Creating Art with Various GAN Architectures

Onur Tepencelik
A53313861
otepence@ucsd.edu

Ismail Ocak
A53305067
iocak@ucsd.edu

Sean Liu
A92094374
sel118@ucsd.edu

David Lu
A53308591
dblu@ucsd.edu

Abstract

Generative Adversarial Networks (GANs) are a prominent and growing field of machine learning which explores models that are able to generate instances that fit within the original dataset. In this report, we explore the usage of GANs for the task of generating artistic paintings. We discuss a variety of GANs that we implemented for this task using the Best Artworks of All Time and WikiArt Datasets. Experiments and results using a baseline Deep Convolutional GAN (DCGAN), a Creative Adversarial Network (CAN), and a Conditional GAN (CGAN) are presented and analyzed. We managed to achieve a model that is able to generate artwork with our baseline DCGAN architecture, extended it to a CAN architecture capable of generating creative, style-agnostic artwork, and extended that even further to a CGAN architecture that generates style-specific artwork. Our implementations are available on GitHub.[1]

1 Introduction

Generative modeling is the task of learning patterns in data such that a model can produce new examples that fit within the original, true dataset. Generative Adversarial Networks (GANs) [1] are one approach to generative modeling and re-frames this unsupervised learning task into a supervised learning one with the use of two sub-models. The generator is the model that is trained to produce new examples while the discriminator classifies samples as either from the true distribution or the model distribution. These two models are trained together in an adversarial manner until the generator can reproduce the true data distribution and the discriminator is guessing at random; both generally consist of Convolutional Neural Networks (CNNs) for tasks that involve visuals. The generator first takes in a random input and transforms it into a new sample after several convolutional layers. In other words, it randomly generates "fake" images based on the input, and at equilibrium, the discriminator should not be able to tell the difference between the images generated by the generator and the actual images in the training set. The discriminator is then fed either a generated sample or a real sample and performs binary classification based on convolutional features.

For our final project, we produced artwork with various GAN architectures using the Best Artworks of All Time [2] and WikiArt [3] datasets. To achieve this, we first trained a Deep Convolutional GAN (DCGAN) [4] as a proof of concept as well as a baseline model to be expanded on for further experimentation. Next, we implemented a Creative Adversarial Network (CAN) from [5]. Here, the discriminator was modified so that the model could differentiate between different styles of artwork in an effort to generate art that is unique from the original dataset; as such, the discriminator cannot identify the generated artwork as a specific artistic style. Therefore, the generator learns to generate

[1] https://github.com/otepencelik/GAN-Artwork-Generation

creative artwork that cannot be classified with high confidence by the discriminator to a specific style. Finally, we implemented a Conditional GAN (CGAN), in which both the discriminator and generator gain access to artistic style information of the inputs, thereby generating artwork of a defined style. Thus, we explore two sides of the same coin: generation of style-agnostic art and generation of style-specific art. In the following sections, we provide the details of each architecture, their respective training procedures, as well as their generated artwork results along with discussion.

2 Related Work

GANs were first introduced by the popular work of Goodfellow *et al.* [1], with the idea of simultaneously training two networks that depend on each other. The DCGAN was then introduced in [4] as a direct extension of GANs with the usage of CNNs in lieu of fully connected networks for improved image generation. In [6], advancements were made to increase robustness and stability of GANs during training for a wide variety of models, including previously untrainable networks. With research into GANs continuing to move at a rapid pace, this type of network is now found in countless applications, including our chosen domain, the generation of artwork.

In traditional GANs, there is no control over what types of images are generated aside from these images being aesthetically similar to the dataset the model was trained on. The result is that a GAN trained on a diverse dataset consisting of the works of many different artists and art styles will produce a broad range of images. For more targeted generation, conditional GANs like AC-GAN [7], and CycleGAN [8] for style transfer, have been developed. AC-GAN learns images with labels jointly during training by adding a conditioning vector as input to the generator; this conditioning is what allows it to produce specific types of outputs. In style transfer, models like CycleGAN translate an input image from a source domain to a target domain. For example, CycleGAN has demonstrated its ability to translate photographs into artwork in the style of numerous famous artists.

Another approach is introduced in [5], where a CAN learns style and becomes creative by generating art that deviates from these learned styles. The authors also demonstrated that the art produced by the CAN was sometimes rated higher than human art, as evaluated by human subjects. Finally, in a recent paper [9], Xue proposed a two-stage GAN that creates Chinese landscape paintings from end to end, with the first GAN generating edge maps for the second GAN to translate into a painting. The resulting paintings were mistaken as human artwork 55% of the time.

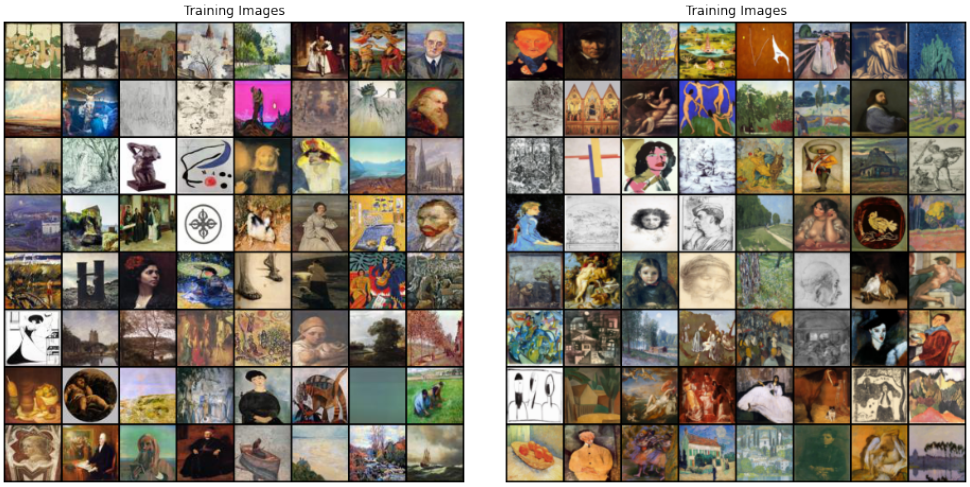
3 Methods

3.1 Datasets

We selected two different datasets to train our models. We started by developing our baseline model using the Best Artworks of All Time dataset [2], which contains over 8,000 pieces of art from 50 of the most influential artists of all time. For our CAN and CGAN architectures, we decided to use a much larger dataset as we obtained unsatisfactory results using [2]. The WikiArt dataset [3] was selected for our later experiments and contains 81,449 pieces of art from 1,119 artists from the 15^{th}- 20^{th} centuries. Each piece of art is labeled and classified into 1 of 27 artistic styles, including Baroque, Impressionism, Minimalism, Pop Art, and Rococo. Figure 1 shows some samples from the two datasets.

3.2 Baseline Model - Deep Convolutional GAN Architecture

For our initial model, we implemented a DCGAN architecture based on [1] and [4], following a PyTorch tutorial [10] that uses the same architecture to generate human faces. The authors of [4] made some changes to traditional CNN architectures to be able to incorporate them into GANs, as the prior attempts to scale up GANs with CNNs were relatively unsuccessful. These implementation changes mainly consist of the replacement of the spatial pooling operations with strided convolutions and the removal of fully connected layers at the end of the CNN architecture. Using strided convolutions allow the network to learn its own spatial downsampling (discriminator) and upsampling (generator), rather than using a deterministic pooling function. Instead of using fully connected layers at the end of the CNN portion of the architecture, the authors proposed to connect the final convolutional



(a) (b)

Figure 1: Sample images from the Best Artworks dataset (left) and the WikiArt dataset (right).

features to the input of the generator and to the output of the discriminator. Along with these two applications, using batch normalization plays an important role in the stability of the model during training. The activation functions used in each layer were also specifically adjusted to increase the stability of learning. As a result, our initial architectures for the generator and discriminator networks are as follows:

Table 1: Baseline Generator Model Summary

Layer	Operations & Parameters	Output Dimension
Input	Random noise generation	[1, 150]
Deconv1	ReLU, BatchNorm, Kernel = 4, Stride = 1	[512, 4, 4]
Deconv2	ReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[256, 8, 8]
Deconv3	ReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[128, 16, 16]
Deconv4	ReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[64, 32, 32]
Deconv5	Tanh, Kernel = 4, Stride = 2, Padding = 1	[3, 64, 64]

Table 2: Baseline Discriminator Model Summary

Layer	Operations & Parameters	Output Dimension
Input	Normalization, Resizing	[3, 64, 64]
Conv1	LeakyReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[64, 32, 32]
Conv2	LeakyReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[128, 16, 16]
Conv3	LeakyReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[256, 8, 8]
Conv4	LeakyReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[512, 4, 4]
Conv5	Sigmoid, Kernel = 4, Stride = 1	[1]

For training, we use the Binary Cross-Entropy Loss (BCELoss) function, defined as:

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

The training procedure is based on the algorithm presented in the original paper on GANs by Goodfellow *et al.* [1], along with some modifications to improve performance. The procedure consists of two parts, the first for training the discriminator and the second for training the generator.

The main objective of the discriminator is to maximize its likelihood to distinguish between real images and fake images. Mathematically, the discriminator is trying to maximize:

$$log(D(x)) + log(1 - D(G(z))) \tag{2}$$

where $D(x)$ represents the probability of the discriminator classifying real images correctly and $(1 - D(G(z)))$ represents the probability of the discriminator classifying the fake images correctly. To train the discriminator, we created two separate batches of images, one with real images and one with fake images created by the current generator. For each batch, we make a forward pass through the discriminator, calculate the loss, make a backward pass and accumulate the gradients. We then update the parameters with these accumulated gradients.

The main objective of the generator is to "fool" the discriminator with its ability to create realistic images that are very similar to the images in the original dataset so that the discriminator cannot distinguish between them. Mathematically, the generator is trying to maximize:

$$log(D(G(z))) \tag{3}$$

which is the probability of the discriminator classifying the fake images wrongly. To train the generator, we use the classification results of the discriminator, calculate the loss and the gradients, and then update the parameters. The discriminator model acts as a teacher for the generator model to learn how to create realistic images.

The main objectives of the generator and discriminator can be combined into one equation, as given in [1]:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{4}$$

where G represents the generator and D represents the discriminator.

3.3 Creative Adversarial Network Architecture

We then implemented a CAN architecture following the approach in [5] in an attempt to extend the results of the DCGAN model. The CAN architecture is an augmentation of a regular GAN or DCGAN architecture, where the discriminator has access to separate information about the inputs, which in our case is the artistic style of the artwork. Our baseline architecture, the DCGAN [4], is trying to learn various concepts of art by looking at all instances in the training set and generating similar instances of artwork that it has seen. On the other hand, as mentioned previously, the authors of [5] proposed an architecture that is able to learn specific artistic styles, and therefore learn to generate creative art by deviating from the styles that it has learned. In other words, the main objective of a CAN architecture is to push the outputs of a DCGAN generator towards an ambiguous style rather than imitating the original dataset. This causes the discriminator to still believe that the generated image is real art, but it cannot classify the image into a specific artistic style since it is style-agnostic, or more "creative".

The generator of the CAN architecture is the same as the generator of the DCGAN, which is provided in Table 1. However, the discriminator is different, with two outputs to be used for evaluation as opposed to one. The discriminator network is divided into two paths after the convolutional layers; one path is for classifying whether the input artwork is real or fake, and the other is trying to classify the artwork with respect to its artistic style. Table 3 summarizes the discriminator of the CAN architecture.

To train the CAN model, we used the same loss functions as the DCGAN model, along with additional loss terms coming from the style classification output of the discriminator. The new objective of the discriminator is to maximize:

$$log(D(x)) + log(1 - D(G(z))) + log(D_c(c = \hat{c}|x)) \tag{5}$$

Table 3: Discriminator Model Summary for CAN Architecture

Layer	Operations & Parameters	Output Dimension
Input	Normalization, Resizing	[3, 64, 64]
Conv1	LeakyReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[64, 32, 32]
Conv2	LeakyReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[128, 16, 16]
Conv3	LeakyReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[256, 8, 8]
Conv4	LeakyReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[512, 4, 4]
Conv5 (Disc)	Sigmoid	[1]
Conv5 (Cls)	Softmax	[27, 1]

where the first two terms are the same as Eq. 2 and the last term represents the probability of the discriminator classifying the artistic style of the artwork correctly. During training, the first two terms are present in the BCE loss function as before, and the last term is included via a multi-class cross entropy loss function.

The new objective of the generator is to maximize:

$$log(D(G(z))) + \sum_{k=1}^{K}(\frac{1}{K}log(D_c(c_k|G(z))) + (1 - \frac{1}{K})log(1 - D_c(c_k|G(z)))) \tag{6}$$

where the first term is the same as Eq. 3 and the summation term accounts for the style ambiguity of the generated images by maintaining a large cross-entropy for $p(c|G(z))$. During training, the first term is present in the BCE loss function as before, and the summation term is included via a uniform cross-entropy distribution among all classes.

Apart from the changes in the architecture and the loss functions, the training procedure of the CAN is the same as the DCGAN, which was explained in detail in the previous section.

3.4 Conditional GAN Architecture

Following the approach in [11], we also implemented a CGAN architecture. The CGAN architecture is a further extension on the GAN, DCGAN, and the CAN architectures, where both the discriminator and the generator have access to the separate information about the inputs. With this approach, it is possible to condition the generator and evaluate the generated output at the discriminator based on the given condition. In our case, the objective of the CGAN is to learn to generate artwork given a specific artistic style. As always, the objective of the generator is to fool the discriminator such that it thinks the generated artwork is real, along with the false belief that the artwork actually belongs to the specific style that was given as input to both networks.

The discriminator and generator networks of the CGAN architecture are very similar to the ones of the DCGAN architecture. The only difference for the generator is that the random noise and the style labels (style encoding, $f : 27 \rightarrow 27$) are concatenated and used as the input to the generator network. The only difference for the discriminator is that the final convolutional layer is removed and the output of the remaining network [512, 4, 4] is flattened. Then, the flattened output and the style labels (style encoding, $f : 27 \rightarrow 4096$) are concatenated to be used as the input to a fully connected network. The architecture summaries can be seen in Tables 4 and 5.

Table 4: Generator Model Summary for CGAN Architecture

Layer	Operations & Parameters	Output Dimension
Input	Random noise generation + Style Encoding	[1, 150+27]
Deconv1	ReLU, BatchNorm, Kernel = 4, Stride = 1	[512, 4, 4]
Deconv2	ReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[256, 8, 8]
Deconv3	ReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[128, 16, 16]
Deconv4	ReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[64, 32, 32]
Deconv5	Tanh, Kernel = 4, Stride = 2, Padding = 1	[3, 64, 64]

Table 5: Discriminator Model Summary for CGAN Architecture

Layer	Operations & Parameters	Output Dimension
Input	Normalization, Resizing	[3, 64, 64]
Conv1	LeakyReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[64, 32, 32]
Conv2	LeakyReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[128, 16, 16]
Conv3	LeakyReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[256, 8, 8]
Conv4	LeakyReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1	[512, 4, 4]
Conv5	LeakyReLU, BatchNorm, Kernel = 4, Stride = 2, Padding = 1, Flatten	[1, 1024×2×2]
Input to FC	Concatenate Conv. Output and Style Embedding	[1, 1024×2 ×2×2]
FC1	LeakyReLU	[1, 1024]
FC2	Sigmoid	[1, 1]

The main objectives of the discriminator and the generator are nearly identical to DCGAN as well, with the only difference being that the probabilities are based on conditions. The discriminator is trying to maximize:

$$log(D(x|c = \hat{c})) + log(1 - D(G(z)|c = \hat{c})) \tag{7}$$

where the terms are the same as in Eq. 2, but conditioned on artistic styles. Similarly, the generator is trying to maximize:

$$log(D(G(z)|c = \hat{c})) \tag{8}$$

and is the same as in Eq. 3, but conditioned on artistic styles. The loss function is identical to the DCGAN architecture as well, the BCE loss.

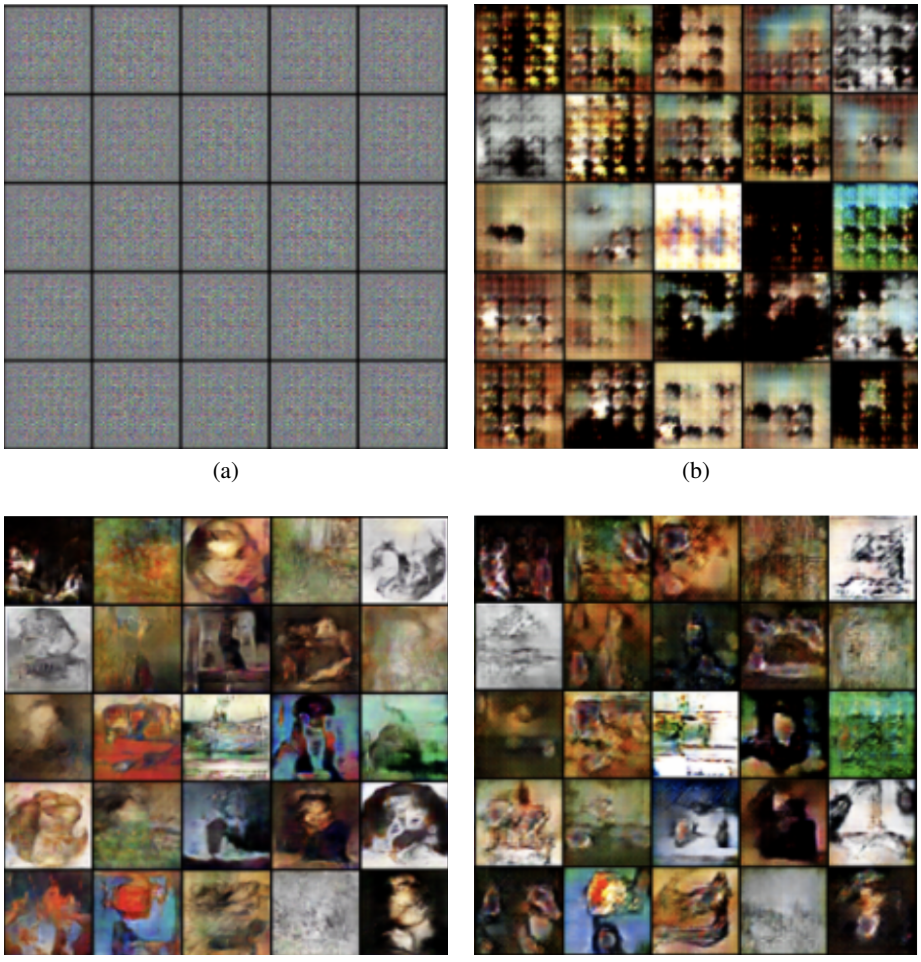
4 Results & Discussion

4.1 Baseline Model - Deep Convolutional GAN

Using the architecture presented in Tables 1 and 2, we trained our baseline model on the Best Artworks of All Time dataset. First, we re-scaled our training data to have a size of 64x64 pixels; all generated artwork images were also 64x64. We trained this model for 150 epochs, which took $\approx$ 1 hour using a single GPU. It was observed that the images generated in earlier epochs were essentially random noise whereas the images in the final epochs were aesthetically similar to the artwork of our training data, as shown in Figure 2. Thus, we can conclude that the generator model learned how to create images to fool the discriminator network.

We mainly evaluated our model based on training loss, along with visual inspections to do qualitative evaluation, as there is no quantitative metric to evaluate artwork. Figure 3 shows the loss for generator and discriminator models. As expected, the loss plot contains some oscillations between the generator and the discriminator, as they are in a competition and if one is performing well, the other is performing worse. The loss for the discriminator model stays about the same or slightly decreases during the training phase while the generator loss decreases in the first 2000 iterations. However, after 2000 iterations ($\approx$ 30 epochs), the generator loss starts increasing. Using the loss plot, we may conclude that as we trained our models, the generator model fell behind the discriminator. Despite this, the generator was still able to improve the quality of the images that it produced. Examining the generated images in Figure 2, we can see that the generator learned to create realistic artwork, by attempting to create portraits, natural images, and modern art pieces.

Although our generative model is able to create realistic art images, the issue of increasing generator loss may be due to over-training the model. However, in this case it seems that some over-training helps us to create better looking art images, as the results keep getting better after epoch 30. We also know that our training data consists of the artwork of various artists. For example, in our dataset, we have the artwork of Salvador Dali, whose art is considered to be Surrealist. We also have some paintings from Cubist artists such as Picasso and Expressionists like Munch. Having such a



(a) (b)

(c) (d)

Figure 2: Images generated by the baseline DCGAN after epochs 1, 30, 80, and 150.

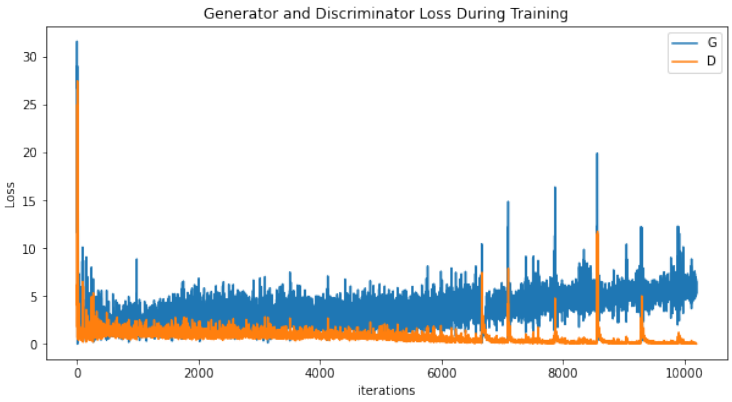


Figure 3: Generator and discriminator loss for the baseline DCGAN architecture.

dataset where we have images from different artistic styles might make it difficult for the generator to produce consistent artwork. Thus, the increase in generator loss may also have resulted from our diverse dataset. More results obtained after training on the WikiArt dataset can be seen in Figure

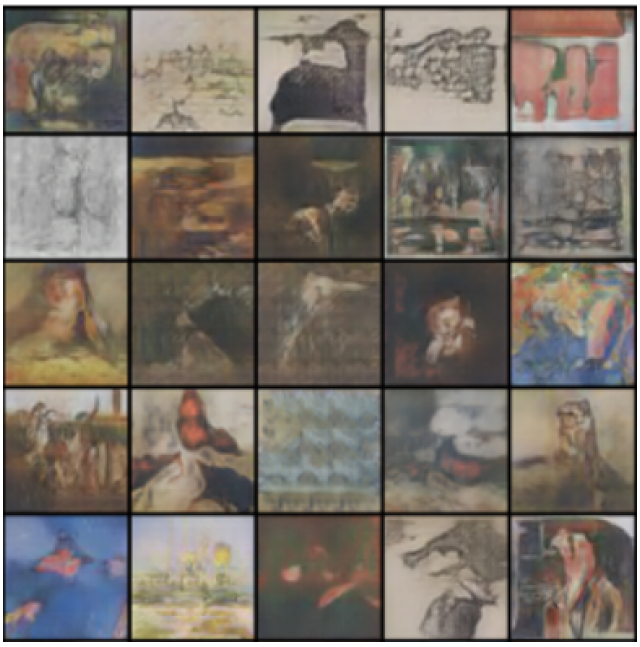


Figure 4: Images generated by the baseline DCGAN on the WikiArt dataset.

4. With the significantly larger dataset which also contains diverse styles, we can see examples of abstract and modern art.

4.2 Creative Adversarial Network

We used the WikiArt dataset [3] with 27 unique artistic styles to train the CAN architecture. Figure 5 displays some examples of the generator outputs of the CAN architecture.

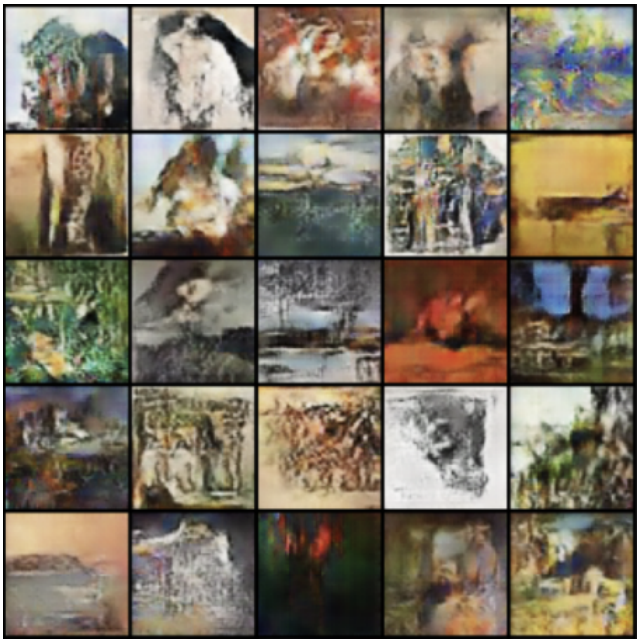


Figure 5: Style-ambiguous, creative artwork generated by the CAN architecture.

We can see that the CAN architecture managed to generate realistic artwork as well. However, it is difficult to assess the creativity of these results. The authors of [5] conducted a research study involving human evaluations for their results. We tried to quantitatively evaluate the creativity of the model by calculating the entropy of the discriminator classifications on fake images. Since the main objective of the CAN architecture is to generate style-ambiguous artwork, the entropy of the discriminator classifications must be high on fake images to justify the creativity of the model.

Figure 6 shows the normalized entropy of the classifications during training. It can be observed that the entropy is very high in the first epochs, which is expected since the output of the first epochs are just random noise and are classified randomly. The entropy is decreasing in the following epochs, however it is oscillating, which shows that the generator is trying its best to become more creative as it gets punished after correct classifications of the discriminator. In the end, the entropy of the model remains above 0.5, hence it can be argued that the model is style-ambiguous, or creative to some extent. Nonetheless, it is hard to claim that entropy is a valid quantitative metric to evaluate the creativity of an artwork generation model. Realistically, there is no meaningful way of assessing these results in terms of creativity, other than human evaluation which is still subjective at best.

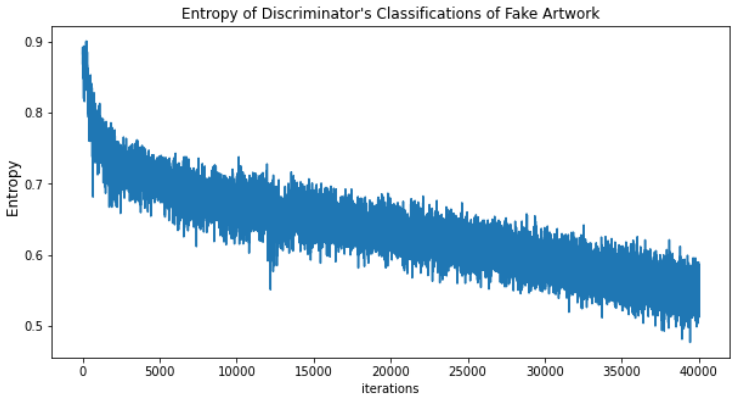


Figure 6: Entropy of discriminator classifications on fake images.

We also attempted to evaluate our model based on the loss functions as well. Figure 7 is the plot that shows discriminator and generator losses across iterations. It can be observed that the loss curves are similar to the ones of the baseline model in Figure 3, with the generator loss being slightly higher due to the additional punishment coming from the correct style classifications of the discriminator. We believe that these loss plots are a good indication of our CAN architecture being trained correctly.

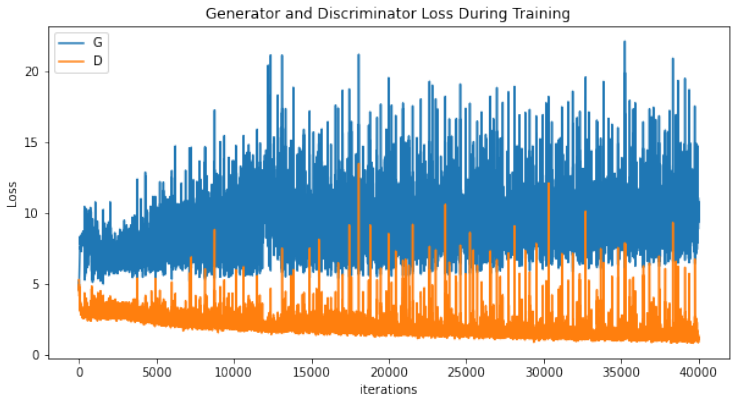


Figure 7: Generator and discriminator loss for the CAN architecture.

4.3 Conditional GAN

The WikiArt dataset [3] was used again to train the CGAN architecture with 27 different artistic styles. To generate images with specific styles, we implemented the CGAN algorithm as in the original CGAN paper [11]. The loss function in CGAN architecture is the same as in our baseline model. Initially, we implemented CGAN architecture using fully-connected layers and did not use any convolutional layers. Using fully-connected layers, our model suffered from the mode collapse problem that is common in GANs. Mode collapse occurs when the generator produces the same output or a small set of outputs repeatedly. To counteract this, we introduced a convolutional network architecture with batch normalization. Using a convolutional network architecture improved the quality of our images and we were able to start creating images in specific styles.

We also found that the addition of Gaussian noise to the inputs of the discriminator greatly enhanced our generated art. The added noise was zero-mean with an initial variance of 0.5, which was decayed with each epoch. This noise prevented the discriminator from overpowering the generator in the early training phase, keeping the two networks in balance and enabling the generator to improve its generated artwork. Increasing the batch size to 128 from our initial 64 also helped significantly in resolving the mode collapse problem even further. Since using a larger batch size increases the number of examples used in a given iteration, the gradients used to update parameters are computed on a more diverse sample of the dataset, which allows for better learning. Finally, we also experimented with Wasserstein GANs [12] to create art in different artistic styles. However, Wasserstein GANs did not improve the quality of the generated images and did not help with creating meaningful images in different artistic styles.

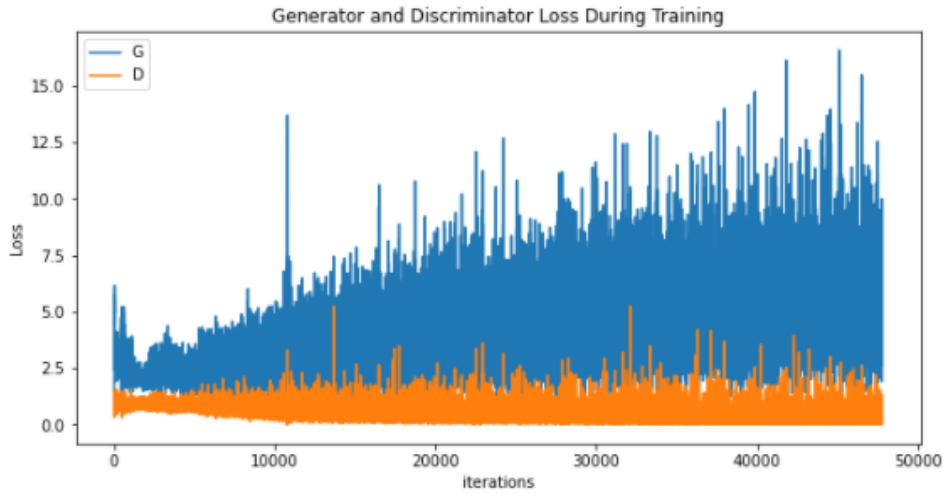


Figure 8: Generator and discriminator loss for the CGAN architecture.

The loss curves are shown in Figure 8 and are similar to those of our baseline DCGAN. The increasing loss of the generator suggests that the discriminator is starting to outperform it; this could be mitigated in the future by introducing further noise to the discriminator, either by increasing the noise that is fed into it or in the form of dropout. However, qualitative analysis of the produced artwork lead us to believe that training of this CGAN architecture was successful.

Example outputs for six artistic styles are displayed in Figure 9 along with samples from the WikiArt dataset. Comparing these qualitative results, our CGAN performed well in emulating most styles, with the Color Field Painting, Impressionism, and Rococo-styled outputs especially resembling those from the original dataset. In the Analytical Cubism style, mode collapse is observed. This is likely due to the small training set available for this style, with only 110 images; in contrast, the Impressionism style had over 13,000 images available. Please refer to Appendix A to view outputs of our CGAN in all 27 styles with WikiArt samples for comparison.

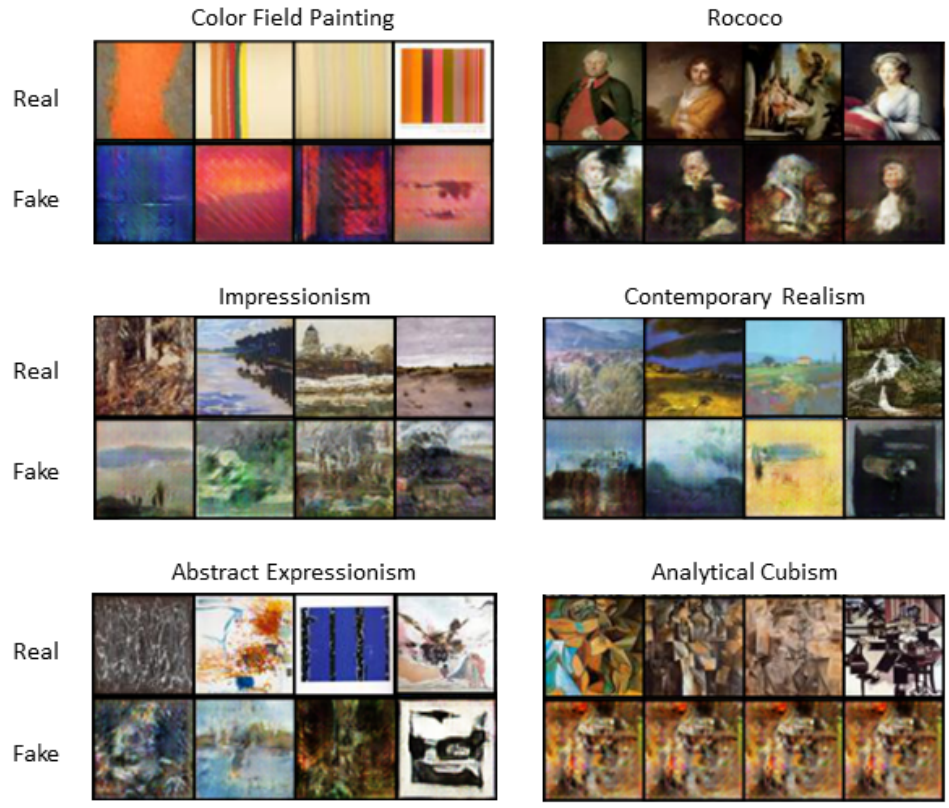


Figure 9: Examples of various artistic styles with "Real" WikiArt samples (top row) and "Fake" generated images (bottom row). Mode collapse is observed in the Analytical Cubism style.

5 Conclusion

We have demonstrated the efficacy of GANs in producing original artwork from scratch. Starting with our baseline DCGAN model, we generated realistic artwork that could fit within the original dataset. Extending this model to the CAN architecture, we then implemented a network capable of creating artwork that is ambiguous in style, and attempted to evaluate its creativity with a quantitative metric. Finally, this architecture was extended even further to implement the CGAN, which proved to be effective at generating artwork that is clearly identifiable in terms of artistic style. Through our experimentation, we were successful at achieving the stated goal: to explore the generation of style-agnostic art and the generation of style-specific art.

6 Individual Contributions

6.1 Onur Tepencelik

- Co-authored the implementation of the baseline model
- Implementation and experimentation of the CAN model
- Various sections of the report

6.2 Ismail Ocak

- Co-authored the implementation of the baseline model
- Implementation and experimentation of the CGAN model
- Various sections of the report

6.3 Sean Liu

- Co-authored the implementation of the baseline model
- Implementation and experimentation of the CAN model
- Various sections of the report

6.4 David Lu

- Co-authored the implementation of the baseline model
- Implementation and experimentation of the CGAN model
- Various sections of the report

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," *arXiv preprint arXiv:1406.2661*, 2014.

[2] "Best artworks of all time," https://www.kaggle.com/ikarus777/best-artworks-of-all-time.

[3] "WikiArt dataset," https://github.com/cs-chan/ArtGAN/tree/master/WikiArt20Dataset.

[4] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[5] A. Elgammal, B. Liu, M. Elhoseiny, and M. Mazzone, "Can: Creative adversarial networks, generating "art" by learning about styles and deviating from style norms," *arXiv preprint arXiv:1706.07068*, 2017.

[6] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," *arXiv preprint arXiv:1704.00028*, 2017.

[7] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," in *International conference on machine learning*. PMLR, 2017, pp. 2642–2651.

[8] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[9] A. Xue, "End-to-end chinese landscape painting creation using generative adversarial networks," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2021, pp. 3863–3871.

[10] "DCGAN tutorial," https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html.

[11] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[12] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*. PMLR, 2017, pp. 214–223.

Appendix A Complete Conditional GAN Outputs with WikiArt Samples