END TO END 3D FACE MODEL SYNTHESIS USING TEXTUAL DESCRIPTIONS

A Thesis

by

M. Uluç Şahin

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Computer Science

Özyeğin University
June 2020

END TO END 3D FACE MODEL SYNTHESIS USING TEXTUAL DESCRIPTIONS

Approved by:

Asst. Prof. M. Furkan Kıraç, Advisor
Department of Computer Science
Özyeğin University

Prof. Erhan Öztop
Department of Computer Science
Özyeğin University

Prof. H. Kemal Ekenel
Department of Computer Engineering
Istanbul Technical University

Date Approved: 10 June 2020

To my family and friends.

ABSTRACT

Although there are various generative models that successfully generate photo-realistic images, these models have no way of controlling generated images. Research on conditional generative models, which allow us to control generated images, is quite limited. Furthermore, research on generating realistic human face images from given natural language descriptions is limited as well. Generated images are either low quality, or lack variance. To solve this problem, we propose Conditional StyleGAN (cStyleGAN), a variation of StyleGAN that is capable of separating high dimensional features and generating high quality images that are conditioned on supplied text descriptions. Our cStyleGAN is able to generate high quality human face images that align with the given text descriptions. We are also extending CelebA human face dataset with our Description Generation Module by providing additional natural language descriptions for images, which can be used in training of cGANs to generate 2D human face images. 2D images can provide good information. However, being able to see an image from different angles and in different illumination, and being able to see it with depth information as a 3D model can transmit more valuable information compared to 2D images. For this reason, we are also providing an end-to-end architecture for generating 3D facial structures from given natural language descriptions.

ÖZETÇE

Gerçek resim kalitesinde resimler üretebilen çeşitli üretici modeller olmasına rağmen, bu modeller üretilen resimleri kontrol edememektedir. Üretilen resimlerinin kontrolünü sağlayan Koşullu Çekişmeli Üretici Ağlar alanındaki araştırmalar oldukça limitlidir. Dahası, doğal dil yapısında verilen betimleyici açıklamaları kullanarak gerçekçi insan yüzü resmi üretebilen sinir ağları ile ilgili araştırmalar da yetersizdir. Üretilen resimler ya düşük kalitede olmakta, ya da çeşitlilik sağlayamamaktadır. Bu sorunu çözmek için, Stil Çekilmeli Üretici Ağlar (StyleGAN) yapısının bir çeşidi olan, ve yüksek boyuttaki özellikleri ayırıp, yüksek kaliteli, verilen betimlemeler üzerinde koşullanmış resimler üretebilen Koşullu Stil Çekişmeli Üretici Ağ (cStyle-GAN) yapısını öne sürüyoruz. Ayrıca, CelebA veri kümesini, Tanımlama Üretim Modülü'müz (Description Generation Module) ile resimler için ürettiğimiz doğal dil yapısındaki, iki boyutlu resimlerin üretiminde kullanılacak olan koşullu üretici ağların eğitiminde faydalı olacak tanımlamalar ile genişletiyoruz. İki boyutlu resimler iyi bir bilgi sağlamasına rağmen, bir resmi farklı açılardan, farkı ışıklar altında, ve bir derinlik bilgisi ile gözlemlemek daha fazla bilgi iletebilir. Bu yüzden, verilen doğal dil tanımlamalarından üç boyutlu insan yüzü modelleri üretebilen uçtan uca yapımızı da öne sürüyoruz.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis advisor Asst. Prof. M. Furkan Kıraç. He guided me and mentored me patiently throughout my study with his experience and knowledge.

I would also like to thank my family; my parents Sibel and Ali, for supporting me and believing me throughout my life.

TABLE OF CONTENTS

DEDICATION . . . . **iii**

ABSTRACT . . . . **iv**

ÖZETÇE . . . . **v**

ACKNOWLEDGEMENTS . . . . **vi**

LIST OF TABLES . . . . **x**

LIST OF FIGURES . . . . **xi**

I INTRODUCTION **1**

1.1 Contributions 4

1.2 Thesis Outline 5

II BACKGROUND **6**

2.1 Autoencoders 6

2.1.1 Likelihood Ratio 7

2.1.2 Kullback-Leibler Divergence 8

2.1.3 Variational Autoencoders 9

2.1.4 Conditional Variational Autoencoders 10

2.2 Generative Adversarial Networks 11

2.2.1 Naive GANs 11

2.2.2 Wasserstein GANs 18

2.2.3 Conditional GANs 22

2.2.4 Evaluation Metrics of GANs 25

2.3 Natural Language Processing 27

2.3.1 Skip-Gram Model 28

2.3.2 FastText 28

2.4 StyleGAN 29

2.4.1 Batch Normalization 31

2.4.2 Instance Normalization 32
2.4.3 Adaptive Instance Normalization (AdaIN) 33
2.4.4 Mapping Network 34
2.4.5 Synthesis Network 34
2.5 Position-map Regression Network 35
2.5.1 UV Mapping 35
2.5.2 What is Position-map Regression Network 35
2.6 Learned Perceptual Image Patch Similarity (LPIPS) 37
III PREVIOUS WORK 39
IV METHODOLOGY 44
4.1 Perceptual Quality Distance 46
4.2 Conditional StyleGAN 49
4.2.1 Discriminator Type 1 51
4.2.2 Discriminator Type 2 51
4.3 3D Facial Structure Pipeline 53
4.4 Datasets 55
4.4.1 Extended Dataset 56
4.4.2 Shapes Toy Dataset 56
4.5 Description Generation Module 57
4.6 Experiments 59
4.6.1 cStyleGAN on Face2Text Dataset 60
4.6.2 cStyleGAN on Extended Dataset 60
4.6.3 Noise Scaling and Effect of Noise 61
4.6.4 Data Augmentation 62
4.6.5 Ablation Study: Disabling Blocks in Synthesis Network . . . 63
4.7 Results 64
4.7.1 Results of cStyleGAN Trained on Face2Text 65
4.7.2 Results of cStyleGAN Trained on Extended Dataset 67

4.7.3 Comparison with Existing Work 69

4.7.4 Results of 3D Generation 70

V CONCLUSION 72

APPENDIX A — SAMPLES 74

APPENDIX B — CSTYLEGAN LAYERS 80

REFERENCES 85

LIST OF TABLES

1 Number of eliminated images for corresponding reason and total generated description count. 59

2 Applied transformations with their chance of being applied. 63

3 Comparison of PQD scores. Best result achieved with our model when trained with Extended Dataset. 70

4 Comparison of Inception Scores. Best result achieved with our model when trained with Extended Dataset. 70

5 Comparison of 3D generation speed of PRN for different input sizes. 71

6 40 annotations available in CelebA Dataset. 77

LIST OF FIGURES

1 An autoencoder architecture example. 7

2 A variational autoencoder architecture example. 9

3 An example architecture of Conditional Variational Autoencoder. . . 11

4 Generative Adversarial Network architecture. 12

5 An example of mode collapse. Images generated with different inputs are the same. 15

6 Architecture of conditional GAN. 23

7 Network architecture of StyleGAN proposed by [9]. Above the dashed line is mapping network while below is synthesis network. 31

8 Network architecture of Position-map Regression Network. 37

9 Proposed end-to-end system architecture. 46

10 PQD between similar quality samples and CelebA dataset. One example from each category is shown. 49

11 Architecture of cStyleGAN Generator. Above the dashed line is mapping network while below is synthesis network. 50

12 Architecture of Discriminator Type 1. 52

13 Architecture of Discriminator Type 2. 52

14 Height-map estimation obtained from generated 2D images. 54

15 Comparison of default PRN models and improved models. 55

16 Examples on toy dataset with Discriminator Type 1. Only one in four images is aligned with given descriptions. 57

17 Examples on toy dataset with Discriminator Type 2. All images are correctly aligned with descriptions. 57

18 Examples images from cStyleGAN trained on Face2Text dataset. . . . 60

19 Generator and Discriminator losses over training epochs. 61

20 Results of same description with different noise values. 62

21 Examples from model trained with augmented training set. 63

22 Results of disabling different early blocks in synthesis network of generator. 64

23 cStyleGAN mode collapsed on Face2Text dataset with 1.5 million iterations for each resolution. 66

24 Results of cStyleGAN on Face2Text dataset trained 600K iterations per resolution. 66

25 Results of cStyleGAN trained on Extended dataset. 67

26 Results of cStyleGAN trained on Extended dataset in zero-shot (unseen text) setting. 68

27 Comparison of outputs from Pro-StackGAN and cStyleGAN. 69

28 Examples of 2D images and corresponding 3D facial structures generated from textual descriptions. 71

29 Face2Text data training samples with human written descriptions. . . 74

30 Samples from CelebA dataset which does not exist in Face2Text dataset, with added descriptions generated by our module. 75

31 Voxelated human face model with 64x64x64 resolution. 76

32 Voxelated human face model with 128x128x128 resolution. 76

CHAPTER I

INTRODUCTION

Inspired by how neurons in human brain works, history of artificial neural networks has its roots as early as early 1940s. In 1943, neurophysiologist Warren McCulloch and mathematician Walter Pitts published their paper [1] where they discussed relation between neurons and logical calculus, followed by Donald Hebb in his book named The Organization of Behaviour [2]. In 1954, Massachusetts Institute of Technology (MIT) developed Hebbian Networks, which translated neural networks onto computational systems for the first time. Followed by many others [3][4][5][6][7], neural networks have increased in popularity as computational power increased. Computers became more accessible and got faster, and translating neural networks onto computer systems became a viable option. Especially in the last decade, along with increased popularity of neural networks in various scientific domains, we even have started to witness that they are becoming an integrated part of our daily lives.

As research on neural networks increased, different types of neural networks with different use cases have emerged. In this work, we have used Generative Adversarial Networks (GANs) [8], which are a type of generative neural networks that are used to generate new samples from underlying data distribution of a given dataset. Samples generated by GANs are new instances, meaning that they don't exist in the dataset, but are similar to available instances in learned samples. GANs achieve this by training two neural networks that compete against each other in a minimax game. A generator tries to generate images that can trick the discriminator, and a discriminator that tries to detect if a given image instance is coming from the dataset or generated by a generator.

It is important to understand what is happening in the background of used neural networks, various techniques and other decisions. Without background knowledge, it is not possible to go further and improve upon existing work. In this thesis, mathematical background of used techniques as well as techniques themselves are explained in detail. For the same reason, autoencoders are discussed in their separate section.

There are various problems about GANs, such as instability in training, lack of proper evaluation metric, long training times, and not being able to control the output of network. Which are partly solved by improvements made in the last years. We are utilizing Wasserstein Metric for better training stability and as a better evaluation metric, we are also using progressively growing method of GANs for improved training times and further increased stability in training. Specifically, we have utilized StyleGAN [9], which is an improved Progressively Growing GAN (ProGAN) [10] that allows separating style and content information of an image. Output conditioning is achieved in our work by feeding a conditional variable as an extra input to our GAN. In this work, we propose Conditional StyleGAN (cStyleGAN), which achieves competitive performance with previous text to face generation methods while generating high quality human face images that align with the given text descriptions. Our approach allows increased image fidelity and resolution in generated human face images while maintaining description-image alignment.

There is insufficient research in generating human face images from given natural language descriptions. Likewise, existing data in this literature is insufficient. Although there are datasets such as CelebA [11], which contains 202599 celebrity face images, these datasets do not contain natural language descriptions written by humans. CelebA contains 40 annotations for each image in the dataset, which describe visual properties of the person in the image. We have developed a module that generates natural language descriptions from these annotations, which is used for training our cStyleGAN. Another dataset is Face2Text [12], which contains 4076 human face images that are annotated by humans. However, this dataset alone is insufficient for our task. We have combined the data existing in Face2Text dataset with the data obtained by generating descriptions with our module for using in the training of our network.

Describing a human face has important applications, such as describing a criminal to police for criminal identification. However, visualising a human face described by other people is a hard task for humans, and given descriptions may not always be accurate. Person who describes the face may not be remembering all the details, or even worse, may remember the details wrong. In such circumstances, using a tool that generates human faces from given descriptions would be invaluable. Seeing an image of a person which suits given description may help remembering the details, or in the best case, person being tried to be described may be visualized exactly by neural network, which removes the need of manually sketching the person. Despite focused studies in this field, this problem is far from being solved. Generated images are either low quality, or not completely aligned with given descriptions. In this study, we have achieved competitive results with state of the art methods, and achieved high quality images with high variance, which are aligned with descriptions, with the ability of observing generated faces in three dimensions.

2D images can provide important information. However, being able to see an image from different angles and in different illumination, and being able to see it with depth information as a 3D model can transmit more valuable information compared to 2D images. For this reason, we also generate 3D facial structures of 2D human face images generated from textual descriptions by using Position-map Regression Network (PRN) [13]. Generated 3D models can be easily used in 3D software, and can be utilized like any other 3D model.

1.1 Contributions

Our contributions in this work has different key-points. In the existing research, generated images are either low quality, not completely aligned with given descriptions, or lacking variety. We have improved upon existing methods and have achieved high quality images with high variety which are aligned with descriptions. Utilizing StyleGAN architecture, we have implemented cStyleGAN, which is capable of separating high dimensional features, such as hair color, hair type and length, facial structure features, gender etc. Generated images are conditioned on conditioning variable given as input, meaning that we can change said features of generated output in a controllable manner by giving natural sentence descriptions. Our model achieves competitive results with state of the art methods in text to face generation domain, and successfully generates high quality human face images that are aligned with the given text descriptions. Additionally, we are providing an end-to-end structure for generating 3D facial structures that are aligned with the given text descriptions. We have improved results of 3D generated facial structures by applying various processes to 3D generation pipeline.

We are also extending CelebA dataset by generating natural language descriptions from annotations available in CelebA dataset. Our system is able to generate descriptions for 203K images in few seconds, which can be used with (or in place of) annotations and descriptions written by humans to train neural networks. Our Description Generation Module uses a rule based algorithm to generate sentences in grammatically correct form, with added randomness to descriptions for minimizing memorizing problem in neural network during training. Furthermore, the system eliminates low quality annotations by using multiple methods: annotations that are conflicting are eliminated, such as marking beard as not available while marking a goatee as available. Also, annotations that contain no information about an important feature, such as hair, is eliminated. Finally, images which do not have certain amount of annotations marked as available are eliminated, where the number is a parameter that can be changed according to needs of users.

Finally, we are proposing our novel GAN evaluation measure, Perceptual Quality Distance (PQD). To our knowledge, this is the first GAN evaluation measure that directly uses perceptual properties of images. We show that PQD distance is inversely proportional (lower is better) to image quality and similarity of image to real dataset, and we show our results on images of various qualities. We show that our model achieves better PQD score compared to existing work, which is also aligned with Inception Score rankings reported in this thesis.

1.2 Thesis Outline

Rest of this thesis is structured as follows:

Chapter 2 talks about previous work that our work is built upon: Chapter 2.1 talks about Autoencoders and variations, which GANs are based upon. Chapter 2.2 has explanations of GANs and variations, along with techniques, loss functions and mathematics used in GANs. Chapter 2.3 covers the Natural Language Processing techniques that is relevant for our work. Chapter 2.4 covers the StyleGAN which our work is based upon and discusses our contribution, cStyleGAN. Chapter 2.5 explains how PRN works and how it is beneficial in our work for creating 3D facial structures.

Chapter 3 discusses previous work in the domain of generating human face images from natural language descriptions.

In Chapter 4 we show our experiments and results on both Face2Text dataset and our extended dataset, and we discuss upon our results.

In Chapter 5 we conclude this thesis with an overview of our work and discussion.

CHAPTER II

BACKGROUND

In this Chapter, we discuss previous work that this thesis is built upon. First, we discuss about autoencoders, which GANs are built upon. Then we extend our discussion to GANs. We discuss about NLP to show methods that are beneficial for our work. We talk about StyleGAN, which our cStyleGAN is based upon. Finally, we talk about PRN, which is used for generating 3D models in our work.

2.1 Autoencoders

Autoencoders are a type of neural network architecture that are used for learning efficient, low dimensional data encodings for a given input. In its simplest form, an autoencoder consists of two main parts; an encoder and a decoder, where encoder maps given input into a low dimensional representation, which we refer as bottleneck in this work, and the decoder, which uses this representation for generating an image, ideally same as the given input. Currently, some of the most popular applications of autoencoders are denoising and dimensionality reduction. In our work, we may think of autoencoders as a stepping stone for Generative Adversarial Networks. We discuss about autoencoders in this work as an introduction to GANs.

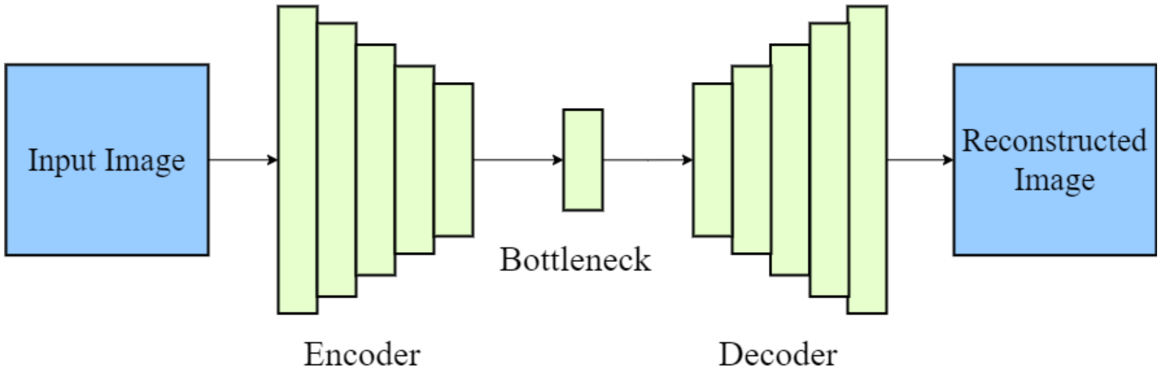


Figure 1: An autoencoder architecture example.

Although there may be different cost functions for autoencoders with different purposes, such as denoising autoencoders, cost function for calculating how good the generated image in a basic autoencoder is simply calculated as pixelwise difference between input image and generated image. Two main types of loss function are used for autoencoders that try to reconstruct given input image; Mean Squared Error (MSE), and Kullback-Leibler (KL) divergence. In this work, we talk about Likelihood Ratio (LR) as it is used in KL divergence, and we discuss KL Divergence itself. We omit the discussion about MSE as it is not used in our work.

2.1.1 Likelihood Ratio

Let us assume we have two probability distributions p, generated by our neural network, and q, distribution of the dataset being used. A metric is required for measuring how good p is compared to q. LR can be described as ratio of two probability distributions, which can be used for this purpose.

$$LR = p(x)/q(x), \tag{1}$$

where x is any sample. We want p to be as close as q. LR measures the probability of a sample being in p compared to q, a LR value larger than 1 means that the sample x is most probably in p, while a value smaller than 1 indicates that x is most probably in q.

We can compute LR for a dataset by multiplying LR values of independent samples in the dataset:

$$LR = \prod_{i=0}^{n} \frac{p(x_i)}{q(x_i)}. \tag{2}$$

Calculations may get quite expensive if there are a lot of data. For decreasing computational complexity, it is desirable to use log10 form of (2):

$$log_{LR} = \sum_{i=0}^{n} log\left(\frac{p(x_i)}{q(x_i)}\right), \tag{3}$$

which has summation instead of multiplication.

2.1.2 Kullback-Leibler Divergence

In simplest words, Kullback-Leibler Divergence (KL Divergence) measures divergence between p and q. In the following examples, we make the assumption that both p and q are continuous distributions, however, KL Divergence is applicable for discrete distributions as well. KL Divergence for continuous probability distributions can be found by taking expected value of likelihood ratio:

$$D_{KL}(P \mid\mid Q) = \sum_{i=0}^{n} p(x_i) log\left(\frac{p(x_i)}{q(x_i)}\right). \tag{4}$$

KL Divergence is valid for both discrete and continuous probability distributions. For continuous probability distributions, Equation (4) changes slightly and uses integral instead of summation:

$$D_{KL}(P \mid\mid Q) = \int_{-\infty}^{\infty} p(x) log\left(\frac{p(x)}{q(x)}\right) dx. \tag{5}$$

KL Divergence considers one probability as the ground truth while comparing distributions. Changing places of distributions in formula would mean that changing the ground truth, therefore it should be noted that KL Divergence is not symmetrical:

$$D_{KL}(P \mid\mid Q) \neq D_{KL}(Q \mid\mid P). \tag{6}$$

While calculating $D_{KL}(P \mid\mid Q)$, in cases where $q(x)$ takes the value 0 results in $p(x)/q(x)$ value to be infinity. Value of KL divergence would not make sense if there is no overlap between two probability distributions. This is not desirable from neural network perspective since it would cause gradients to go in unwanted directions. A solution is adding continuous noise to input to achieve overlapping at between the two probability distributions.

2.1.3 Variational Autoencoders

Autoencoders may not generate meaningful output for an input which is taken from an unknown probability distribution. Output for such input would be randomly generated pixels. Autoencoder has no idea if the vectors that we are passing to decoder are valid vectors or not. Variational Autoencoders (VAE) on the other hand, can get a random input and still generate meaningful outputs. VAEs solve this problem by forcing encoder to produce a probability distribution function over encodings instead of producing single encoding of a particular image. Instead of different single values, VAEs have distribution function for each value in the latent variable. VAEs restrict latent variable μ generated by encoder to some predefined distribution (such as normal distribution). Encoder in VAE generates parameters for such distribution instead of generating values in latent space directly. So, the decoder trained with such values generates meaningful images when a random μ from this distribution is selected as input to the decoder.

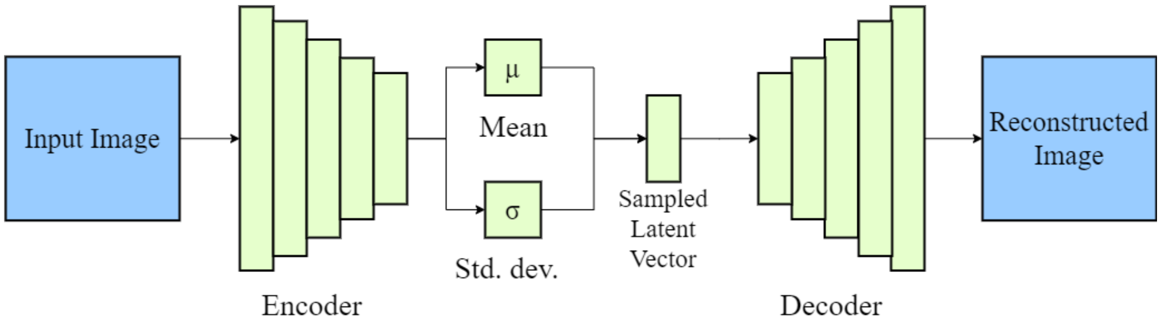


Figure 2: A variational autoencoder architecture example.

2.1.4 Conditional Variational Autoencoders

Although VAEs generate meaningful images even from randomly selected values, it is not possible to control generated output using VAEs. In VAEs, there is no such parameter that controls the output for a given input. That is, values in latent representation, which are generated by encoder, cannot be interpreted (at least with current technology), and the output of VAE cannot be controlled. This also means that while VAEs are good for reconstructing input, it cannot work as a controlled generative model. If we feed a random latent vector to decoder it may not give a reasonable image, or the generated image may be different than what we desired it to be.

To solve this problem, another input called Conditioning Variable is fed into both encoder and decoder during the training. So that values in latent space generated by encoder are conditioned on this extra input, and decoder learns to output images that represent desired shape or form corresponding to given conditioning variable. For example, if MNIST [14] dataset is being used, it is feasible to feed a 10 dimensional one hot vector that represents classes in MNIST dataset. If we feed one hot vector which has "1" in the first dimension along with pictures of zero during training, then the network learns to output picture of number zero if we give a one hot vector which has "1" in the first dimension along with a randomly selected latent vector. In layman's terms, decoder is correlating a class of image with a certain value and generates from that class when correlated input is given.

An important point to note here is, Conditional VAEs (cVAEs) rely on provided conditioning variable to generate an image from a certain class while it relies on latent vector taken from a probability distribution to chose other properties of selected class. For example, such as angle, thickness etc. are decided according to values in the sampled latent values if we are generating images of numbers in MNIST dataset.

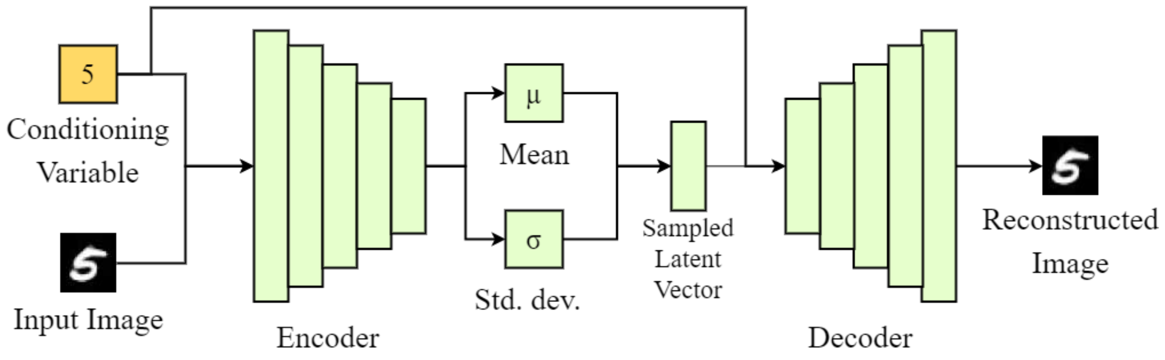


Figure 3: An example architecture of Conditional Variational Autoencoder.

2.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are type of generative neural networks that are used to generate new samples from underlying data distribution of a given dataset. Similar to autoencoders, samples generated by GANs are new instances, meaning that they do not exist in the dataset, but are similar. In this section, we discuss Conditional GANs (cGANs) [15] that are used in our work, and Naive GANs which Conditional GANs are based upon. We also discuss various loss functions and metrics that are used in GANs in this section.

2.2.1 Naive GANs

We use the word "adversarial", which means "involving or characterized by conflict or opposition" [1]. GANs are composed of two different neural networks which compete against each other in a minimax game. This is where the word adversarial is coming from. A generator, which is a generative model that tries to generate samples that can trick the discriminator, and a discriminator, which is a discriminative model that classifies a given sample as either real, meaning that instance is coming from the dataset, or as fake, meaning that instance is generated by a generator.

Aim of generator is to output samples that are close to real samples, which can

[1]https://www.lexico.com/en/definition/adversarial

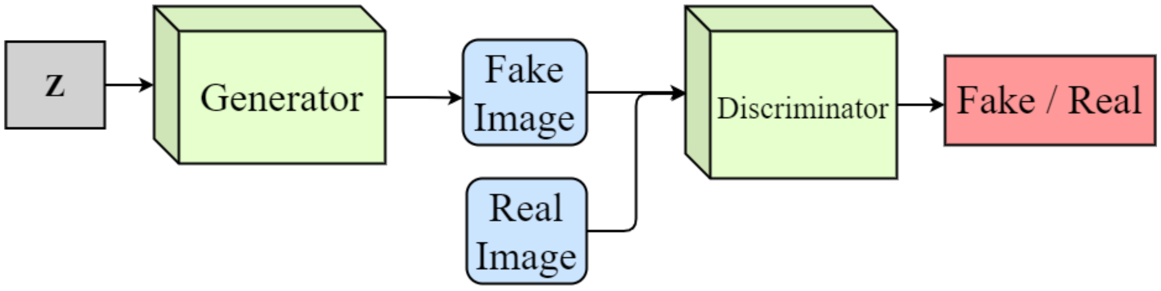


Figure 4: Generative Adversarial Network architecture.

be mistaken as real by the discriminator. Here, generator only takes some noise, or a latent space from a random distribution as input. Discriminator similarly, receives single input in the form of a sample along with corresponding label. However, this sample may be coming from two different sources: a fake sample generated by the generator, or a real sample taken from the dataset. Labels are given during training, so that the discriminator learns to discriminate between real and fake samples as the training continues. As the discriminator gets better in distinguishing between fake and real samples, the generator would need to generate samples that are closer to real data in order to continue tricking the discriminator. This minimax game ensures that the generator is good at outputting samples that are close to underlying probability distribution of dataset used during training.

Here the discriminator outputs values between 0 and 1, 0 if it is perfectly sure that it is a fake sample, and 1 if it is perfectly sure that it is a real image, or vice versa. In the case of perfectly trained, ideal generator, discriminator would output 0.5, which means that samples generated by a generator are so close to real samples that discriminator cannot decide if a sample is coming from the generator or from dataset.

Discriminator is trained with both fake samples and real samples. It is easy to label data as fake or real because we know which samples are fake, and which samples are coming from dataset. These labels are consistent throughout the training, meaning that real samples stay in the real category and fake samples always stay in fake category. However, we cannot say that there is a similar consistency for the generator. Generator's ability to generate realistic samples are measured by discriminator, since the discriminator is always changing throughout the training, there is no such consistent measure available for generator. It is harder to train generator compared to discriminator in GANs because of discussed conditions. Discriminator is a network that only learns how to classify between real and fake samples, whereas generator needs to learn how to generate realistic looking, diverse samples that are similar to real data.

Another point that should be noted is that generator and discriminator are trained in alternate. We discussed that consistency for measuring discriminator does not apply for generator. To compensate, in some architectures, discriminator is trained k iterations for each training iteration of generator, which ensures the metric that measures generator's performance is based on a more optimal, better trained discriminator.

Back propagation algorithm of GAN works in a similar fashion with other neural networks. Loss obtained from discriminator is back-propagated through network for updating weights and parameters. However, since we do not have a distinctive loss function for generator, updating weights of generator network depends on loss obtained from the discriminator.

2.2.1.1 Loss Function of Naive GANs

In GANs, we have two different loss functions; one for discriminator, and another one for generator. GAN's aim is finding the underlying probability distribution so that the generator can generate samples that are as close as possible to real samples.

In 2.2.1 we stated that discriminator and generator are competing against each other in a minimax game. Equation for this minimax game is shown in (7).

$$\min_{G}\max_{D} V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1 - D(G(z)))], \qquad (7)$$

where G is Generator, D is Discriminator, x is real sample and z is noise. D tries to maximize probability of classifying samples into correct label, while G tries to minimize $log(1 - D(G(z))$. Note that while D is directly affecting the error function while G does this through D. G can only achieve lower loss (or error) values by making sure that D has high loss value. G can only affect $log(1 - D(G(z))$, so, $log(D(x))$ part has no direct effect on generator during the training.

In the GAN paper [8], it is stated that this error function may cause generator to stuck. Discriminator's job is easy at the beginning since generated samples are very different from real samples. This causes *log (1 - D(G(z))* to saturate. To solve this problem, [8] states that it is better to train G to maximize *log D(G(z))* instead of training G to minimize *log (1 - D(G(z))*. This provides a better training for GAN early on.

2.2.1.2 Pros and Cons of Naive GANs

Neural network models are generally known as data hungry. Usually, obtaining data is hard and may be expensive depending on the type and amount of data is required. In case of supervised learning, we need data that is labeled so that we can train our model. Although amount of data required is considerably high, an advantage of GANs is that they do not need labeled data as it is a form of semi-supervised learning.

An important weakness of GANs is that there is no evaluation metric which measures performance of them. Both Inception Score and Fréchet Inception Distance are not directly evaluating the quality of generated samples. There is not a suitable metric that can be used to compare two different GANs. Despite GANs being a popular area of research, results are often measured by human eye. GANs are also harder to train compared to VAEs since there is not a clear evaluation metric, and thus no clear objective, to minimize in GANs.

Training of GANs is not always stable. During the training GAN, it is not unusual to face mode collapse. In case of mode collapse, some modes of multi-modal data are missed by the generator. In the example of a GAN trained on MNIST dataset, the generator may generate only few classes no matter what input is given to it. For example, generator may only generate images of "0" and "5" and not others, or even worse, generator may generate the same image of a single class among all the classes. This occurs when generator finds a single solution that the discriminator cannot distinguish from real image. In naive GANs, there is no incentive for generator to generate different distribution of samples, so, generating same image that successfully tricks the discriminator is completely acceptable by generator. In such a case, minimax equation reaches to an equilibrium and the training stops. Training must be restarted or continued from a previous checkpoint. An example of mode collapse on model trained on CelebA dataset can be seen in Figure 5.

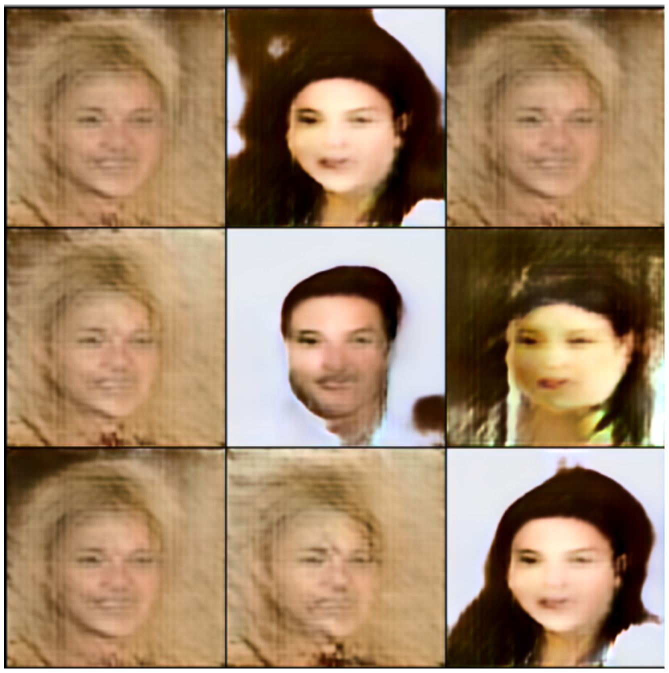


Figure 5: An example of mode collapse. Images generated with different inputs are the same.

Although not exactly a disadvantage, we should point that practically it is not possible to control the samples generated by GANs. GANs in this form (without a conditioning variable) have no way of controlling the output. Similar to VAEs, theoretically it is possible to control output by changing latent variables that effect certain properties of generated samples. However, this is practically not viable with current technology. Conditional GANs solve this problem and allow controlling output with additional input variable, which is discussed in Section 2.2.3.

2.2.1.3 Jensen-Shannon Divergence

Jensen-Shannon Divergence (JSD) is a method based on KL Divergence. Similar to KL Divergence, (JSD) is used for measuring the similarity between two probability distributions. Unlike KL Divergence, JSD has finite boundary which is quite useful for generative neural networks. JSD is defined as follows:

$$JSD(P \mid\mid Q) = \frac{1}{2}KL(P \mid\mid M) + \frac{1}{2}KL(Q \mid\mid M), \tag{8}$$

where $M = \frac{1}{2}D(P \ + \ Q)$.

Square root of JSD is Jensen-Shannon Distance, which is a metric (satisfies properties of a distance measure). It should also be noted that JSD is symmetric unlike KL Divergence:

$$JSD(P \mid\mid Q) = JSD(Q \mid\mid P) \tag{9}$$

The most important contribution of JSD is that it works even if there is no overlap between the two probability distributions. JSD is bounded by 1 for two probability distributions:

$$0 \leq JSD(P \mid\mid Q) \leq 1 \tag{10}$$

However, JSD falls short in taking the distance between two probability distributions into consideration. Let us assume we have three Gaussian probability distributions $p1$, $p2$ and $p3$ with μ values of 1, 10 and 50 respectively, and σ values of 1 for each. Assuming there is no overlap between these three distributions, although similarity between $p1$ and $p2$ is higher than the similarity between $p1$ and $p3$, JSD between $p1$ and $p2$ is equal to JSD between $p1$ and $p3$.

If two probability distributions are far away from each other, meaning that they have no overlap, then JSD would take value of $log2$. Derivative at this point would be 0, we cannot get useful gradient information, and the training gets very slow or completely halts.

2.2.1.4 Wasserstein Distance (Earth Mover Distance)

In Section 2.2.1.3, we discussed that JSD falls short in taking the distance between two probability distributions into consideration. Wasserstein Distance (also known as Earth Mover's Distance) is a metric that is used for measuring how different the two probability distributions are from each other. Similar to JSD, Wasserstein Distance is symmetric. In this work, the term Wasserstein Distance is used to denote 1^{st} Wasserstein Distance (or Wasserstein - 1).

It would be correct to make the following comment about KL Divergence, JSD and Wasserstein Distance: assuming we have x values on x axis, and $p(x)$ and $q(x)$ values on y axis where p and q is a probability distribution. KL and JSD would not take horizontal space (distance on x axis) between distributions into consideration whereas Wasserstein Distance does when calculating how different these two probability distributions are. In case where there is no overlap between distributions, KL Divergence would be infinity whereas JSD and Wasserstein Distance would give definite values.

Wasserstein Distance is also known as Earth Mover's Distance (EMD) in computer science. There is a famous example given when EMD is being explained: assume that there are two piles of dirt with same mass, EMD between these two piles of dirt is how much energy required to transform one pile of dirt into another. Here, energy is measured by distance each unit of dirt moved. So, we can explain Wasserstein Distance as minimum amount of work required to change one shape of probability distribution into shape of other probability distribution. The p^{th} Wasserstein Distance is defined as follows:

$$W_p(\mu, \nu) := \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \int ||x, y||^p d\gamma(x, y) \right)^{1/p}, \tag{11}$$

where $p \geq 1$ and $\Gamma(\mu, \nu)$ denotes all joint distributions γ for (x, y) with marginals μ and ν. When $p = 1$, the formula is equal to EMD.

Another important feature of Wasserstein Distance is that it is continuous and differentiable almost everywhere. This is especially helpful in GAN training for obtaining meaningful gradients during training, which is an important part of success of wGANs.

2.2.2 Wasserstein GANs

We have discussed that GANs may face mode collapse during training, where generated samples are same even if the inputs are different. Wasserstein Distance is an important step towards solving this problem, as well as alleviating vanishing gradient problem even though it does not completely solve it. Arjovsky et al. [16] used Wasserstein-1 Distance in their GAN architecture, which is called Wasserstein Generative Adversarial Network (wGAN). wGANs also offer a better loss function in the form of Wasserstein Distance where it is correlated with quality of images.

In Section 2.1.2 we have discussed that KL divergence is infinite (or not defined) when there is no intersection between two probability distributions. So, adding a noise to form an intersection between two probability distributions is a common approach in neural networks. In wGANs, since Wasserstein Distance works even if there is no overlapping between the two probability distributions, adding noise for creating an overlapping between them is not necessary, although still used for purposes such as avoiding memorizing training data.

It is beneficial to discuss how Wasserstein Distance is used in wGANs. First, we should look at the definition of **Lipschitz Continuity**: if a real-valued function $\mathbf{f}: \mathbb{R} \rightarrow \mathbb{R}$ satisfies the property $||\mathbf{f}||_{\mathbf{L}} \leq \mathbf{K}$, then it is called K-Lipschitz Continuous. Functions that are continuously differentiable everywhere are called Lipschitz Continuous if a real constant exists such that $\mathbf{K} \geq \mathbf{0}$ for all $\mathbf{x_1}, \mathbf{x_2} \in \mathbb{R}$,

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2|. \tag{12}$$

If we look at how [16] transformed Wasserstein Distance and used following equation in wGAN:

$$W_p(p_r, p_g) = \frac{1}{K} \sup_{||f||_L \leq K} \mathbb{E}_{x \sim p_r}[f(x)], \tag{13}$$

we see that sup (supremum) is used as opposed to inf (infimum) in Equation (11), which is the least upper bound. Supremum in this formula is defined over all 1-Lipschitz functions, meaning that it is Lipschitz Continuous and continuously differentiable everywhere. Assuming that parameterized family of functions $\{f_w\}_{w \in W}$ are all K-Lipschitz for some K, then Wasserstein Distance between p_r and p_g can be written as:

$$W_p(p_r, p_g) = \max_{w \in W} \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_r(z)}[f_w(g_\theta(z))]. \tag{14}$$

We used the word ”discriminator” up until this point, it is important to note that ”discriminator” in wGAN is actually a ”critic” and not a discriminator. In Naive GANs, discriminator outputs values between 0 and 1 depending on how close given sample is to dataset. However, in wGANs, critic does not bound values between 0 and 1. Commenting on Equation (14), critic in wGAN does not directly tell us if the given sample is fake or real, but instead learns the Equation (14) and outputs the

Wasserstein Distance between the probability distributions. Aim of critic is separating outputted values obtained from real samples and fake samples.

To enforce Lipschitz Constraint, gradients are clipped between values, usually in the range $[-\mathbf{0.01}, \mathbf{0.01}]$. Although this enforces Lipschitz Continuity, it is not a great way of enforcing it. Clipping weights limits the capability of model since it basically removes gradient information. Gradient Penalty (WGAN-GP) [17] aims to ensure Lipschitz Continuity without sacrificing model's capabilities, which is discussed in Section 2.2.2.1.

Wasserstein Distance is continuous and differentiable almost everywhere, which means that, unlike Naive GANs where there is delicate balance between training the generator and discriminator, discriminator in wGAN can be trained until optimality. This is usually provided by training discriminator k times for each generator update, in most cases $k = 5$ is used. If the discriminator is optimally trained, then the gradients for updating generator network is more reliable. In the case of JSD, gradients become 0 if the discriminator is trained until optimality, which causes vanishing gradient problem and results in worse training conditions.

Using Wasserstein Distance helps alleviating the mode collapse problem, where generated outputs are same no matter what input is given. It is beneficial to discuss why mode collapse happens before discussing how wGANs solve this problem. As explained by [18], we can explain why mode collapse happens in following fashion: first, re-write minimax equation for Naive GAN with respective generator and discriminator parameters θ_G and θ_D:

$$f(\theta_G, \theta_D) = \mathbb{E}_{x \sim p_{data}}[log(D(x; \theta_D))] + \mathbb{E}_{z \sim \mathcal{N}(0,1)}[log(1 - D(G(z; \theta_G); \theta_D))], \qquad (15)$$

where x is data, z is latent variable, and p_{data} is the data distribution. Solving for optimal discriminator parameters $\theta^*_D(\theta_G)$ for each update step of generator G is not computationally feasible. So, the parameters θ_G and θ_D are updated in an alternated fashion. The optimal solution $\theta^* = \{\theta_G^*, \theta_D^*\}$ is a fixed point. If $f(\theta_G, \theta_D)$ is convex in θ_G and concave in θ_D, then trust region updates are guaranteed to converge to a fixed point. However, this is not the case in practice, and updates are not constrained appropriately. As a result, generator collapses many values into same region where it can achieve a low loss value.

Mode collapse occurs when generator finds a small set of outputs that can trick discriminator well. If the discriminator is stuck in local minima, then it cannot penalize these outputs and generator continues to output same values. In case of wGAN, since we can train discriminator to optimality, it is harder for it to stuck in local minima, which helps improving mode collapse problem.

2.2.2.1 Gradient Penalty (WGAN-GP)

Lipschitz Continuity must be ensured in order to make use of Wasserstein Distance in GANs. A way of ensuring this is simply clipping gradients between certain values, usually in the range $[-\mathbf{0.01}, \mathbf{0.01}]$. However, this limits capabilities of the model and makes training harder. As proposed by [17], Lipschitz Continuity can be ensured by constraining gradient norm of critic's output with respect to output. Following this statement, new loss function can be written as Equation (16)

$$L = \mathbb{E}_{\tilde{x} \sim \mathbb{P}_g}[D(\tilde{x})] - \mathbb{E}_{x \sim \mathbb{P}_r}[D(x)] + \lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}}[(||\Delta_{\hat{x}} D(\hat{x})||_2 - 1)^2], \tag{16}$$

where λ is constant taken as 10 by [17], $\mathbb{P}_r$ is data distribution, $\mathbb{P}_g$ is generated sample distribution. Note that first two terms in (16) is the original critic loss, whereas third term is added gradient penalty. $\hat{x}$ is sampled from $\tilde{x}$ as follows:

$$\hat{x} = t\tilde{x} + (1-t)x \; with \; 0 \leq t \leq 1 \tag{17}$$

When gradient penalty is used, batch normalization should be avoided. Batch Normalization creates correlations between samples composing the batch, which harms the effectiveness of gradient penalty. Gradient penalty penalizes norm of critic's gradient for each output individually, when batch normalization is used, problem is changed to outputting a batch of samples from an input batch of samples, which is not compatible with working systematic of gradient penalty. To solve this problem, batch normalization layers should be omitted from the network architecture, and layer normalization [19] should be used instead.

2.2.3 Conditional GANs

Similar to Conditional VAEs, we can add a conditioning variable to Naive GANs which allows us to control properties of generated outputs. Naive GANs have no way of controlling modes of generated output. They have no conditioning input, and generated samples are not conditioned on anything practically controllable. In the case of MNIST example, a generated image may be a "0", "1" or any other digit, which we cannot control. To solve this problem, an additional input is given to both discriminator and generator, which can be used to control the output of the network.

Compared to Naive GANs, generator and discriminator of a conditional GAN (cGAN) takes an additional input, a conditioning variable, which is used to train generator and discriminator in such a way that generated samples are correlated with given conditioning variable. Both generator and discriminator are now conditioned on conditioning variable, minimax equation for cGAN is shown in Equation (18):

$$\min_G \max_D V(D,G) = E_{x \sim p_{data}(x)}[logD(x|y)] + E_{z \sim p_z(z)}[log(1 - D(G(z|y)))], \quad (18)$$

where y is conditioning variable. Note that as opposed to to Naive GANs, discriminator tries to minimize $logD(x|y)$, and generator tries to minimize $log(1-D(G(z|y)))$, which are both including conditional probabilities conditioned on y.

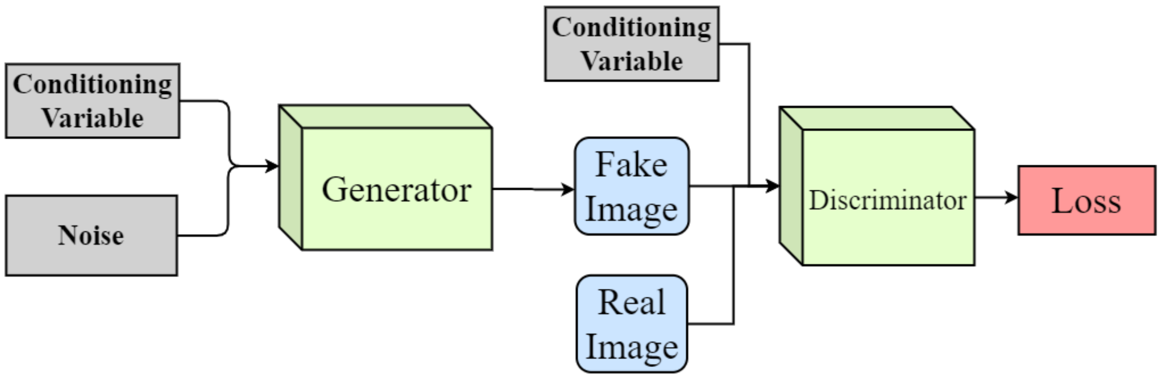


Figure 6: Architecture of conditional GAN.

2.2.3.1 Matching-aware Discriminator (GAN-CLS)

One way of improving cGANs to generate samples that are better aligned with conditioning variable is using Matching-aware discriminator(GAN-CLS) [20]. In 2.2.1, we discussed that discriminator takes samples from two different sources: samples from dataset, and samples generated by generator. In the case of cGANs, conditional variables corresponding to these inputs are also being given to discriminator. However, in this format, discriminator has no way of knowing if the conditioning label matches the given input or not. Discriminator in cGAN should get good at distinguishing fake samples from real samples, and it should also penalize if a generated sample is not conforming the given conditioning variable. In the case of cGANs without Matching-aware discriminator, generated samples that are not aligned with given conditioning variables are not penalized by the discriminator.

To solve this problem, a third type of input is given to discriminator: a sample from dataset with incorrect conditioning variable. In the beginning of training, it is easy for discriminator to distinguish fake samples from real samples since generator is not generating samples close to real distribution. As training progresses, generator gets better at outputting realistic samples. To be able to decrease the loss, the generator should also learn to generate samples that are aligned with given conditioning variable.

Another contribution of [20] is interpolating between two conditioning variables.

In their work, they used word embeddings (which is discussed in more detail in Section 2.3) as conditioning variables, however, it it also viable to apply this method for other types of conditioning variables if it is feasible to interpolate between them. In cGAN, generator learns to generate samples that correspond to given conditioning variables. Word embeddings that are obtained from given sentences are sparse, there are gaps in the data manifold. Interpolating between embeddings allows network to fill these gaps, allowing it to generate a more diverse set of samples. Here, generator should also optimize the following additional term:

$$\mathbb{E}_{t_1,t_{2\sim p_{data}}}[log(1 - D(G(z, \beta t_1 + (1 - \beta)t_2)))], \tag{19}$$

where z is noise, and β is interpolating parameter between conditioning variables t_1 (correct conditioning variable) and t_2 (incorrect conditioning variable).

2.2.3.2 Conditioning Augmentation

In Chapter 2.1, we have discussed that encoder part of autoencoders generate single encoding for a particular image, while encoder in Variational Autoencoders generate a probability distribution function over encodings. Similar to VAEs, in Conditioning Augmentation (CA) [21], latent variables are randomly sampled from an independent Gaussian Distribution from mean and variance of conditioning variable (in the case of [21], conditioning variables are text embeddings). Intuition behind CA is, latent space for text embeddings are high dimensional for representing the meaning of words, however, when data is limited, it is not possible to fill all the dimensions of embeddings with meaningful variables, which causes discontinuity in the latent space. This is not a desirable situation for training the generator. CA solves this by sampling latent variables from Gaussian Distribution, which encourages smoothness in latent space. Additionally, it allows small disturbances in latent space by adding a degree of randomness since latent variables are sampled from a Gaussian Distribution, which in return allows generated images to be more diverse.

Additionally, following regularization term is used in CA to avoid overfitting and for better smoothness:

$$D_{KL}(\mathcal{N}(\mu(\varphi_t), \Sigma(\varphi_t)) \,||\, \mathcal{N}(0, I)), \tag{20}$$

where $\mathcal{N}(\mu(\varphi_t), \Sigma(\varphi_t))$ is an independent Gaussian Distribution, $\mu(\varphi_t)$ is mean and $\Sigma(\varphi_t))$ is covariance matrix of the text embedding φ_t. It can be seen that the regularization term is KL Divergence between Gaussian Distribution and Conditioning Gaussian Distribution.

2.2.4 Evaluation Metrics of GANs

There are different approaches for evaluating the performance of GANs. In this section, two quantitative evaluation metrics will be discussed: Inception Score (IS) [22] and Fréchet Inception Distance (FID) [23]. Although only IS is being used in this work, it is still important to discuss FID as it is a popular evaluation metric used when evaluating GANs, and because it is relevant to IS.

2.2.4.1 Inception Score

IS is used for evaluating the variety and quality of samples generated by a generator. Calculation of IS relies on an external, pre-trained Inception v3 model [24] which is used for calculating class probabilities for generated samples.

IS aims to evaluate two key points: quality of samples, and how diverse the generates samples are. Inception v3 model can classify between 1000 classes of ILSVRC 2012 dataset [25], which means that the highest possible IS score obtainable is 1000. Lowest possible IS of 0 means that generated samples lack both diversity and quality while an IS of 1000 means that the generated images are diverse, high quality images. An important weakness of IS is, since real sample distribution is not utilized during the calculation, it cannot measure how close generated samples are to real samples. IS calculation can be seen in Equation (21):

$$\mathrm{IS(G)} = \exp\left(\mathop{\mathrm{E}}_{x\sim p_g}[D_{\mathrm{KL}}(p(y|x)\ ||\ p(y))]\right)\ , \tag{21}$$

where x is samples generated by generator from distribution p_g, D_{KL} is KL divergence between conditional class distribution $p(y|x)$ and class distribution $p(y)$, and y is label for sample x. We can see that KL is directly proportional to IS. That is, a set of generated samples cover different classes and if the generated individual samples belong in one distinct class, then KL and IS values are high. Meaning that, generator can generate images belonging to different classes, and each generated sample particularly fits into one distinctive class.

As discussed, IS encourages generated samples to be meaningful and diverse, however, it is not an ideal metric for comparing distribution of generated samples with distribution of real samples since it is not using information from real samples, which FID attempts to solve.

2.2.4.2 Fréchet Inception Distance

Another metric that is used to evaluate performance of GANs is Fréchet Inception Distance. IS has no notion of real samples when measuring quality and diversity of generated images. It is more desirable to use a metric that utilizes real samples, so that we can compare generated samples with samples from real data. FID is used for measuring distance between two probability distributions. In our case, it is used for calculating the distance between probability distribution of real samples and probability distribution of generated samples, which is a better approach if results are desired to be similar with real samples.

Similar to IS, FID is dependent on an external, pre-trained Inception v3 model. Here, output layer of Inception v3 is removed and pooling layer before the output layer is used for obtaining logits that is used to calculate FID. These logits are summarised as multivariate normal distribution by calculating mean and variance. Same process is repeated for obtaining mean and variance of both real sample distribution and fake sample distribution, which then be used in calculation of FID to find distance between the two distributions. Equation of FID can be seen in (22):

$$\text{FID} = ||\mu_r - \mu_g||^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}), \tag{22}$$

where μ_r and μ_g are feature-wise mean for real samples and generated samples, Σ_r and Σ_g are covariance matrices for real samples and generated samples respectively.

2.3 Natural Language Processing

We can express our ideas, emotions, knowledge, or anything that can be expressed in verbal or written form through sentences using natural language. Another person can understand what we are trying to communicate or transmit with the use of words and sentences. However, neural networks has no way of understanding these natural language description in the same way that we do. For neural networks, it should be in a format that it is well structured and well defined.

For this purpose, we use "embedders", which is a form of encoder. These encoders can be used to encode words and sentences so that the information that wanted to be transmitted can be encoded in well structured array form, which can be used to represent words mathematically, so we can feed into neural networks in a form that these networks can understand what information we are trying to convey.

Natural Language Processing (NLP) is a huge topic itself, and it is not suitable to discuss every aspect of NLP in this thesis. Therefore, in this chapter, we will discuss only the part that is related to our work, which is Word Embeddings, where each individual word is represented in the encoded array form, Sentence Embeddings, where whole sentences are encoded instead of individual words, the FastText model

[26] which is used to obtain said encodings, and finally, we discuss Skip-Gram Model that FastText uses. In this section we omit discussion about other word embeddings since they are not used in our work, such as Word2Vec [27], GloVe [28], ELMo [29] etc.

2.3.1 Skip-Gram Model

Skip-Gram model works in the following fashion: For each word in corpus, it also takes words surrounding the current word in a range defined by a parameter. Then, feeds the words to neural network during training, so that the network can predict probability of a word appearing in the window of word that is selected. However, it is not possible to feed words as subsequent characters to neural network, a numerical or mathematical representation is needed. This can simply be done by creating a one-hot vector for each word in the vocabulary, where dimension of one-hot vector is same as vocabulary size. One-hot vector can be fed into the neural network, which in return outputs probability of a word appearing next to given words. However, similar to autoencoders discussed in Section 2.1, we are not interested in output, but instead, we are interested in values in middle layers, which we call embeddings. Since we are looking from a window perspective, each word will ideally have close probability with words that most frequently appear in its window. As a result, since output will be similar for words that are close to each other, values in hidden layers will be close to each other as well. So, two different words with close meaning will have similar embeddings.

2.3.2 FastText

FastText uses a similar method to Skip-Gram Model introduced by [27]. Instead of computing embeddings of words directly, it computes embeddings for each character n-gram. For example, if $n = 2$, for the word ”when”, FastText will represent embedding of this word as a sum of n-grams of ”wh”, ”he” and ”en”. This method allows computing representations for words that are not available in the corpus, which is an important feature when working with human generated text descriptions. This approach is known for generating better embeddings for words that are not used frequently, it is also a good way of computing embeddings with words that has typos. In the case of human written sentences, probability of including a typo is always present. FastText minimizes loss of information due to a typo.

Word embeddings are useful for obtaining information about them in mathematical form which can be used in neural networks. However, two problems arise when feeding a sentence to a neural network: first, amount of words in sentences are usually different, which changes the shape of input each time a sentence of different length is present. Second, dimension of input gets bigger as sentence gets longer, which slows down training since neural network needs to process an input with bigger dimensions for longer sentences. First problem is easily solvable by padding each sentence to a fixed length. This is a viable method even though this may result in some sentences to get shorter, which causes a loss of information, and may cause short sentences to get unnecessarily long with padding.

For the second problem, it is a good idea to use sentence embeddings instead of using embeddings for each individual word that forms the sentence as a way of fixing input dimension size. Sentence embeddings can be obtained by taking element-wise mean of embeddings of word that form the sentence. However, this causes loss of information about each word's location in the sentence. This method is more suitable for applications where location of words are not important.

2.4 StyleGAN

In this section, we will discuss about StyleGAN introduced by [9], a Wasserstein GAN architecture that is capable of separating high dimensional features by using convolutional blocks, and is able to generate high quality images.

Before going into details of StyleGAN, it is beneficial to discuss Progressively Growing GAN (ProGAN), which StyleGAN is based upon. In ProGAN, generator architecture is structured in the form of convolutional layers or blocks, where each block is responsible of generating a certain resolution image and takes input as output of previous block. Training starts from smaller resolutions, and resolution of output is iteratively increased by activating blocks until resolution is as large as desired size. Similarly, discriminator layers are activated as generator starts to output bigger images to match the resolution. By starting from smaller resolutions, we are simply asking the network to solve a much smaller, easier problem, and iteratively increasing the difficulty of the task. This allows a more stable training, while also achieving much faster training times since number of iterations required for higher resolutions are drastically decreased.

StyleGAN makes significant changes in traditional generator architecture, while keeping discriminator architecture same. In contrast to regular generator architectures, StyleGAN includes two different networks in generator structure: mapping network and synthesis network. Before discussing these two sub-networks in more detail, it is beneficial to discuss Adaptive Instance Normalization (AdaIN) [30], as it is an aspect that should be understood before trying to understand what mapping network and synthesis network does. We also discuss Batch Normalization and Instance Normalization here, as they are the steps that lead to AdaIN. StyleGAN network architecture can be seen in Figure 7.

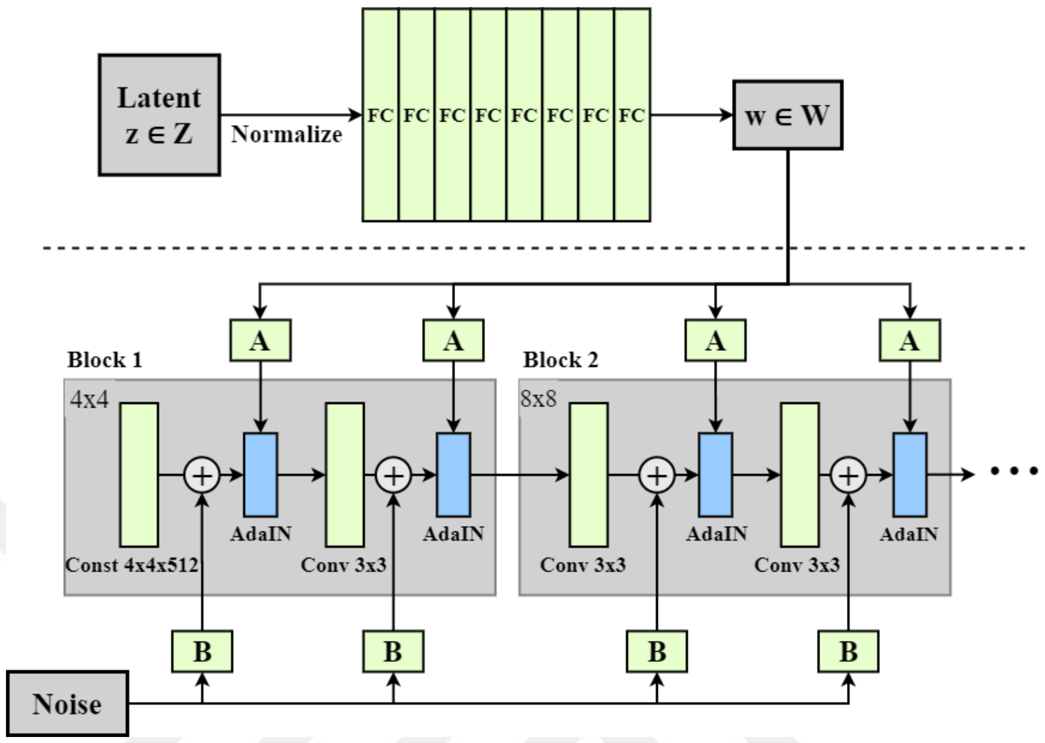


Figure 7: Network architecture of StyleGAN proposed by [9]. Above the dashed line is mapping network while below is synthesis network.

2.4.1 Batch Normalization

Normalizing inputs is a widely used method in the neural network training. Although different inputs may have the same importance, their range of values may differ greatly as if their importances are different. Normalizing the input to have 0 mean would achieve faster convergence, and in some cases, it would allow network to not get stuck in a local minima.

During the training of most neural networks, each layer depends values obtained from previous layers. A common problem faced during training is how quickly these values can change and affect subsequent layers. Since layers get their inputs from previous layers (except the input layer), changes made in gradients of previous layers also effect next layers, combined with this, updating gradients of next layers may cause a domino effect that may cause network to be unstable even with a carefully selected learning rate.

Similarly, in Batch Normalization [31], values in hidden layers can be normalized (or standardized) with mean and standard deviation of inputs in mini-batch. This technique aims to achieve better synchronisation across different layers in the neural network in order to prevent discussed domino effect. As [31] showed, Batch Normalization algorithm is as follows:

```
Algorithm 1: Batch Normalizing Transform algorithm
    Input: Values of x over a mini-batch B = { $x_{1...m}$ }, Parameters to be learned: $\gamma, \beta$
    Output: {$\mathrm{y}_i = BN_{\gamma,\beta}(x_i)$}
$\mu_\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ // Mini-batch mean
$\sigma_\beta^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2$ // Mini-batch variance
$\hat{x}_i \leftarrow \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}}$ // Normalize
$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta(x_i)}$ // Scale and shift
```

As can be seen from the Algorithm 1, γ and β are learned during training and calculated over all samples in mini-batch. Here, $BN_{\gamma,\beta}(x)$ is dependent on both current training sample, and other samples in the current mini-batch.

For a better understanding, we can write Batch Normalization in the form:

$$y_{tijk} = \frac{x_{tijk} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}, \quad \mu_i = \frac{1}{HWT}\sum_{t=1}^{T}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm},$$
$$\sigma_i^2 = \frac{1}{HWT}\sum_{t=1}^{T}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - mu_i)^2, \tag{23}$$

where T form a batch, x_{tijk} represent T's $tijk$-th element, while k and j span spatial dimensions, i is the feature channel and t is the index of sample in batch.

2.4.2 Instance Normalization

Instance Normalization (IN) [32] normalizes each channel independent of mini-batch in contrast to Batch Normalization. This is especially important in topics such as style transfer, where each image may have different properties that does not effect style, such as contrast and brightness. In the case of Batch Normalization, these properties would effect mini-batch mean and mini-batch variance. IN prevents instance-specific mean and covariance shift which simplifies learning process. We can spot differences compared to (23) when we look at equation of IN in (24):

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}}, \quad \mu_{ti} = \frac{1}{HW} \sum_{l=1}^{W} \sum_{m=1}^{H} x_{tilm},$$
$$\sigma_{ti}^2 = \frac{1}{HW} \sum_{l=1}^{W} \sum_{m=1}^{H} x_{tilm}(x_{tilm} - mu_{ti})^2. \tag{24}$$

We see that summation $\sum_t = 1^T$ is omitted and averages taken over pixels over individual samples for each separate sample in the mini-batch.

2.4.3 Adaptive Instance Normalization (AdaIN)

[30] made the following observation that IN also performs as style normalization technique by normalizing mean and variance. Interpreting this observation, we can say that AdaIN is developed based on the idea that IN can be used to adjust style of generated images by normalizing certain features. It is a method that is especially useful in style transfer because of its speed, and cheapness considering memory and computational requirements.

AdaIN is an extention to existing IN method. It is defined as follows:

$$\text{AdaIN}(x, y) = \sigma(y) \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y), \tag{25}$$

where normalized content input is scaled with $\sigma(y)$ and biased by amount of $\mu(y)$ where y is style. Here, each feature map x is normalized separately. We can describe intuition behind AdaIN with a simple example similar to example in [30]: assume that we an image **A** which has a lot of vertical lines. In the style transfer literature, it is possible to transfer this property to another image **B**, obtaining another image which has lots of vertical lines. If we have a feature channel that detects vertical lines, then this channel will produce high activation for vertical features. AdaIN will produce an output which have same high activation values for this feature, which then can be decoded (with the help of a decoder) into image space [33]. Variance of this feature space is transferred to AdaIN output and then to the final output image generated by the network.

2.4.4 Mapping Network

In style transfer literature, style y is computed from input directly. However, StyleGAN follows a different approach where it is computed from vector W, which is learned by mapping network for given input. Mapping network maps input to an intermediate latent space a vector W, which is then given to AdaIN layers in synthesis network in the form of style vectors.

Mapping network also helps in "disentangling" the latent space. For example, different properties such as hair length and gender may be correlated by network if the dataset has a bias between these two properties. If all males in the dataset are short haired, network may not be able to separate hair length from gender, which is entanglement. Mapping network helps disentangling such latent variables with learned fully connected layers.

2.4.5 Synthesis Network

In contrast to traditional generator architectures, synthesis network starts with a learned constant. Synthesis network is composed of blocks which start with an upsampling layer, followed by convolutional layer which then followed by AdaIN layer with two repetitions for convolutional and AdaIN layers. Each block is responsible for generating an output with certain resolution. With the exception of first block where it starts from a learned constant, each block takes input of previous block, and after being upsampled to next resolution, it is given into convolution layers and AdaIN layers for adding more high resolution and finer details. Gaussian noise is added after each convolution. Intuition behind this architecture is making each block responsible from scale specific styles. Synthesis network can be taught as an architecture where each block adds certain component of style which composes the overall style of generated image.

Synthesis network has important implications. Most of the time, neural networks act as a black box, where it is very hard to interpret, or not possible to interpret at all. Synthesis network can be seen as a step towards successfully interpreting these black boxes, understanding how low level features work in latent space.

2.5 Position-map Regression Network

In this chapter, we discuss about UV Mapping, then we describe what Position-map Regression Network (PRN), how it uses UV Mapping, and how PRN is used in this thesis to generate 3D human face structures from 2D images that are generated by our cStyleGAN.

2.5.1 UV Mapping

UV Mapping is a method that is used to project points in 2D space into 3D model space. It is a mapping that tells us where each point in 2D image corresponds to in 3D model's surface. Here, ”U” and ”V” represent coordinates in 2D space of the image (UV Texture Map). Aim of UV Mapping is creating a ”mapping” from ”U” and ”V” coordinates to ”X”, ”Y”, and ”Z” coordinates in 3D space. If we think about a paper as UV Texture Map and a sphere as our 3D object that we want to paint with this texture, UV Mapping can be taught about as wrapping this paper onto the sphere.

2.5.2 What is Position-map Regression Network

To summarize PRN in one sentence, we could say that PRN is a neural network architecture that is used for generating textured 3D face models from 2D face images. Here, we can split this process into two main parts: face alignment, and regressing 3D facial geometry.

One method of generating 3D facial geometry is concatenating all points in 3D space to a 1D vector, and using a neural network to predict it. However, each point in 3D space has a correspondence with neighbouring points. Representing these points in 1D space causes loss of spatial information. Predicting each point in 1D form requires a fully connected layer which increases training difficulty. Other model based methods [34][35][36] train models that predict parameters instead of predicting coordinates directly. However, this method is discussed as being hard as training models that generate good results require special care.

As a solution to these problems, PRN proposes UV Position Map to represent 3D facial structure. UV Position Map is 2D representation of points in 3D structure. We can see that it is a similar idea discussed in Section 2.5.1. However, PRN uses UV Mapping as a way of storing 3D coordinates instead of creating a mapping from 2D image to already existing 3D structure. Here, UV Coordinates are based on 3D structure for keeping semantic meaning. Note that UV Mapping in this method also contains dense alignment information. For training such a system, end-to-end 2D images with corresponding 3D structures are required. Authors state that PRN is trained on 300W-LP [34], which contains more than 60K images with fitted 3D Morphable Model (3DMM) [37].

PRN transfers input RGB image into position map image. This is done by an encoder-decoder architecture where both encoder and decoder parts utilize convolutional layers. A novel loss function, which measures the difference between ground truth position map and the network output is proposed by authors for learning parameters of this network. A gray-scale weight mask is used where each point in weight mask represents the weight of corresponding point in position map. This mask and position map has the same width and height to provide pixel to pixel correspondence from mask to position map. Since facial points such as eyes, noses and mouth have the highest importance, mask is adjusted so that these areas have the highest weights in loss function for enforcing model to learn accurate points for these areas. Since neck area is often occluded by hairs or clothes, this region has the lowest importance, and thus lowest weight.

The encoder part of the network is composed of a convolution layer, which is followed by 10 residual blocks [38] that reduce input image into feature maps. The decoder part contains 17 transposed convolution layers that is used for generating position map. Architecture of PRN is shown in Figure 8:

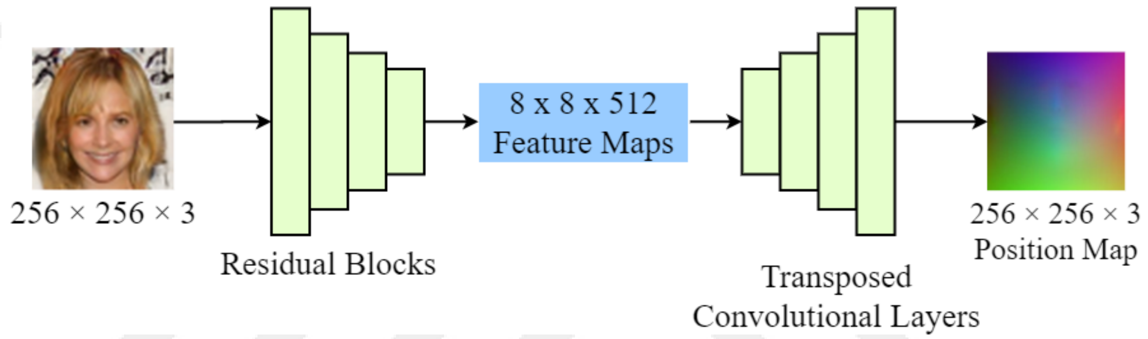


Figure 8: Network architecture of Position-map Regression Network.

2.6 *Learned Perceptual Image Patch Similarity (LPIPS)*

Comparing different data has various difficulties in different domains. For example, Euclidean Distance is useful for comparing two vectors. However, if we want to compare two image, measures such as Euclidean Distance is not very useful. Two images that are looking very similar may have large Euclidean Distance values. A small shift in pixels' locations would not change the perceived image's similarity for human perception, but it would mean a big increase in Euclidean Distance. Similarly, blurring an image would not change Euclidean Distance between two images as much, but it would change the visual properties of image greatly. Euclidean Distance is just an example to show why it is difficult to measure similarity between two images, encountering similar problems is also possible in methods such as Structural Similarity Index (SSIM) [39].

Perceptive similarity of images in computer vision is still an important problem. However, there are important improvements in this field with the help of deep neural networks. Zhang et al. [40] used deep features that are obtained from intermediate layers of neural networks for comparing image similarity, which they call Learned Perceptual Image Patch Similarity (LPIPS). They used SqueezeNet [41], AlexNet [42], and VGG [43] for testing their method and reported notable results for all three architectures. Equation for calculating similarity between two images is as follows:

$$d(x, x_0) = \sum_{l} \frac{1}{H_l W_l} \sum_{h,w} ||w_l \odot (\hat{y}^l_{hw} - \hat{y}^l_{0hw})||^2_2, \tag{26}$$

where d is similarity, x and x_0 are images, and $\hat{y}^l$, $\hat{y}^l_0 \in \mathrm{R}^{H_l \,\mathrm{x}\, W_l \,\mathrm{x}\, C_l}$, where l is current layer. Features $\hat{y}^l$ and $\hat{y}^l_0$ are extracted from layer l, and unit normalized in channel dimension. Then, activations are scaled by the vector w^l, and ℓ_2 distance is computed.

Although Euclidean Distance is still used, using information obtained from middle layers of a network trained for measuring image similarity can help measuring distance between semantic similarity of two images. The paper [40] also discusses that results are aligned with human judgments about similarity between two images.

CHAPTER III

PREVIOUS WORK

Work in this field is focused on two major areas; generating high quality images, and generating images that are aligned with given text descriptions. Zhang et al. [21] proposed StackGAN which is composed of two different GANs that first generate low resolution images from given descriptions, and then increase resolution of generated images by adding finer details. They also introduced Conditioning Augmentation method, which helps generating more diverse images. In Conditioning Augmentation technique, instead of using word embeddings directly, they randomly sample latent variables from an independent Gaussian distribution where mean and covariance are functions of text embedding generated from given descriptions.

Akanimax [44] made use of Conditioning Augmentation in his work and used ProGAN and StackGAN for synthesizing facial images, which we will refer as Pro-StackGAN throughout this thesis. Writers also employed GAN-CLS loss. In regular conditional text-to-image GANs, discriminator loss is calculated by using two different inputs; image with corresponding description, and generated image with given description. In GAN-CLS loss, a third input, which is an image with mismatched description is also taken into consideration when calculating discriminator loss. They used an older version of Face2Text dataset, which includes only 400 images with annotations. Although generated images were somewhat aligned with given descriptions, image quality was poor and sometimes almost unrecognizable. Still, results are notable considering how small of a dataset have been used.

In Text2FaceGAN, Nasir et al. [45] utilized DC-GAN [46] with GAN-CLS loss for generating human face images from text descriptions. Similar to our work, they utilized CelebA dataset, and generated captions from CelebA annotations. However, they have used annotations for all 203K images in the dataset whereas in our work low quality annotations are eliminated for increasing quality of generated descriptions. Also, their text generation algorithm lacks randomness that is important for neural networks to avoid memorizing.

Text to image domain has increased in popularity with improvements made in GANs. Although not in text to human face domain, there are various works which generate images from textual descriptions. In [20], Reed et al. also utilized DC-GAN for generating bird and flower images. Their networks are trained on CUB-200 dataset [47], which includes 11788 images, and Oxford-102 flowers dataset [48], which includes 8198 flower images. They have conditioned both the generator and the discriminator in their network by using text embeddings which they obtain from the text encoder they trained. They also proposed GAN-CLS loss, and utilized it in their discriminator, which helps the generator to generate images that are better aligned with descriptions, since the discriminator can distinguish between images with correct descriptions and images with incorrect descriptions.

In the work [21], Zhang et al. also synthesized bird and flower images from given textual descriptions with their networks, which they trained on CUB-200 dataset and Oxford-102 flowers dataset similar to [20]. In their proposed StackGAN architecture, low resolution images are generated before generating high resolution images with finer details. They accomplished this by implementing two different generator and discriminator in their network structure, which are separated as Stage 1 and Stage 2. In this first stage, low resolution images are generated by the generator. Discriminator of the first stage learns to discriminate images generated only by the first stage generator. Stage 2 improves upon results generated by Stage 1 and adds finer details while also increasing the resolution of generated images. Instead of directly generating high resolution images that are conditioned on descriptions, first, a low resolution image is generated, then higher resolution image is generated, and more details are added. This method helps stabilizing training as it is easier to generate low resolution images. It also helps increasing the training speed as it is faster to generate low resolution images.

Work in [49] improved upon existing StackGAN and introduced tree like structure of multiple discriminators and generators. In their proposed architecture StackGAN-v2, there are multiple discriminators and generators which operate on different resolution scales of images. This architecture works in similar manner with StackGAN where each discriminator-generator pair for a given resolution is similar to a stage in StackGAN. According to their experiments, StackGAN-v2 does not face mode collapse and generate images with higher quality compared to StackGAN.

Xu et al. [50] proposed AttnGAN architecture, which allows synthesizing finer details in different subregions. AttnGAN pays attention to different relevant words for different areas of generated images. Instead of correlating whole image with a sentence, different subregions which are related to corresponding words are associated by the network. They have utilized both COCO dataset [51], and CUB dataset in their work. Their network succesfully generated bird images from descriptions, and also generated various scene images which are learned from COCO dataset. They displayed strengths of their network by analysing different layers of AttnGAN and showing that the network is automatically selecting the conditioned word for generating different parts of the image. They also proposed deep attentional multimodal similarity model (DAMSM) for computing image-text matching score. Their DAMSM module is composed of two different parts: a text encoder, which is a bi-directional LSTM [52] network that is used for extracting word-specific information, and an image encoder, which is a convolutional neural network that maps given images into vectors. They state that, intermediate layers of their image encoder learn local features while later layers learn global features of given images. This image encoder is based on Inception-v3 model.

In a different work, Hinz et al. [53] utilized AttnGAN, and COCO dataset for training their network. Similarly, they have generated various scene images from given text descriptions. However, their work makes modifications upon AttnGAN and their proposed architecture is called OP-GAN. They state that OP-GAN focuses specifically on individual objects. Objects generated are placed in meaningful locations in the generated image. Background is generated simultaneously while generating the individual objects by OP-GAN. To accomplish this, they have utilized object pathway similar to work in [54]. They proposed a metric called Semantic Object Accuracy (SOA) for scoring image-text alignment by using a pre-trained object detector. SOA evaluates image-text alignment according to if described objects are in the given image or not.

There are also works on 3D object generation conditioned on external inputs. However, work in this field is usually based on voxels rather than generating 3D models based on vertices. Wu et al. [55] proposed their architecture 3D-GAN, which generates voxelated 3D objects from given latent space. In this work, generated objects are not based on text descriptions and directly generated from given latent space variables. As en extension, they proposed 3D-VAE-GAN, which generates 3D voxelated objects from given 2D images. Here, 2D images of objects are conditioning variables, which generated 3D objects are aligned with. They have succesfully generated 3D objects of chair, table, sofa, gun, and cars with up to 64 x 64 x 64 resolution. Objects in this work are lacking different colors and composed of same color.

Work in [56] generated colored 3D chair and table shapes composed of voxels with up to 128 x 128 x 128 resolution. They utilized conditional Wasserstein GAN for generating colored 3D voxelated structures. Their network is trained on chair and table models available in ShapeNet dataset [57], which contains with 8,447 table and 6,591 chair instances. Therefore, their model can only generate table and chair objects.

Improving upon [56], Fukamizu et al. [58] utilized a similar structure used in StackGAN, and generated 3D voxelated shapes in two steps: first, a 3D object with smaller resolution is generated. Then, in the stage 2, objects with higher resolution are generated based on smaller resolution objects. Similarly, their network generated objects based on text descriptions. Generated objects in the first stage are reflecting properties described in the description roughly, also, shape of generated object is not high quality. In the second stage, objects have higher quality and better aligned with given descriptions. This method helps stabilizing the network training as well as increasing training speed considerably. Especially in 3D objects, amount of data generated is too much due to the nature of 3D objects.

CHAPTER IV

METHODOLOGY

In this section we discuss our methodology and show our experimental results. We have tested our proposed network conditional StyleGAN (cStyleGAN) on both Face2Text dataset, and on Extended dataset. Extended dataset is the combination of Face2Text dataset, and images with descriptions generated from CelebA annotations by using our Description Generation Module. Additionally, we have tested two different conditional discriminator architecture. In the first version, results are not successfully conditioned on outputs. In the second version of discriminator architecture, obtained results are perfectly aligned with given descriptions. We discuss the reasoning behind why the first discriminator is failed to condition generated images, and why the second discriminator succeeded, in their respective subsections. We also show our 3D facial structure generation pipeline and structures generated by our system using images generated with cStyleGAN in this section.

We discuss our proposed Perceptual Quality Distance and show the results of it on models that are trained on Face2Text dataset and Extended dataset separately, we show that such method can be used for evaluating quality of images generated by generative models. We also show that it gives high values for irrelevant images such as samples taken from our toy shapes dataset, which means the similarity between two images are very low.

Before showing results with our model with real data, we test our model to see if it correctly conditions output on given text descriptions. To test this, we used a simple toy dataset of shapes with three different colors, which are generated with a simple Python script. This is also the methodology that we used during our research.

First, we made sure that network is correctly conditioning outputs on given conditioning inputs, then more extensive research for increasing the image quality have been conducted.

We discuss our Description Generation Module that we used for generating descriptions by using 40 annotations available for each image in CelebA dataset. We show that our model generates more realistic images while still keeping images aligned with given descriptions when additional data from CelebA dataset is used. As an additional test, to further show the effects of Extended dataset, we trained Pro-StackGAN on both Face2Text dataset, and on our combined dataset.

We have performed ablation studies for understanding effects and importance of different parts in our system. In previous sections, we have discussed that each block in Synthesis Network of generator learns different parts of desired output. For example lowest resolution (4x4 pixels) block learns to generate a non-detailed, overall representation of a human face image while second block which outputs an 8x8 image may learn finer details and so on. Although this is correct in theory, we need to do experimentation for seeing effects of different blocks, which we tested in Section 4.6.5.

Our end-to-end system architecture that generates 3D facial structures from text descriptions can be seen in the following Figure:

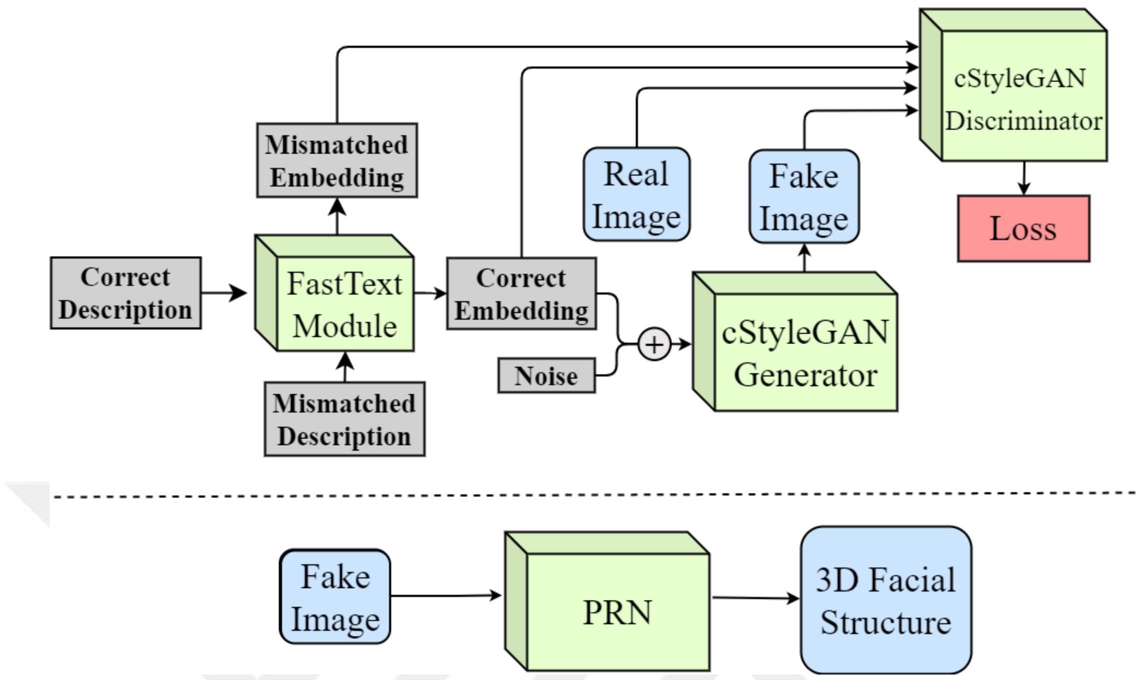


Figure 9: Proposed end-to-end system architecture.

4.1 Perceptual Quality Distance

In Section 2.2.4 we have discussed the widely used Inception Score (IS), and Fréchet Inception Distance (FID) for evaluating GANs. IS gives high scores if images are high quality and diverse. However, there is no such criteria about how close the images are to real dataset. A network that generates images which are both high quality and diverse would achieve high IS, even if the images are nothing like the real dataset which generated images are based on. Improving on this, FID utilizes information of real dataset for measuring the distance. Both of these methods are dependent on pre-trained Inception v3 model. IS uses KL Divergence, while FID calculates mean and variance for measuring the distance. However, these methods do not use perceptive properties of images directly. FID calculates mean and covariance of images in order to summarize activations that are used to compare real images and generated images. IS on the other hand, disregards real image information completely and calculated score is based only on generated images' label distributions computed by Inception v3 model. Although these methods are proved to be useful, an alternative method that directly uses perceptive properties of generated images and real images would be valuable for evaluating performances of GANs.

Our proposed method, Perceptual Quality Distance (PQD), is based on LPIPS discussed in Section 2.6, which measures the similarity between two images. In our case, we have used LPIPS method for measuring the perceptive similarity between generated images and targeted dataset for calculating quality of images which our model generated. In theory, if generated images are high quality and close to the training dataset, then the PQD value would be low, meaning that generated images are similar to the ones in the training dataset, which also means that generated images are high quality. In our tests, we have seen that PQD is inversely proportional to quality of human face images generated when compared with CelebA dataset. We have observed that PQD value is high when images generated by our network is compared with an irrelevant dataset, such as our toy shapes dataset. This is expected since human face images are not similar with colored geometrical shapes.

If we assume that x_1 is real image from dataset A and x_2 is image generated by network trained on A, our acceptance criteria is as follows:

C1: $PQD(x_1, x_2)$ is small if x_2 is perceptually close to x_1 and/or has high quality.

C2: $PQD(x_1, x_2)$ is large if x_2 is not similar to x_1 and/or has low quality.

Our method is based on AlexNet model publicly available in Torchvision [59] library for applying LPIPS method to images. We first generate a number of images from network that is being evaluated, then compare each generated image to images from CelebA dataset to measure quality and perceptual similarity of generated images with CelebA dataset. To increase robustness of our method, we have eliminated outliers when calculating distances by applying two-pass elimination system: on the first pass, we have eliminated outlier scores when comparing an image with images from CelebA dataset, then average score for current image is calculated. On the second pass, we have eliminated outlier average scores among all images for further increasing robustness against outliers. For eliminating outliers, we have calculated standard deviation and mean of PQD score of current image with CelebA dataset, and eliminated scores that are more than one standard deviation away from mean. Same procedure is also applied to scores in the second pass. Calculation of PQD is shown in Equation (27).

$$PQD(x_g, x_r) = \sum_{m}^{M} t_m \sum_{n}^{N} t_n F(x_{gn}, x_{rm}) \frac{1}{N} \frac{1}{M}, \tag{27}$$

where x_g is generated image, x_r is real image, M is number of real images used, and N is number of generated images used. F is LPIPS function. t_n is 0 if PQD score is one std. away from mean of PQD scores of current image with all real images compared, otherwise 1, t_m is 0 if average PQD score of current image is one std. away from mean of average PQD scores of other generated images, otherwise 1.

To test our proposed method, we have computed PQD between samples generated from our networks and 100 images chosen from CelebA dataset and reported results. We have categorised images according to their quality based on human perception. In the first category, irrelevant images from toy shapes dataset is used. Then, images are categorised by increasing quality. PQD is supposed to decrease as images get higher quality, and in our case, as they get closer to CelebA dataset. PQD between images and CelebA dataset can be seen in Figure 10.

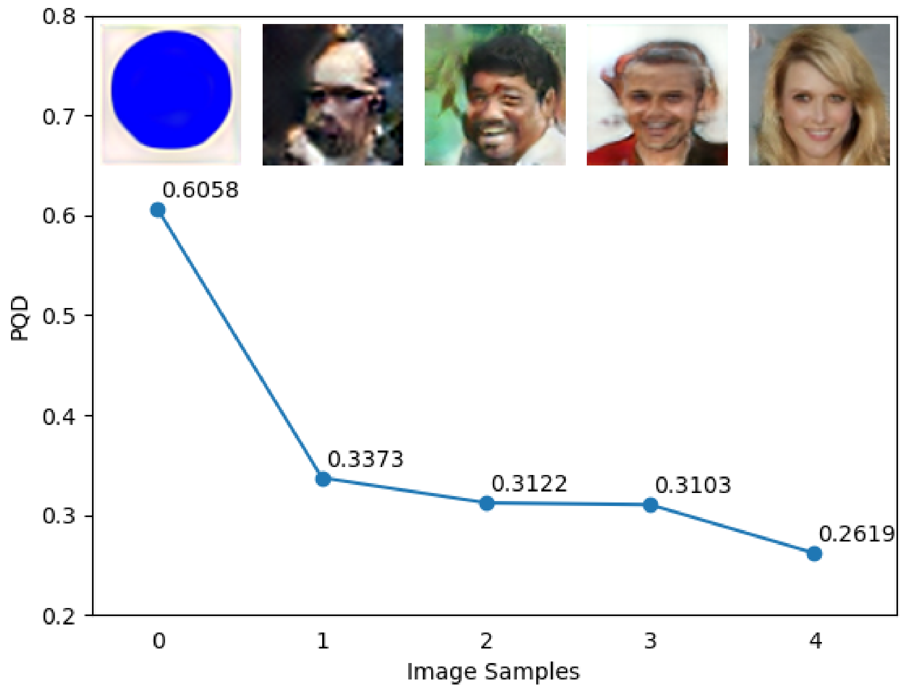


Figure 10: PQD between similar quality samples and CelebA dataset. One example from each category is shown.

4.2 Conditional StyleGAN

We have discussed conditional GANs in Section 2.2.3 in more detail. Here, we discuss how we changed StyleGAN architecture in order to condition outputs on given text embeddings. Our cStyleGAN is able to generate images that are aligned with given natural language descriptions with the help of sentence embeddings.

cStyleGAN, similar to StyleGAN, does not make big differences in discriminator architecture. However, it uses a non-traditional generator architecture. Generator in cStyleGAN can be seen in following figure:

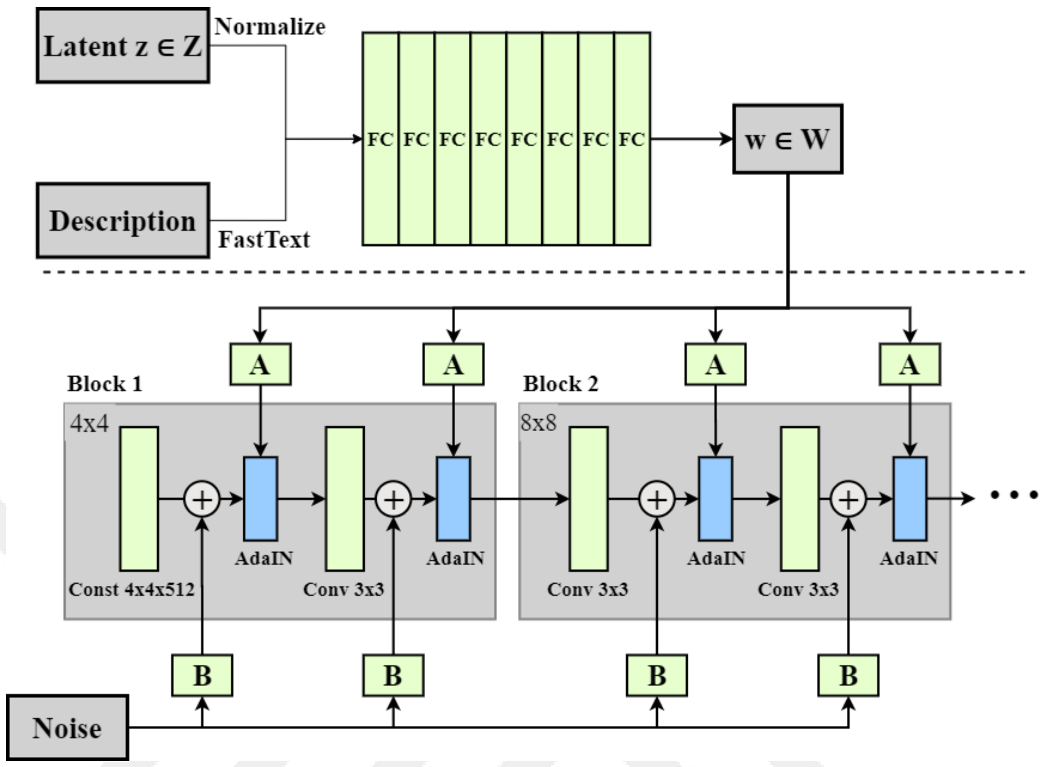


Figure 11: Architecture of cStyleGAN Generator. Above the dashed line is mapping network while below is synthesis network.

Mapping network in the cStyleGAN, in contrast to StyleGAN, takes noise vector concatenated with sentence embeddings. Although we have used FastText for encoding sentences, changing input layer for using different embedding types is a matter of changing amount of neurons in the input layer. An important application of mapping network is that it acts as a substitute for Conditioning Augmentation discussed in 2.2.3.2. We observed that adding an additional CA network before generator results in a qualitative loss of quality and loss of conditioning variable-output alignment in generated images. For this reason, an additional CA network have not been used in cStyleGAN.

For enforcing conditioning variable stronger and achieving better description-output alignment, similar to GAN-CLS loss, we have added a third input to discriminator in order to convert it to matching-aware discriminator: real samples with description of another sample which does not match. Our architecture uses Wasserstein GAN for reasons discussed in Section 2.2.2. We have combined properties of Wasserstein Distance with GAN-CLS loss.

In our model, we have used FastText model for encoding natural language sentences, which produces 300 dimension sentence embeddings. However, input shape changes as layers in discriminator activated since each subsequent layer has a different input shape compared to previous one. As a reshape method, we have used fully connected layers that takes embeddings as input and outputs with shape according to input shapes of discriminator layers. We observed that this method works considerably better compared to adding conditional variables to layer output before feeding it through another fully connected layer at the end of discriminator architecture. In current version, result obtained from convolutional layers is directly fed into fully connected layer since it already contains information from conditioning variable.

4.2.1 Discriminator Type 1

In our first discriminator type, we have concatenated embeddings (which are our conditioning variable) with the output of last convolutional layer. Then we fed concatenated input into fully connected layer for obtaining discriminator output. This method produced results that are not aligned with given descriptions. Architecture diagram of this Discriminator Type 1 can be seen in Figure 12.

4.2.2 Discriminator Type 2

Instead of concatenating embeddings with output of the convolution layers and then feeding it to the fully connected layer, we concatenated the embeddings with the image and fed it directly into the convolution layers. We have used an additional fully connected layer for reshaping embeddings to convert input shape into suitable form that can be feed into our first convolution layer. We have observed that our model works best with this discriminator type. Architecture diagram of Discriminator

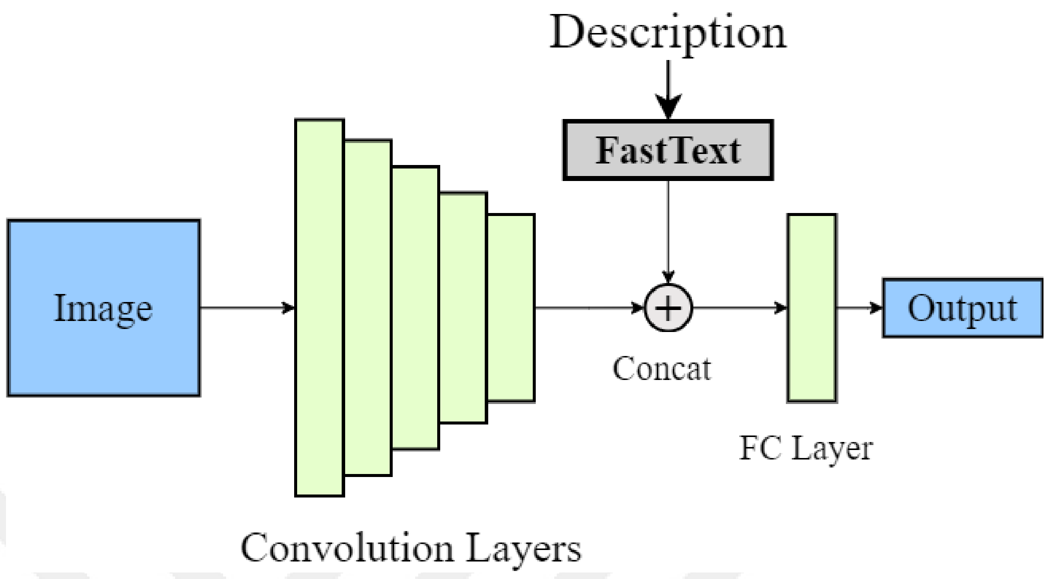


Figure 12: Architecture of Discriminator Type 1.

Type 2 can be seen in Figure 13.

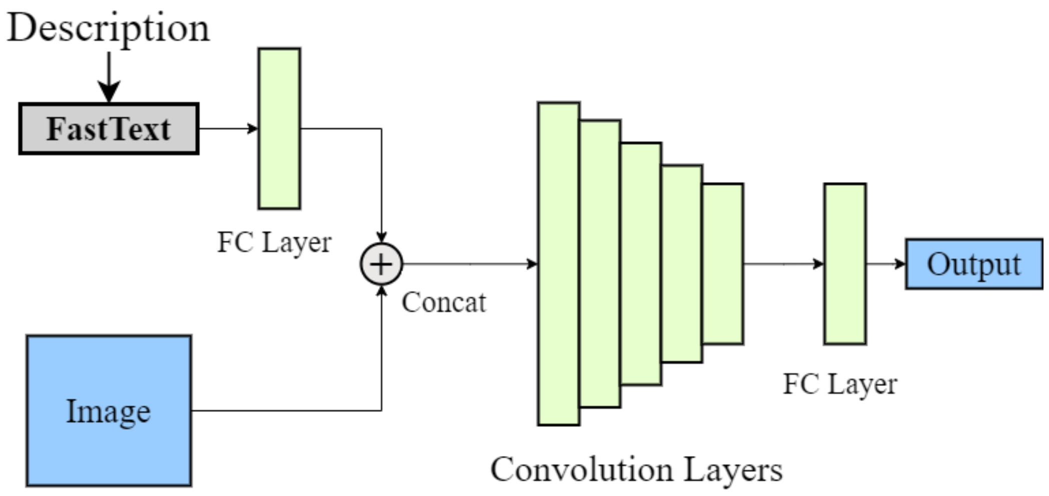


Figure 13: Architecture of Discriminator Type 2.

Compared to Discriminator Type 1, this type utilizes embeddings throughout the architecture instead of only utilizing at the fully connected layer at the end. Weights of the convolution layers also use information from the embeddings, which helps in conditioning outputs better.

We have used Discriminator Type 2 in our model throughout this thesis as it produces the best results.

4.3 3D Facial Structure Pipeline

We have used Position-map Regression Network (PRN) for generating 3D facial structures from 2D images. PRN uses UV Position Map to represent 3D facial structure. UV Position Map is 2D representation of points in 3D structure. PRN transfers input 2D image into position map image using encoder-decoder architecture. In our case, position map generated by PRN is 3D facial structures of images generated by given text descriptions. We have chosen PRN over other methods, such as using a GAN architecture that creates voxels (similar to [56]). There are multiple reasons for this decision: first, number of voxels generated by GAN is not enough in our case. Even the smallest changes in human face may cause the person to look like another person. Representing a human face in 3D with voxels with this much detail requires a high number of voxels, which is a very hard task. Secondly, there are a many unneeded voxels when a human face is generated with voxels, such as voxels behind the head, or voxels represents the location of inside the head, which are unnecessary. Example images of voxelated 3D human face models can be seen in Appendix A.3.

We have applied various processing to generated 3D models for improving the quality. Facial structures generated by PRN was lacking detail if the texture resolution is low. This is not a problem caused by PRN and it is natural to have blurry textures when low resolution textures are used. However, we can increase quality of the texture by up-scaling the image with existing AI methods. We have used ES-RGAN [60] for increasing texture resolution applied to 3D models. Although this may cause unwanted relics on textures for 2D applications, we have observed that overall quality of texture on 3D model increased when up-scaling is applied to low resolution textures. We also observed jagged lines on default generated model. We have normalized vertices in x, y and z channels separately in a size 7 window for smoothing out the surface of the model. This would cause a small loss in details of generated model. To add finer details to model and to emphasize depth information of facial depth further, we have used color information in input images to calculate an estimation of height of certain areas in the images for creating an estimated height-map, and applied this height-map estimation to facial structure. We have converted image to grayscale by taking max of RGB values for each pixel, then the values are normalized between the range $[-0.5, 0.5]$. We have clipped values that are outside of range $[-0.49, 0.49]$ for eliminating unwanted parts that are pure black or pure white, which may cause unrealistic edges in the 3D model. We also clipped values between the range $[-0.35, 0.26]$ for emphasizing effect of higher and lower areas. Finally, we have applied a rectangular mask that crops face area for making our height-map usable by PRN. Results of generated height-map estimations can be seen in Figure 14. A comparison of default 3D facial structures generated by PRN and our improved models can be seen in Figure 15.

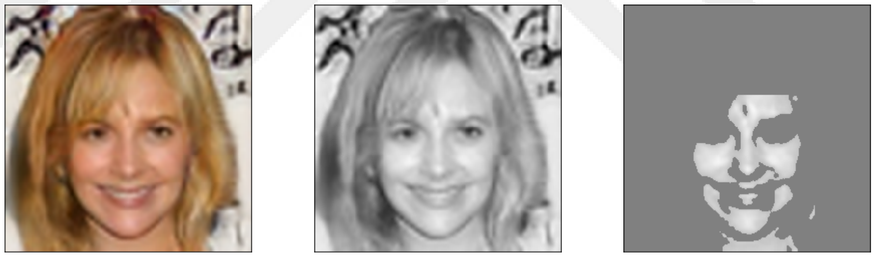
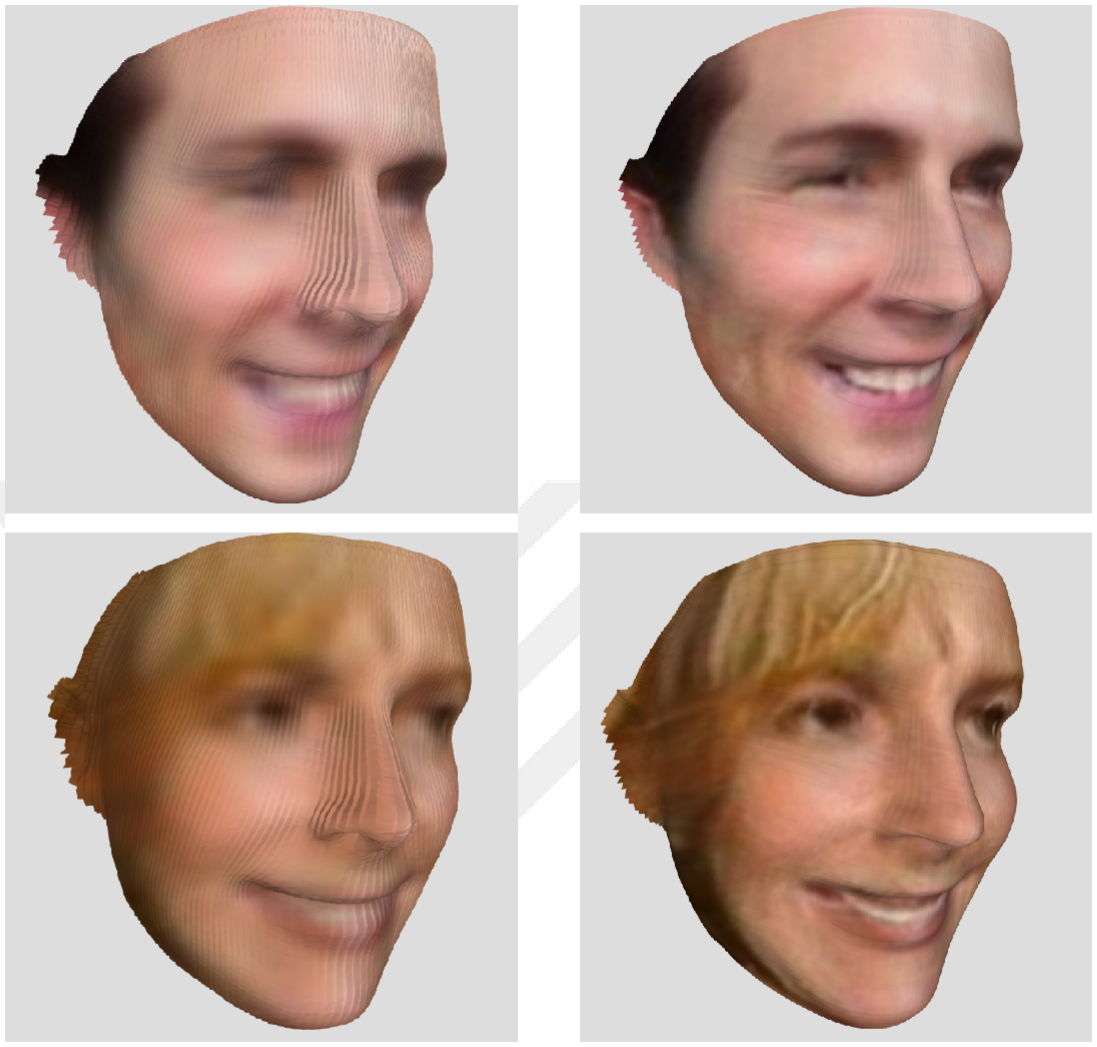

(a) Original Image (b) Grayscale (c) Height-map Estimation

Figure 14: Height-map estimation obtained from generated 2D images.



(a) Default PRN Results (b) Improved Models

Figure 15: Comparison of default PRN models and improved models.

4.4 Datasets

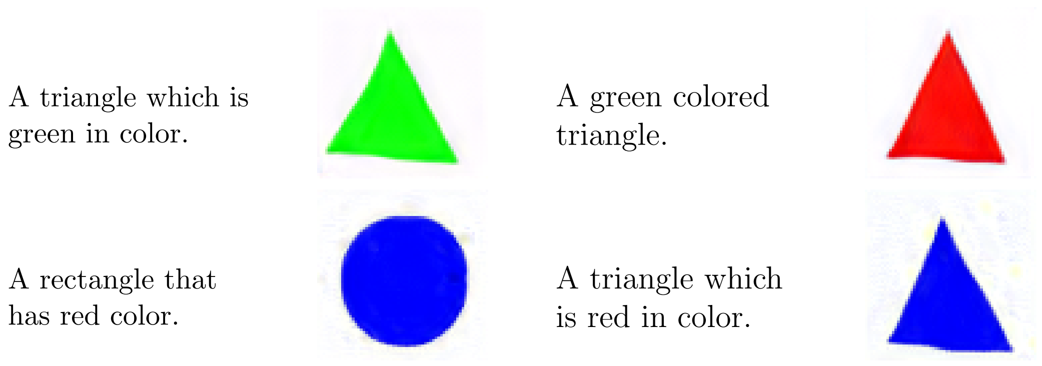
Neural networks require sufficient amount of data for achieving good results. In this work, two datasets are utilized for increasing the quality of the results: Face2Text and CelebA. Face2Text dataset contains 4076 images with descriptions written by humans. CelebA dataset contains 202599 celebrity face images with 40 annotations that describe the qualitative properties present in the each image. However, these annotations are numerical values and are not in natural language form, which are needed to be converted to sentence form using Description Generation Module. We also tested our model with a toy dataset of colored shaped for ensuring that our model correctly conditions outputs on given conditioning variables.

4.4.1 Extended Dataset

Face2Text dataset contains 4K images from CelebA dataset with textual descriptions annotated by humans. Although it is possible to train a GAN with this data, it is insufficient to train for the state of the art results with such a limited dataset. Descriptions for additional images from CelebA dataset have been generated using our Description Generation Module to cope with this problem. An additional 48069 instances chosen from image-text pairs where descriptions are generated from CelebA annotations are added, which results in 52145 image-description pairs in total. We refer to this combined dataset as ”Extended dataset” throughout this thesis. Additional images in this dataset are selected according to the quality of available annotations, which we discussed in more detail in Section 4.5.

4.4.2 Shapes Toy Dataset

We have generated a dataset that contains three different shapes of circle, square and triangle, with three different colors each, red, green, and blue. This dataset is used for testing the cStyleGAN on how successful it is when conditioning output on given inputs. In our experiments, we always made sure that output is conditioned on inputs correctly before trying to increase the generated image quality. This test also shows that our model can be applied not only on human face images, but also images in other modalities. In our tests, we saw that cStyleGAN with Discriminator Type 1 is not successfully conditioning outputs on given inputs (shown in Figure 16), while Discriminator Type 2 perfectly conditions (shown in Figure 17). Only one in four outputs are aligned with given inputs when Discriminator Type 1 is used, while all of them are aligned when Discriminator Type 2 is used.

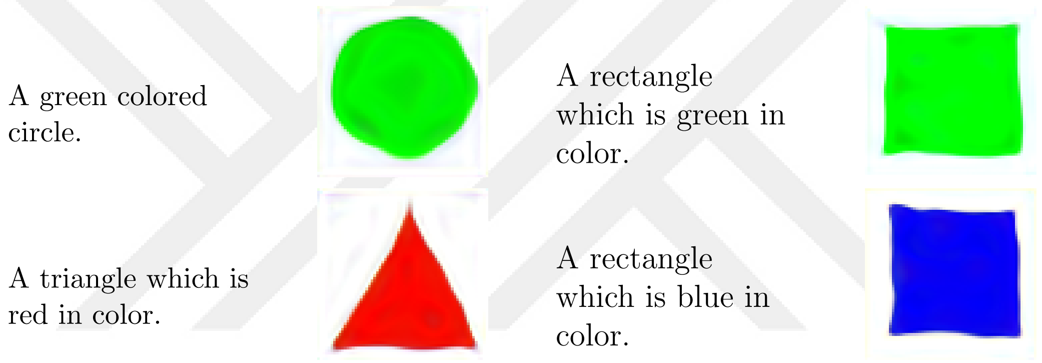

A triangle which is green in color.

A green colored triangle.

A rectangle that has red color.

A triangle which is red in color.

Figure 16: Examples on toy dataset with Discriminator Type 1. Only one in four images is aligned with given descriptions.


A green colored circle.

A rectangle which is green in color.

A triangle which is red in color.

A rectangle which is blue in color.

Figure 17: Examples on toy dataset with Discriminator Type 2. All images are correctly aligned with descriptions.

This is merely a test for ensuring that outputs are aligned with given conditioning variables as input. In Section 4.6.1 and Section 4.6.2 we show our results on real datasets.

4.5 Description Generation Module

GANs require large amounts of data to generate high quality outputs. Face2Text dataset contains 4K images + descriptions. Although it is possible to generate outputs that are aligned with descriptions of this data, it is insufficient for generating photo-realistic images.

There are 40 annotations available for each image in CelebA dataset. These annotations are used to describe certain physical properties of the person in the image, such as gender, hair color, hair length, and so on. Each one of 40 annotations have the value of either "1" or "-1", where "1" means person in the image has that quality while a "-1" means not having that quality. For example, if "mustache" and "black hair" annotations of an image is "-1" and "1" respectively, that means the person in the image does not have mustache and has black hair.

A rule based generation system have been used to generate additional descriptions by using these annotations. Images that have very few number of "1"s as annotations are eliminated since they are lacking enough information to describe given images. In our work, we have chosen to eliminate an image if amount of annotations marked as "1" for that image is less than a certain value, where we chosen this value as 8. This eliminates images with inadequate information. Additionally, images with annotations that has no information about hair are eliminated. Finally, images with conflicting annotations, for example, having "no beard" marked as "1" and "goatee" marked as "1", are eliminated.

For the remaining images, a sentence is generated according to annotations by using a rule based system. A description is chosen from predefined set of multiple descriptions for each annotation to increase diversity of generated descriptions. This randomness helps the model to avoid memorizing given sentences and generate meaningful images when a different, never seen before description is queried. Order of sentences in generated descriptions are also randomized by changing order of descriptive sentences in one description. We have used a permutation based system for randomizing order of sentences except the first sentence in $(n-1)!$ ways where n is number of sentences in the description.

In our experiments, descriptions are generated for 48K undescribed images using this method, resulting in 52K total image-description pairs when combined with

Face2Text dataset. CelebA dataset has an uneven ratio between male and female images, which may create an unwanted bias in our model. Our module preserves 1 to 1 ratio between male and female images when generating descriptions. Number of eliminated images for parameters used in our experiments with elimination reasons are shown in Table 1. Note that eliminations are not mutually exclusive and may be counted in multiple reasons.

Table 1: Number of eliminated images for corresponding reason and total generated description count.

Reason	Number of images eliminated
Have less than specified amount (8) of annotations marked as ”1”	84251
Conflicting annotations marked as ”1”	77236
Skipped for preserving male to female ratio	70138
Generated description amount	48069

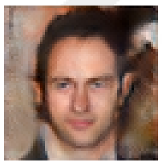
Our algorithm generates sentences over annotations in one pass. For each image, only one pass is applied. Our algorithm runs in $O(nl)$ time where n is number of images, l is number of annotations for each image, which is 40 in CelebA dataset.

4.6 Experiments

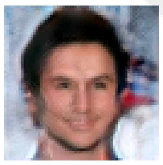
Here we share our experiments with our model cStyleGAN on two different datasets: Face2Text dataset, and Extended dataset. As a comparison to existing work, we also compare our results with Pro-StackGAN. In our experiments, for cStyleGAN, we have used Adam optimizer [61] with learning rate 0.001 for resolutions up to 128x128, and 0.0015 for 128x128, for both generator and discriminator with $\beta 1 = 0.0$ and $\beta 2 = 0.99$. We have trained our model for 1.5 million iterations per each resolution, starting from 8x8 up to 128x128. Although, we have early stopped our training during 128x128 resolution phase due to loss in image quality, decreased loss in generated images and increased generator loss. We have used batch-size of 8 for all resolutions. Embedding size generated from FastText module set as 300 dimensions.

4.6.1 cStyleGAN on Face2Text Dataset

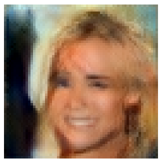
Earlier work of this thesis was developed using Face2Text dataset. We have already ensured that output is aligned with given conditioning variables by using our toy shapes dataset. However, another important aspect is image quality of generated outputs. Face2Text dataset contains 4K human face images which are annotated by humans. This data is insufficient for our task as neural networks, especially GANs require large amounts of data. Images generated from given descriptions with model trained on only Face2Text dataset can be seen in Figure 18.



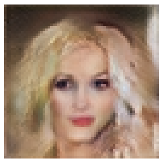
A young man with short brown hair and small eyes. His eyebrows are thick. His nose is small and his lips are thin. A stubble is growing on his face. He has got a well - defined jawline.



A man with short, brunette hair, thick eyebrows, light eyes, a long nose and a stubble beard and moustache. He has thin, smiling lips and a prominent chin.



A woman with curtained, blond hair, a beaky nose, dark eyes, thin eyebrows and a wide smile.



A woman with long blonde hair, big green eyes and full pouting lips. She is wearing makeup.

Figure 18: Examples images from cStyleGAN trained on Face2Text dataset.

4.6.2 cStyleGAN on Extended Dataset

We have trained our network on Extended dataset for 1.5 million iterations per each resolution. This corresponds to 30 epochs per each resolution starting from 8x8 and going up to 128x128 on 52K data instances, which in total makes 150 epochs. However, in our experiments, we have observed that best results are achieved when the model finished training for 64x64 resolution (120 epochs), before activating another block in synthesis network to train for 128x128 resolution. During our experimentation, we have observed that generator loss explodes as discriminator loss goes to smaller values after epoch 120, which is the end of training for 64x64 images. Generated images do not improve after this point, and get worse. Therefore, we have early stopped our training. Our best results are achieved at epoch 120, so, our reported results are from our model trained for 120 epochs. Losses of generator and discriminator over epochs can be seen in Figure 19.

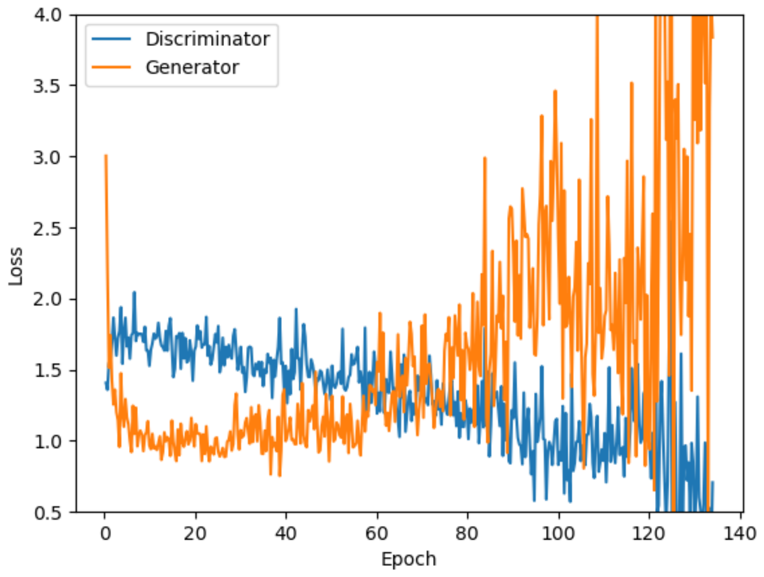


Figure 19: Generator and Discriminator losses over training epochs.

4.6.3 Noise Scaling and Effect of Noise

Earlier during our experiments, we have observed that if scale of noise is too high compared to input text embeddings obtained from descriptions, then generated images are mostly based on the noise and not correctly aligned with descriptions given. This was a trivial problem, but an important one. We have found that simply scaling noise with a constant value during training fixes the problem. In our experiments, we have observed that average of absolute values in noise was 14 times higher than average of absolute values in text embeddings. Therefore, we have scaled noise with 0.07 (or roughly $\frac{1}{14}$), which solved the problem. This is same as sampling noise from the distribution $\mathcal{N}(0, 0.07)$.

We also tested effect of noise on generated images. We have fixed the input descriptions and generated images with different noise values sampled from same normal distribution of $\mathcal{N}(0, 0.07)$. We observed that properties of generated image changed while still being aligned with given descriptions. Examples of this experiment can be seen in Figure 20.

A serious looking woman with straight blond hair. She has arched eyebrows. Her eyes are brown and big and her lips are thin. She has got a heavy lower lip.

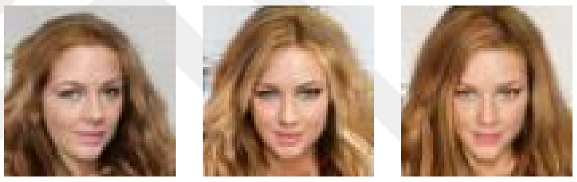


Figure 20: Results of same description with different noise values.

4.6.4 Data Augmentation

Data augmenting during GAN training may cause generator to learn generating augmented images as well, which is an undesired result in our case. We have tried augmenting our training set for increasing variety in the training set for any possible improvements. Some transformations have big effects on the image, so we decided to apply those transformation with a probability only. Transformations and their chance of being applied with the order they applied (from top to bottom in table) are shown in following table:

Table 2: Applied transformations with their chance of being applied.

Applied Transform	Probability of Being Applied
Random Horizontal Flip	50%
5 degrees of random rotation	100%
Adding Gaussian Noise $\mathcal{N}(0, 0.07)$	30%
Erasing random patch from image	1%

Aim of this test is achieving an improvement in the generated image quality. So, we tested this method with Extended dataset, however, we have encountered the transformations that we applied also in generated images, which is unwanted. Image quality was not increased noticeably. Noise and rotation can be seen in most generated images. We haven't encountered erased patches in generated images. Examples of this test can be seen in Figure 21.

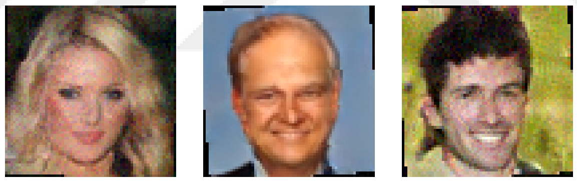


Figure 21: Examples from model trained with augmented training set.

4.6.5 Ablation Study: Disabling Blocks in Synthesis Network

We have disabled different blocks in our experiments and compared outputs to see effects of individual blocks in synthesis network. In this experiment, we refer to first block which generates 4x4 resolution output as Block 1, second block which generates 8x8 resolution output as Block 2 and so on. A number of combinations of blocks are disabled to see the effects. Eliminating first few layers has dramatic effect on the output since subsequent layers depend on output of previous layers. First, we have disabled Block 2, which outputs 8x8 resolution image. Image generated roughly represented a human face image, got blurry and lost detail. Even the parts that are easier to spot in image such as hair were barely recognizable. Next, we only disabled Block 3. This time information were not lost up to Block 3, so output has more detail compared to previous test. Although still very blurry, this time a general facial structure is present in the image, such as eyes and nose. From these results, we can infer that Block 2 is more directed towards learning overall facial key-points. When we disabled both Block 2 and Block 3, we saw that even the rough details of facial structure are lost again. Results of this experimentation can be seen in Figure 22.

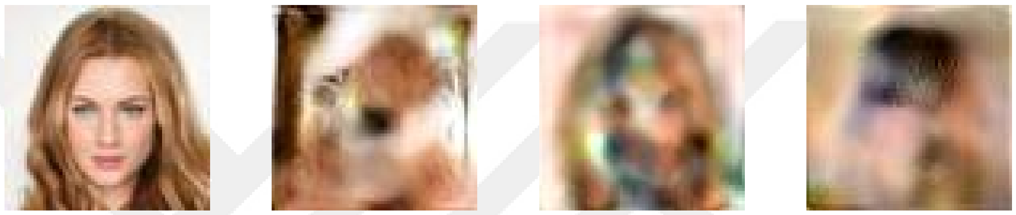

(a) Original Output (b) Block 2 Disabled (c) Block 3 Disabled (d) Block 2 and 3 Disabled

Figure 22: Results of disabling different early blocks in synthesis network of generator.

Disabling Block 2 resulted in loss of both general facial structure details and overall image composition, only a very rough human face is visible. Disabling only Block 3 resulted in more detailed but still very blurry face image. Disabling both Block 2 and Block 3 has a similar result of only disabling Block 3 since Block 3 is dependent on the output of Block 2.

4.7 Results

In this section we show our results of cStyleGAN in one-shot setting, and discuss the implications and weak points of the results. We show results of 3D images generated from 2D human face images. We also compare existing work (Pro-StackGAN) and cStyleGAN trained on both Face2Text and Extended datasets separately to show effectiveness of Extended dataset. All results of cStyleGAN are aligned with given descriptions. However, we do not show descriptions for every result as it would make this section messy. Instead, we are showing examples also with descriptions in Section

4.7.2 separately.

4.7.1 Results of cStyleGAN Trained on Face2Text

In this section we show results of our cStyleGAN when trained only on Face2Text dataset. As expected, image quality is worse compared to cStyleGAN trained on Extended dataset. We have trained our model with same parameters explained in Section 4.6. cStyleGAN achieved Inception Score of 1.9 ± 0.2 when trained only on Face2Text dataset. When we trained until the end of 64x64 resolution with 1.5 million iterations for each resolution, we observed that model faces mode collapse. This is a good test for showing a weakness of Inception Score as it is not taking variety into consideration as long as generated images belong to different classes.

First we show cStyleGAN trained only on Face2Text with 1.5 million iterations for each resolution to show how cStyleGAN mode collapses when there is not sufficient data for desired iterations. Although Wasserstein GANs are better in terms of mode collapse compared to Naive GANs, it still can happen when model is over-trained. Results can be seen in Figure 23.

Our model mode collapsed with 1.5 million iterations on each resolution when trained with Face2Text dataset, we decided to decrease iterations for each resolution to 600K since data is smaller. We have observed that our model does not mode collapse with this setup. Results can be seen in Figure 24.

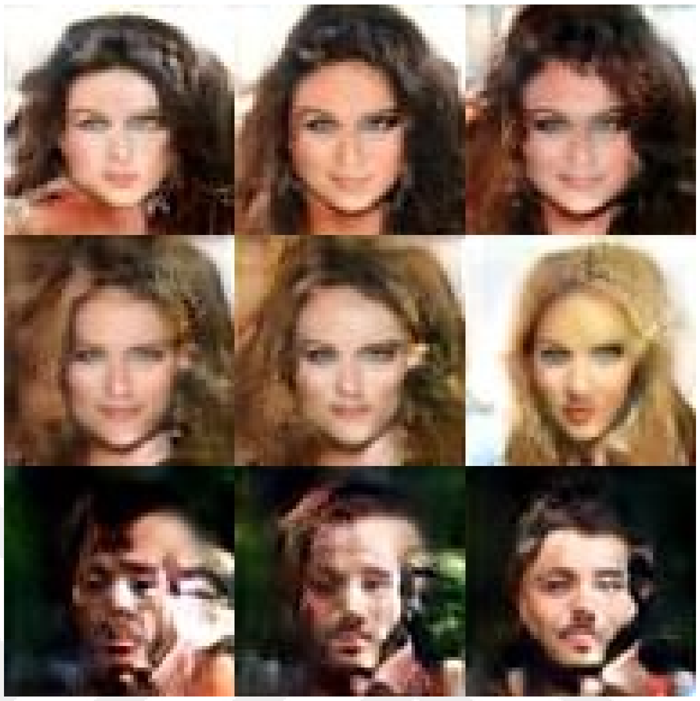


Figure 23: cStyleGAN mode collapsed on Face2Text dataset with 1.5 million iterations for each resolution.

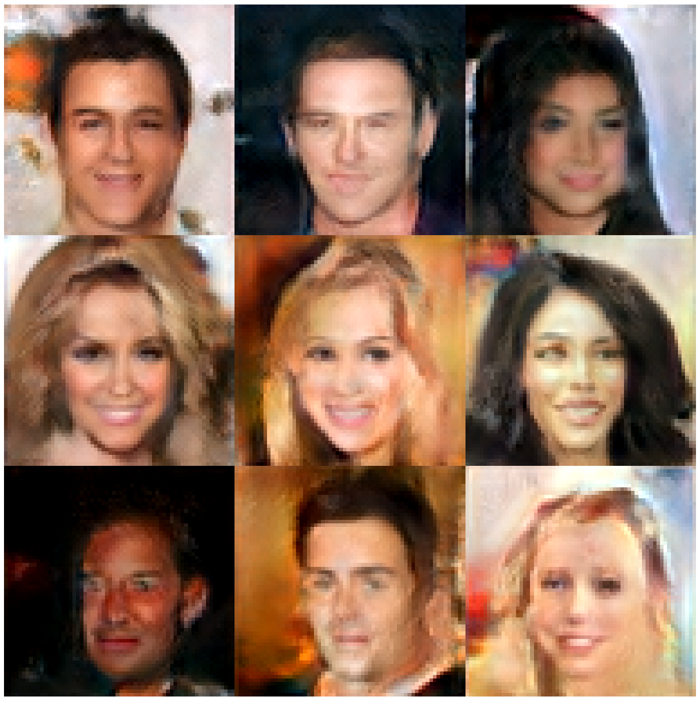


Figure 24: Results of cStyleGAN on Face2Text dataset trained 600K iterations per resolution.

4.7.2 Results of cStyleGAN Trained on Extended Dataset

Results of images generated by cStyleGAN trained on Extended dataset can be seen in Figure 25.

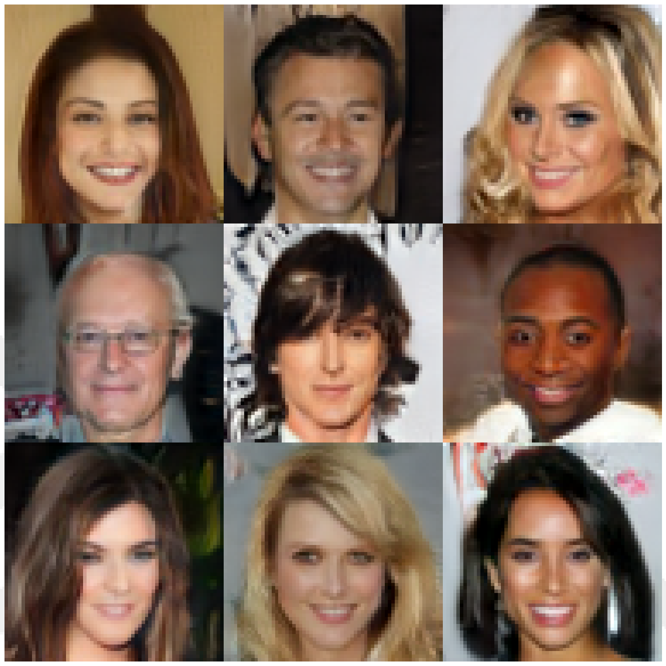
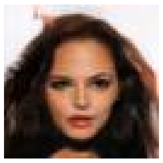
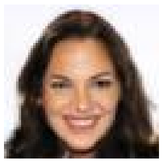


Figure 25: Results of cStyleGAN trained on Extended dataset.

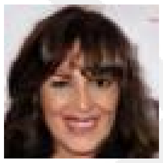
We show capabilities of our model by doing the following test: first, we generate a 2D face image from a given description. Then, we make small changes in given description and regenerate the image to show how our model captures word specific information in given descriptions. We observed that our model is able to make drastic changes according to differences in given descriptions. For example, changing "black hair" to "blond hair" drastically changes hair color in generated images. Similarly, changing "looks serious" to "smiling" changes expression of generated face in a noticeable way. However, properties that are relatively harder to notice such as "heavy lower lip" is not expressed as successfully as other properties. This is a problem that can be improved with a larger and more diverse dataset. Results of this method can be seen in Figure 26, changed keywords compared to previous image (from left to right) are marked with red color.



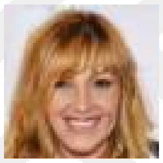
A woman with wavy black hair. She has arched eyebrows. Her eyes are brown and small and her lips are thin. She has got a heavy lower lip. She looks serious.



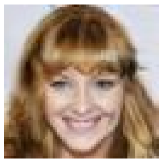
A woman with straight black hair. She has arched eyebrows. Her eyes are brown and small and her lips are thin. She has got a heavy lower lip. She is smiling.



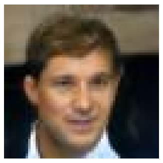
A woman with wavy black hair with bangs. She has arched eyebrows. Her eyes are brown and small and her lips are thin. She has got a heavy lower lip. She is smiling.



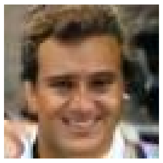
A woman with straight blond hair with bangs. She has arched eyebrows. Her eyes are brown and small and her lips are thin. She has got a heavy lower lip. She is smiling.



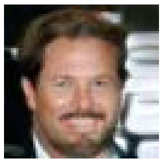
A pale skinned woman with straight blond hair with bangs. She has arched eyebrows. Her eyes are brown and small and her lips are thin. She has got a heavy lower lip. She is smiling.



A male with brown hair. The man has a serious look on his face. He has big nose. He has bags under his eyes. He has no beard.



A male with brown hair. The man is smiling. He has big nose. He has bags under his eyes. He has no beard.



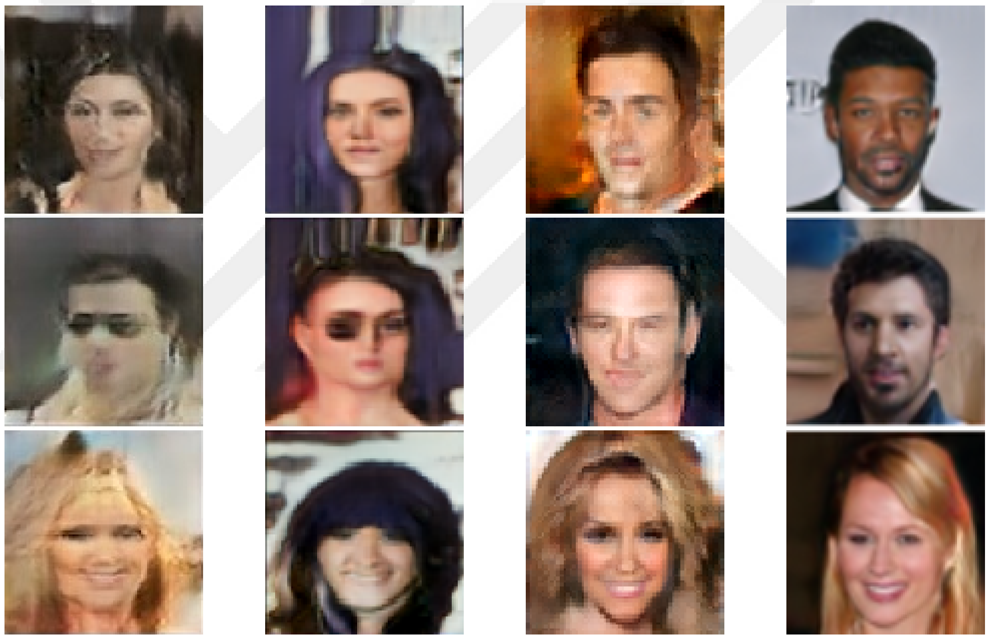
A male with brown hair. He has oval face. The man is smiling. He has small, pointy nose. He has bags under his eyes. He has a goatee.

Figure 26: Results of cStyleGAN trained on Extended dataset in zero-shot (unseen text) setting.

4.7.3 Comparison with Existing Work

We have tested Pro-StackGAN as a comparison and to show improvements made by our Extended dataset. In this section we combine results of different models as a comparison for better a understanding of our results.

As expected, results of models trained on Extended dataset have higher quality. Examples from both Pro-StackGAN and cStyleGAN trained on Face2Text dataset and Extended dataset can be seen in Figure 27.



(a) Pro-StackGAN on Face2Text

(b) Pro-StackGAN on Extended

(c) cStyleGAN on Face2Text (600K iterations)

(d) cStyleGAN on Extended

Figure 27: Comparison of outputs from Pro-StackGAN and cStyleGAN.

A comparison of PQD scores achieved by trained networks are shown in Table 3.

Table 3: Comparison of PQD scores. Best result achieved with our model when trained with Extended Dataset.

Model	Dataset	PQD
Pro-StackGAN	Face2Text	0.36279
Pro-StackGAN	Extended	0.31031
cStyleGAN (ours)	Face2Text	0.28584
cStyleGAN (ours)	Extended	0.26478

Our model trained with same parameters explained in Section 4.6 achieved Inception Score of 2.4 ± 0.1 when trained on Extended dataset. Work in [45] have also reported inception score on generated images. A comparison of Inception Scores achieved with our model compared to existing work can be seen in Table 4.

Table 4: Comparison of Inception Scores. Best result achieved with our model when trained with Extended Dataset.

Inception Score	
Text2FaceGAN	1.4 ± 0.7
Ours (Face2Text Dataset)	1.9 ± 0.2
Ours (Extended Dataset)	2.4 ± 0.1

4.7.4 Results of 3D Generation

Examples of 3D facial structures generated from our 2D images are shown in Figure 28.

A serious looking woman with straight blond hair. She has arched eyebrows. Her eyes are brown and big and her lips are thin. She has got a heavy lower lip.

A young male with black straight hair. He has bags under his eyes. He has no beard. The man is attractive. He has big nose. The man is smiling.

A young female with blond hair with bangs. The woman is attractive. She has bags under her eyes. The woman is smiling. She has pointy and big nose.

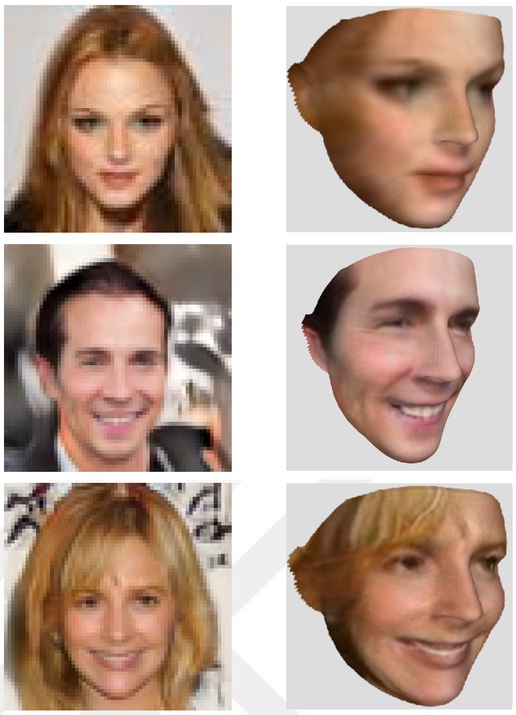


Figure 28: Examples of 2D images and corresponding 3D facial structures generated from textual descriptions.

We also tested computation time of PRN as it is important in most applications. Speed of PRN directly correlates with 2D input image. We have tested speed of PRN on a GTX 1070 GPU with Python 3.7.6 and TensorFlow 1.14.0. A comparison of 3D generation speed averaged over 10 tries for each different input resolutions can be seen in Table 5. PRN could not detect a face in resolutions of 32x32 and under.

Table 5: Comparison of 3D generation speed of PRN for different input sizes.

Input Resolution	Time
8x8	N/A
16x16	N/A
32x32	N/A
64x64	1.471 seconds
128x128	3.713 seconds
256x256	12.517 seconds

CHAPTER V

CONCLUSION

This thesis aims to improve image quality and diversity of generated images in the domain of human face images. We propose cStyleGAN for generating high quality human face images that are conditioned on given natural language descriptions. cStyleGAN is a conditional Wasserstein GAN which is modified from existing StyleGAN and trained with our extended dataset, which is a combination of Face2Text dataset and images with descriptions generated by our Description Generation Module. Our cStyleGan trained on extended dataset achieved 2.4 ± 0.1 inception score.

Due to lack of research in this field, available datasets are insufficient in this domain. Although there are datasets such as CelebA with data as much as 203K, these datasets do not contain descriptions, or do not contain annotations in natural language form, or both. Our Description Generation Module aims to generate descriptions for images using numerical annotations in CelebA dataset to increase data which we can use in conditional networks. We believe that this approach can help researchers that work on conditional models.

We have tested our proposed network with different settings and different datasets. We have observed that additional images with descriptions obtained by Description Generation Module helped in improving image quality noticeably while still keeping images aligned with given conditioning variables. We have observed that mode collapse can happen even with Wasserstein GANs and encountered mode collapse during some our tests. Extra data used during training also helped alleviating this problem.

We have showed results of our proposed GAN evaluation method, Perceptual Quality Distance (PQD). We have shown that our PQD results are also aligned with popular GAN evaluation metrics such as Inception Score and Fréchet Inception Distance. We have tested our trained models and discussed that PQD score decreases as models generate higher quality images that are closer to real images taken from training dataset. We believe that this method can prove to be a useful alternative to currently available metrics.

Images are a good way for obtaining information visually. However, 3D models can provide more information as it is not possible to see an entity from different angles just by looking a 2D image. Converting 2D images into 3D models can help obtaining more information with the help of illumination, depth information, and being able to seeing it from different perspectives. We generate 3D human face models from 2D images by using PRN, which can carry more information to observers. We also improved 3D results by upscaling textures for more fidelity and applying our estimated height-map to models for better 3D facial shapes.

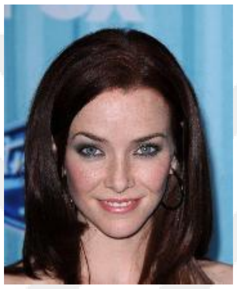
As future work, evaluation methods for measuring output alignment of conditional GANs with given conditioning variables can be explored, which can be useful for measuring performance of conditional GANs while also keeping input-output alignment. There is still no widely-used evaluation method for measuring input-output alignment in different conditional GANs.

Controllable image generation can have important applications. Especially in human face image generation, controlling the properties of generated images may help in identifying suspects. Manually drawing suspects from given descriptions from eyewitnesses is a time consuming and hard process, drawing such images requires training and proficiency. We believe that developments in conditional image generation will aid people in this field.

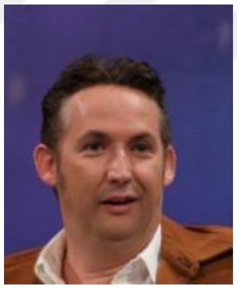
APPENDIX A

SAMPLES

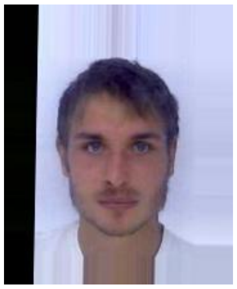
A.1 Face2Text Data Training Samples with Descriptions



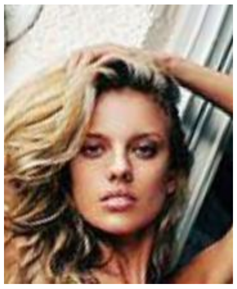
A young woman with voluminous brown hair parted at the side. She is light - skinned and her eyes are blue, around which there is a small amount of make - up. The girl has freckles mostly on her cheeks. Her lips are thin and are painted pink. They are slightly parted and one can see her teeth



A man with short, dark - brown hair, protruding ears, a long nose, a weak chin and a long face with an open - mouthed smile.



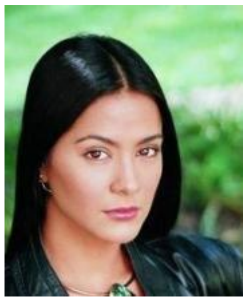
A blue eyed man with messy brown hair, a large nose, sparse peach fuzz beard and an intimidated expression.



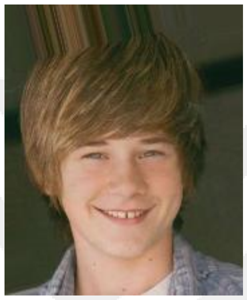
A young woman with blonde hair and dark roots. Her eyes are dark - coloured and there is dark make - up around them. Her lips are somewhat thick and she looks serious.

Figure 29: Face2Text data training samples with human written descriptions.

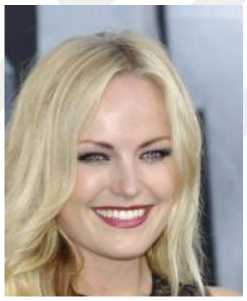
A.2 Extended Data Training Samples with Descriptions



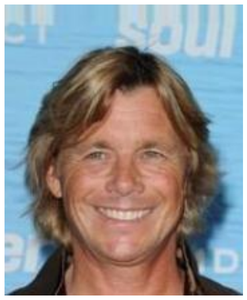
A young female with black straight hair. The attractive woman is wearing heavy make up. She has oval face. She has arched eyebrows.



A young male that has pale skin with brown straight hair with bangs. He has big nose. The man is attractive. He has no beard. The man is smiling.



A young female with blond wavy hair. The attractive woman is wearing heavy make up. She has oval face. The woman is smiling. She has pointy nose. She has narrow eyes.



A middle aged male with brown hair. He has no beard. The man is smiling. He has narrow eyes with bags under his eyes. He has a face with double chin. He has big nose.

Figure 30: Samples from CelebA dataset which does not exist in Face2Text dataset, with added descriptions generated by our module.

A.3 Voxelated 3D Human Face Models

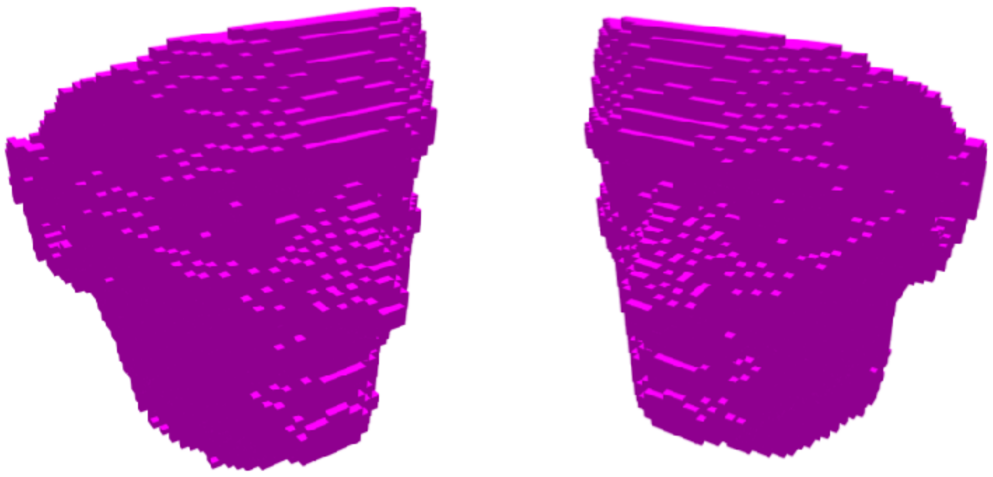


Figure 31: Voxelated human face model with 64x64x64 resolution.

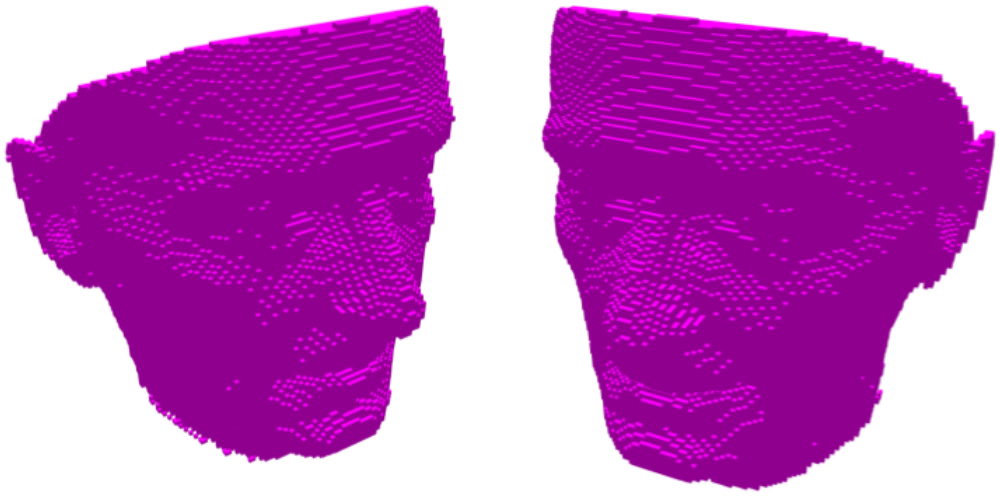


Figure 32: Voxelated human face model with 128x128x128 resolution.

A.4 CelebA Annotation Index Map

Table 6: 40 annotations available in CelebA Dataset.

Annotation Index	Meaning
0	5_o_Clock$_shadow$
1	Arched_Eyebrows
2	Attractive
3	Bags_Under_Eyes
4	Bald
5	Bangs
6	Big_Lips
7	Big_Nose
8	Black_Hair
9	Blond_Hair
10	Blurry
11	Brown_Hair
12	Bushy_Eyebrows
13	Chubby
14	Double_Chin
15	Eyeglasses
16	Goatee
17	Gray_Hair
18	Heavy_Makeup
19	High_Cheekbones
20	Male
21	Mouth_Slightly_Open
22	Mustache
23	Narrow_Eyes
24	No_Beard
25	Oval_Face
26	Pale_Skin
27	Pointy_Nose
28	Receding_Hairline
29	Rosy_Cheeks
30	Sideburns
31	Smiling
32	Straight_Hair
33	Wavy_Hair
34	Wearing_Earrings
35	Wearing_Hat
36	Wearing_Lipstick
37	Wearing_Necklace
38	Wearing_Necktie
39	Young

A.5 Samples from Description Generation Module

Annotation: [’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’]

Generated Description: An old female with blond wavy hair. She has pointy nose. The attractive woman is wearing heavy make up..

Annotation: [’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’]

Generated Description: A young male with brown straight hair. He has narrow eyes with bags under his eyes. The man looks serious. He has no beard.

Annotation: [’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’]

Generated Description: A young male with black hair with bangs. He has no beard. He has pointy nose. The man is smiling.

Annotation: [’-1’, ’1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’1’, ’1’, ’-1’, ’1’, ’1’, ’-1’, ’-1’]

Generated Description: A middle aged female with blond wavy hair. The attractive woman is wearing heavy make up. She has arched eyebrows. She has pointy nose. The woman is smiling.

Annotation: [’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’]

Generated Description: A young male with brown straight hair with bangs. He has no beard. He has bags under his eyes. He has big nose. The man is smiling.

Annotation: [’-1’, ’1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’1’, ’-1’, ’1’, ’-1’, ’1’, ’1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’1’]

Generated Description: A young female with blond straight hair. She has pointy nose. She has a face with rosy cheeks. The woman is smiling. She has arched eyebrows. The attractive woman is wearing heavy make up.

Annotation: [’1’, ’-1’, ’1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’]

Generated Description: A young male with black hair. He has bags under his eyes. The man is attractive. The man has a slightly open mouth

Annotation: [’-1’, ’1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’1’, ’-1’, ’1’, ’-1’, ’-1’, ’1’]

Generated Description: A young female with black hair and a receding hairline. The woman is smiling. She has arched eyebrows. She has oval face. The attractive woman is wearing heavy make up. She has big nose.

Annotation: [’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’]

Generated Description: A young male with black hair. He has a goatee. He has chubby oval face. He has big nose.

Annotation: [’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’, ’1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’-1’, ’1’]

Generated Description: A young male with black straight hair with bangs. He has no beard. He has oval face. The man is smiling. The man is attractive.

APPENDIX B

CSTYLEGAN LAYERS

B.1 cStyleGAN Discriminator Layers

Biases excluded.

from_rgbs.0.conv.weight_orig torch.Size([16, 4, 1, 1])

from_rgbs.1.conv.weight_orig torch.Size([32, 4, 1, 1])

from_rgbs.2.conv.weight_orig torch.Size([64, 4, 1, 1])

from_rgbs.3.conv.weight_orig torch.Size([128, 4, 1, 1])

from_rgbs.4.conv.weight_orig torch.Size([256, 4, 1, 1])

from_rgbs.5.conv.weight_orig torch.Size([512, 4, 1, 1])

from_rgbs.6.conv.weight_orig torch.Size([512, 4, 1, 1])

from_rgbs.7.conv.weight_orig torch.Size([512, 4, 1, 1])

from_rgbs.8.conv.weight_orig torch.Size([512, 4, 1, 1])

convs.0.conv.0.conv.weight_orig torch.Size([32, 16, 3, 3])

convs.0.conv.2.conv.weight_orig torch.Size([32, 32, 3, 3])

convs.1.conv.0.conv.weight_orig torch.Size([64, 32, 3, 3])

convs.1.conv.2.conv.weight_orig torch.Size([64, 64, 3, 3])

convs.2.conv.0.conv.weight_orig torch.Size([128, 64, 3, 3])

convs.2.conv.2.conv.weight_orig torch.Size([128, 128, 3, 3])

convs.3.conv.0.conv.weight_orig torch.Size([256, 128, 3, 3])

convs.3.conv.2.conv.weight_orig torch.Size([256, 256, 3, 3])

convs.4.conv.0.conv.weight_orig torch.Size([512, 256, 3, 3])

convs.4.conv.2.conv.weight_orig torch.Size([512, 512, 3, 3])

convs.5.conv.0.conv.weight_orig torch.Size([512, 512, 3, 3]) convs.5.conv.2.conv.weight_orig torch.Size([512, 512, 3, 3])

convs.6.conv.0.conv.weight_orig torch.Size([512, 512, 3, 3])

convs.6.conv.2.conv.weight_orig torch.Size([512, 512, 3, 3])

convs.7.conv.0.conv.weight_orig torch.Size([512, 512, 3, 3])

convs.7.conv.2.conv.weight_orig torch.Size([512, 512, 3, 3])

convs.8.conv.0.conv.weight_orig torch.Size([512, 513, 3, 3])

convs.8.conv.2.conv.weight_orig torch.Size([512, 512, 4, 4])

embedding_shapers.0.linear.weight_orig torch.Size([1, 300])

embedding_shapers.1.linear.weight_orig torch.Size([1, 300])

embedding_shapers.2.linear.weight_orig torch.Size([65536, 300])

embedding_shapers.3.linear.weight_orig torch.Size([16384, 300])

embedding_shapers.4.linear.weight_orig torch.Size([4096, 300])

embedding_shapers.5.linear.weight_orig torch.Size([1024, 300])

embedding_shapers.6.linear.weight_orig torch.Size([256, 300])

embedding_shapers.7.linear.weight_orig torch.Size([64, 300])

embedding_shapers.8.linear.weight_orig torch.Size([16, 300])

fc.linear.weight_orig torch.Size([1, 512])

B.2 cStyleGAN Generator Layers

Biases excluded.

fcs.mapping.1.linear.weight_orig torch.Size([400, 400]

fcs.mapping.3.linear.weight_orig torch.Size([400, 400]

fcs.mapping.5.linear.weight_orig torch.Size([400, 400]

fcs.mapping.7.linear.weight_orig torch.Size([400, 400]

fcs.mapping.9.linear.weight_orig torch.Size([400, 400]

fcs.mapping.11.linear.weight_orig torch.Size([400, 400]

fcs.mapping.13.linear.weight_orig torch.Size([400, 400]

```
fcs.mapping.15.linear.weight_orig torch.Size([400, 400]
convs.0.constant torch.Size([1, 600, 4, 4]
convs.0.style1.transform.linear.weight_orig torch.Size([1200, 400]
convs.0.style2.transform.linear.weight_orig torch.Size([1200, 400]
convs.0.noise1.weight_orig torch.Size([1, 600, 1, 1]
convs.0.noise2.weight_orig torch.Size([1, 600, 1, 1]
convs.0.conv.conv.weight_orig torch.Size([600, 600, 3, 3]
convs.1.style1.transform.linear.weight_orig torch.Size([1024, 400]
convs.1.style2.transform.linear.weight_orig torch.Size([1024, 400]
convs.1.noise1.weight_orig torch.Size([1, 512, 1, 1]
convs.1.noise2.weight_orig torch.Size([1, 512, 1, 1]
convs.1.conv1.conv.weight_orig torch.Size([512, 600, 3, 3]
convs.1.conv2.conv.weight_orig torch.Size([512, 512, 3, 3]
convs.2.style1.transform.linear.weight_orig torch.Size([1024, 400]
convs.2.style2.transform.linear.weight_orig torch.Size([1024, 400]
convs.2.noise1.weight_orig torch.Size([1, 512, 1, 1]
convs.2.noise2.weight_orig torch.Size([1, 512, 1, 1]
convs.2.conv1.conv.weight_orig torch.Size([512, 512, 3, 3]
convs.2.conv2.conv.weight_orig torch.Size([512, 512, 3, 3]
convs.3.style1.transform.linear.weight_orig torch.Size([1024, 400]
convs.3.style2.transform.linear.weight_orig torch.Size([1024, 400]
convs.3.noise1.weight_orig torch.Size([1, 512, 1, 1]
convs.3.noise2.weight_orig torch.Size([1, 512, 1, 1]
convs.3.conv1.conv.weight_orig torch.Size([512, 512, 3, 3]
convs.3.conv2.conv.weight_orig torch.Size([512, 512, 3, 3]
convs.4.style1.transform.linear.weight_orig torch.Size([512, 400]
convs.4.style2.transform.linear.weight_orig torch.Size([512, 400]
convs.4.noise1.weight_orig torch.Size([1, 256, 1, 1]
convs.4.noise2.weight_orig torch.Size([1, 256, 1, 1]
convs.4.conv1.conv.weight_orig torch.Size([256, 512, 3, 3]
convs.4.conv2.conv.weight_orig torch.Size([256, 256, 3, 3]
convs.5.style1.transform.linear.weight_orig torch.Size([256, 400]
convs.5.style2.transform.linear.weight_orig torch.Size([256, 400]
convs.5.noise1.weight_orig torch.Size([1, 128, 1, 1]
convs.5.noise2.weight_orig torch.Size([1, 128, 1, 1]
convs.5.conv1.conv.weight_orig torch.Size([128, 256, 3, 3]
convs.5.conv2.conv.weight_orig torch.Size([128, 128, 3, 3]
convs.6.style1.transform.linear.weight_orig torch.Size([128, 400]
convs.6.style2.transform.linear.weight_orig torch.Size([128, 400]
convs.6.noise1.weight_orig torch.Size([1, 64, 1, 1]
convs.6.noise2.weight_orig torch.Size([1, 64, 1, 1]
convs.6.conv1.conv.weight_orig torch.Size([64, 128, 3, 3]
convs.6.conv2.conv.weight_orig torch.Size([64, 64, 3, 3]
convs.7.style1.transform.linear.weight_orig torch.Size([64, 400]
convs.7.style2.transform.linear.weight_orig torch.Size([64, 400]
convs.7.noise1.weight_orig torch.Size([1, 32, 1, 1]
convs.7.noise2.weight_orig torch.Size([1, 32, 1, 1]
convs.7.conv1.conv.weight_orig torch.Size([32, 64, 3, 3]
convs.7.conv2.conv.weight_orig torch.Size([32, 32, 3, 3]
convs.8.style1.transform.linear.weight_orig torch.Size([32, 400]
convs.8.style2.transform.linear.weight_orig torch.Size([32, 400]
convs.8.noise1.weight_orig torch.Size([1, 16, 1, 1]
convs.8.noise2.weight_orig torch.Size([1, 16, 1, 1]
convs.8.conv1.conv.weight_orig torch.Size([16, 32, 3, 3]
convs.8.conv2.conv.weight_orig torch.Size([16, 16, 3, 3]
to_rgbs.0.conv.weight_orig torch.Size([3, 600, 1, 1]
to_rgbs.1.conv.weight_orig torch.Size([3, 512, 1, 1]
to_rgbs.2.conv.weight_orig torch.Size([3, 512, 1, 1]
to_rgbs.3.conv.weight_orig torch.Size([3, 512, 1, 1]
to_rgbs.4.conv.weight_orig torch.Size([3, 256, 1, 1]
to_rgbs.5.conv.weight_orig torch.Size([3, 128, 1, 1]
to_rgbs.6.conv.weight_orig torch.Size([3, 64, 1, 1]
to_rgbs.7.conv.weight_orig torch.Size([3, 32, 1, 1]
to_rgbs.8.conv.weight_orig torch.Size([3, 16, 1, 1]
```

Bibliography

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[2] D. O. Hebb, *The organization of behavior: a neuropsychological theory.* J. Wiley; Chapman & Hall, 1949.

[3] B. Widrow and M. E. Hoff, "Adaptive switching circuits," tech. rep., Stanford Univ Ca Stanford Electronics Labs, 1960.

[4] B. Widrow and M. E. Hoff, "Associative storage and retrieval of digital information in networks of adaptive "neurons"," in *Biological Prototypes and Synthetic Systems*, pp. 160–160, Springer, 1962.

[5] K. Nakano, "Associatron-a model of associative memory," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 3, pp. 380–388, 1972.

[6] T. Kohonen, "Correlation matrix memories," *IEEE transactions on computers*, vol. 100, no. 4, pp. 353–359, 1972.

[7] J. A. Anderson, "A simple neural network generating an interactive memory," *Mathematical biosciences*, vol. 14, no. 3-4, pp. 197–220, 1972.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[9] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4401–4410, 2019.

[10] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[11] Z. Liu, P. Luo, X. Wang, and X. Tang, "Large-scale celebfaces attributes (celeba) dataset," *Retrieved August*, vol. 15, p. 2018, 2018.

[12] A. Gatt, M. Tanti, A. Muscat, P. Paggio, R. A. Farrugia, C. Borg, K. P. Camilleri, M. Rosner, and L. Van der Plas, "Face2text: collecting an annotated image description corpus for the generation of rich face descriptions," *arXiv preprint arXiv:1803.03827*, 2018.

[13] Y. Feng, F. Wu, X. Shao, Y. Wang, and X. Zhou, "Joint 3d face reconstruction and dense alignment with position map regression network," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 534–551, 2018.

[14] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

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

[16] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[17] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in neural information processing systems*, pp. 5767–5777, 2017.

[18] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled generative adversarial networks," *arXiv preprint arXiv:1611.02163*, 2016.

[19] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[20] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," *arXiv preprint arXiv:1605.05396*, 2016.

[21] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 5907–5915, 2017.

[22] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in neural information processing systems*, pp. 2234–2242, 2016.

[23] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in neural information processing systems*, pp. 6626–6637, 2017.

[24] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[25] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[26] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[27] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[28] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

[29] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *arXiv preprint arXiv:1802.05365*, 2018.

[30] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1501–1510, 2017.

[31] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[32] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016.

[33] A. Dosovitskiy and T. Brox, "Inverting visual representations with convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4829–4837, 2016.

[34] X. Zhu, Z. Lei, X. Liu, H. Shi, and S. Z. Li, "Face alignment across large poses: A 3d solution," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 146–155, 2016.

[35] Y. Liu, A. Jourabloo, W. Ren, and X. Liu, "Dense face alignment," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 1619–1628, 2017.

[36] P. Dou, S. K. Shah, and I. A. Kakadiaris, "End-to-end 3d face reconstruction with deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5908–5917, 2017.

[37] V. Blanz and T. Vetter, "A morphable model for the synthesis of 3d faces," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 187–194, 1999.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[39] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[40] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 586–595, 2018.

[41] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size,” *arXiv preprint arXiv:1602.07360*, 2016.

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[43] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[44] Akanimax, “Akanimax.t2f: text to face generation using deep learning,” *GitHub repository, https://github.com/akanimax/T2F*, 2018.

[45] O. R. Nasir, S. K. Jha, M. S. Grover, Y. Yu, A. Kumar, and R. R. Shah, “Text2facegan: Face generation from fine grained textual descriptions,” in *2019 IEEE Fifth International Conference on Multimedia Big Data (BigMM)*, pp. 58–67, IEEE, 2019.

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

[47] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, “Caltech-UCSD Birds 200,” Tech. Rep. CNS-TR-2010-001, California Institute of Technology, 2010.

[48] M.-E. Nilsback and A. Zisserman, “Automated flower classification over a large number of classes,” in *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

[49] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas, “Stackgan++: Realistic image synthesis with stacked generative adversarial networks,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 8, pp. 1947–1962, 2018.

[50] T. Xu, P. Zhang, Q. Huang, H. Zhang, Z. Gan, X. Huang, and X. He, “Attngan: Fine-grained text to image generation with attentional generative adversarial networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1316–1324, 2018.

[51] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[52] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[53] T. Hinz, S. Heinrich, and S. Wermter, "Semantic object accuracy for generative text-to-image synthesis," *arXiv preprint arXiv:1910.13321*, 2019.

[54] T. Hinz, S. Heinrich, and S. Wermter, "Generating multiple objects at spatially distinct locations," *arXiv preprint arXiv:1901.00686*, 2019.

[55] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling," in *Advances in neural information processing systems*, pp. 82–90, 2016.

[56] K. Chen, C. B. Choy, M. Savva, A. X. Chang, T. Funkhouser, and S. Savarese, "Text2shape: Generating shapes from natural language by learning joint embeddings," in *Asian Conference on Computer Vision*, pp. 100–116, Springer, 2018.

[57] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[58] K. Fukamizu, M. Kondo, and R. Sakamoto, "Generation high resolution 3d model from natural language by generative adversarial network," *arXiv preprint arXiv:1901.07165*, 2019.

[59] S. Marcel and Y. Rodriguez, "Torchvision the machine-vision package of torch," in *Proceedings of the 18th ACM international conference on Multimedia*, pp. 1485–1488, 2010.

[60] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao, and C. C. Loy, "Esrgan: Enhanced super-resolution generative adversarial networks," in *The European Conference on Computer Vision Workshops (ECCVW)*, September 2018.

[61] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[62] M. Kettunen, E. Härkönen, and J. Lehtinen, "E-lpips: Robust perceptual image similarity via random transformation ensembles," *arXiv preprint arXiv:1906.03973*, 2019.