Generating logo images with Generative Adversarial Networks

Hristo Todorov
HSMS 'Prof. Emanuil Ivanov'
Kyustendil, Bulgaria
httodoroff@gmail.com

Hristo Kanev
HSMS 'Acad. Boyan Petkanchin'
Haskovo, Bulgaria
hristokanevkrash@gmail.com

Dr. Stoyan Vellev
SAP Labs Bulgaria
Sofia, Bulgaria
stoyan.vellev@sap.com

July 11, 2020

ABSTRACT

The aim of this paper is to summarize our research on one very difficult problem for neural networks – creating images requiring a great amount of creativity, such as logo images. By using generative adversarial networks and clustering we are striving to achieve diversity of the results and grouping into categories based on common features. We will try to clarify and address the main challenges in creating effective algorithms for solving this problem and to determine the applicability of the contemporary machine learning techniques to a harder challenge requiring creativity and originality.

1 Introduction

One of the most interesting subjects in Deep learning are definitely the Generative Adversarial Networks (GANs for short) [**goodfellow2014generative**]. The French-American scientist Yann Lecun described them as "the coolest idea in machine learning in the last 20 years". One of their main applications is image generation. There has also been outstanding progress in the areas of image super-resolution, music generation, image compression etc. However, GANs suffer from some problems like vanishing gradients, mode collapse, non-convergence and more. They are also restricted to generating data that is in some way similar to the data they have been trained with and cannot reach the level of creativity that is native to humans.

The aim of this paper is to clarify and address the main challenges in creating an effective algorithm for solving this problem. In order to achieve this, we have developed an algorithm that works almost autonomously and even though it requires very little interaction with the potential client, it is still capable of creating a wide variety of unique and professional-looking logos. Logo images are a perfect benchmark for GANs [**alex2017logo**], because they need to be unique and easy to remember, their colors need to be well combined and most importantly - they need to have meaning. The best logos are not just visually appealing - they are also tightly coupled with their company or organization's trademark, brand and values. Achieving this is a big challenge for the contemporary machine learning techniques.

2 Related work

There is wide ongoing research related to new areas of application of machine learning. Generative models have become a vital part of machine learning research recently. They are forced to discover and efficiently internalize the essence of the data they are trained with in order to generate it. The introduction of the Generative Adversarial Networks [**goodfellow2014generative**] made a huge breakthrough in the area of data generation, because they can be trained in a completely unsupervised manner, which makes them very useful for solving a wide variety of problems like image-to-image translation [**isola2016imagetoimage**], image super-resolution [**ledig2016photorealistic**], music generation [**yang2017midinet**], fashion design [**article**] and more.

The problem of logo generation was stated in 2017 by Sage, et al. [**alex2017logo**]. Furthermore, this problem has also been tackled by Mino, et al [**mino2018logan**]. They proposed the usage of clustering, based on common colors. Our experiments show however that the color of an image is not a vital feature of it and can be easily altered.

3 Dataset

The development of a successful deep neural network requires the usage of well selected training data. The Large Logo Dataset[1] perfectly satisfies the criteria, because it contains over 600 000 high-quality diverse logos. It can be used freely for academic purposes.

4 Clustering

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). This task can be solved using many different algorithms and we decided to use one of the most popular ones - K-means clustering. It is relatively simple, but despite that the results it produces are good. By clustering the images we can synthesize labels, which indicate the category of each of the images. The logos in the same cluster will have the same label.

4.1 Feature extraction

When performing a classification task, only few of the features of the object that needs to be classified are vital. Most of the time, elements of an image like background, noise or just any secondary element actually make the classification task harder and they should be ignored. The more accurate and important the extracted features are, the more successful the clustering will be. We have experimented with two algorithms for feature extraction - the first one uses a pretrained neural network and the second one uses an autoencoder.

4.1.1 Transfer learning with VGG16

There is a wide variety of features that can be extracted from an image such as color, style etc., but the symbolism is definitely the most important feature. The style and the color can easily be altered (such algorithms are presented further), but the symbolism is the base of every good logo image and any modification to it would eventually change the whole identity of the logo. That is why we chose symbolism to be the main feature to be extracted. In order to extract it from the images we will need a neural network that has already been trained to classify logo images into some predefined categories of symbolism - if we had such a network, the clustering would be absolutely pointless.

Many researchers have encountered similar problems and that has led to the invention of a new research area in machine learning - Transfer learning. Transfer learning focuses on storing knowledge while solving one problem and then applying it to a different but related problem. In our case, we will be using a convolutional neural network called VGG16 [**simonyan2014deep**]. It has been trained to classify images from the big dataset ImageNet (it contains over 13 million real-life images with their corresponding labels). There is a wide variety of neural networks trained on ImageNet to choose from like ResNet50, AlexNet and Xception to name a few, but despite its simplicity, VGG16 is sufficient and the usage of a different neural network architecture would not improve the results. Even though these kinds of neural networks are made to classify real-life objects, they also perform well on much more abstract data like logo images.

[1] `https://data.vision.ee.ethz.ch/sagea/lld/`

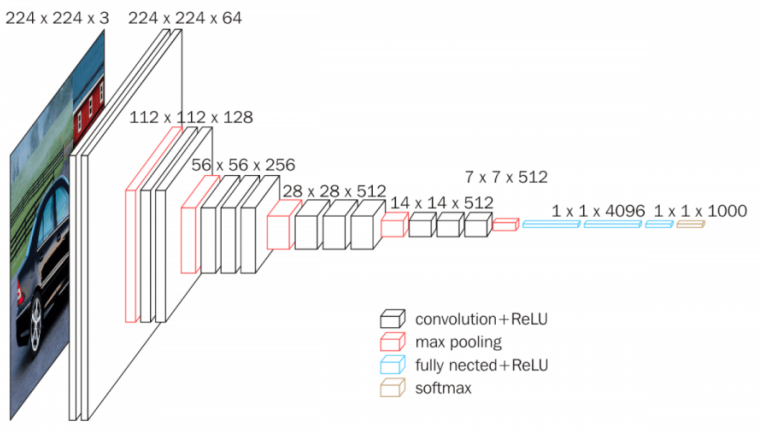

Figure 1: Architecture of VGG16

4.1.2 Autoencoder

An autoencoder is a type of neural network that is used to learn efficient data codings in an unsupervised manner. It is typically used for dimensionality reduction. It is composed of two parts: encoder and decoder. The encoder learns to compress the input dimensions and compress the input data into encoded representation. The decoder learns to reconstruct the original state of the input data after it has been compressed and encoded. It can be trained just like a simple feedforward neural network by using backpropagation [**Rumelhart:1986we**]. If we denote the encoder by ϕ, the decoder by ψ, the original data by X and the latent space (encoded data) by F, then

$$\phi : X \to F \tag{1}$$

$$\psi : F \to X \tag{2}$$

$$\phi, \psi = \arg\min_{\phi,\psi} \|X - (\psi \circ \phi)X\|^2 \tag{3}$$

Autoencoders are also used for noise reduction, image compression, anomaly detection, image generation and more. What makes them suitable for feature extraction is that the latent space (encoded data) is in a lower dimension than the original input data and it contains only the most important features which are needed for further reconstruction. Autoencoders are a top choice if the input data is too abstract and cannot be successfully classified. For the encoder we have developed a simple convolutional neural network that is very similar to the discriminator, one of the two essential components of any Generative Adversarial Network. The decoder is completely identical to the other one - the generator. Their architectures are presented in the next section.

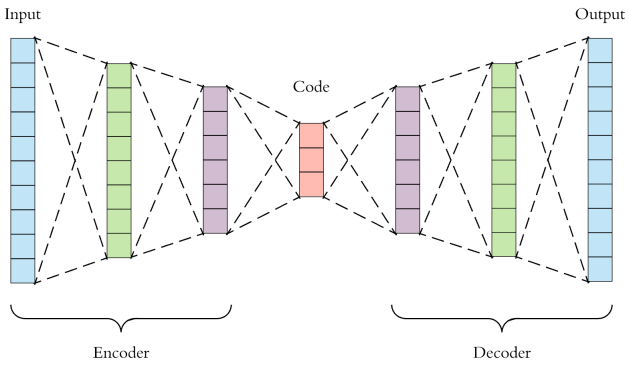

Figure 2: Autoencoder architecture

4.2 K-means clustering

K-means clustering views every list of features, extracted from a certain image with one of the feature extraction algorithms, as a data point in the Euclidean space. After clustering, objects that are similar will be assigned to the same centroid. By getting the centroid index of every image, we are creating synthetic labels that can be later used during the process of generation.

Algorithm 1 K-means clustering

1: Start with initial guesses for the cluster centers (centroids)
2: **while** there are changes in centroid values **do**
3: For each data point, find the closest cluster center (by using Euclidean distance)
4: Replace each centroid by average of data points in its partition
5: **end while**

5 Generative Adversarial Networks

Generative Adversarial Networks are currently a very popular topic among researchers. They made a huge breakthrough in data generation and are currently used for solving a wide variety of problems like image super-resolution, music generation, 3D models reconstruction for images, fashion design and more. A Generative Adversarial Network is actually composed of two separate neural networks: the first one is called the generator or the Artist and the second one is called the discriminator. They contest in a minimax two-player game where the generator acts like a forger and tries to capture the data distribution and the discriminator acts like a critic and estimates the probability that a sample came from the training data rather than it was generated. Both of the networks can be trained with backpropagation and there is no need for any Markov chains.

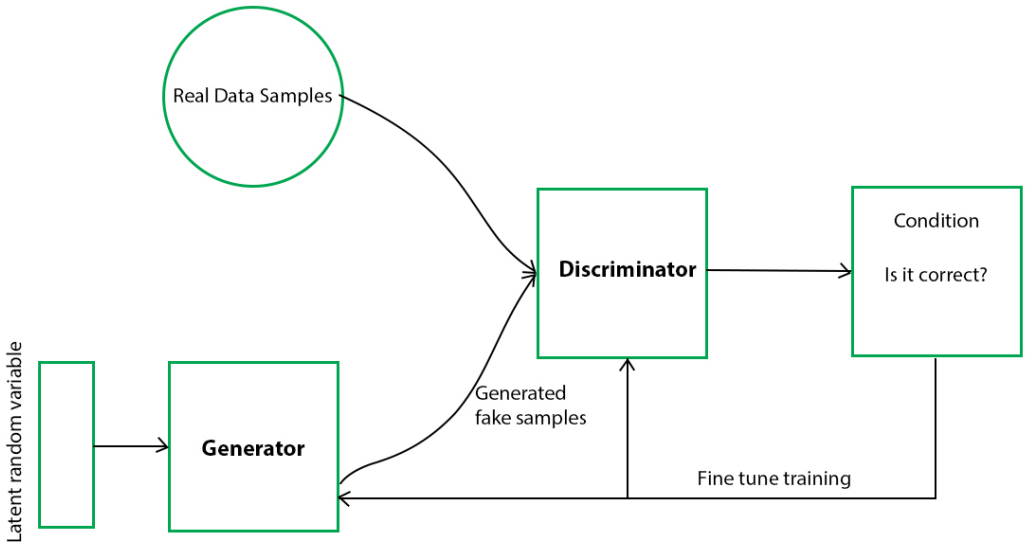

Figure 3: Architecture of a Generative Adversarial Network

The main objective of a GAN is to reach the state where the data generated by the generator is looking so genuine that the discriminator is no longer capable of distinguishing it from the real training data. Any further training would be pointless and would eventually lead to worse results from both the generator and the discriminator. There must also be an equilibrium between the generator and the discriminator's training rates. They may be separate networks, but despite that, they heavily depend on one another, and if one of them becomes way better than the other, then different problems might occur. Because that, they are very unstable and extremely hard to train.

The discriminator we developed is a simple convolutional neural network. It maps an image of a shape $(32, 32, 3)$ to a probability expressing whether the same image is real.

The generator we developed is a deconvolutional [**5539957**] neural network which maps a random noise of size (100) to an image of size $(32, 32, 3)$.

5.1 Training a Generative Adversarial Network

To learn the generator's distribution p_z over data x, let us define a prior on input noise variables $p_z(z)$, then represent a mapping to a data space as $G(z; \theta_g)$, where G is a generator with parameters θ_g. We also define a discriminator $D(x, \theta_d)$ with parameters θ_d. $D(x)$ represents the probability that x came from the training data rather than from p_g. We train D to maximize the probability of assigning correct label to both the training images and the generated images. We also simultaneously train G to minimize $\log(1 - D(G(z)))$. To paraphrase this, G and D play the following two-player minimax game with value function $V(G, D)$:

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

The output of the discriminator must be in the range $[0; 1]$ (it is a probability), so the activation function of its last layer must be sigmoid.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

The loss (cost) function shown in (4) is called binary cross-entropy. Even though it is the most popular loss function for GANs, because it was used in the original paper [**goodfellow2014generative**], it turns out that there are many problems with it. The choice of cross-entropy loss means that points generated far from the boundary provide very little gradient information to the generator on how to generate better images, because cross-entropy is actually a relative metric. This problem is referred to as the vanishing gradient problem or the loss saturation.

The vanishing gradient problem can be solved if a different loss function is used instead of cross-entropy. The Least Squares Generative Adversarial Network [**mao2016squares**], or LSGAN for short, is an extension to the regular GAN architecture. It uses least squares loss as a cost function that is trying to minimize the Pearson χ divergence. We can define the objectives of the discriminator and the generator with the following two formulas:

$$\min_D V_{LSGAN} = \frac{1}{2}\mathbb{E}_{x \sim p_{data}(x)}[(D(x) - 1)^2] + \frac{1}{2}\mathbb{E}_{z \sim p_z(z)}[(D(G(z))^2] \tag{6}$$

$$\min_G V_{LSGAN} = \frac{1}{2}\mathbb{E}_{z \sim p_z(z)}[(D(G(z))^2] \tag{7}$$

The LSGAN architecture also provides more stable training and the generated images are higher-quality. However, another common problem is presented. Sometimes the generator produces an especially plausible output that successfully deceives the discriminator. The generator starts producing only that output over and over again and the discriminator's best strategy is to learn to always reject it. But if both networks get stuck in a local minimum, then it's really easy for the next generator iteration to find the most plausible output for the current discriminator. Each iteration the generator over-optimizes for a particular discriminator and the discriminator never manages to learn its way out of the trap. As a result the generator starts producing a narrow range of outputs. That form of GAN failure is called mode collapse and it is one of the most serious problems with GAN training.

This problem has bothered researchers for three years until a new architecture was introduced in 2017. It is called Wasserstein Generative Adversarial Network (WGAN for short) [**arjovsky2017wasserstein**]. Despite the fact that it deals with vanishing gradients, it also reduces significantly the chance of mode collapse appearance. It requires more changes to the original GAN architecture than LSGAN does. The output of the discriminator must not be in the range $[0; 1]$, but in the range $(-\infty; +\infty)$. We simply replace the sigmoid activation function we use in the discriminator's output layer. The discriminator's output is no longer a probability, but rather just a number expressing how successful the generated images is. The larger this number is, the more successful they are. It is the opposite for real images - lower number means better images. The discriminator actually does not expresses a probability whether an image is real or not, but rather expresses how real they look. Because of that, it is no longer called a discriminator, but a critic. The critic's weights must also be clamped after each gradient update in order to be in the range $[-0.01; 0.01]$. The usage of the RMSProp optimizer is recommended.

The WGAN loss function is constructed using the Wasserstein metric (also called Kantorovich–Rubinstein metric). The Wasserstein metric is a distance function defined between probability distributions on a given metric space M. The discriminator tries to maximize $D(x) - D(G(z))$ and the generator tries to minimize $D(G(z))$.

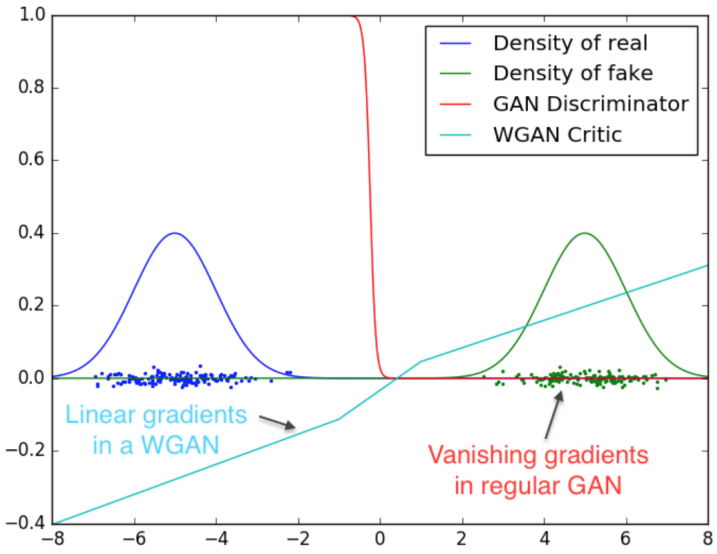

Figure 4: Comparison between gradients of a regular GAN and a WGAN

5.2 Conditional Generative Adversarial Networks

Before starting to design a logo, designers have a conversation with their clients about what components they want to appear in it. Regular GANs generate images absolutely randomly, because the only input to the generator receives is a random noise. One way to get the desired output is to cluster the images after they have been generated. However, a lot of unnecessary data will be generated and there is no gurantee that the generator will produce output associated with the desired symbolism. An architecture called Conditional Generative Adversarial Network (CGAN for short) [**mirza2014conditional**], has recently been introduced. It restricts the GAN to generating images only from a certain specified class. Both the generator and the discriminator receive an additional input - label of the desired outputs. We synthesized labels of the training data by means of feature extraction and clustering. In the generator the prior input noise and the label are combined in joint hidden representation.

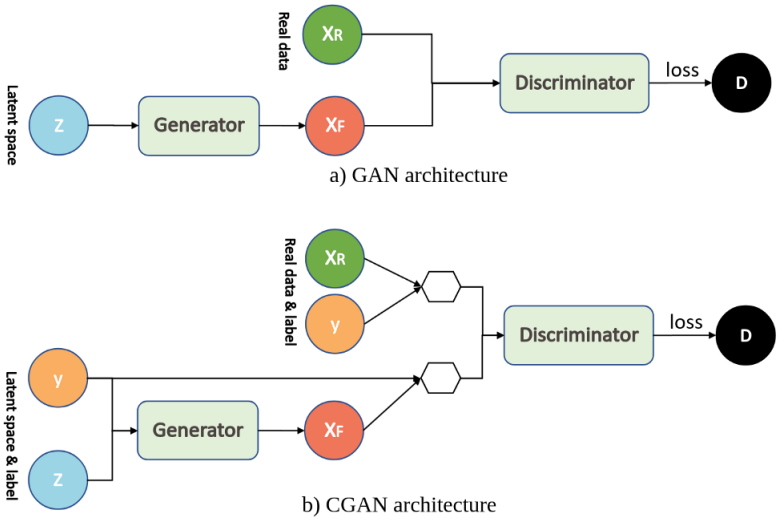

Figure 5: Comparison between a regular GAN and a CGAN

6 Image processing algorithms

6.1 Blurring

Blurring algorithms are quite useful for rounding some sharp edges of the generated images. There is some variety of noise types that can be used, but so far, the best one to use is Gaussian noise. We apply Gaussian noise with the probability density function of a Gaussian random variable and its value is given with this formula:

$$p_g(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \tag{8}$$

Where z represents the gray level, μ the mean value and σ the standard deviation. In the case of digital image processing Gaussian noise can be reduced using a spatial filter, though when smoothing an image, an undesirable outcome may result in the blurring of fine-scaled image edges and details because they also correspond to blocked high frequencies. Conventional spatial filtering techniques for noise removal include: mean (convolution) filtering, median filtering and Gaussian smoothing.

6.2 Denoising autoencoder

Autoencoders can be also used for noise reduction as stated in 4.1.2. Denoising autoencoders try to achieve a good representation by changing the reconstruction criterion. Indeed, they take corrupted input and their objective is to recover its original state (without any corruptions). Before feeding the images to the autoencoder, we add some type of noise to them, but when calculating the loss function, we compare the reconstructed output to the original, undistorted input.

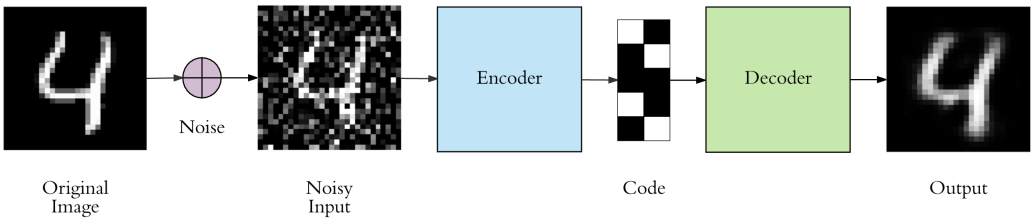

Figure 6: Architecture of a Denoising autoencoder and an example usage on the MNIST dataset

6.3 Color changing

We have implemented an algorithm which is used to change the colors of an already generated image, with ones which are on a different end of the color spectrum. A client may want their logo to be in a different color that better suits their needs. It is very important that the colors after the change are still well-combined. The way we achieve this change of colors is with the following formula for all the channels of an image:

$$p(x, y) = |p(x, y) - n| \tag{9}$$

Where $p(x, y)$ stands for the pixel with coordinates x and y and n is a value from 0 to 255 which shows where from the color wheel to get the colors.

6.4 Complementing

We are using this algorithm to transform an image from one end of the color wheel to the exact opposite. The way we accomplish this is by first determining the average value for all the color channels:

$$\min(r, g, b) + \max(r, g, b) \tag{10}$$

After we get this value we simply subtract the original value of the pixel from it to create a new pixel value with complemented color channels.

7 Experiments & Analysis

Python was chosen as the main programming language for the implementation due to the many high-quality machine learning libraries available. The core of the project, the machine learning and the image processing algorithms, are implemented using Tensorflow, NumPy, scikit-learn, OpenCV, PIL and matplotlib. For the training of the neural networks, Nvidia CUDA has been utilized. A web interface for the project has also been developed so that potential users can test it. In this way, important feedback for the further development of the project can be gathered.

The experiments conducted provide interesting insights into which components of a Generative Adversarial Network are more likely to induce a failure. The Generative Adversarial Networks are challenging to train, so the hyperparameters have to be chosen precisely. The GAN has been trained for over 100 epochs and despite that the results are satisfactory, further training will improve the quality of the output. The generated images are diverse and unique, even though those with round form dominate over the others.

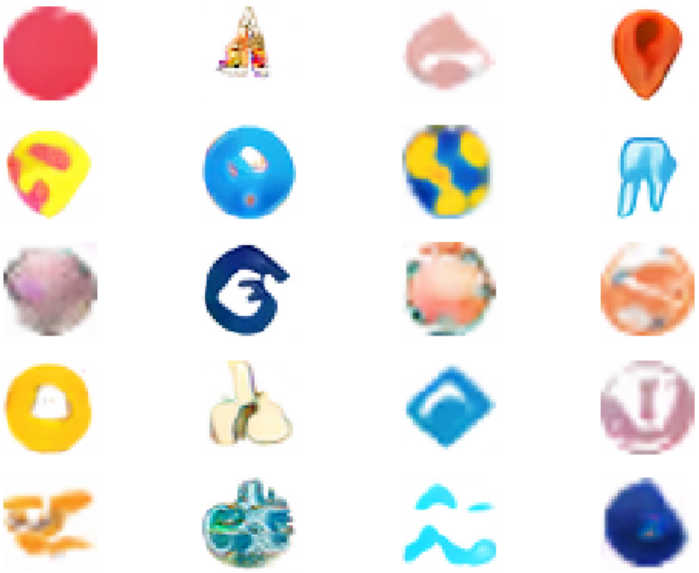

Figure 7: Sample images generated by CLSGAN

8 Conclusion and future work

This paper's aim is to clarify and address the main challenges in creating effective algorithms for producing data that requires high amount of creativity, such as logo images. The results are promising, but we have concluded that machine learning algorithms are still not capable of automating the job of a designer. The Generative Adversarial Networks are capable of producing a wide range of unique high-quality data, but they still suffer from some problems and their training is a really tough challenge. Clustering is an efficient way of unsupervised data labeling, but it heavily depends on the data obtained from the feature extraction algorithms. By using different image processing algorithms the quality of the generated images can be significantly improved and they can be altered so that they suit the client's needs. In the future, we will try to find a better way for GAN training, we will also develop more image processing algorithms so that the generated logo images become more professional-looking and expand the research into other kinds of UI elements like pictograms in order to prove the generality of this approach.