REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

CREATING PICTURES WITH ARTIFICIAL INTELLIGENCE

14011903 — Alibek ERKABAYEV

SENIOR PROJECT

Advisor
Assist. Prof. Dr. Oguz Altun

January, 2021

ACKNOWLEDGEMENTS

In this study, our teacher who assists us with his mentoring and always supports us in terms of technique and motivation. I thank **Assist. Prof. Dr. Oguz Altun**

Alibek ERKABAYEV

TABLE OF CONTENTS

LIST OF FIGURES **v**

LIST OF TABLES **vi**

ABSTRACT **vii**

ÖZET **viii**

1 Introduction **1**

2 Related Works **2**
2.1 GAN 2
2.2 StyleGAN 3
2.3 StyleGAN2 4
2.4 Conditional StyleGAN 4

3 Feasibility **5**
3.1 Technical Feasibility 5
3.1.1 Software Feasibility 5
3.1.2 Hardware Feasibility 5
3.1.3 Communication Feasibility 5
3.2 Economic Feasibility 5
3.3 Legal Feasibility 6
3.4 Time Feasibility 6

4 System Analysis **7**
4.1 Requirements Analysis 7
4.1.1 StyleGAN 7
4.1.2 StyleGAN2 10
4.1.3 Conditional StyleGAN 11
4.1.4 Proposed Data 15
4.2 Logo Generator Network Structure 16

5 System Design **17**

5.1 Dataset 17
5.1.1 Preprocessing Dataset 17
5.1.2 Environment Configuration 18
5.1.3 User Interface 18

6 Application **20**

7 Experimental Results **21**

8 Performance Analysis **25**

9 Result **26**

References **27**

Curriculum Vitae **28**

LIST OF FIGURES

Figure 2.1 GAN 2
Figure 3.1 Timeline of Project 6
Figure 4.1 StyleGAN Structure 8
Figure 4.2 Calculation of the adaptive instance normalization (AdaIN) in the StyleGAN. 9
Figure 4.3 Comparison of the components of StyleGAN (a-b) and StyleGAN2 (c-d) 11
Figure 4.4 StyleGAN 12
Figure 4.5 Mapping Network 12
Figure 4.6 AdaIN 13
Figure 4.7 Stochastic Variation 14
Figure 4.8 Introducing Conditions 14
Figure 4.9 WGAN-GP 15
Figure 4.10 Network Structure 16
Figure 5.1 QMUL-OpenLogo structure 18
Figure 5.2 User Interface 19
Figure 7.1 Model Init Results 21
Figure 7.2 Model 200 iteration Results 22
Figure 7.3 Model 1000 iteration Results 23
Figure 7.4 Real Images from Dataset 24
Figure 8.1 Performance Analysis of StyleGAN2 with example FFHQ dataset 25

LIST OF TABLES

Table 3.1 Expense Table 6

ABSTRACT

Creating Pictures with Artificial Intelligence

Alibek ERKABAYEV

Department of Computer Engineering
Senior Project

Advisor: Assist. Prof. Dr. Oguz Altun

Designing a logo is a long, complicated, and expensive process for any designer. However, recent advancements in generative algorithms provide models that could offer a possible solution. Logos are multi-modal, have very few categorical properties, and do not have a continuous latent space. Yet, conditional generative adversarial networks can be used to generate logos that could help designers in their creative process. We propose StyleGAN and Conditional StyleGAN for this project. The project's results offer a first glance at how artificial intelligence can be used to assist designers in their creative process and open promising future directions, such as including more descriptive labels which will provide a more exhaustive and easy-to-use system.

Keywords: StyleGAN, StyleGAN2, Conditional StyleGAN, LogoGANv2, LOGO generate, Artificial Intelligence, Machine Learning

ÖZET

Yapay Zeka ile Resim Oluşturma

Alibek ERKABAYEV

Bilgisayar Mühendisliği Bölümü
Bitirme Projesi

Danışman: Dr. Ögr. Üyesi Oguz Altun

Bir logo tasarlamak, her tasarımcı için uzun, karmaşık ve pahalı bir süreçtir. Ancak, üretken algoritmalardaki son gelişmeler, olası bir çözüm sunabilecek modeller sağlar. Logolar çok modludur, çok az kategorik özelliğe sahiptir ve sürekli bir gizli boşluğa sahip değildir. Yine de, koşullu üretken karşıt ağlar, tasarımcılara yaratıcı süreçlerinde yardımcı olabilecek logolar oluşturmak için kullanılabilir. Bu proje için StyleGAN ve Koşullu StyleGAN'ı öneriyoruz. Projenin sonuçları, yapay zekanın tasarımcılara yaratıcı süreçlerinde yardımcı olmak için nasıl kullanılabileceğine ve daha kapsamlı ve kullanımı kolay bir sistem sağlayacak daha açıklayıcı etiketler dahil olmak üzere gelecek vaat eden yönlere nasıl açık olabileceğine ilk bakış sunuyor.

Anahtar Kelimeler: StyleGAN, StyleGAN2, Conditional StyleGAN, LogoGANv2, LOGO üretimi, Yapay Zeka,Makine Öğrenmesi

1
Introduction

Logos are very important to represent the impression of companies and brands. Designing a logo for a new brand is a lengthy and tedious back-and-forth process between a designer and a client. Recent advancements in generative models suggest a possible use of artificial intelligence as a solution to this problem.

This project uses GANs models to produce new logo images. GAN consist of two neural networks, a generator and a discriminator, that are contended against each other, trying to reach a Nash Equilibrium in a minimax game. It works by trying to mimic real-world images collected from the Wikipedia pages. For this, we use a dataset – LLD – of 600k+ logos crawled from the world wide web. Nowadays we have different type of GANs architecture. We will use StyleGANv2 developed by Nvidia and conditional StyleGAN.

Finally project is assigned to produce logos with help of machine learning.

2
Related Works

2.1 GAN

Generative adversarial networks are effective at generating high-quality and large-resolution synthetic images [1].

The generator model takes as input a point from latent space and generates an image. This model is trained by a second model, called the discriminator, that learns to differentiate real images from the training dataset from fake images generated by the generator model. As such, the two models compete in an adversarial game and find a balance or equilibrium during the training process.

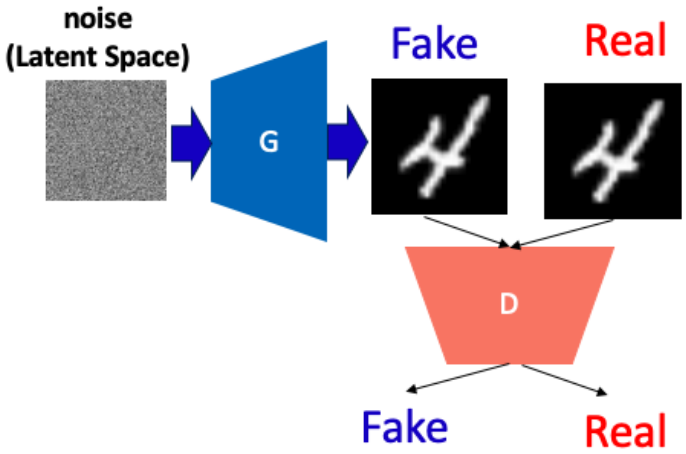

Figure 2.1 GAN

Many improvements to the GAN architecture have been achieved through enhancements to the discriminator model. These changes are motivated by the idea that a better discriminator model will, in turn, lead to the generation of more realistic synthetic images.

As such, the generator has been somewhat neglected and remains a black box. For example, the source of randomness used in the generation of synthetic images is not well understood, including both the amount of randomness in the sampled points and the structure of the latent space.

This limited understanding of the generator is perhaps most exemplified by the general lack of control over the generated images. There are few tools to control the properties of generated images, e.g. the style. This includes high-level features such as background and foreground, and fine-grained details such as the features of synthesized objects or subjects.

This requires both disentangling features or properties in images and adding controls for these properties to the generator model.

2.2 StyleGAN

The Style Generative Adversarial Network, or StyleGAN for short, is an extension to the GAN architecture to give control over the disentangled style properties of generated images [2].

The StyleGAN is an extension of the progressive growing GAN that is an approach for training generator models capable of synthesizing very large high-quality images via the incremental expansion of both discriminator and generator models from small to large images during the training process.

In addition to the incremental growing of the models during training, the style GAN changes the architecture of the generator significantly.

The StyleGAN generator no longer takes a point from the latent space as input; instead, there are two new sources of randomness used to generate a synthetic image: a standalone mapping network and noise layers.

The output from the mapping network is a vector that defines the styles that is integrated at each point in the generator model via a new layer called adaptive instance normalization. The use of this style vector gives control over the style of the generated image.

Stochastic variation is introduced through noise added at each point in the generator model. The noise is added to entire feature maps that allow the model to interpret the style in a fine-grained, per-pixel manner.

This per-block incorporation of style vector and noise allows each block to localize both the interpretation of style and the stochastic variation to a given level of detail.

2.3 StyleGAN2

StyleGAN2 is a generative adversarial network that builds on StyleGAN with several improvements. First, adaptive instance normalization is redesigned and replaced with a normalization technique called weight demodulation. Secondly, an improved training scheme upon progressively growing is introduced, which achieves the same goal - training starts by focusing on low-resolution images and then progressively shifts focus to higher and higher resolutions - without changing the network topology during training. Additionally, new types of regularization like lazy regularization and path length regularization are proposed [3].

2.4 Conditional StyleGAN

In order to enable control the style of the generated output, conditions are introduced. There are two major differences between the conditional and un-conditional StyleGAN architectures, namely the way the input to the generator w is produced and in how the discriminator calculates its loss [4].

Additional information that is correlated with the input images, such as class labels, can be used to improve the GAN. This improvement may come in the form of more stable training, faster training, and/or generated images that have better quality.

Class labels can also be used for the deliberate or targeted generation of images of a given type.

A limitation of a GAN model is that it may generate a random image from the domain. There is a relationship between points in the latent space to the generated images, but this relationship is complex and hard to map.

Alternately, a GAN can be trained in such a way that both the generator and the discriminator models are conditioned on the class label. This means that when the trained generator model is used as a standalone model to generate images in the domain, images of a given type, or class label, can be generated.

3
Feasibility

The feasibility study carried out during the development of the system is described in this section.

3.1 Technical Feasibility

3.1.1 Software Feasibility

The planned project will be implemented using Linux OS and Windows OS. The data gathered on the Internet and categorized. The project will be implemented using the Python programming language, Tensorflow, Reactjs, and Django frameworks.

3.1.2 Hardware Feasibility

In the software to be realized for the project, strong hardware is needed as it will use machine learning and deep learning. It is planned to utilize the Tesla T4 or Tesla v100 GPU provided by the Google Colab service via the cloud after running the Intel i7 in our local environment with a smaller section of our data set with the Nvidia 940M series graphics card with 12 GB of RAM.

3.1.3 Communication Feasibility

It is supposed to use GitHub technology and Google Colab. This is to ensure you work on a project regardless of location.

3.2 Economic Feasibility

Since we will be able to read our data through Google Drive if needed, it is not considered to benefit from this service, which also charges a fee of 5.79 TL per month for 100GB usage. For training, the model needs the Google Colab Pro subscription,

cause of model has a big size and the training time is longer than the given 9-hour free version of Google Colab. Pro subscription fee is 9.99$.

Table 3.1 Expense Table

	Piece	Unit Price	Total Price	Brand
Computers	1	4500TL	4500TL	ASUS X555LN Notebook
Software Developers Expense	1	75TL(Daily)	10650TL	Student
100GB space	3	5.79TL	17.37TL	Google Drive
Pro subscription	3	9.99$	29.97$	Google Colab

Note: Since the use of application development programs is provided with student licenses and open-source, no expenditures have been made on behalf of application development programs.

3.3 Legal Feasibility

The project complies with current laws and regulations, and any patents, etc. not to violate the protected right.

3.4 Time Feasibility

The design and implementation process of the application is as stated in the Gantt Diagram in Figure 3.2.

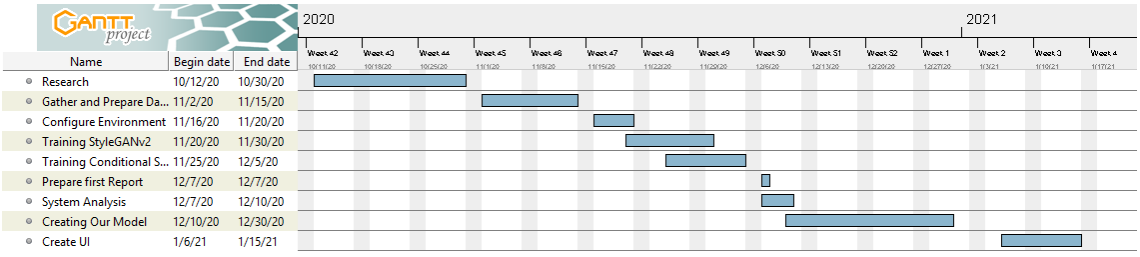

Figure 3.1 Timeline of Project

4
System Analysis

Our main purpose of choosing the python programming language for development are open source, supported libraries and has a wider range of uses and the community to use and develop the application produce is larger. To achieve completion of the project from all the processing steps as much as possible and reaching the next step will affect the total success we will achieve at the end of the project. Another factor that will directly affect success is that the analysis will take place over the GPU rather than the CPU. Thus, the application will be provided to work in a shorter time and at a lower cost.

4.1 Requirements Analysis

4.1.1 StyleGAN

The StyleGAN is described as a progressive growing GAN architecture with five modifications, each of which was added and evaluated incrementally in an ablative study [1, 2].

The incremental list of changes to the generator are:

Baseline Progressive GAN.
Addition of tuning and bilinear upsampling.
Addition of mapping network and AdaIN (styles).
Removal of latent vector input to generator.
Addition of noise to each block.
Addition Mixing regularization.
The Figure 4.1 below summarizes the StyleGAN generator architecture.

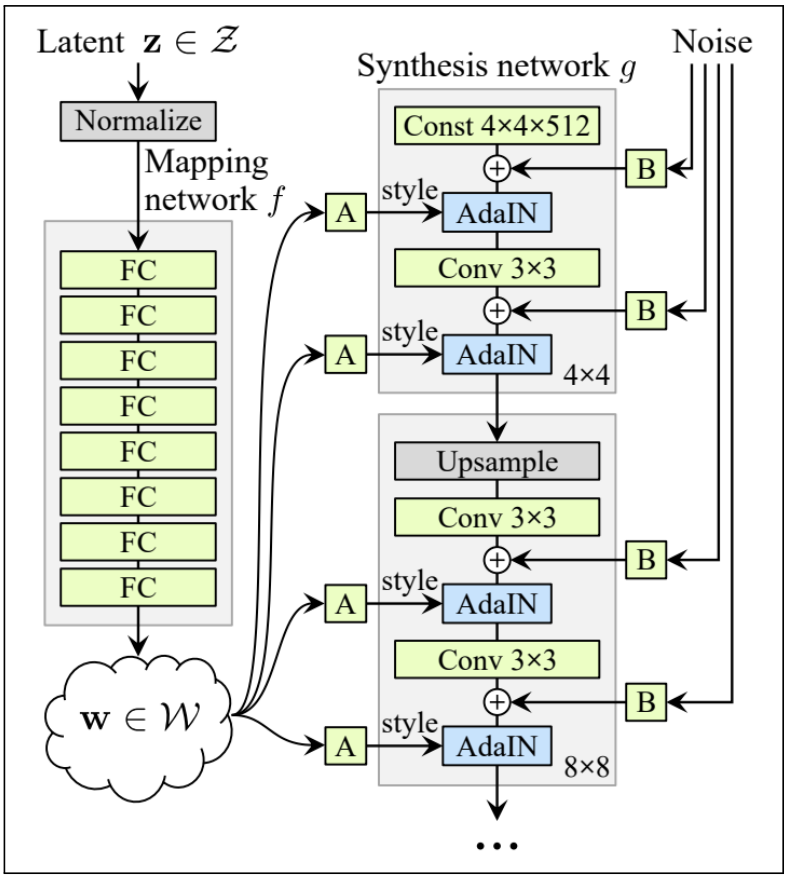

Figure 4.1 StyleGAN Structure

4.1.1.1 Baseline Progressive GAN

The StyleGAN generator and discriminator models are trained using the progressive growing GAN training method.

This means that both models start with small images, in this case, 4×4 images. The models are fit until stable, then both discriminator and generator are expanded to double the width and height (quadruple the area), e.g. 8×8.

A new block is added to each model to support the larger image size, which is faded in slowly over training. Once faded-in, the models are again trained until reasonably stable and the process is repeated with ever-larger image sizes until the desired target image size is met, such as 1024×1024.

4.1.1.2 Bilinear Sampling

The progressive growing GAN uses nearest neighbor layers for upsampling instead of transpose convolutional layers that are common in other generator models.

The first point of deviation in the StyleGAN is that bilinear upsampling layers are unused instead of nearest neighbor.

4.1.1.3 Mapping Network and AdaIN

Next, a standalone mapping network is used that takes a randomly sampled point from the latent space as input and generates a style vector [5].

The mapping network is comprised of eight fully connected layers, e.g. it is a standard deep neural network.

The style vector is then transformed and incorporated into each block of the generator model after the convolutional layers via an operation called adaptive instance normalization or AdaIN.

The AdaIN layers involve first standardizing the output of feature map to a standard Gaussian, then adding the style vector as a bias term.

$$\text{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

Figure 4.2 Calculation of the adaptive instance normalization (AdaIN) in the StyleGAN.

The addition of the new mapping network to the architecture also results in the renaming of the generator model to a “synthesis network".

4.1.1.4 Removal of Latent Point Input

The next change involves modifying the generator model so that it no longer takes a point from the latent space as input.

Instead, the model has a constant 4x4x512 constant value input in order to start the image synthesis process.

4.1.1.5 Addition of Noise

The output of each convolutional layer in the synthesis network is a block of activation maps.

Gaussian noise is added to each of these activation maps prior to the AdaIN operations. A different sample of noise is generated for each block and is interpreted using per-layer scaling factors.

This noise is used to introduce style-level variation at a given level of detail.

4.1.1.6 Mixing regularization

Mixing regularization involves first generating two style vectors from the mapping network.

A split point in the synthesis network is chosen and all AdaIN operations prior to the split point use the first style vector and all AdaIN operations after the split point get the second style vector.

This encourages the layers and blocks to localize the style to specific parts of the model and corresponding level of detail in the generated image.

4.1.2 StyleGAN2

The previous state-of-the-art architecture for generating images was the StyleGAN model. A distinctive feature of the model is the generator architecture. The generator takes as input an intermediate representation of the input object. Generator layers go through adaptive instance normalization (AdaIN). Despite its superior performance

over competing approaches, the original StyleGAN generates images with noticeable artifacts [3].

In the StyleGAN2 generator, unnecessary operations at the beginning were removed, the summation of bias terms was removed outside the style block. The updated architecture allows replacing instance normalization (AdaIN) with "demodulation". The demodulation operation is applied to the weights of each convolutional layer.

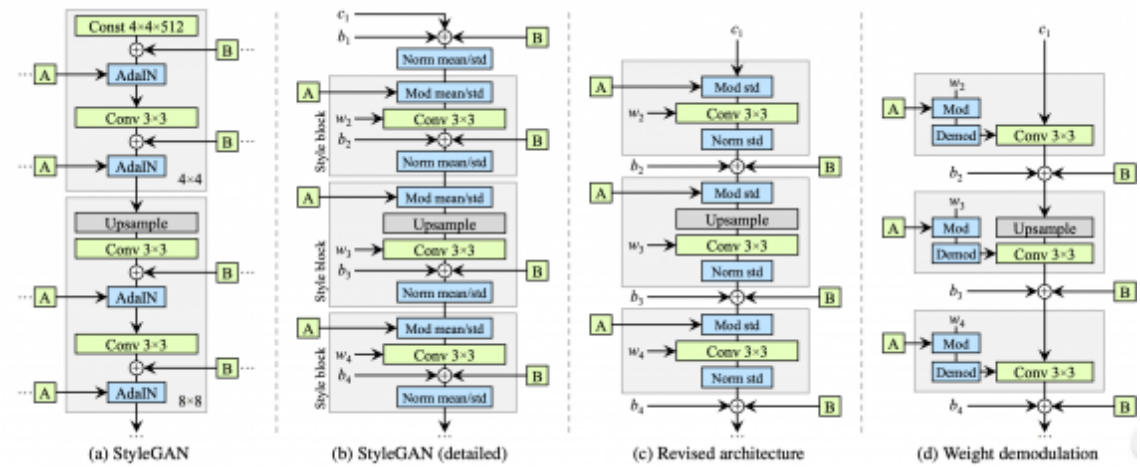

Figure 4.3 Comparison of the components of StyleGAN (a-b) and StyleGAN2 (c-d)

4.1.3 Conditional StyleGAN

The StyleGAN architecture aimed to improve on feature entanglement present in the ProGAN architecture through various extensions as marked by (a), (b) and (c) on the below Figure 4.4:

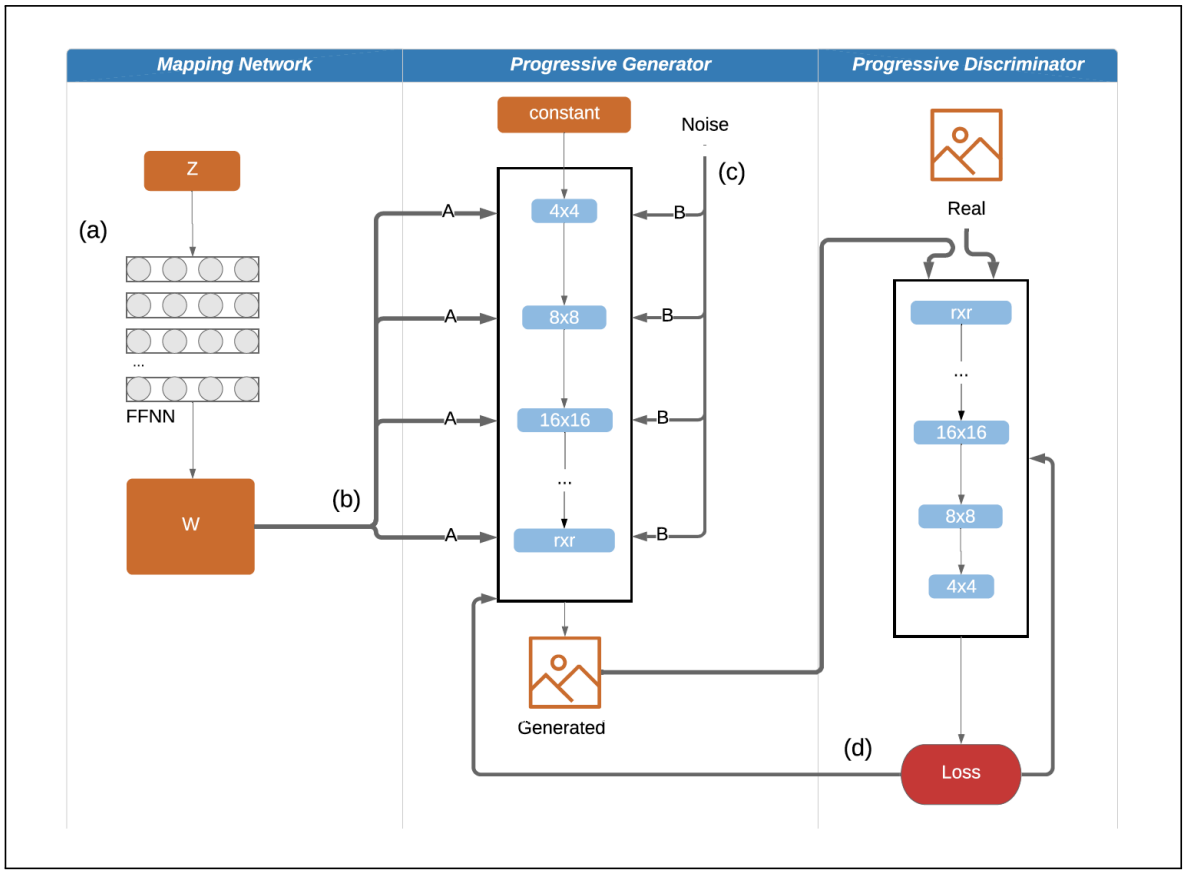

Figure 4.4 StyleGAN

4.1.3.1 Mapping Network (a)

As opposed to feeding a random vector straight into the generator, the input is projected onto an intermediate latent space w by being fed through a mapping network.

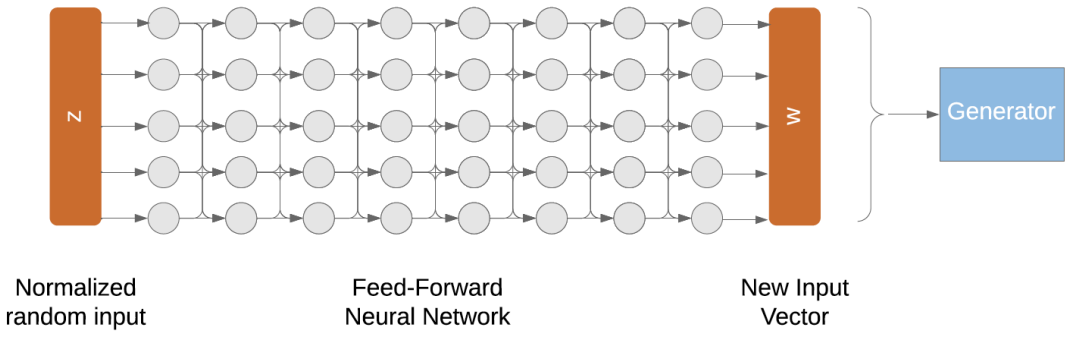

Figure 4.5 Mapping Network

4.1.3.2 AdaIN (b)

In context of the mapping network, the intermediate vector w is transformed into the scale and bias of the AdaIN equation. Noting that w reflects the style, AdaIN, through its normalization process, de

termines the importance of individual parameters in the convolutional layers. Thus, through AdaIN, the feature map is translated into a visual representation.

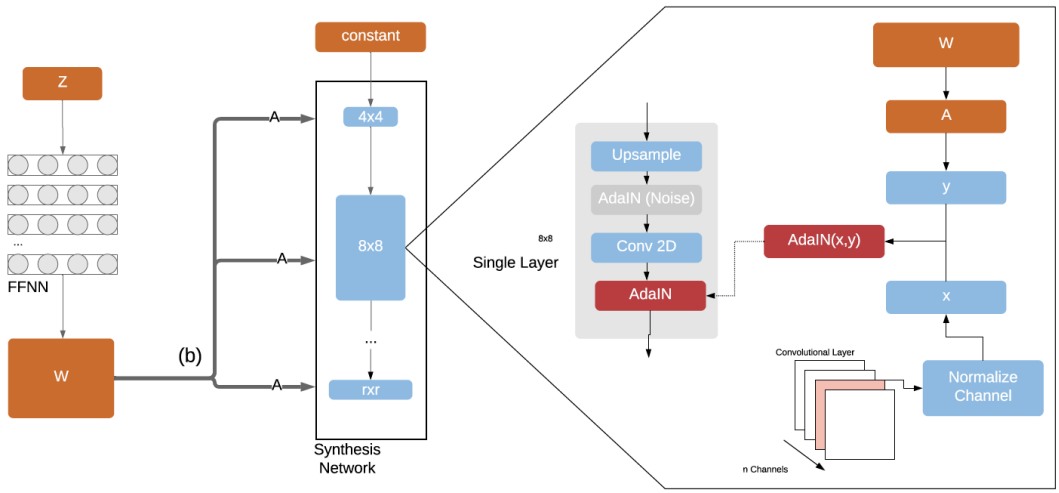

Figure 4.6 AdaIN

4.1.3.3 Stochastic Variation (c)

The noise inputs take the form of two-dimensional matrices sampled from a Gaussian distribution. These are then scaled to match the dimensions within the layer and applied to each channel [2]. This introduces variation within that feature space.

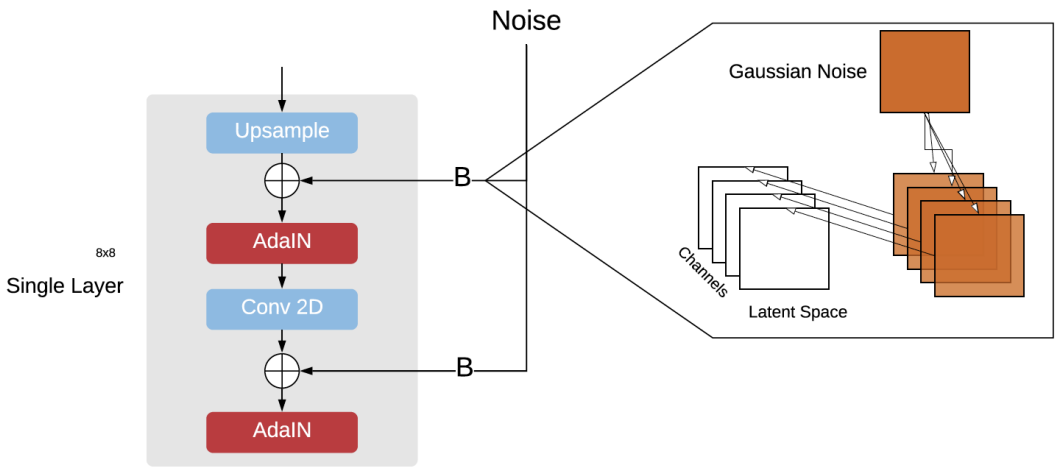

Figure 4.7 Stochastic Variation

4.1.3.4 Introducing Conditions

In order to enable control the style of the generated output, conditions are introduced. There are two major differences between the conditional and un- conditional StyleGAN architectures, namely the way the input to the generator w is produced and in how the discriminator calculates its loss.

Firstly, noise is introduced to the one-hot encoded class conditions, which are then concatenated with the input space z before being fed into the mapping network.

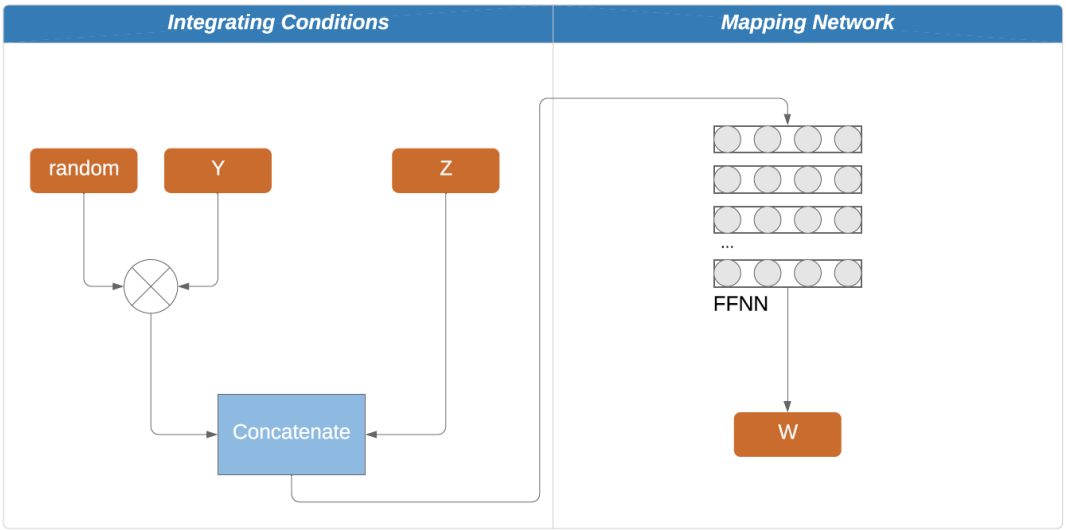

Figure 4.8 Introducing Conditions

Secondly, the WGAN-GP takes on a conditional format:

$$\nabla_{\theta D} \underbrace{[f_{\theta D}(x|y) - f_{\theta D}(G(w|y))]}_{\text{Discriminator Loss}} + \underbrace{\lambda[(|| \nabla_{\hat{x}} D(\hat{x}|y)||_2 - 1)^2]}_{\text{Gradient Penalty}}$$

Figure 4.9 WGAN-GP

4.1.4 Proposed Data

4.1.4.1 LLD - Large Logo Dataset

The logos in this dataset where crawled by parsing the HTML pages with Scrapy tool to look for the favicon declaration as described on Wikipedia [6]. If this failed, we attempted to download the default URL http://url/favicon.ico. [7]

After downloading, all logos where directly converted to RGB with 32 pixel size, rejecting all non-square logos. Some statistics:

- Unreadable image files: 71.596
- Non-square images: 36.401
- Total logos saved: 662.273
- of which duplicates removed: 114.063

This resulted in 548.210 logos, in the following resolution:

- Native 32p: 158.881 (24
- Downscaled: 148.132 (22
- Total logos saved: 662.273
- Upscaled: 355.260 (54

4.1.4.2 QMUL-OpenLogo

Existing logo detection benchmarks consider artificial deployment scenarios by assuming that large training data with fine-grained bounding box annotations for each class are available for model training. Such assumptions are often invalid in realistic

logo detection scenarios where new logo classes come progressively and require to be detected with little or none budget for exhaustively labelling fine-grained training data for every new class. Existing benchmarks are thus unable to evaluate the true performance of a logo detection method in realistic and open deployments. In this work, we introduce a more realistic and challenging logo detection setting, called Open Logo Detection. Specifically, this new setting assumes fine-grained labelling only on a small proportion of logo classes whilst the remaining classes have no labelled training data to simulate the open deployment. Further, we create an open logo detection benchmark, called QMUL-OpenLogo, to promote the investigation of this new challenge. QMUL-OpenLogo contains 27,083 images from 352 logo classes, built by aggregating and refining 7 existing datasets and establishing an open logo detection evaluation protocol.

4.2 Logo Generator Network Structure

According to the research we have done, it has been revealed that we need to use two different networks for our project to work effectively. StyleGAN2 network will be used to generate a new logo from random logo samples. Conditional Stylegan will produce logos according to the category or label we provide. The final structure of our net is as shown in the Figure 4.10.

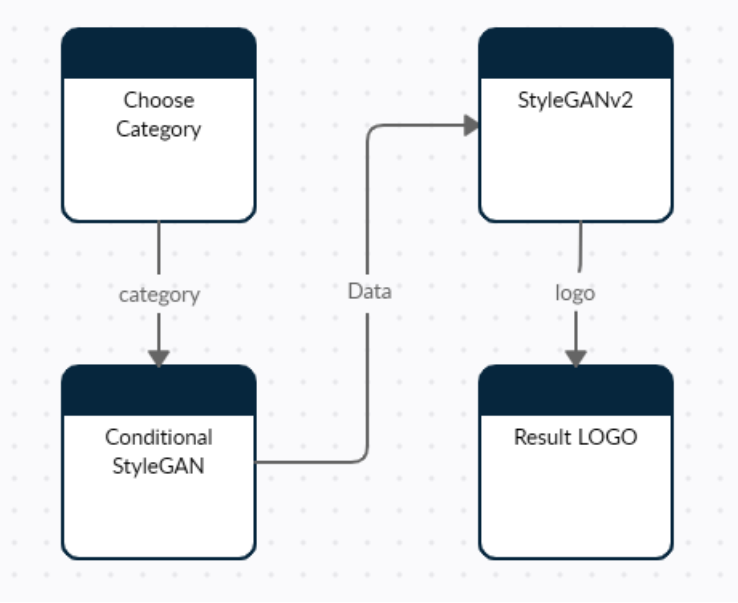

Figure 4.10 Network Structure

5
System Design

The design phase carried out after the analysis phase is completed is reported in this section. This section provides basic information about the designed system, but consists of "Software Design" and "Input-Output Design" subsections.

5.1 Dataset

The dataset used to train Logo Generation is the LLD-icons dataset [7], which consists of 486'377 32 $\times$ 32 icons. And second dataset used for categorized training QMUL-OpenLogo dataset [8].

5.1.1 Preprocessing Dataset

First of all dataset QMUL-OpenLogo has cropped with bounding boxes to separate the logos from the image. In the Annotation folder, the files that are saved in Pascal Voc format and store the bounding-box information are placed. In the JpegImages folder, there are images placed in jpg format. With a script written in Python, we can separate the logos used annotations and jpg images.

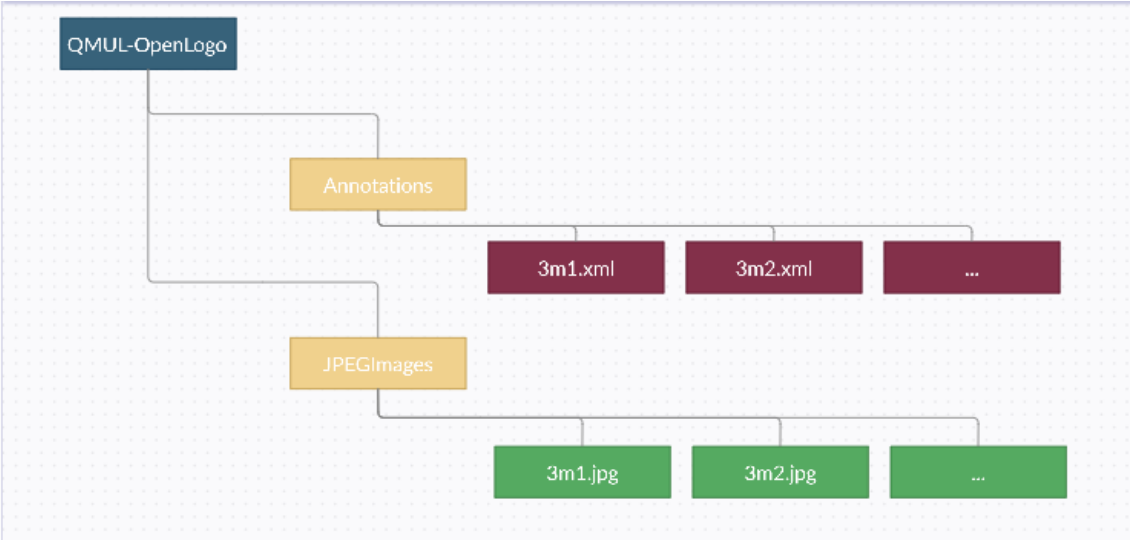

Figure 5.1 QMUL-OpenLogo structure

After cropping QMUL-Open Logo dataset need to resize the same size as LLD dataset, so all our logos to 32 pixels so the network can use this dataset for training. OpenCV used for this operation.

5.1.2 Environment Configuration

For training networks need computer with good GPU configuration, Google Colab Pro used for training. All preprocessed dataset uploaded to Google Drive and connected to Google Colab. After uploading all datasets, open-source StyleGAN2 and Conditional Stylegan networks cloned to Google Colab.

5.1.3 User Interface

In the end, when training end and models created we need to create a User Interface for easy to use trained models and see results. For creating User Interface Django framework used for backend and Reactjs used for frontend.

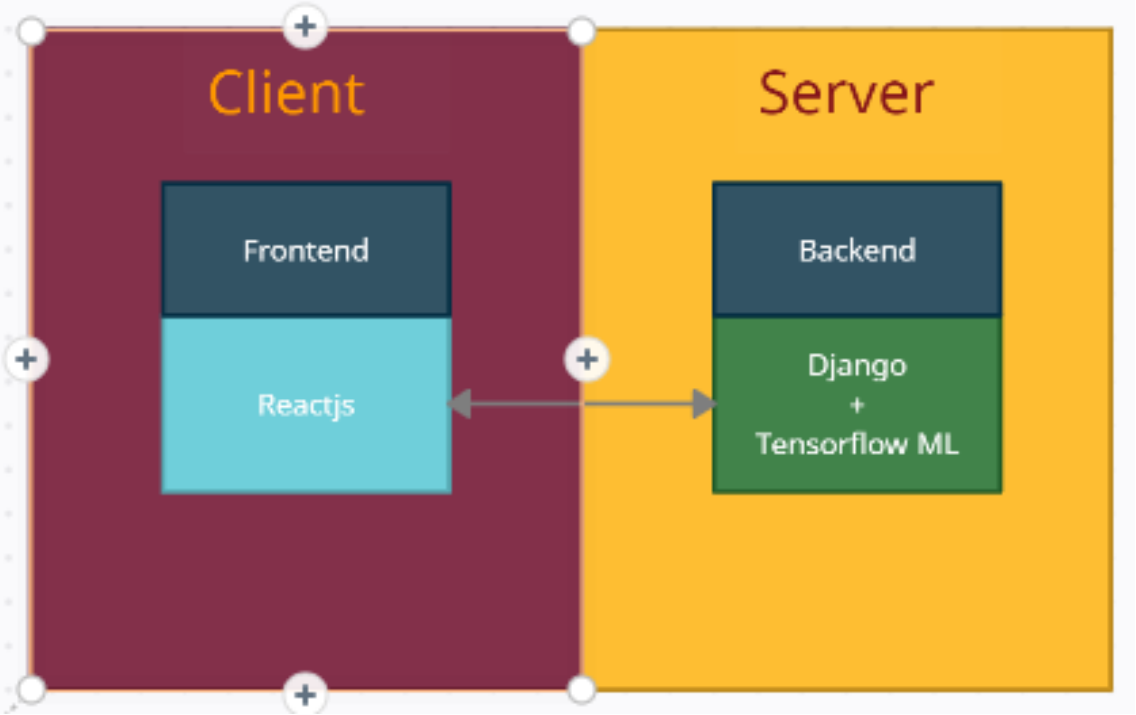

Figure 5.2 User Interface

6
Application

The application refers to a system for this project. Results were obtained using Jupyter Notebook on Google Colab. One of the conveniences provided by Jupyter Notebook is that it allows you to show intermediate outputs by dividing the processes into steps. In this way, each result obtained can be controlled step by step in the values of the variables, and when any errors are detected, they can go back and correct the error and run it again. PyCharm was used for the test. This is because Jupyter Notebook supports the Python programming language.

7
Experimental Results

Experimental Results made in this section is explained.

At first we can see that there is noise.

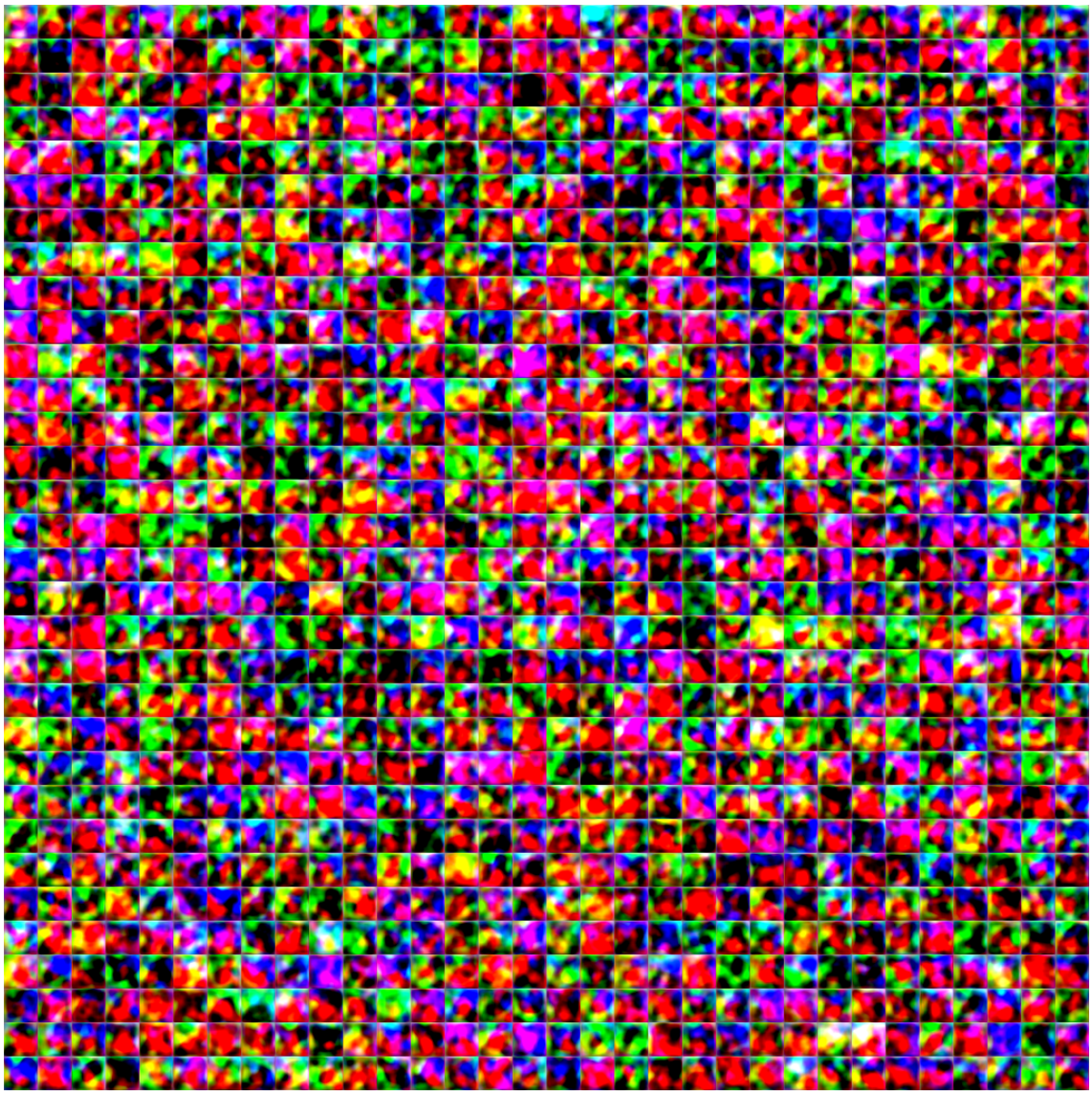

Figure 7.1 Model Init Results

200 and 1000 iterations were made and their effects on the results were checked. According to the results we looked at, fragment errors on the image are less in iterations with more.

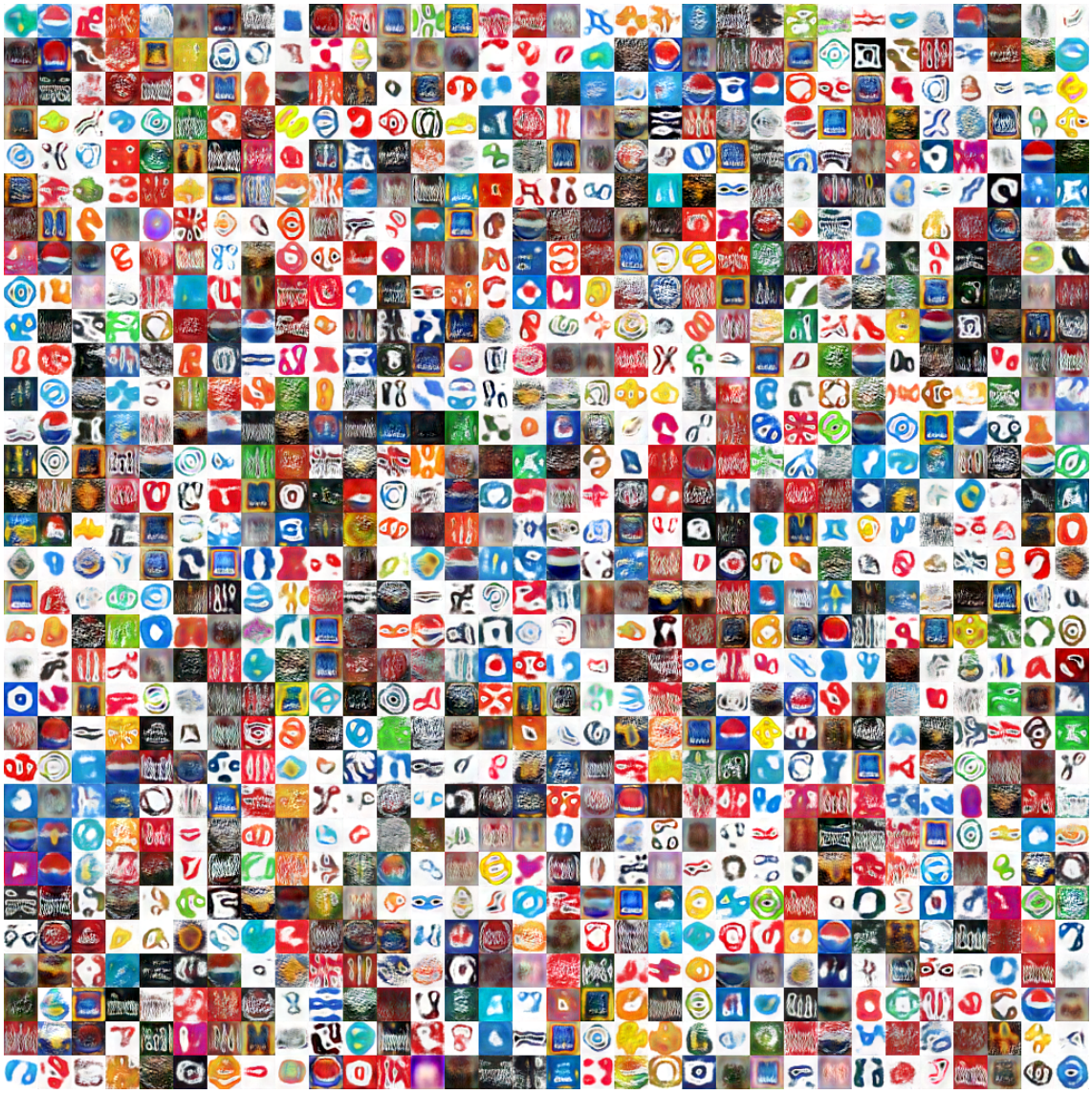

Figure 7.2 Model 200 iteration Results

The results we get at 1000 iterations of oil are cleaner.

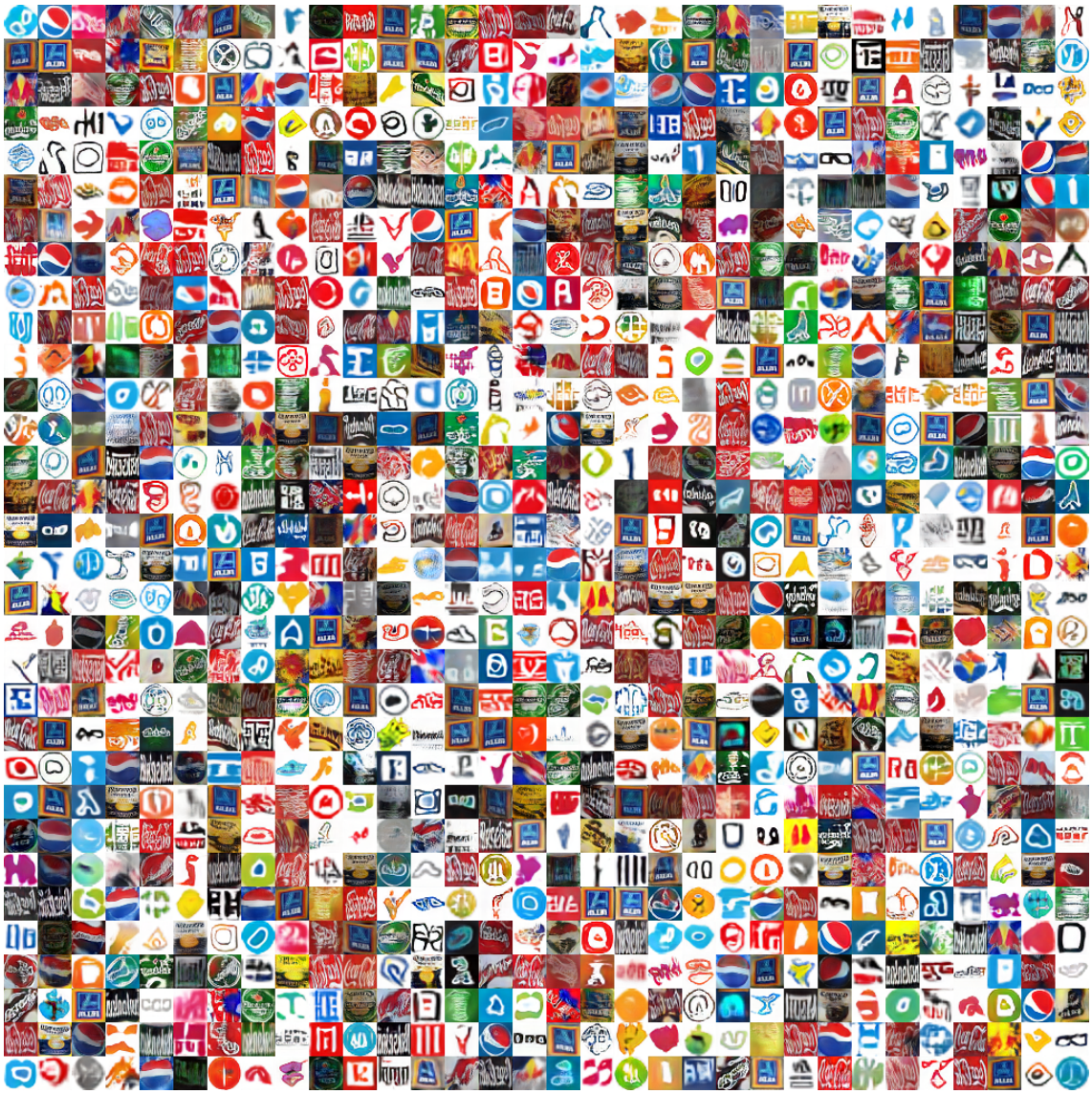

Figure 7.3 Model 1000 iteration Results

Generated images in 1000 iteration closer real ones.

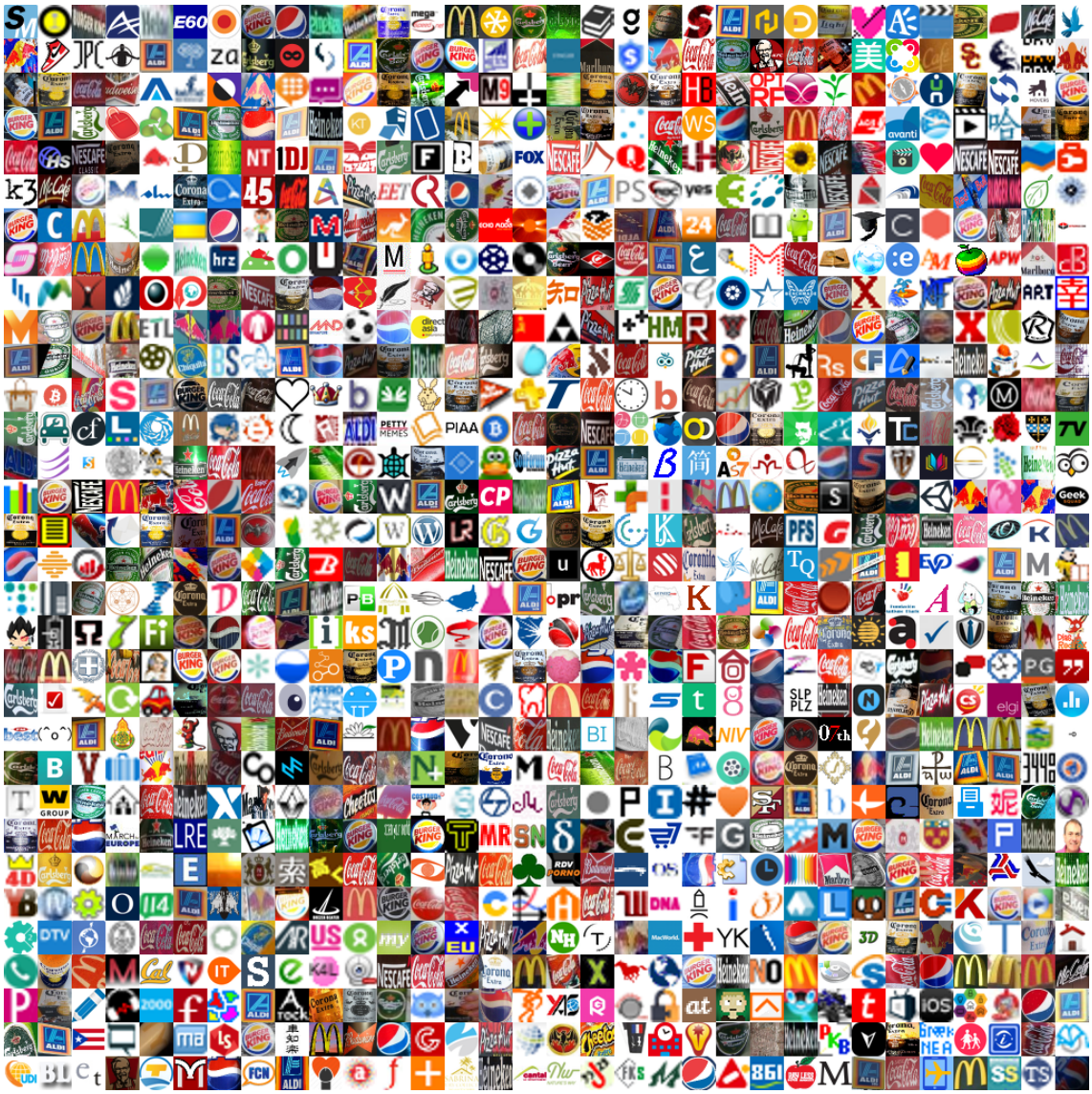

Figure 7.4 Real Images from Dataset

Due to the feature of Google Colab, GPU usage rights are granted for 12 hours before the browser is closed. After this time, the session falls, timeout and the model training is unfinished. To prevent this, the following code snippet is run from the "console" tab using the browser developer tools feature.

```
function ClickConnect(){
console.log("Working");
document.querySelector("colab-toolbar-button#connect").click()
}
setInterval(ClickConnect,60000)
```

8
Performance Analysis

Performance Analysis made in this section is explained.

We have verified that the results match the paper when training with 1, 2, 4, or 8 GPUs. Note that training FFHQ at 1024×1024 resolution requires GPU(s) with at least 16 GB of memory. The following table lists typical training times using NVIDIA DGX-1 with 8 Tesla V100 GPUs:

Configuration	Resolution	Total kimg	1 GPU	2 GPUs	4 GPUs	8 GPUs	GPU mem
config-f	1024×1024	25000	69d 23h	36d 4h	18d 14h	9d 18h	13.3 GB
config-f	1024×1024	10000	27d 23h	14d 11h	7d 10h	3d 22h	13.3 GB
config-e	1024×1024	25000	35d 11h	18d 15h	9d 15h	5d 6h	8.6 GB
config-e	1024×1024	10000	14d 4h	7d 11h	3d 20h	2d 3h	8.6 GB
config-f	256×256	25000	32d 13h	16d 23h	8d 21h	4d 18h	6.4 GB
config-f	256×256	10000	13d 0h	6d 19h	3d 13h	1d 22h	6.4 GB

Figure 8.1 Performance Analysis of StyleGAN2 with example FFHQ dataset

On Google Colab has 1 Tesla V100 GPU and used dataset resolution 32x32, so performance analysis for project is 32 Hour for 1000 iterations.

Conditional StyleGAN is the modified StyleGAN and training times longer than optimized StyleGAN2. For training, the Conditional StyleGAN model needs to train 40% longer and this mean 44 Hours need for 1000 iterations.

9
Result

Results made in this section is explained.

As a conclusion of our experiments, we could see that we can get the same result by using StyleGAN2 instead of using Conditional StyleGAN to produce a categorized logo. The only difference is that we produce 1 model for the conditional StyleGAN. For StyleGAN2, we have to produce a separate model for each category. but we can get a better result in terms of performance. And using also big dataset with mixed synthesis images is better than real logo datasets. when models trained with real logos we can produce every time real or similar logos.

References

[1] J. Brownlee. (2020). "A gentle introduction to stylegan the style generative adversarial network," [Online]. Available: https://machinelearningmastery.com/introduction-to-style-generative-adversarial-network-stylegan/.

[2] T. A. Tero Karras Samuli Laine, "A style-based generator architecture for generative adversarial networks," 2019. [Online]. Available: https://arxiv.org/abs/1812.04948.

[3] "Analyzing and improving the image quality of stylegan," 2020. [Online]. Available: https://arxiv.org/abs/1912.04958.

[4] G. S. Cedric Oeldorf, "Loganv2: Conditional style-based logo generation with generative adversarial networks," 2019. [Online]. Available: https://arxiv.org/abs/1909.09974.

[5] "Improved training of wasserstein gans," 2017. [Online]. Available: https://arxiv.org/abs/1704.00028.

[6] (). "Favicon: How to use," [Online]. Available: https://en.wikipedia.org/wiki/Favicon#How_to_use.

[7] "Logo synthesis and manipulation with clustered generative adversarial networks," 2017. [Online]. Available: https://arxiv.org/abs/1712.04407.

[8] H. Su, X. Zhu, and S. Gong, "Open logo detection challenge," in *British Machine Vision Conference*, 2018.

Curriculum Vitae

FIRST MEMBER

Name-Surname: Alibek ERKABAYEV
Birthdate and Place of Birth: 06.03.1995, Türkmenistan
E-mail: alibek060395@gmail.com
Phone: 0554 510 54 15
Practical Training: YTU Donanım anabilim dalı

Project System Informations

System and Software: Linux OS, Windows OS, x64 Python3.6, Tensorflow==1.15, Nvidia GPU
Required RAM: 16GB
Required Disk: 20GB