IconiGAN: A Machine Learning Project

Marco Manco
s281564
Politecnico di Torino
s281564@studenti.polito.it

Alexander Carlo Gustavo Spira
s287560
Politecnico di Torino
s287560@studenti.polito.it

Lorenzo Valente
s281995
Politecnico di Torino
s281995@studenti.polito.it

Abstract

A logo is one of the most important parts of a brand. It represents the original and unique symbol that reflects the value of the brand and creates a first impression on a customer. Small or large, whatever the proportion they may be, all activities - industrial,craft, commercial - shops, service firms, professional firms - all need a compelling, strong and recognizable logo that helps the business have more success, which is why creating an effective brand identity is essential. What we created is a Generative Adversarial Network [9] that can be used to generate random icons that can be used as inspiration or as a starting point for the creation of an original logo. In particular we have taken advantage of using convolutional layers to create a Deep Convolutional GAN (DCGAN) [5] that was able to analyze and extract high-level features from our dataset. Starting from a large pool of logos, the network develops various and heterogeneous examples which cover a lot of shapes and colors, from geometrical structures to figures that recall little objects, animals and plants.

1. Introduction

Designing a logo for a new company or updating the one existing is not an easy task since the process usually is very long and complicated. The problem is that the logo should be original and have the right meaning for the company. For this purpose a Generative Adversarial Network (GAN) falls perfectly in this area since it is capable of generating almost anything we want, from MNIST digits [9] to cities [10] and fake celebrities [11]. We trained it to create logos starting from icons that have a more clean and repetitive structure than a logo which often contains the brand name. Logo creation case has not yet been fully explored and studied in a way that it can be considered standardized or proven successful. This is probably because a logo is an artistic result, so it isn't dictated by rules or some kind of order of lines and curves which the model can learn and try to recreate. As a consequence the latent space is not a continuous one, so not every logo will be aesthetically pleasant.

For what concerns the architecture, before we cited the GAN that, in a few words, is an architecture that has two different neural networks that allows it to learn and imitate the distribution of the input data. In our case we used not a simple GAN but a DCGAN which is a Deep Convolution GAN. It uses Deep Convolutional networks instead of the fully-connected networks of the GAN. This particular change makes the network more suitable for images and videos that can exploit spatial correlations. But these networks usually in the long run have an increasing error, so we opted to use residual blocks.

Overall the model structure is based on deconvolutions, strided convolutions, and residual blocks. With the upResBlock module for the generator of convolutional GANs, our method can further enhance the generative power of the feature extraction while synthesizing image details for the specified size. With the downResBlock module for discriminator combined with upResBlock for generator, the proposed method can speed up the back-propagation of gradients and not suffer from the vanishing or exploding gradients problems.

2. Data

The dataset used for the training was the LLD-icons dataset [7]. It consists of 548,210 standardized 32 x 32-pixel favicons, that were crawled on April 7th 2017. We used a sample of around 5000 of these icons for our project because we wanted to have a great variety of colors and forms without compromising too much the time to run the tests.

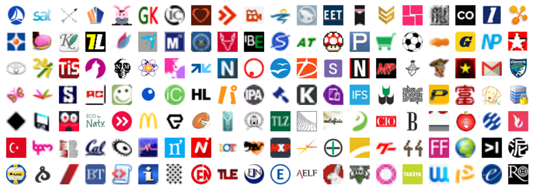
Figure 1. Sample of 5kLLD Dataset [7]

This dataset of icons instead of logos (like Logo-2k) [6] was chosen because a logo in general includes the brand name, and in our case having alphanumerical elements was a problem. This is because they were affecting our results by having high frequency variance in the colors of near pixels. So the perfect logo would have been one with low frequency variance for near pixels, for example something simple as a single letter or a simplified representation of something.

2.1. Data Augmentation

We used some data augmentation to support and strengthen our nets. In fact, we introduced some basic rotation, zoom and shear transformations to diversify the dataset's logos (rotation range = 0.125, shear range = 0.1, zoom range = 0.25), but also horizontal and vertical flip because they didn't affect negatively our samples like it would have in other applicative fields (e.g. face, animals, etc.), instead they reinforced them. Lastly, we used fill mode as "nearest" to complete the augmentation.

3. Methods

We started creating a DCGAN following the original paper [5] and then modifying the model following some other projects. At first we looked at the GitHub project GANs-Logo-Generator [1] and at the pokemon GAN project on GitHub [2] to understand the layer configurations and other things like filter dimension and stride for both the generator and the discriminator. After we tried various other configurations and studied the quality of the resulting images and the generator error.

Among these experiments, we identified some hyperparameters to change such as batch, filter size, input noise size and number of total epochs.

To understand if our approach was giving good results, we had to run a few hundred epochs and then look at the results. We then chose the best configuration where the outcomes were considered good and diversified. Also, we looked for a configuration where the generator loss didn't explode but it remained constant during all the process.

3.1. GAN

This framework consists of two neural networks, a generator and a discriminator, which compete simultaneously with each other. The generator takes a noise vector and it tries to recreate a correct image without knowing how the dataset is set up. The discriminator analyzes images from the generator's samples and from real ones.

Development is given by a *minmax* process where the framework updates the Discriminator's weights (locking the Generator's ones), subsequently Generator updates its own weights with the back-propagation of the errors.

3.2. DCGAN

While classic GAN mainly used fully connected models (which involved a great number of parameters), a new approach was sought and many key-ideas were changed. First of all, FC layers were replaced with convolutional layers, then max pooling was changed with convolutional stride. Also the activations' function were switched with ReLU in hidden layers, Tanh for Generator's output and lastly LeakyReLU in the Discriminator.

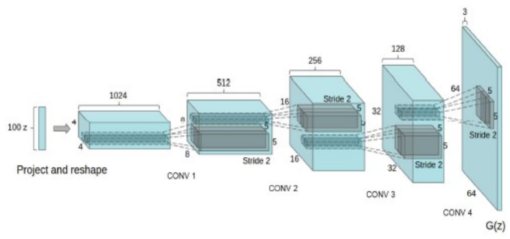

Figure 2. DCGAN generator architecture [5]

3.3. Residual Blocks

In traditional neural networks, each layer feeds into the next layer. In a network with residual blocks, each layer feeds into the next layer and directly into the next activational layer. In fact, in the traditional nets there is a limit to the number of layers added that results in an accuracy improvement. [4] [8]

If we have sufficiently deep networks, we can occur in vanishing or explosive gradients. Also, if we keep increasing the number of layers, we will see that the accuracy will saturate at one point and eventually degrade. This phenomenon is caused by *overfitting*. With Residual blocks, hidden layers could learn as fast as the final ones, giving us the ability to train deeper networks.

3.4. Model and Loss

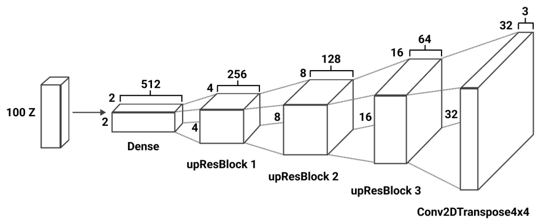

Figure 3. Generator architecture

GANs exploit Adversarial Loss, where one component tries to minimize it and the other one increases it.

This loss is formed by two parts: the first tranche is updated when dataset images are labeled as real while the second one is increased when generator samples are treated as real.

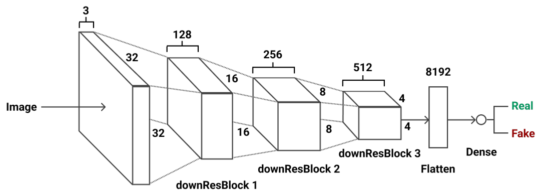

Figure 4. Discriminator architecture

$$\min_{G}\max_{D} V(D,G)$$

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

Figure 5. Minmaxlite

Inspired by ResBlock, we propose upResBlock and downResBlock by combining deconvolution, strided convolution, and ResBlock.

The motivation behind up and down residual blocks is to utilize the upsides of ResBlock, i.e., creating pictures rapidly from noise and more efficient backward propagation of gradients.

The model was trained on Colab with GPU set as a hardware accelerator, with the 5kLDD dataset. The time required to train the final model was 1 hour and 20 minutes.

3.5. upResBlock

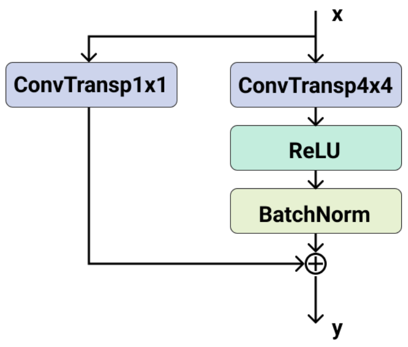

Figure 6. upResBlock

As illustrated in Figure 6, the upResBlock allow making full use of the information of the former layers. In fact, our upResBlock could make all information pass through and make backward propagation of gradients more efficiently with the help of shortcut:

$$y = B(R(C(x))) + C(x) \tag{1}$$

where y indicates the output of the layer considered; C means the convolutional layers; R means the the activation function ReLU and B indicates the batch normalization.

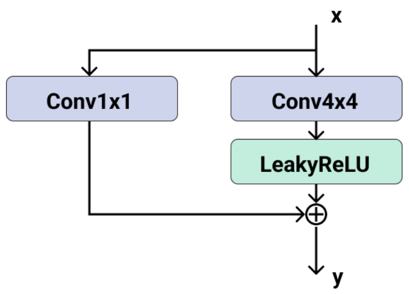

Figure 7. downResBlock

3.6. downResBlock

To make it possible for a discriminator to preserve the full features, especially the useful features, which increases the difficulty of the convergence of loss, we introduce our downResBlock. The structure of this block can be formulated as:

$$y = L(C(x)) + C(x) \tag{2}$$

where L indicates the activation function LeakyReLU and C is same as Equation 1. Through such a construction, our downResBlock could save the feature from previous layers which helps the convergence of loss.

3.7. High-level changes

Starting from the paper's model, we have performed some high-level changes to adapt the network to our needs: in particular during the first phase, we tried to add some convolutional layer with stride equal to one to increase the complexity of our nets but the results were poor and therefore changed our approach. In the second phase, we introduced residual blocks and these brought us positive outcomes.

We also thought of using a simpler GAN but we encountered an underfitting problem so we discarded it.

3.8. Implementation Problems

We faced mainly two problems: mode collapse and loss explosion.

"A mode collapse can be identified when reviewing a large sample of generated images. The images will show low diversity, with the same identical image or same small subset of identical images repeating many times. A mode collapse can also be identified by reviewing the line plot of model loss. The line plot will show oscillations in the loss over time, most notably in the generator model." [12]

We have solved this problem by adjusting the ratio between generator and discriminator (a good scale that was found is 3:1). In fact the discriminator was too weak so we improved it.

Another problem was the loss explosion that caused an instability in the generator and led to undefined and abstracted images.

Residual blocks were proposed to resolve them. Through residual blocks, networks are easier to optimize and could gain higher performance in a variety of tasks. It's good to note that the Residual Block can't change the size of the feature map. Therefore, we propose two novel blocks dubbed upResBlock and downResBlock.

4. Experiments

Before experimenting we looked at what projects had already been done with similar objective. The first one we looked at was the GitHub project GANs-Logo-Generator [1] since the objective was similar to ours. For what concerns the results of this project we think that, even if the training was done for more than 6 days with 230 000 epochs and with 256 of batch size, the generated results were incomplete and very simple. They aren't as well defined and sharp as some of our cases. Since we were not satisfied with this, we looked for something else and found the pokemon GAN project on GitHub [2] which instead had very defined and sharp result images but the dataset was smaller than ours. This approach took great advantage of data augmentation but the thought behind this project was right and we followed it.

4.1. Changing Dataset

As previously said, the first experiments were conducted using a DCGAN that was following the model from the original paper of the DCGAN [5] and after that started with adjustments coming from different other projects. One of our first experiments was to find the right dataset because we didn't know what to look for and what would have been the best one. After using the Logo-2k, which was the first we picked, we immediately understood that we needed images with very few, or even better, with no writings in them. This is because the high frequencies of colors and lines were very difficult for the network to study and replicate. The results were very bad images since the model was trying to recreate a word that wasn't composed of letters but by shapes, as you can see in Figure 8.

Even with some changes to the network, the results didn't get any better so we came to the conclusion that the problem were the input images.

So after looking for other dataset we came across the LDD-icon that had very few images with writings and, in general, with alphanumerical elements. Immediately the results improved, giving more geometrical shapes that were looking like logos. The problem was that the resulting images were too vague, not defined and didn't resemble figures of any type, as you can see in Figure 9.

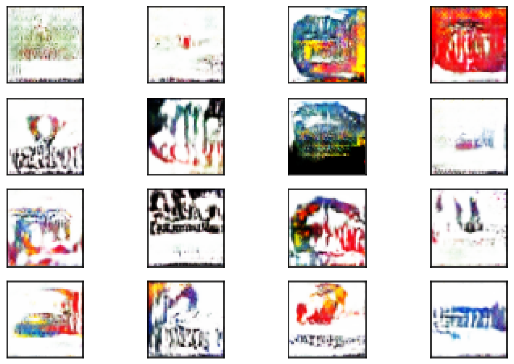
Figure 8. Image with logo-2k

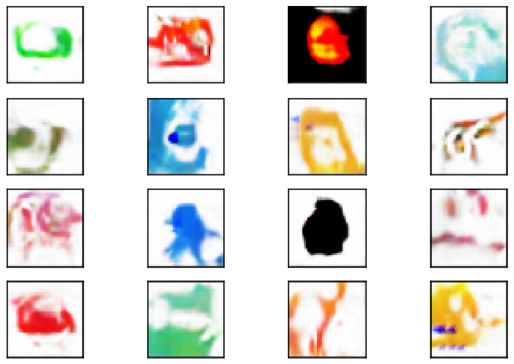
Figure 9. First image with 5kLLD

Not only were the images not good enough, but also the error of the generator was immediately exploding after a small regression, and in general was pretty high. It was close to 2 when it should have been considered stable and then went to 2.75 after the increment as shown in Figure 10.

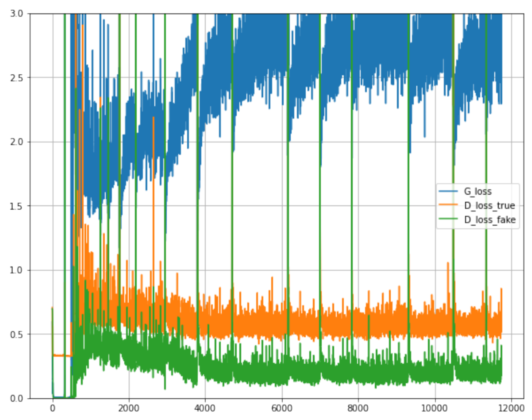

Figure 10. Losses of an early experiment

So, we understood that something was wrong and we started searching how to train a stable GAN. After looking online, we discovered that the generator error should have had a standard trend: after a rough start it should decrease and keep stable for a while and then increase again. Also the discriminator error should have followed a trend which was a slight increase and then either keep stable or decrease. [12] In our case this was already correct.

4.2. Changing Hyperparameters

One of the first attempts was to modify hyperparameters and the model complexity. Increasing the complexity of the model was needed because we had a lot more data of the projects with which we were starting, for example the pokemon GAN had only 819 images while we used little more than 5000 samples. By increasing the complexity we mean to make more layers or change the layers values so that there are more parameters. But this process had to be done carefully, because both the generator and the discriminator had to evolve together. If just one was strengthened the whole model would have been unbalanced and the results would look terrible. This is because the two components would not collaborate and just make one of the two errors explode depending on which one was less powerful. The best ratio of the number of parameters between the Generator and Discriminator was 3 to 1. Of course, depending on how many parameters, the time for the training and the time for convergence could either increase or decrease.

In this case, since we had too many images, the solutions were two: increase parameters or train longer. The first solution was too time consuming for our resources, so we opted to increase the training process by increasing the epochs. We went from 150 epochs to 350 and sometimes 400. So, the process was taking the same time as before (less parameters but more epochs) and was giving us optimal results for both the generator error and for the resulting images.

Along the way we experimented with different improvements. Among these we found that having the filter size as a submultiple of the image size was avoiding the blockiness of the results, especially in early stages. In fact, paper implementation had 5x5 filters while we used 4x4 and 2x2. We also started by working with batches of 64 and just later found that it is better to have it set to 128.

Before we used the residual blocks we encountered, for different configurations, similar problems. The results as previously said were too random, which means that there wasn't a dominant shape or some kind of order for the colors.

4.3. Results

The final model, as shown in Figure 11, has an error that follows the trend explained earlier.

In addition, after 350 epochs, you can see the final images generated in Figure 12.

The images we selected as example may appear blurry because of the low resolution of 32x32 pixels which is the resolution of the images of the dataset used for the training. Other than that some of them, as previously said, can recall, with some fantasy, to some real object or living beings stylized and simplified. Some can be seen as a negative of the background since some logos originally also have a background color. Sometimes there are too many details that, given the low resolution, appear to be like a splash of colors or vice versa appear to be nothing because the figure is thin or composed with a similar color of the background. Most of the time there are circular figures with some kind of shape inside that could be compared to an alphanumeric value.

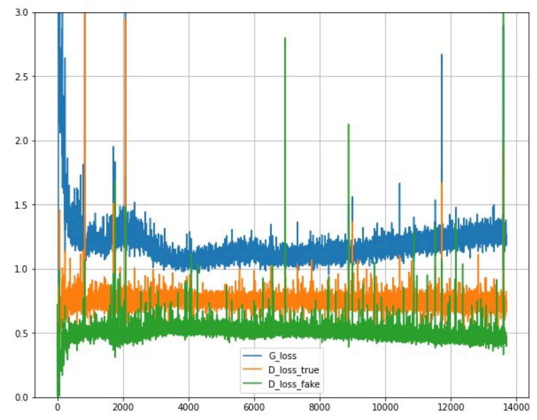

Figure 11. Final Model Losses

5. Conclusion

Although a logo is an artistic result and it isn't dictated by rules or some kind of order of lines and curves, the model has learned how to extrapolate some information and to recreate some good outcomes. Obviously, not all the creations are aesthetically beautiful but we can say that our goals have been achieved.

It was a long and complicated work because we had to recreate a GAN that worked correctly and that had a good relationship between the number of parameters of the generator and the discriminator. In fact, unbalanced networks produced only poor images and the errors were too high.

After we found the right ratio, we focused on the image quality and network error. At first, we saw how the wrong dataset produced unpleasant images. So, we started looking for the dataset that would help us the most. But this was not easy at all because there is not a perfect one if not one created by yourself according to your needs. At the end, we found a functional compromise and we focused on network optimization.

In class we saw how the invention of resnet had led to the creation of deeper and deeper networks but with contained errors. We have therefore exploited the theoretical notions

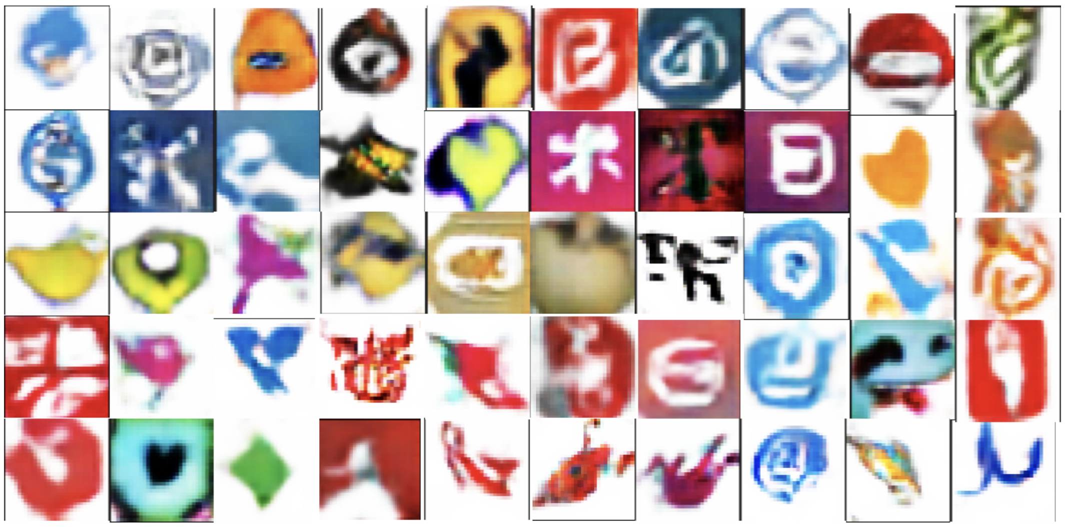

Figure 12. Generated logos

to develop residual blocks that have brought excellent results.

5.1. Future Extensions

Our goal was to create a network that works best on both the results and the loss side. We also thought about what could be improved further. An interesting idea would be to condition the network by the shape of the logo (e.g. spherical, square, rectangular) and by color (e.g. red, yellow, etc.).

It would also be interesting to have logos generated according to the company name (if the name is Lamps the generation would be influenced by that and try to create something accordingly to what the name represents) or business context (like if the company is of some sort of high quality clothing brand or a steel factory).

Some updates could be done by increasing the resolution to something more standard like 256x256 pixels, by improving the complexity of the network with longer training and with more parameters supported by more expensive hardware.

References

[1] GANs-Logo-Generator GitHub project: `https://github.com/NVukobrat/GANs-Logo-Generator`

[2] Pokemon GitHub project: `https://github.com/kvpratama/gan/tree/master/pokemon`

[3] Tips for training GANs: `https://machinelearningmastery.com/how-to-train-stable-generative-adversarial-networks/`

[4] ”Up and Down Residual Blocks for Convolutional Generative Adversarial Networks”: `https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9344669`

[5] ”UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS”: `https://arxiv.org/pdf/1511.06434.pdf%C3%AF%C2%BC%E2%80%B0`

[6] Logo-2k website: `https://paperswithcode.com/dataset/logo-2k`

[7] LLD-Icon website: `https://data.vision.ee.ethz.ch/sagea/lld/`

[8] Residual Block explanation: `https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec`

[9] ”Generative Adversarial Nets”: `https://arxiv.org/pdf/1406.2661.pdf`

[10] This X Does Not Exist website: `https://thisxdoesnotexist.com/`

[11] ”PROGRESSIVE GROWING OF GANS FOR IMPROVED QUALITY, STABILITY, AND VARIATION”: `https://arxiv.org/pdf/1406.2661.pdf`

[12] Practical guide to GAN failure modes: `https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/`