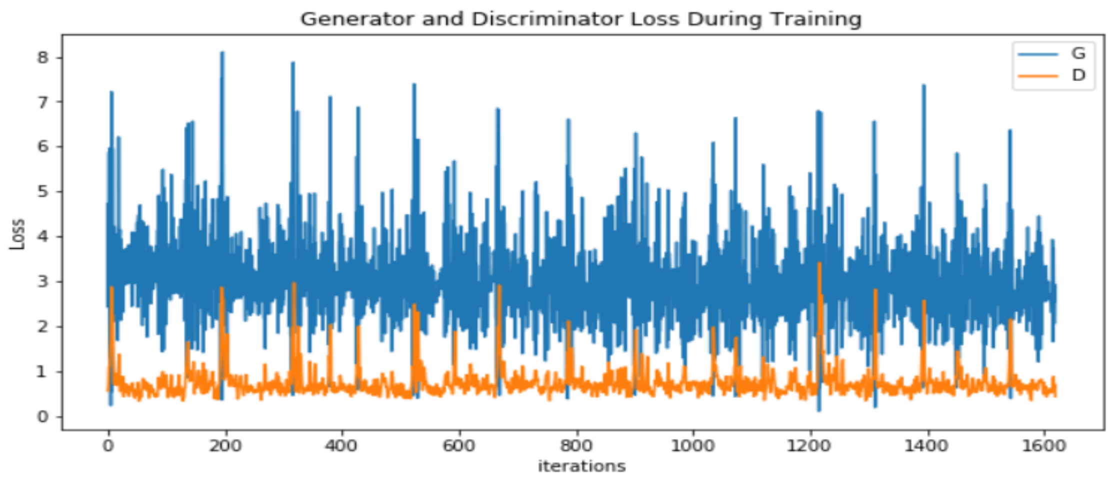

Figure 8. Loss at epoch 30

With 30 epochs, we observed a significant decrease in generator loss, that shows that our model indeed not trained enough at 10 epochs.

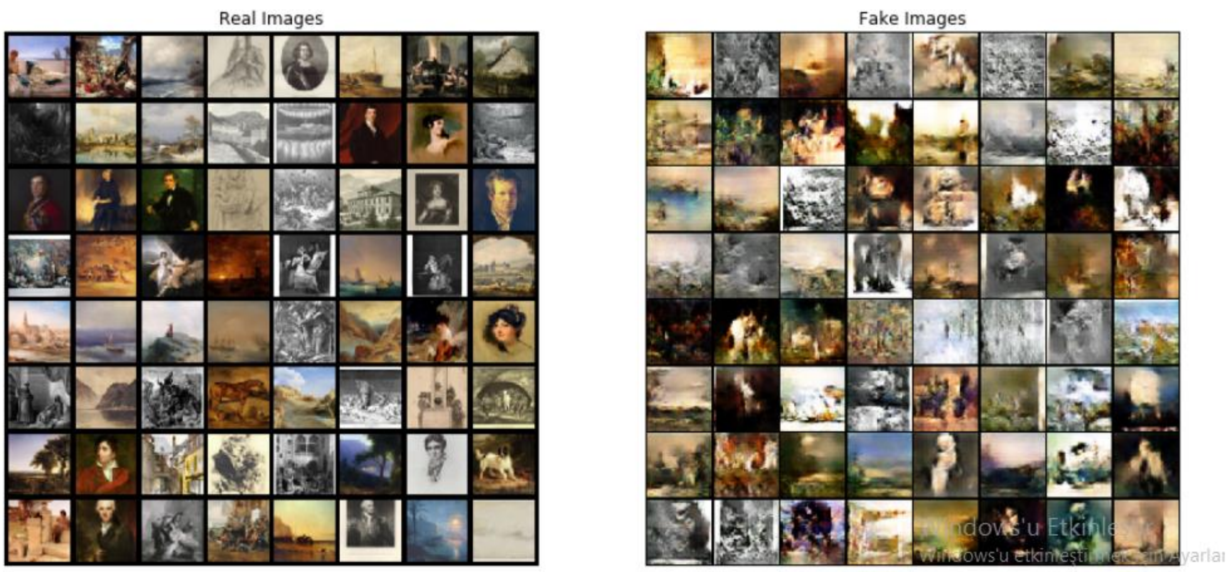

Figure 9. Results of 30 epoch.

As it seems from Figure 9, the outputs are looking like more realistic. The noise is significantly dropped, and outputs are much more like artistic images.

For the final results, we tried 40 epochs.

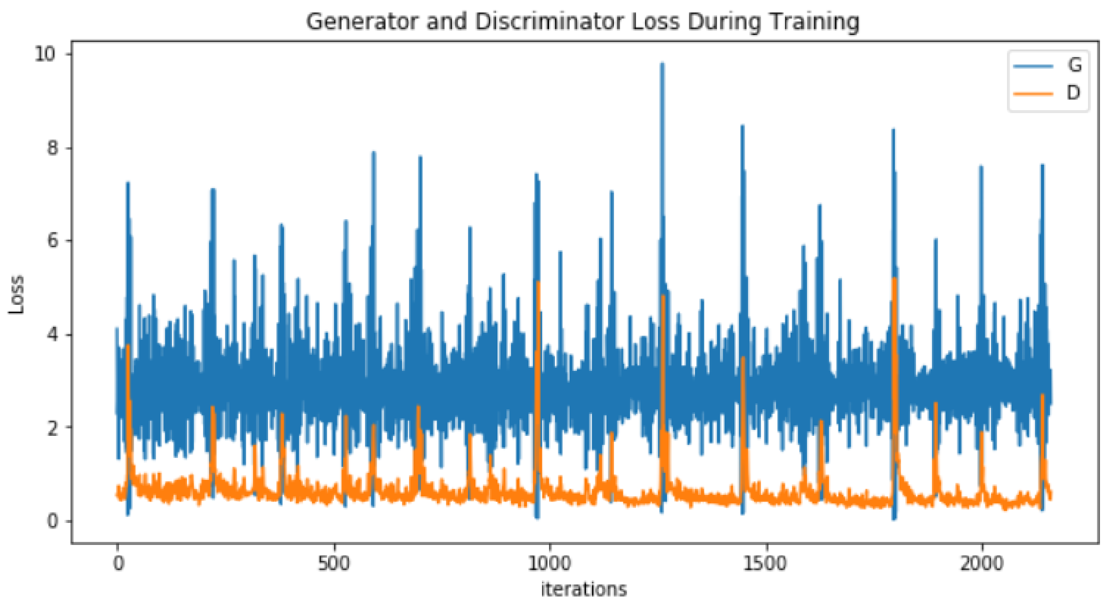

Figure 10. Loss of 40 epochs.

Comparing the losses with 30 epochs and 40 epochs, this time difference is not that significant, but still with 40 epochs, there is a slight improvement in loss.

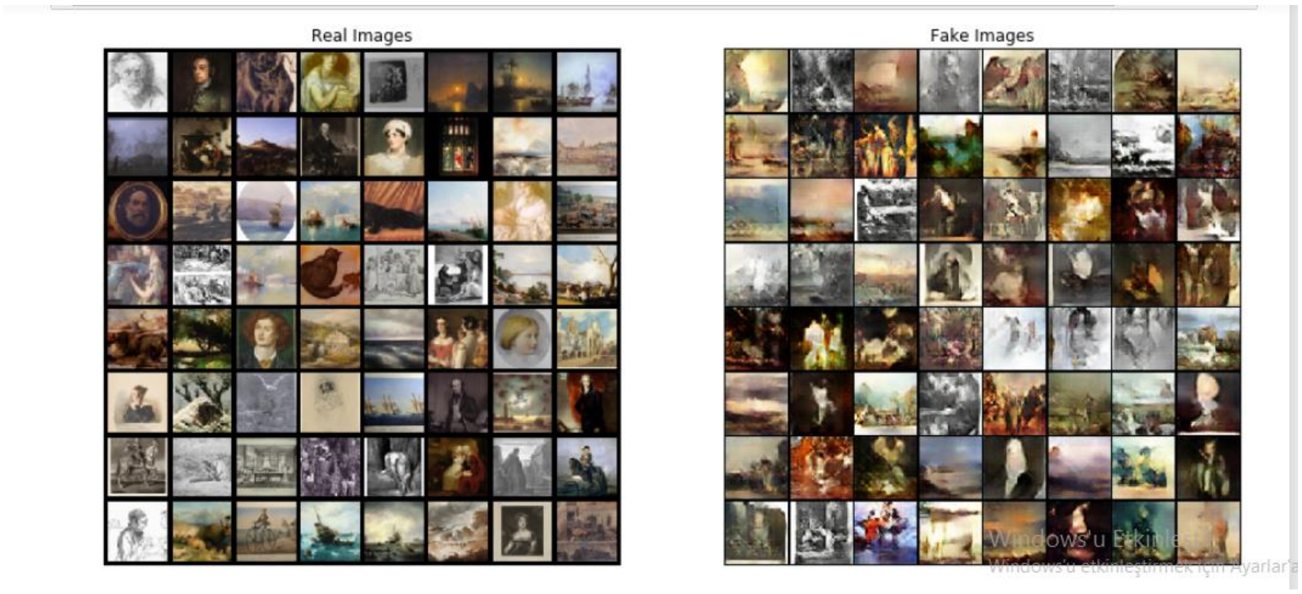

Figure 11. Results of epoch 40

From the Figure 11, the output of epoch 40 seems realistic Romanticism art images.

4.2. Conclusion

As a conclusion, we managed to train a stable DCGAN architecture and managed to get realistic outputs from it. The model still can be improved, training with more epochs may still get better results, or making generator and discriminator deeper can produce better results with the same epoch numbers.

References

1. https://towardsdatascience.com/generative-adversarial-networks-history-and-overview-7effbb713545
2. A Brief History of Generative Adversarial Networks, Parth Sharma
3. Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Network, Radford et al.
4. https://www.scaler.com/topics/deep-learning/dcgan/
5. https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html