Style Transfer via CycleGAN

Group 4
Ioannis Linardos (s1866303)
Jayanth Chinthu Rukmani Kumar (s2216248)
Siba Siddique (s2356678)

***Abstract*—In machine learning, the study of generative adversarial networks (GANs) is currently one of the most exciting topics. In brief, in GANs, the generator and the discriminator compete in a game with each other with the goal of generating new data that can pass for real data. In this project, we will explore two types of GANs - Deep Convolutional GAN (DCGAN) and CycleGAN, with more emphasis on the latter. DCGAN is used to generate emojis, and CycleGAN is used to transform the style of one image to another. The results show that the cycle consistency loss has improved the results of the CycleGAN model. In the case of DCGAN, mode collapse was observed.**

***Keywords*—** Style transfer, CycleGAN, DCGAN, Adversarial Networks, Generative Models

Introduction

The present project concerns Generative Adversarial Networks (GANs), a state of the art generative deep learning model. The idea was introduced by Goodfellow et al. [1] and is inspired by the game theoretic idea of adversarial games. Here, we focus on a special GAN architecture called CycleGAN, which can be applied to image-to-image translation of unpaired datasets [2].

In this particular application, we will use two datasets of emojis: Windows-style and Apple-style emojis. The purpose is to convert between Apple-style and Windows-style emojis. As an intermediate development step, we will train a Deep Convolutional GAN (DCGAN) to generate emojis.

In the next section, we will discuss into the theory of GANs in more detail, focusing on using a CycleGAN for style transfer. Thereafter, we describe our vanilla GAN and CycleGAN implementations and present the results. The paper ends with a discussion of the results and the challenges we encountered.

Literature Review

Generative adversarial networks (GANs) were introduced in 2014, and have performed well in image generation [1]. Before that, deep generative models had the issue of approximating intractable probabilistic computations like maximum likelihood estimation. Therefore, there was a need of more advanced network that would overcome these issues. GANs solve this problem by replacing probabilistic computations with an adversarial game between two networks. The first one is the generator G which generates the data and the second one is a discriminator D which computes the probability that a data point is real. During training, the generator tries to "outsmart" the discriminator and vice versa. In this process, both models become better.

Mathematically speaking, the discriminator D aims at maximizing $D(x)$, where x is a real sample, and minimize $D(G(z))$, where z is a random vector that is used as an input to G to generate a fake sample $G(z)$. On the other hand, G is trained to maximize $D(G(z))$ (or equivalently minimize $1 - D(G(z))$). The combination of these two processes is called adversarial loss. In practice, we minimize the logarithm of these numbers because it helps with the computations (it transforms products into sums). This leads to a two-player zero-sum game. Ideally, the discriminator will always return 0.5, meaning that it cannot distinguish between real and fake data.

Generative Adversarial Networks can also be adapted to be used in image-to-image translation. In this case, the aim is not to generate new data but to transform a given image to another one. For example, photos of horses can be transformed to photos of zebras and vice versa [2]. In the past, this has been done by using paired datasets. In this example this would mean having horses and zebras in the same locations. However, paired datasets are rarely available because they are difficult to gather.

Cycle GANs is a new model that overcomes the issue by allowing the use of unpaired datasets X and Y [2]. It typically involves two mapping functions

$G : X \rightarrow Y$ and $F : Y \rightarrow X$ that receive input from one dataset and translate it to the other; they are playing the role of a conditional generator. Moreover, there are two discriminators D_X and D_Y that calculate the probability that a sample belongs to the dataset X or Y respectively.

Each of the generators G and F with their respective discriminators D_Y and D_X can be trained separately in pairs as vanilla GANs using adversarial loss, in the manner that was explained above. The innovation in the cycleGAN model is the use of cycle consistency loss [2]. The idea is that G and F are essentially inverse mappings of each other. So, $G(F(y)) = y$ and $F(G(x)) = x$. Therefore, the two separate GANs can be trained together minimizing the difference between the input and the output. Essentially, it is similar to training an autoencoder which transforms the input to an intermediate stage, which has a different style, before transforming it back to the original.

The combination of these two types of loses is what makes the Cycle GANs successful. The adversarial loss of each pair works towards making the transformed data come from the required distribution and the cycle consistency loss constrains the output so that it does not diverge too much from the input.

Research Questions

The research questions are structured as follows:

1) Does the proposed cycle GAN architecture work in style transferring with the given datasets?
2) How important is the use of cycle consistency in the Cycle GAN model?
3) What challenges appear in the process?

Methodology

Dataset

The training set consists of 4284 emojis, split about equally between Apple-style and Windows-style emojis. The size of each emoji is $72x72$ pixels RGB. Although there are a lot of similar emojis, there is no one-to-one correspondence, namely the two datasets are not paired.

DCGAN

We first trained a vanilla Deep Convolutional GAN (DCGAN) to generate images using only the Apple-style emojis dataset. The network is called deep convolutional to describe the generator's and discriminator's architecture.

The discriminator was a 2D CNN consisting of 4 convolutional layers using the ReLU activation function. Between each layer there is a batch normalization layer and in the output there is a fully connected layer and a sigmoid activation function. The channel sizes were as follows $3 \rightarrow 32 \rightarrow 64 \rightarrow 128 \rightarrow 1$. The kernels are $4x4$ convolutions and the stride is 2. Two zeros were padded in the borders so that at each layer the data are downsampled by a factor of 2. This is a common architecture in deep models for image classification. The emoji's are not very high resolution, meaning that the network need not be very deep. Deeper architectures would suffer from problems like vanishing gradient or overfitting.

The generator was a deconvolutional or transpose CNN, a network that consists of layers that perform transpose convolution operations. This is the inverse of the convolutional operation; it upsample the data. The input was randomly generated noise from a $U \sim [-1, 1]$ distribution of length 100. All in all there were four deconvolutional layers with ReLU activation function. The channel sizes were as follows $100 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 3$. The kernels were again $4x4$ convolutions with stride 2 and padding 1.

During training, we used the sum of squares generator and discriminator losses respectively. This loss function tries to minimize the sum of squared differences between the real and the target value. The weights were updated using the Adam method. The weights of the generator and discriminator were trained at different steps, each using its own part of the adversarial loss as we described, but they were both updated at every iteration. This is a design decision that may have consequences in regards to the convergence of the network as it is discussed below.

Cycle GAN

The Cycle GAN model consists of two generators and two discriminators. The Windows and Apple discriminators both have the same architecture as the discriminator network as the vanilla DCGAN

since they perform the same operation essentially. Each discriminator aims at learning to discriminate between emojis that belong to the original distribution (Apple or Windows). Moreover, there are a Windows-to-Apple and an Apple-to-Windows generator.

The two generators have the same architecture. Each generator consists of three parts, an encoder, a transformation block and a decoder. The encoder part consists of two convolutional layers. The channel sizes are $3 \rightarrow 32 \rightarrow 64$, the kernels are $4x4$ with stride 2 and padding 1. Its purpose is to encode the essential features of the input. The transformation part is were the transformation of the features from the one style to the other happens. It consists of a residual block, a 64 channel convolutional layer with $3x3$ kernels (padding and stride 1) with a skip connection. The decoder decodes the transformed features to the final output using two deconvolutional layers with sizes $64 \rightarrow 32 \rightarrow 3$ with $4x4$ kernels (stride 2 and padding 1). The activation function in all the layers is ReLU and between convolutional and deconvolutional layers we use batch normalization.

In the Cycle GAN, there are multiple loss functions to take into consideration. At first, four losses were considered; two discriminator and two generator losses trained in pairs as adversarial losses. Namely, the two generators are trained separately as we explained. Once more, we used the sum of square losses and the weights were trained using the Adam optimizer to update the generator and discriminator weights. The weights are updated at different steps but at every iteration. Afterwards, the innervation that makes cycle GANs successful was implemented, that of cycle consistency. The idea is that if an an Apple emoji passes first through the Apple-to-Windows generator and then through the Windows-to-Apple generator (or conversely with Windows emojis), the end result should be the same. This gives rise to two more loss functions, Windows $\rightarrow$ Apple $\rightarrow$ Windows and Apple $\rightarrow$ Windows $\rightarrow$ Apple, where the sum of squared differences between the twice transformed and original images should be the same. We will try to train the GAN both with and without cycle consistency to compare the results.

RESULTS

DCGAN

In the Figures 1, 2 and 3 some samples of generated images are presented after 200, 2400 and 5200 training iterations respectively. It can be seen that the generator failed to produce images that resemble emojis. In the early iterations, the generated images were darker, something that started to change in the later iterations. It could be proposed that, as the model was trained more, more colors appeared in the generated samples. This means that there was some form of training going on and that the problem of vanishing gradients did not appear. However, we can observe the problem of mode collapse; all the generated samples are essentially the same. In any case, the result exhibit the difficulty to successfully train a generative model.

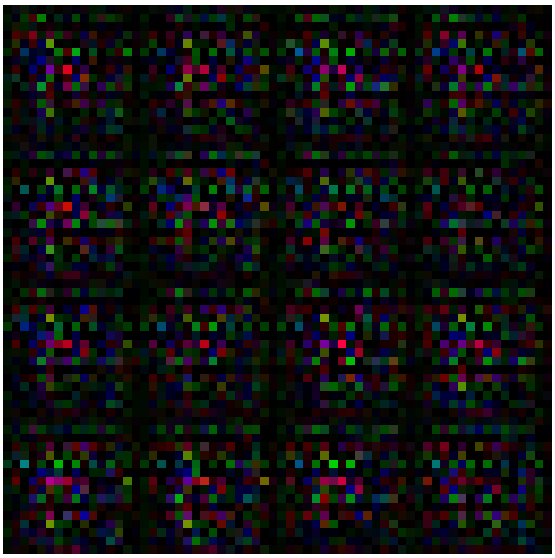

Fig. 1: Sample of generated images after 200 iterations

These results led us to try a new approach in which we trained the discriminator with a lower learning rate. In particular, the learning rate of the discriminator was 0.01 times that of the discriminator. We did that to avoid the problem of having a pessimistic discriminator. We can see the generated samples in Figures 4, 5 and 6. The problem of mode collapse was lessened in this case but the results are not much better.

Cycle GAN

In Figures 7 and 8 we can see the style transfer from Apple to Windows and from Windows to

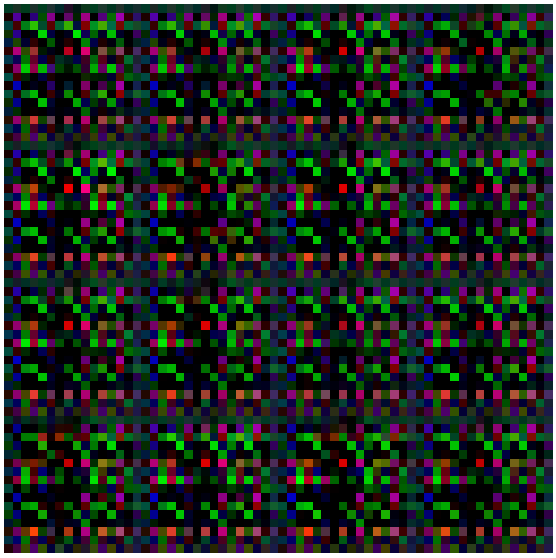

Fig. 2: Sample of generated images after 2400 iterations

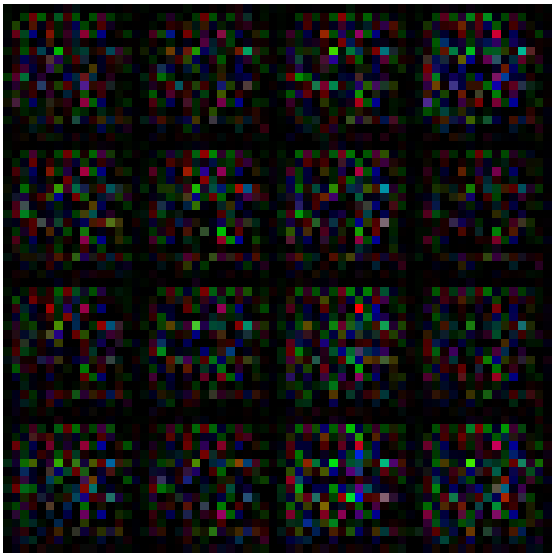

Fig. 4: Sample of generated images after 200 iterations with smaller learning rate in the discriminator

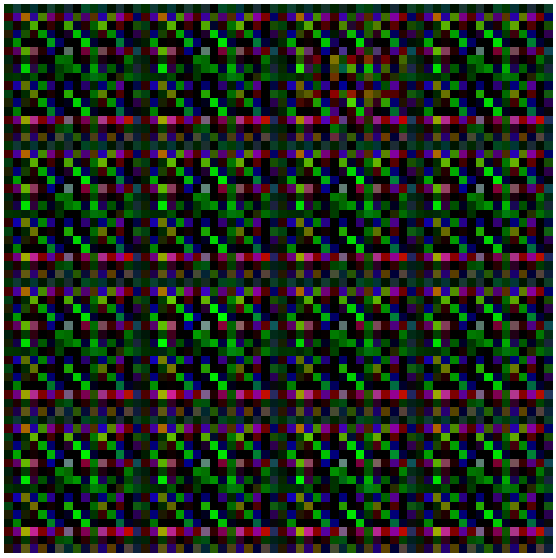

Fig. 3: Sample of generated images after 5200 iterations

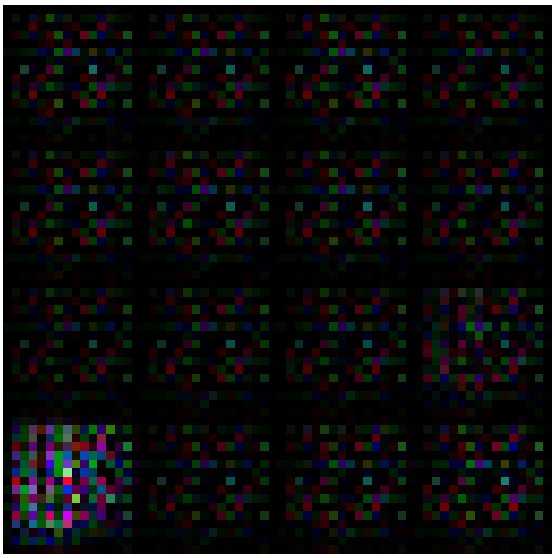

Fig. 5: Sample of generated images after 2400 iterations with smaller learning rate in the discriminator

Apple respectively when the model is trained for 600 iterations and cycle consistency is not used. The exhibit shows that the samples are very noisy but there is some vague resemblance with the original image, mainly in terms of colors.

In Figures 9 and 10, the results after 100 iterations of training with pretrained weights are presented without cycle consistency. The weights have been trained for 40000 iterations and therefore they converge much faster. The results are much clearer; in most cases the transformed emojis display distinctly the same picture as the original.

In Figures 11 and 12 the first results with cycle consistency loss are presented. These samples were generated by models trained for 600 iterations. The results are already significantly better than without the use of cycle consistency and of comparable quality as with the pretrained weights. We can observe that in the case of Windows to Apple translation the purple colour appears quite often. Moreover, in the Apple to Windows translation, the green colour appears often. This may be a form of mode collapse where the generator learnt that this feature ”fools” the discriminator.

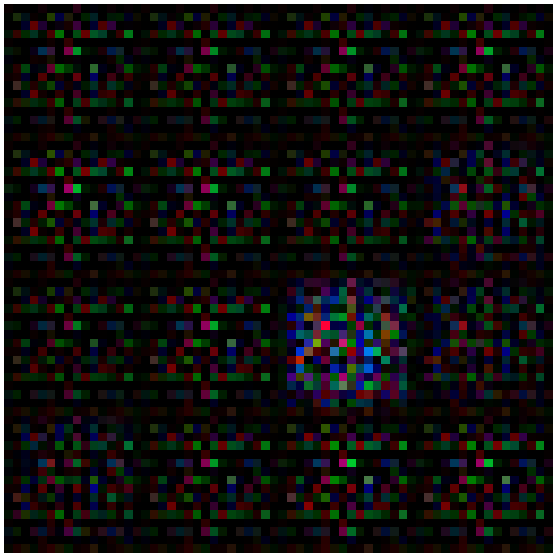

Fig. 6: Sample of generated images after 5200 iterations with smaller learning rate in the discriminator

In Figures 13 and 14, we can see the results of using cycle consistency loss alongside the pretrained weights (trained for 40000 iterations). These results are in most cases the clearest whatsoever. The generated images show an obvious resemblance with the original while not being the same (something expected since we intend to transfer the style anyway). However, the results are still blurry.

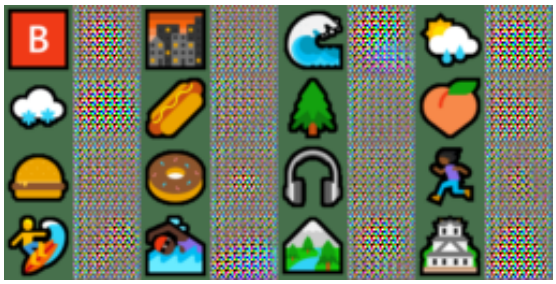

Fig. 7: Windows to Apple style transfer without cycle consistency after 600 iterations

Discussion

Cycle Consistency Loss

It can bee seen at the presented results that the use of cycle consistency loss helps significantly with the network convergence. In style transfer, we expect the end result to resemble the original. Therefore, it makes sense that minimizing the distance between the twice transformed picture and the original will be a useful constraint of the model. Similar results

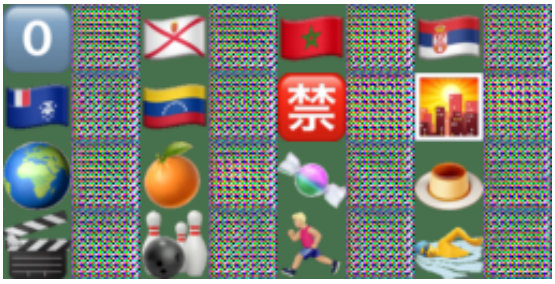

Fig. 8: Apple to Windows style transfer without cycle consistency after 600 iterations

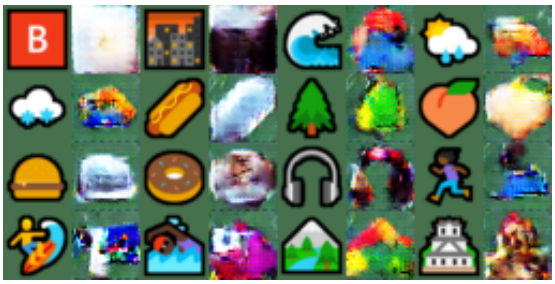

Fig. 9: Windows to Apple style transfer without cycle consistency after 100 iterations with pretrained weights

seem to appear with higher numbers of training iterations. Thus, it could be concluded that cycle consistency speeds up the model's convergence.

On the other hand, it could be argued that cycle consistency is in fact too restrictive to the model's creativity. As it was expected, the results of using cycle consistency are more loyal to the original while the results of training for more iterations exhibit some divergent features. Therefore, the use of this loss function should be considered per case depending on how much we desire to diverge from the original dataset. In the case of emojis style transfer, the effect seems to be more positive than negative.

GAN Challenges

In our first effort to train a vanilla DCGAN, we experienced first-hand the problem of mode collapse because all the samples were essentially identical. However, the Cycle GAN model seems to be less vulnerable to this problem. The main reason seems to be the fact that it is a conditional GAN, it does not receive noise as input and the output will in one form or another resemble the input, especially considering the fact that the generators are not very deep networks.

Fig. 10: Apple to Windows style transfer without cycle consistency after 100 iterations with pretrained weights

Fig. 11: Windows to Apple style transfer with cycle consistency after 600 iterations

In regards the problem of convergence, the theoretical guarantees are difficult to practically verify. However, we can see that the results seem to improve with more training (see pretrained weights) which hints at the model converging. Moreover, as we discussed previously, the use of cycle consistency loss also seems to help. One thing that could be better tuned in our model is not training the discriminator at every iteration because this tends to lead to a pessimistic discriminator which may cause vanishing gradients.

A qualitative problem that we encountered is the difficulty in evaluating the models, specifically in comparing the generated results in the relevant section. In this case, it was almost inevitable to resort to empirical approaches based on our experience with emojis.

Conclusion

Generative Adversarial Networks have achieved interesting results in image generation. Cycle GANs are a variation of GANs that can be used to transfer image styles between unpaired datasets. In this project, we tried to implement this model in emoji style transfer between Windows and Apple emojis. The model was tried with and without cycle consistency loss and for different iterations. The model seems to improve with further training while the use of cycle consistency seems to improve the results.

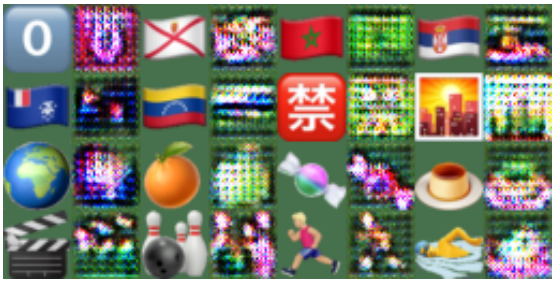

Fig. 12: Apple to Windows style transfer with cycle consistency after 600 iterations

Fig. 13: Windows to Apple style transfer with cycle consistency after 100 iterations with pretrained weights

Fig. 14: Apple to Windows style transfer with cycle consistency after 100 iterations with pretrained weights

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[2] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.