



Technische Universität München

Master's Thesis

INSTITUTE FOR HUMAN-MACHINE COMMUNICATION
TECHNISCHE UNIVERSITÄT MÜNCHEN
Univ.-Prof. Dr.-Ing. habil. G. Rigoll

Conditional Face Image Generation and Its Application in 3d Pedestrian Modelling

Arka Bhowmick

Advisor: Stefan Hörmann, M.Sc.

Started on: 01.09.2019
Handed in on: 31.03.2020

Abstract

Generative adversarial network (GAN), an aspiring area in the field of deep learning, has made it possible to generate realistic images of human faces, change the style of an image and generate voice and texts. Generation of realistic images using GANs is an interesting research area and in this thesis that is the main concern.

Textures are an important part in creating 3D models and face textures are integral in creating 3D human models. Generating face textures from 3D scans of faces is already a known method. But due to its complexity, an endeavour is made in this thesis to leverage the state of the art generative network to generate good quality textures that can be efficiently fitted on an available 3D face model. Developing a method to perform selective modifications on generated face textures and developing an efficient method of fitting a texture to a 3D model is discussed in this thesis as well.

As already mentioned generation of textures is not the only concern here. To assure the diversity of the generated images a face attribute predictor is trained that assures the diversity of generated images. It is shown that method like regression can prove very efficient in identifying and changing specific facial features without changing any other facial properties. Two new indices, the face index and intercanthal index, are proposed that helps in an accurate geometrical analysis of the generated faces and textures. Post texture generation, it is shown how efficiently the textures can be fitted on a 3D model even without any UV map (2D mesh of a 3D model) information (like the different co-ordinates of the eyes, lips, nose and other facial features). It is shown how a simple and efficient Support Vector Machine (SVM) technique can be used to align and fit a 2D texture onto a 3D model efficiently. In this thesis, effect of texture augmentations in training generative networks is also discussed.

Index Terms- generative adversarial networks, 2D face textures, texture modification, UV mapping, texture rectification.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Problem Description	2
1.3	Contribution	2
1.4	Structure	3
2	Background	5
2.1	Linear Regression	5
2.2	Support Vector Machine	6
2.3	Convolutional Neural Networks (CNN)	7
2.3.1	Convolution Operation	8
2.3.2	Non-linearity /Activation Function	9
2.3.3	Pooling or Subsampling operation	10
2.3.4	Fully Connected Layer	10
2.3.5	Training the network using backpropagation	10
2.4	General Overview of Generative models	11
2.5	Generative Networks	13
2.5.1	Generative Adversarial Network (GAN)	13
2.6	StyleGAN architecture	15
2.6.1	State of the Art GAN Model (StyleGAN)	15
2.7	Face Landmark Detection	18
2.8	3D model and UV mapping	19
3	Related Work	21
3.1	Generative Adversarial Networks	21
3.2	Evaluating GAN output	23
3.3	Image Data Modification	24
3.4	Face Texture Generation	25

4	Methodology	27
4.1	Dataset Preparation	27
4.1.1	CelebFaces Attributes Dataset (CelebA)	27
4.1.2	Texture Dataset	28
4.2	Training	29
4.2.1	Effect of image augmentation during training	31
4.3	Degree of Realism Assessment	32
4.4	Modification of Generated Images (Faces and Textures)	35
4.4.1	Style mixing using the StyleGAN model	36
4.4.2	Modification using Regression	36
4.5	Best Fit Texture Resolution	40
4.6	Rectifying Bad Fit Textures	42
5	Results and Discussion	47
5.1	Training of the StyleGAN architecture	47
5.1.1	Evaluating Training progress using FID score	47
5.1.2	Degree of Realism for generated faces	48
5.2	Variational Analysis	50
5.3	Finetuning with Texture	51
5.4	Modification of Generated Textures	54
5.4.1	Style mixing implementation for textures	54
5.4.2	Modification of Images using Regression Analysis	56
5.5	Determining the best fit texture for the 3D face model	61
5.6	Rectification of distorted texture	65
6	Conclusion and Outlook	67
6.1	Conclusion	67
6.2	Future Work	68
	References	69

Introduction

1.1 Motivation

Synthetic data is a crucial asset in the field of machine learning. In the field of autonomous driving the use of synthetic or artificial data is significant. An autonomous driving system needs to be trained using a simulation environment because using real data is not always a practical solution. The simulation environment consists of buildings, cars, signal posts, pedestrians, etc. In order to train a system to be robust and reliable, it needs to be trained with a lot of variations, so that the system is prepared to face any kind of situation. One of the important components in the simulation environment are the pedestrians and creating these pedestrians involves 3D models and 2D textures [Figure: 1.1(a)].

Creating 2D textures e.g. for faces using the conventional methods of obtaining 3D scans and extracting texture using some complex mathematical process leads to inferior quality textures [Figure: 1.1(b)]. And in order to bring variations there should be more 3D scans and hence the entire process is inefficient and time consuming. Textures are also manually designed by designers. Although these manually designed textures are of good quality, it is still very time consuming.

Based on the above-mentioned facts an endeavour is made in this thesis to automate this texture generation process for pedestrian modelling in simulation environments. By automating this process more high-quality textures with variations can be generated in significantly less amount of time, than that could be done using the conventional methods.

This approach might also prove beneficial for designing computer games. The use of human figures in computer games is extensive and if this texture generation can be automated then it will be possible to generate different figures in significantly less amount of time. Another important application is training of face recognition systems. Generating face textures with a lot of variations which could then be wrapped around a 3D model to create a face [Figure: 1.1(a)], will help in training the face recognition system, making the surveillance system more robust and reliable and will simultaneously reduce the effort of collecting photographs of different faces required for the training.

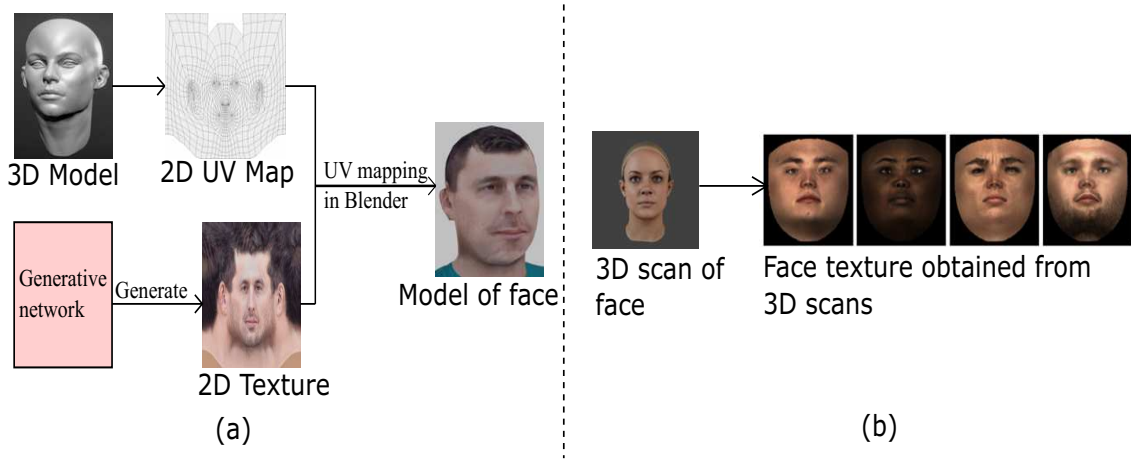


Figure 1.1: (a) Creating a face model using 3D model and 2D texture generated using a generative network; (b) 3D scan and textures generated from 3D scans.

Automating the generation of textures is possible leveraging the recent development in the field of deep learning in the form of generative networks (explained in Background chapter). With the help of generative networks, it is possible to create high quality textures with substantial amount of variations and that too without obtaining any 3D scans or doing any manual design.

Hence, this thesis provides a proof of concept on how this generation, creating variations and fitting of 2D textures could be automated.

1.2 Problem Description

The basic idea is demonstrated in figure 1.1(a) where a model of a face is created using a 3D model and 2D texture. The purpose of the thesis is generation of 2D textures for 3D modelling of the pedestrian models (human figures) using deep learning techniques. Here the main concern has been the generation and fitting of face textures. This involves the 2D face texture generation, performing quality checks to assess the realism of the generated textures, making sure that the generated textures are quite diverse in nature, wrapping of the 2D face textures on a 3D face model, assessing the quality of the face after the 2D texture being wrapped on it and in case of distortions after the wrapping step, finding a solution for it. The figure 1.2 gives an overview for the workflow of the thesis.

1.3 Contribution

The main contributions in the thesis can be summarized in the following points:

- The effect of data augmentation while training GAN architecture is analysed.

- Variational analysis is performed on the generated textures to make sure the generated textures are diverse in nature.
- Degree of realism is calculated for the assessment of the generated textures. Two new metrics are introduced apart from the conventional Frechet Inception Distance (FID) score. The two new metrics are InterCanthal Index and Face Index which are described in the *Methodology* chapter.
- StyleGAN's style-mixing method is implemented on the generated 2D face textures to produce random variations on the textures.
- The concept of linear regression and vector orthogonalization is used to implement conditional variation on the generated textures.
- The generated 2D face textures are then fitted on the 3D model using the concept of UV mapping.
- The texture wrapped 3D model is analysed for distortions or irregularities.
- Experiments are performed to separate bad textures (those that causes distortions after wrapping on the 3D model) and acceptable textures.
- Support Vector machine (SVM) is implemented to classify acceptable textures and bad textures. And last but not the least, the bad textures are converted to the acceptable ones leveraging the SVM implementation that is described in details in the *Methodology* chapter.

1.4 Structure

The thesis is divided into the following sections:

- The *Background* chapter explains the different concepts that are used for the successful completion of the thesis. The topics that are discussed are *Linear Regression*, *Support Vector Machine*, *Convolutional Neural Networks (CNN)*, *Generative Networks*, *Support Vector Machines (SVM)*, and last but not the least *UV Mapping*.
- The *Related Work* chapter tries to give an overview of the different researches that have been done, which are relevant to the thesis. Related works in the field of GAN architecture, texture generation and generated image modifications.
- The *Methodology* chapter describes the main contributions of the thesis in details, the ones that are being described in the *Contribution* section above.
- The *Results and Discussion* chapter summarizes the different evaluations that has been performed during the thesis, like the variational analysis of the generated textures, degree of realism assessment, effect of data augmentation during GAN training, etc. The above mentioned topics in the *Contribution* section is actually divided between the *Methodology* and *Results and Discussion* chapter, where the *Methodology* chapter describes the methods that have been implemented and the *Results and Discussion* chapter summarizes the different outcomes of the methods that are being implemented. It keeps the thesis structure clean and tidy.
- Last but not the least the *Conclusion* chapter gives an overview of the entire thesis and the final outcome and also throws some light on the possible future areas of

research building on this work.

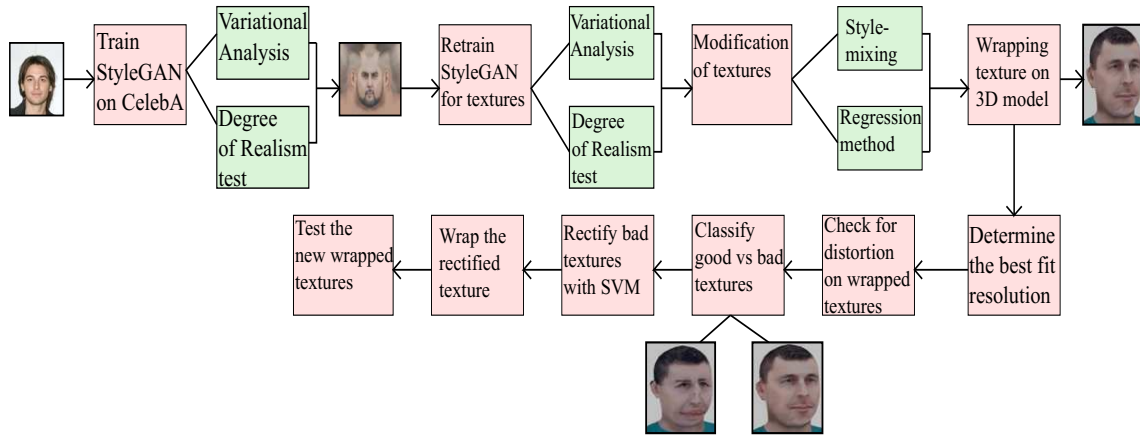


Figure 1.2: The workflow of the thesis

Background

The purpose of this chapter is to explain the different concepts that is used throughout this thesis, for better understanding of this thesis. This chapter is divided into six parts.

- Section 2.1 explains Linear Regression, which is basic machine learning.
- In section 2.2 the concept of Support Vector Machine is explained.
- Section 2.3 explains Convolutional Neural Networks (CNN) which is an itergral part of deep learning.
- Section 2.4 and 2.5 explains Generative Networks in General that is the building block of the thesis.
- Section 2.6 describes the State of the art architecture for GAN i.e. the StyleGAN architecture.
- In section 2.7 face landmark detection is discussed using dlib face landmark detector.
- Section 2.8 explains the process of fitting of a 2D texture on a 3D model.

2.1 Linear Regression

One of the very basic concept that is used in the thesis is Linear Regression for the conditional modification of fake images, which is explained in the *Own Work* chapter.

Linear regression is a linear approach that models the relationship between dependent and independent variables. Given a data $y_i, x_{i1}, \dots, x_{ip}$, where y is the dependent variable and x_{i1}, \dots, x_{ip} is a p dimensional vector, which is the independent variable. The variable X which is a p dimensional vector is called the regressor variable. The model takes the form:

$$y_i = b_0 + b_1x_{i1} + \dots + b_px_{ip} = \mathbf{X_i^T b} \quad (2.1)$$

Here,

- y is response vector of length n .
- X is an $n \times (p + 1)$ matrix of independent variables.
- b is a $(p + 1)$ vector of regression coefficients.

This kind of linear regression i.e. represented by [Eq: 2.1], is known as *multiple linear regression*, also known as *multivariate linear regression*.

The image [Figure: 2.1] gives a geometrical interpretation of Linear Regression. The basic idea of Linear Regression is about fitting a straight line through the data-points.

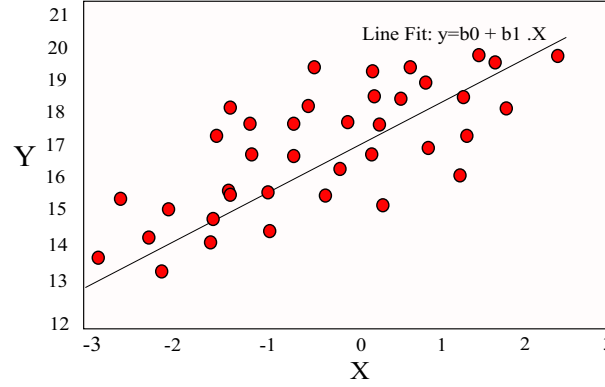


Figure 2.1: Scatter plot for linear regression. The best values of b approximates the best fit line.

2.2 Support Vector Machine

A Support Vector Machine (SVM) is a discriminative classifier that is defined by separating boundary or hyperplane. Given a labelled training data, SVM outputs a boundary, which is $H0$ in figure 2.2, which categorizes new examples. The name support vector machine comes from the support vectors that lie closest to the decision surface (or hyperplane), the red and the green dots that lie on the lines $H1$ and $H2$ [Figure: 2.2] are the support vectors and these are the data-points that are most difficult to classify. Another interpretation for SVM can be that it maximizes the margin around the separating hyperplane. The d in figure 2.2 is the margin of separation, i.e. the separation between the hyperplane and the support vector for a given weight vector w and bias b . The input to the SVM is a set of input/output training pair samples (the input sample features x_1, x_2, \dots, x_3 and the output result y). The SVM outputs a set of weights w , one for each feature, whose linear combination predicts the value of y . From figure 2.2 the separating hyperplane is $H0$ defined by the equation $w.x + b = 0$, where w is the weight vector, x is the input vector and b is the bias. In order to maximize the margin, i.e. the distance $2d$, $\|w\|$ has to be minimized with the condition that there are no datapoints between $H1$ and $H2$ [Equation: 2.2, 2.3]:

$$x_i.w + b \geq +1 \text{ when } y_i = +1 \quad (2.2)$$

$$x_i.w + b \leq -1 \text{ when } y_i = -1 \quad (2.3)$$

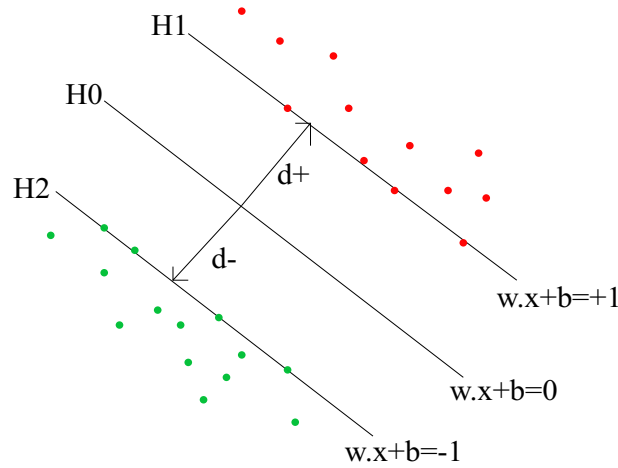


Figure 2.2: Support Vector Machine overview

2.3 Convolutional Neural Networks (CNN)

A CNN is a deep learning algorithm that takes in an input image and assign various aspects/objects in the image and is capable of differentiating an image based on some predefined classes. A CNN gets its name from the type of hidden layers present in it. A CNN consists of *convolutional layers*, *pooling layers*, *fully connected layers*, and *normalization layers*.

To mention a little bit of history, the very first CNN architecture was LeNet, which was a pioneering work by Yann LeCun (1988) [LBBH98], which helped to gear up the field of deep learning. Below is an image that gives an intuitive idea about how the CNN architecture works.

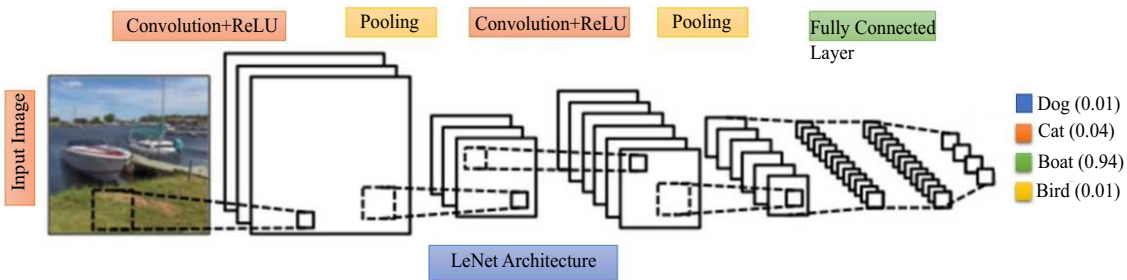


Figure 2.3: A simple CNN architecture [LBBH98]

This CNN architecture[Fig: 2.3] classifies the image into 4 classes namely the dog, cat, boat and bird. It is evident from the picture that boat gets the highest probability. Though just saying it seems to be a lot more intuitive, there is some serious mathematics that goes behind such architectures. There are 4 main operations that are being carried out in the CNN architecture:

1. Convolution.

2. Background

2. Activation Function (Non Linearity like ReLU).
3. Pooling or Subsampling.
4. Classification (Fully Connected Layer)

These operations are basic to all classes of neural networks and hence understanding them is key to understanding the Thesis properly.

Now we all know that image is a matrix of pixel values. **Channel** of a image is referred to as the number of colour elements that the image has. An image from a standard digital camera will have 3 channels namely Red, Green and Blue. A **gray scale** image is an image with only a single colour channel. The pixel values are between 0 and 255. Hence these channels are also known as the 8 bit colour channels.

2.3.1 Convolution Operation

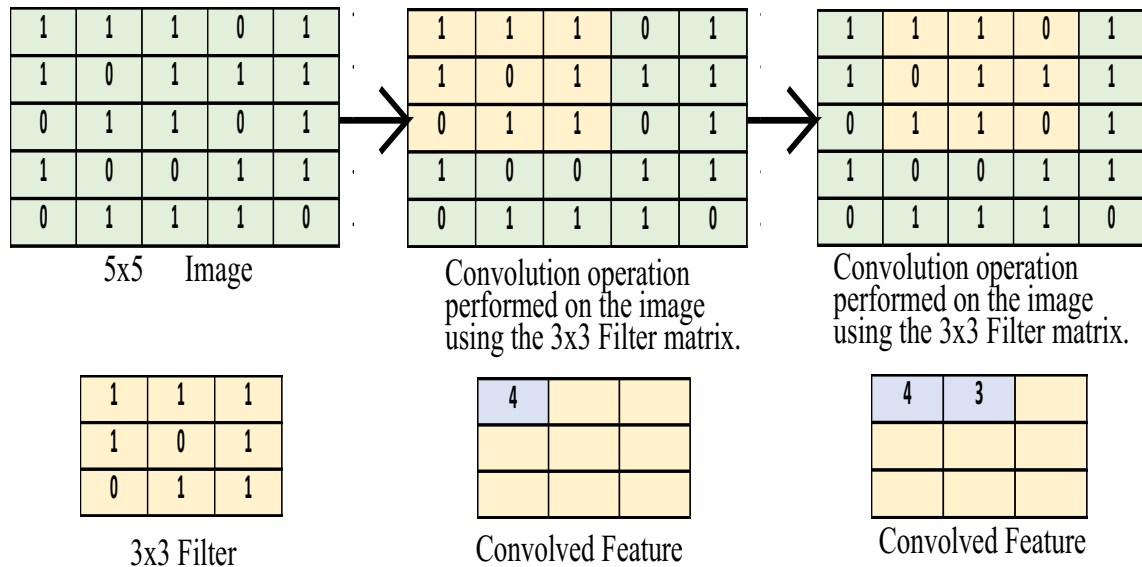


Figure 2.4: (Left to right) Explanation of the convolution operation.

ConvNets (convolution network) derive their name from convolutional operator. The basic purpose of the convolution is extracting features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

As it is already mentioned that image is a matrix values, let us take an image matrix of 5x5 image pixel values are only 0 and 1 (first figure from top left) [Figure: 2.4].

The small 3x3 matrix (first figure from bottom left) [Figure: 2.4] is also called the **Kernel or Filter**. The filter is slid over the 5x5 image matrix and multiplication is performed with the elements in the 3x3 matrix with elements of the 5x5 image matrix, over the block on which the filter is superimposed as marked by the yellow block (top middle from left) [Figure: 2.4]. The multiplication is followed by addition of the results

of multiplication of each individual cell in the block. Hence convolution is a purely linear operation. The convolution operation can be summarized by the following equation:

$$G[m, n] = (f * h)[m, n] = \sum_j \sum_k h[j, k] f[m - j, n - k] \quad (2.4)$$

here f is the input image and h is the kernel. The indices of the rows and columns of the resultant matrix are marked with m and n respectively. Here in figure 2.4, it is evident that the kernel has moved one distance, and this is known as the **stride**. Here stride is 1. The matrix formed after convolution is known as the **Feature Map** (bottom middle figure from left)[Figure: 2.4]. The number of feature maps obtained depends on the number of filters used for convolution. It is important to mention the filters acts as feature detectors for the original input image. The spatial size of the output volume of the feature maps can be calculated by the following formula:

$$Feature\ map\ volume = \frac{W - K + 2p}{S} + 1 \quad (2.5)$$

where W is the input feature (image) volume size (e.g. $W = 5$ from figure 2.4), the kernel field size of the convolutional layer neurons is denoted by K , S denotes the stride that is applied (in the above case 1) and the amount of zero padding if used on the border is denoted by p .

2.3.2 Non-linearity / Activation Function

The output of a neuron is basically a liner operation that can be summarized by the formula $Output = (weight \cdot input) + constant$, where $Output$ is the neuron output, $input$ is the input to the neuron, $weight$ defines the strength of the connection between input and output neuron and the $constant$ is also known as the bias term in machine learning. The output can range between $-\infty$ to $+\infty$. The neuron is unaware of the bounds of the value. Hence in order to decide whether the neuron will be activated or not, the purpose of the activation function comes into play, where the output of the neuron is bound between certain values and at the same time the activation function introduces non-linearity to the model. There are many different kinds of activation functions like the *Step*, *Sigmoid*, *Tanh*, *ReLU*, etc. Among them ReLU activation function is one of the most commonly used.

The ReLU or Rectified Linear Unit[Aga18] is an activation function. Since most of the data in real world is non linear in nature, it is the job of the activation function to introduce non-linearity in the model, otherwise it is just a simple linear operation (i.e convolution). ReLU can be described as $A(x) = \max(0, x)$, where A defines the ReLU operation and x is the neuron output. The ReLU is not bound, though it is a good approximator. The range of ReLU is $[0, \infty]$. ReLU is an element wise operation and removes all the non zero values from the convoluted matrix or feature map and in doing so technically it removes sharp edges from the convoluted image.

2.3.3 Pooling or Subsampling operation

Pooling is also known as subsampling/downsampling or spatial pooling. Spatial pooling can be different types: max, average, etc.

In case of max pooling a spatial neighbourhood (e.g 2x2) window is defined and the largest element from the rectified feature map within that window is taken. Instead of taking the largest element, the average of all the elements in that window can be considered, which in that case will be known as average pooling. An example of max pooling operation with a spatial window of 2x2 on a rectified feature map is shown below:

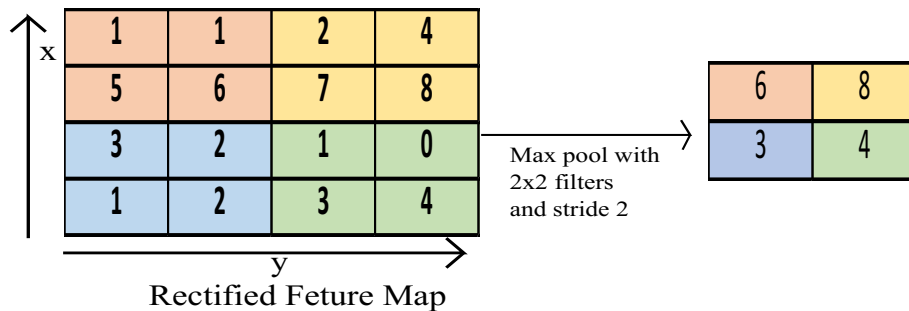


Figure 2.5: Max Pooling operation

We slide the window 2x2 by 2 cells and take the maximum value in each region. The Max Pooling serves some important functions:

1. As it is evident from figure 2.5 that it reduces the dimensionality of the rectified feature map. Hence it makes input more manageable.
2. Reduces the number of parameters and computations and in a way controls overfitting.
3. Makes the network invariant to small transformations and distortions.
4. Makes the network scale invariant. This is very useful because in real life we may have the image of same thing but with different scales.

2.3.4 Fully Connected Layer

The Fully Connected layer is the classical neural network architecture, in which all neurons connect to all neurons in the next layer.

The rectified feature maps i.e the outputs from the Convolution and pooling layers serves as high-level features of the input image and the fully connected layer use these features for classifying the input image into various classes.

2.3.5 Training the network using backpropagation

The overall training process of the CNN can be summarized as follows:

- **Step 1:** First of all we initialize all the filters with random Gaussian values. It is advised that the initialization should not be done with all zeros.
- **Step 2:** The input to the network is training image of size $N \times N$, that goes through all the convolution, pooling and fully connected layers and finds out the output probabilities for each class. (E.g.: In our above example we have 4 classes and we can think of an output such as $[0.2, 0.3, 0.4, 0.1]$). Since the filter weights are randomly assigned, for the 1st training example, output probabilities are also random.
- **Step 3:** The error is calculated as follows:

$$Total\ Error(loss) = \sqrt{\frac{\sum_{i=1}^T (Predicted_{o/p} - Actual_{o/p})^2}{T}} \quad (2.6)$$

Here the mean squared error(MSE) loss has been used for the example where T is the number of classes. Here $Predicted_{o/p}$ is the output predicted or calculated by the network and $Actual_{o/p}$ is the actual reference output provided beforehand.

- **Step 4 :** The next step is to reduce the loss by performing optimization on the loss function. This is done using the **backpropagation method**. The backpropagation method is used to calculate the gradients of the loss function w.r.t the network parameters and then uses **gradient descent** to update all the parameter values to minimize the loss.
 - The weights are adjusted proportionally to their error contribution.
 - When the same input image is again fed to the network then output may become $[0.1, 0.1, 0.7, 0.1]$. This clearly shows that the output indicating to a particular class.
 - This proves that the network has learned to classify something. And this process continues until and unless we are satisfied with the loss minimization.

2.4 General Overview of Generative models

Generative models are a special branch of machine learning models that are used to generate data. In other words, generative models can generate data that we feed them as training data.

The different types of generative models are:

- **Autoencoders:** An autoencoder is a type of model that is used to learn low level representation or embeddings of the training data. It is generally used to mimic the input data to the output. The autoencoder mainly consists of 4 parts namely the **encoder**, **bottleneck**, **decoder** and **reconstruction loss**. The encoder model helps in learning the lower dimensional representation of the input data. The bottleneck is the layer that contains the compressed representation. The decoder

is the network that helps in reconstruction of the original data from the compressed one and the reconstruction loss is the method that gives the intuition of how well the autoencoder is performing. The main benefit of an autoencoder network is dimensionality reduction. There are different variations of autoencoders present like the variational autoencoders[KW19]. A variational autoencoder is a type of autoencoder whose training is regularized to avoid overfitting and to make sure that the latent space or the bottleneck layer representation has good properties that enable generative process which means new content can be generated which was not possible in case of a simple autoencoder.

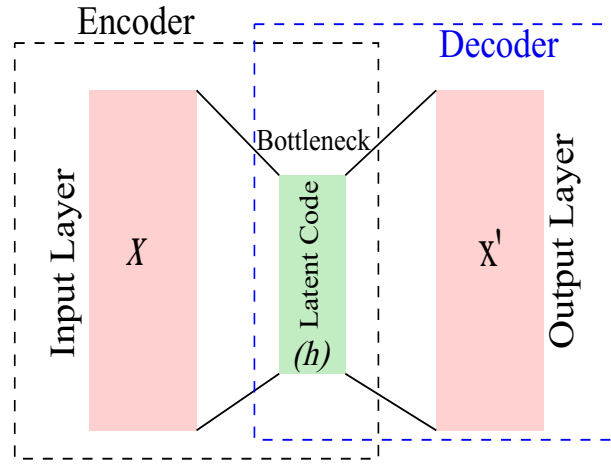


Figure 2.6: Schema of basic Autoencoder. The Input Layer represents training data. *Latent Code* represents the intermediate vector or the embedding of the input X . The Encoder network maps the input x to intermediate state *Latent Code* and the Decoder network maps *Latent Code* to X' .

- **Generative Adversarial Networks (GANs):** The GAN architecture has a generator model $q(x/h)$ to map the intermediate state of h represented in figure 2.6 to the input space X . And there is a discriminative model $p(y/x)$, which tries to associate an input instance x to a yes/no binary answer y , about whether the generator model generated the input or was a genuine sample from the dataset we were training on. Some of the notable GAN architectures are the DCGAN[rad], PROGAN[KALL17], StyleGAN[KLA18]. The workflow is represented by figure 2.7.
- **Auto-regressive models (AR):** Autoregressive models predicts the future behaviour based on the past behaviour. Autoregressive models like **PixelRNN**[vdOKK16], trains a network that models the conditional distribution of each and every individual pixel, given the previous pixel. In AR model, the value of the outcome variable Y at some instance of time t is directly related to the predictor variable X . The difference between simple linear regression and AR models is that Y is dependent on X and previous values for Y .

2.5 Generative Networks

After the elaborate introduction of CNN, this section will generally involve the explanation of Generative Networks. Till date there has been tremendous improvement in the field of discriminative models, that generally map high dimensional inputs to class labels. These successes is mainly based on the backpropagation and dropout algorithms, using piece-wise linear units which have a particularly well behaved gradient. Deep generative networks have had a less impact, because of the difficulty of approximating many intractable probabilistic computations that arise in maximum likelihood estimation and related strategies, and due to the difficulty of leveraging the benefits of piece-wise linear units in the generative context. This problem was addressed by Ian Goodfellow and his team in the year 2014, by the introduction of Generative Adversarial Networks[GPAM⁺14].

2.5.1 Generative Adversarial Network (GAN)

Generative vs. Discriminative Algorithms

To understand GAN architecture it is very important to understand the difference between Generative and Discriminative algorithms.

Discriminative algorithms tries to map a high dimensional input to a class. Hence basically it is solving a classification problem. In mathematical terms it can be represented as $p(y|x)$. Here y is the class variable and x is the feature variable. For example given the properties of an email what is the probability that the email is spam or not.

The question that the generative model tries to answer, is that assuming the email is spam, how likely are the features? While discriminative models care about the relation between x and y , generative models care about how to get x .

The GAN Framework

In the GAN architecture there are two CNNs competing against one another. One is known as the **generator** and the other is known as the **discriminator**. In simple language, the generator generates new data instances while the discriminator evaluates them for authenticity. The discriminator decides whether the data instances fed to it is sampled from the real data or generated by the generator.

The discriminator and generator are represented by 2 functions (in this case neural networks) that are both differentiable w.r.t their parameters and input. The discriminator takes input x (from the real data) and it's parameters are Θ^D . The generator is defined as G and it takes input z (array of random numbers) and it takes Θ^G as it's parameters.

Both D the discriminator and G the generator have cost functions that are defined w.r.t their parameters. The D minimizes $J^D(\Theta^G, \Theta^D)$ by controlling Θ^D and the G minimizes $J^G(\Theta^G, \Theta^D)$ by controlling only Θ^G . Here $J()$ is the cost function or loss function used to optimize the generator and the discriminator network.

2. Background

As the cost functions depends on each others parameters, this scenario represents more of a game than a optimization problem. The solution of a game is finding the NASH equilibrium, which is a tuple (Θ^D, Θ^G) , where Θ^G is the local minimum for G parameters and Θ^D is the local minimum for D parameters.

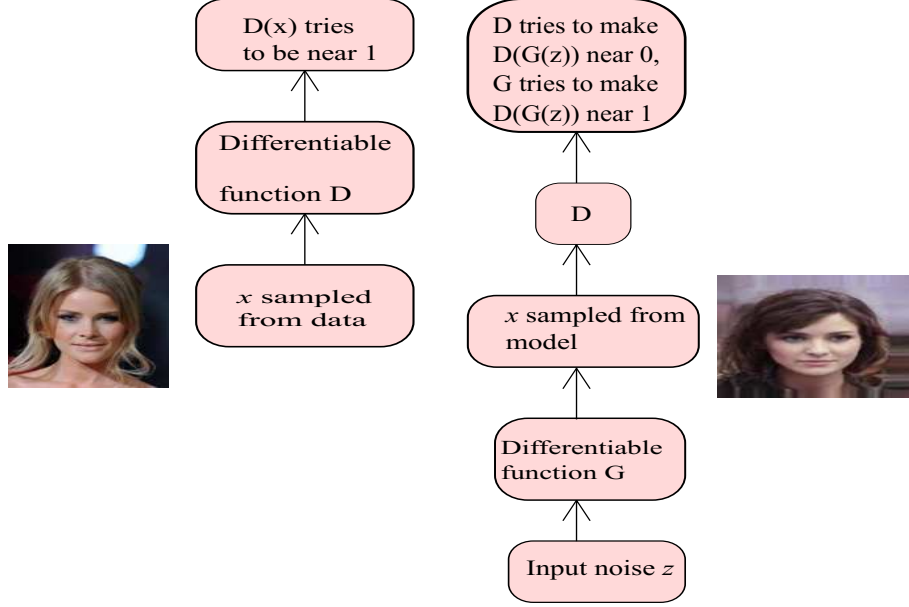


Figure 2.7: The GAN framework represents two adversaries namely Discriminator (D) and Generator(G). Typically D and G is represented by deep neural networks. There are 2 scenarios in this game play. Firstly the D is fed with real training sample x and D returns the probability that it's output is real rather than fake. The goal of D is to make $D(x)$ close to 1. Secondly, G is fed an array of random numbers z that may be sampled from a Gaussian distribution. D is then fed with $G(z)$ and here both D and G works together. D tries to make $D(G(z))$ close to 0 while G tries to make the same quantity close to 1.

Cost function: D and G play the following two-player minimax game with value function $V(G, D)$:

$$\min_G \max_D V(G, D) = \sum_{x \sim p_{data}(x)} [\log D(x)] + \sum_{z \sim p_z(z)} [\log(1 - D(G(z)))] \quad (2.7)$$

Here $p_{data}(x)$ is the data distribution from which the original data is sampled and $p_z(z)$ is the distribution from which the latent vector (noise) is sampled that is fed to the generator G . x is the original data or the image in this case and z is the latent or noise vector that is fed to the generator.

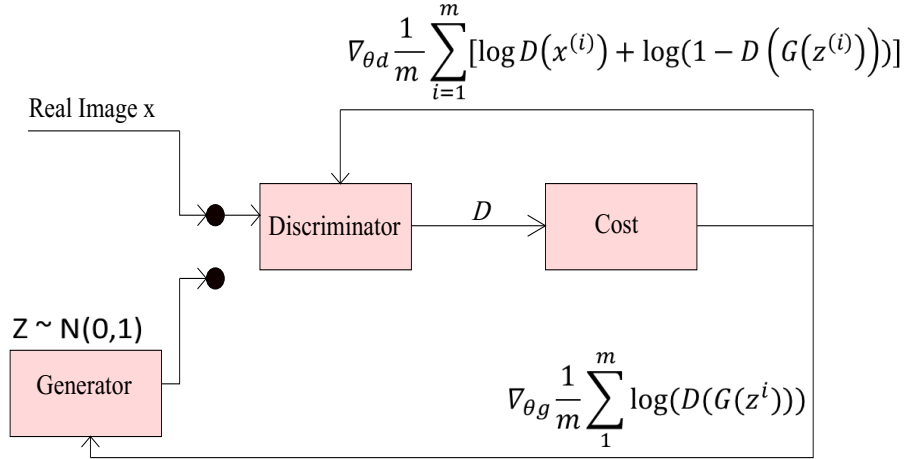


Figure 2.8: GAN loss function [GPAM⁺14]. $D()$ is the discriminator, $G()$ is the generator, m are the total number of samples, x and z is the real image and the latent vector respectively.

2.6 StyleGAN architecture

2.6.1 State of the Art GAN Model (StyleGAN)

The GAN model that is used for this thesis is the StyleGAN architecture[KLA18]. Until now most of the improvements to the GAN network has been done on Discriminator models. But the Style Generative Adversarial Network, or StyleGAN in short, is an extension to the GAN architecture that proposes a new generator architecture. The architecture includes a fully connected mapping network, to map the noise vector to an intermediate latent space. The intermediate latent space is used to control style at each point in the generator model, and the introduction to noise as a source of variation at each point in the generator model.

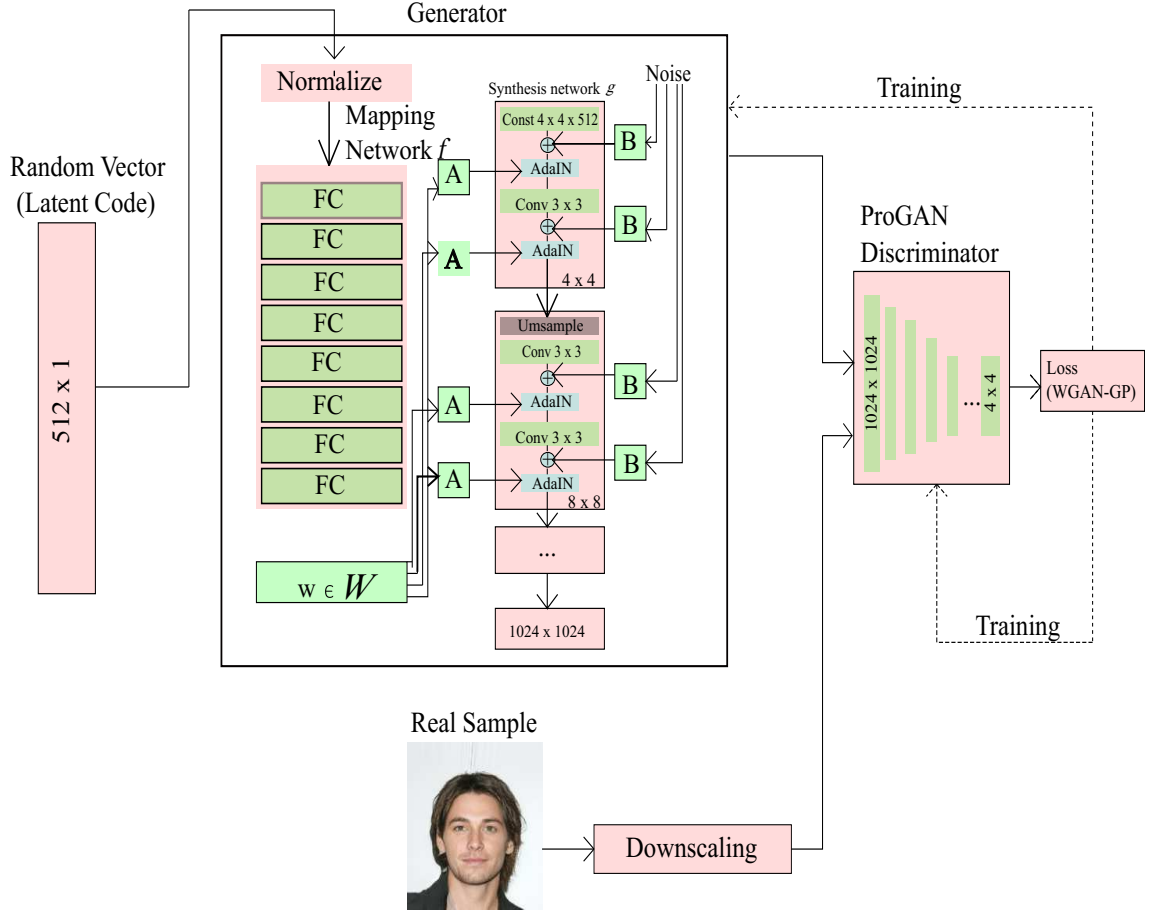


Figure 2.9: Overview of the StyleGAN network

Style-based generator

Traditionally the latent code i.e. the noise vector is directly applied to the input layer. In case of StyleGAN, the entire input layer is omitted and instead starts from a learned constant [Figure: 2.9]. Given a latent code z in the input latent space Z , a fully connected mapping network (non-linear in nature) $f : Z \rightarrow W$ first produces $w \in W$. The dimensionality of the latent code z and the dimensionality of the latent space w is the same. The mapping network f is 8-layer fully connected network.

The **A** represents *learned affine transformation*. The learned affine transformation transforms w to styles $y = (y_s, y_b)$, that controls the **AdaIN** operation [KLA18, HB17]. **AdaIN** refers to "Adaptive Instance Normalization" and it is defined as:

$$AdaIN(x_i, y) = y_{s,i} \frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i} \quad (2.8)$$

where x_i refers to feature map, $\mu(x_i)$ is the mean of x_i and $\sigma(x_i)$ is the standard deviation of x_i . x_i is normalized separately and then biased and scaled using the learned affine

transformations performed in block **A**. Hence the dimensionality of y is twice the number of feature maps on that layer.

Last but not the least, the generator is provided with explicit noise inputs, represented by block **B**. These are single-channel images consisting of uncorrelated Gaussian noise, and a dedicated noise image is fed to each layer of the synthesis network. The noise image is broadcasted to all the feature maps using learned per feature scaling factors and then added to the output of the corresponding convolution.

Properties of Style-based generator

- The generator architecture makes it possible to control the image synthesis via a scale-specific modifications to the styles.
- A concept of *mixing regularization* is employed where a given set of images is generated using two latent codes z_1 and z_2 and as a consequence there are w_1 and w_2 that control the styles. This technique prevents the network from assuming that adjacent styles are correlated.
- Stochasticity in an image refers to the placement of hairs, skin pores, etc. This stochasticity is achieved by adding per pixel noise after each convolution, that is evident from figure 2.9. In the figure that is represented by block **B**.
- The changes in stochastic variation does not have any global effects. The style affects the entire image. Global effects like pose, lighting, etc are completely controlled by the styles, whereas the noise is added independently of each other. If the network would have tried to control the global effects using the noise, then that would be penalized by the discriminator.
- One of the most important concept that has been implemented in the StyleGAN is the truncation trick in w . Here the generator truncates the intermediate vector w , to keep it close to the average intermediate vector to avoid generating poor images. The concept is quite simple, instead of using the intermediate vector directly, w is transformed to

$$w_{new} = w_{avg} + \psi(w - w_{avg}) \quad (2.9)$$

where ψ defines the standard deviation, i.e how far the generated image can be from the average image. w_{avg} is produced by selecting many inputs and then taking their intermediate vector using the mapping network and then taking their average.

Loss function

The loss function that is used for the purpose of StyleGAN is the Wasserstein Loss[ACB17]. The Wasserstein Distance (also known as the Earth Mover Distance) is actually the minimum cost of transporting mass to convert a data distribution q to p . The Wasserstein

2. Background

distance is defined as

$$W(P_r, P_\theta) = \sup \mathbb{E}_{x \sim P_r}[f(x)] - \mathbb{E}_{x \sim P_\theta}[f(x)] \quad (2.10)$$

where,

- f is 1-Lipschitz function, such that $|f(x_1) - f(x_2)| \leq |x_1 - x_2|$.
- $\|f\| \leq 1$
- $\sup - >$ Least upper bound.
- P_r and P_θ are two different probability distributions.

The figure 2.10 shows the WGAN loss function. Compared to the original GAN algorithm, the WGAN undertakes the following changes:

- After every gradient update on the Discriminator function, the weights are clamped to a small fixed range $[-c, c]$.
- Use of a new loss function derived from the Wasserstein distance where there is no more logarithm term. The discriminator model does not act as a critic anymore but acts as an helper for estimating the Wasserstein metric between real and generated data distribution.

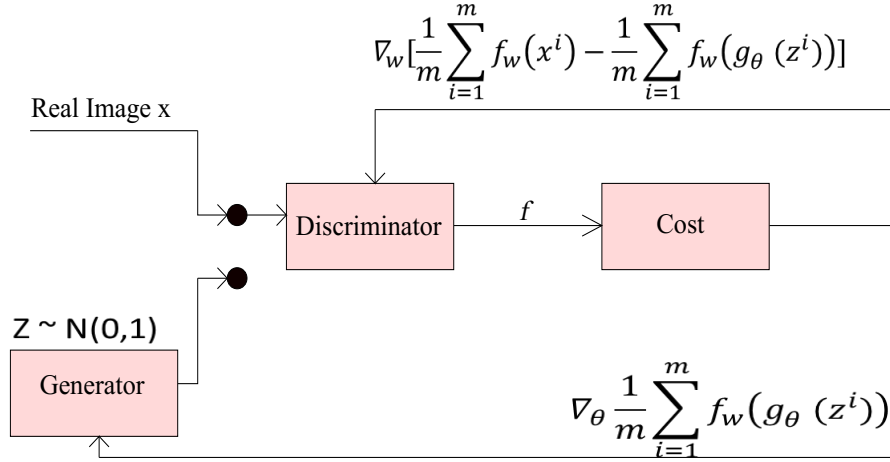


Figure 2.10: WGAN loss function[ACB17]

2.7 Face Landmark Detection

One of the popularly used face detector is the CNN based face detector from dlib. Dlib is a toolkit for making the real world machine learning and data analysis applications in C++. Although, it is written in C++, it has good, easy to use python bindings. The **facial landmark detection** is generally used for this thesis.

This detector is based on the **Histogram of oriented gradients** and **linear SVM**. While the HOG+SVM based face detector works perfectly fine, there is also CNN based

dlib face detector. The HOG based face detector in dlib is meant to be a good frontal face detector. Meanwhile the CNN based detector is capable of detecting faces in almost all angles. But CNN based detection is computationally expensive. Hence, for the purpose of this thesis the **HOG** based detector is used because it serves the purpose quite well. The image [Figure: 2.11] shows an example of the dlib face detector (HOG+SVM based) on a sample image from CelebA dataset. The *Annotated Image* gives the bounding box for the face including the 68 coordinates for the facial landmarks that are marked by red dots.

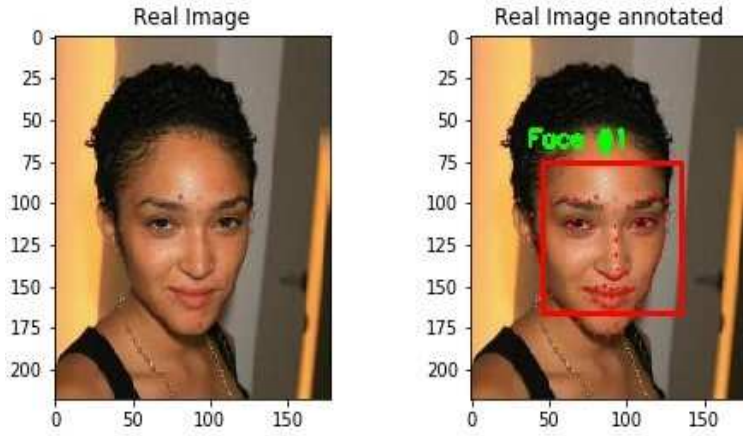


Figure 2.11: Dlib face detector applied on an Image

2.8 3D model and UV mapping

One important part of the thesis is to fit the generated face textures on a 3D model and determine what resolution of texture, fits best on the available 3D model. In order to fit the 2D texture into a 3D model is called **UV mapping**[HLS07]. UV mapping is the 3D modelling process of projecting a 2D image to a 3D model's surface for texture mapping. 'U' and 'V' represents the axes of the 2D texture because 'X', 'Y' and 'Z' are used to denote the coordinates of the 3D object. The figure 2.12 gives an overview of the UV mapping concept. A UV map consists of the 3D model's XYZ coordinates flattened into 2D UV space.

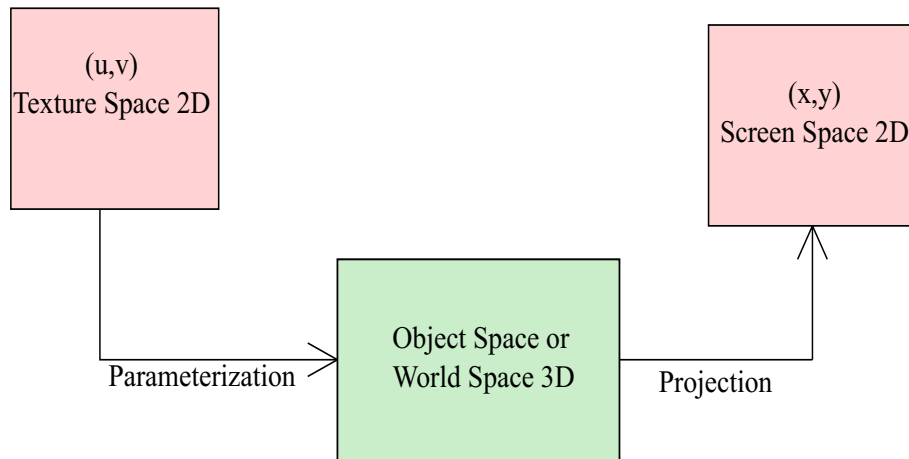


Figure 2.12: UV mapping overview

The figure 2.13 gives a pictorial representation of the UV mapping concept.

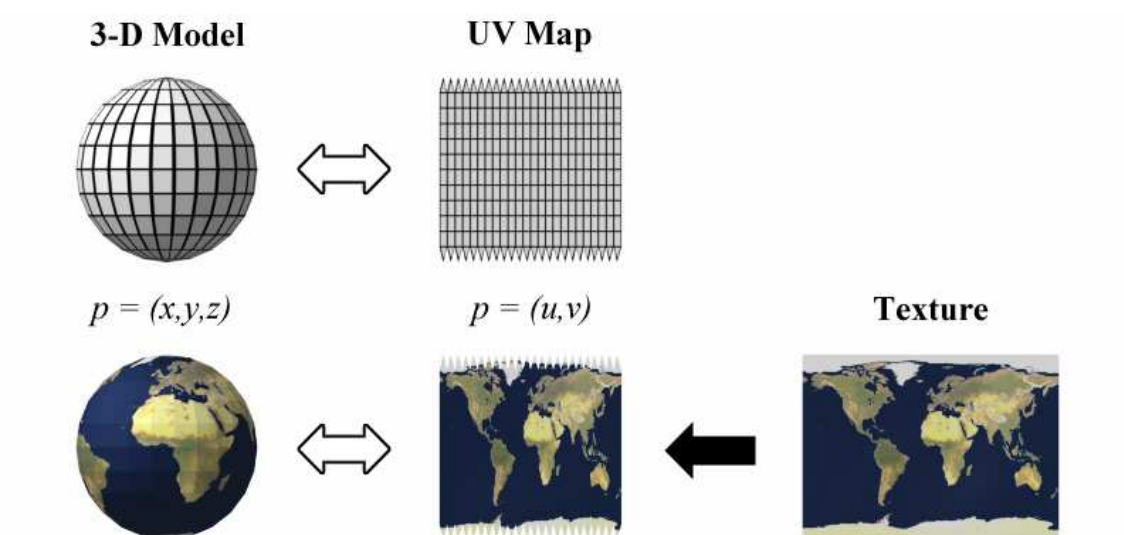


Figure 2.13: The application of a texture in the UV space related to the effect in 3D[HLS07].

Related Work

3.1 Generative Adversarial Networks

In the year 2014, Ian J. Goodfellow, for the first time proposed a new framework for estimating generative models via an adversarial process, in which the generative network G and the discriminative network D is simultaneously trained. The proposed adversarial model has a few advantages. Firstly Markov Chains are never needed, and only back-propagation is used to obtain the gradients, no inference is needed during learning, and moreover a wide variety of functions can be incorporated into the model. These aforementioned advantages are primarily computational. There is also some statistical advantage because the generator network is not directly updated with the data examples, it is updated only with gradients flowing through the discriminator. The other advantage of adversarial nets proposed by Goodfellow is that they can represent very sharp, even degenerate distributions.

There are few other GAN architectures that needs to be mentioned.

Architecture Variant GANs

Many types of GAN variants have been proposed in the architecture, which is summarized in figure 3.2 . These variants have a specific task to perform like image to image transfer, image super-resolution, image completion and text to image generation. The architectures mentioned helps in improving the performance of GANs w.r.t to different scenarios like image quality improvement, image variety improvement, stable training, etc.

Fully Connected GANs: The original GAN paper[GPAM⁺14] that has been mentioned above, uses fully connected layers for generator and discriminator. This architecture variant has been used to generate simple images of MNIST and CIFAR 10 and Toronto Face Dataset. It has not shown good generalization performance for complex image types.

Laplacian Pyramid of Adversarial Networks (LAPGAN): LAPGAN architecture [DCSF15] had been proposed to generate high resolution images, from lower resolution input. LAPGAN utilizes a cascade of CNNs within a Laplacian pyramid framework to generate high quality images.

Deep Convolutional GAN (DCGAN): It is one of the most important milestones in the field of GAN architectures. DCGAN[RMC] was the first architecture that applied a deconvolutional neural network architecture for generator. The DCGAN utilize the spatial up-sampling ability of the deconvolution operation for generator, which makes generation of higher resolution images possible. The figure 3.1 shows the architecture of the DCGAN. Some of the notable features of the DCGAN includes the use of transposed

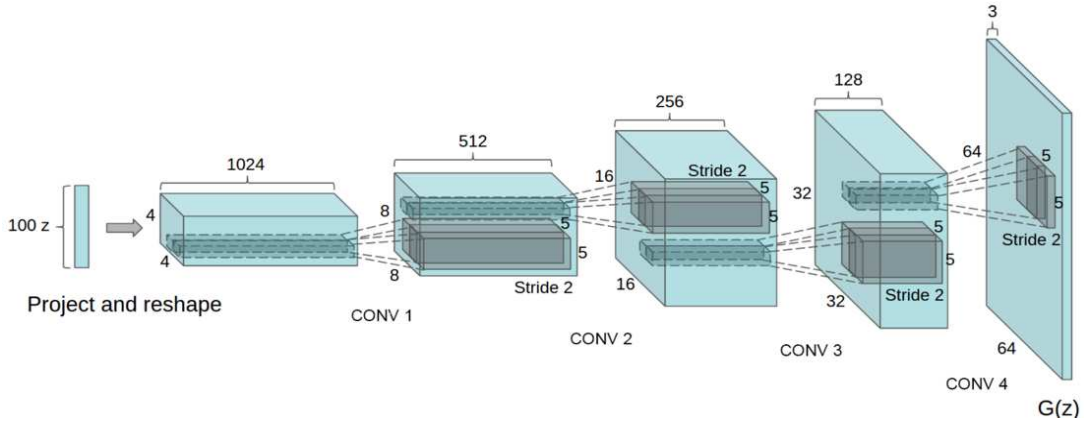


Figure 3.1: DCGAN generator. A 100 dimensional uniform distribution Z is projected to a small spatial content convolutional representation with many feature maps. A series of four fractionally-strided convolutions then convert this high level representation into a 64x64 pixel image [RMC]

convolution for unsampling and downsampling, use of batch normalization that stabilizes learning by normalizing input at each unit to have zero mean and unit variance and use of ReLU activation function in the generator and the use of Tanh function at the output layer of the generator.

Progressive GAN (PROGAN)[KALL17]: This architecture involves the progressive steps towards the expansion of network architecture. It uses the idea of progressive neural networks. This architecture has the advantage of not forgetting and can take advantage of the prior knowledge via lateral connections to previously learned features. Training starts with the low resolution 4x4 pixels images. The generator and the discriminator starts to grow with the training progressing. The progressive nature enables more stable learning. Current state of the art GAN models are based on this architecture, the most prominent example being the *StyleGAN* architecture[KLA18], which is the backbone of the thesis and has been describe in details in the *Background* chapter.

BigGAN: BigGAN[BDS18] has also achieved state of the art results. The BigGAN pulls together the recent best practises in training class-conditional images and scaling up the batch size and model parameters. It has proved that increasing the batch size and the complexity of the model can dramatically improve GAN’s performance.

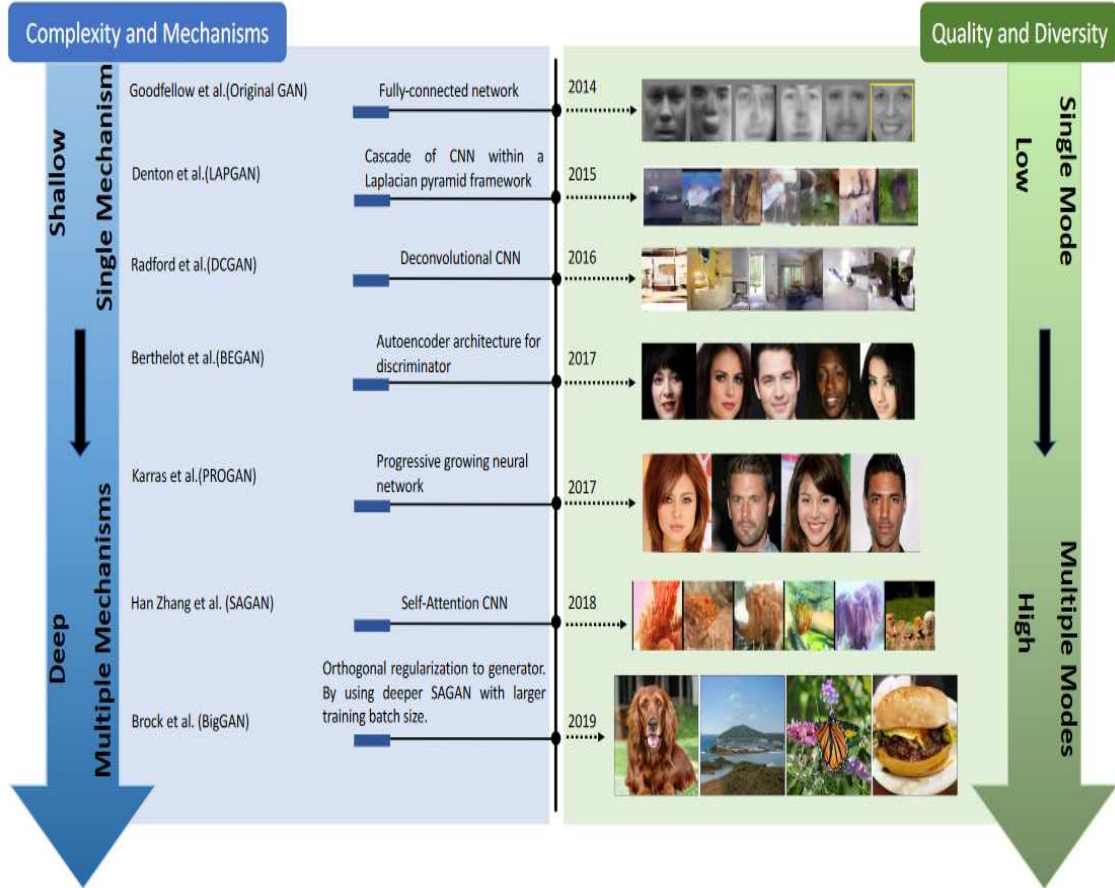


Figure 3.2: Timeline of architecture-variant GANs. Complexity in blue stream refers to size of the architecture and computational cost such as batch size. Mechanisms refer to the number of types of models used in the architecture (e.g., BEGAN uses an autoencoder architecture for its discriminator while a deconvolutional neural network is used for the generator. In this case, two mechanisms are used)

3.2 Evaluating GAN output

As the output of GAN is an image, it is reasonable to judge how realistic the images are. Substantial amount of work has been done in this field. Till date there is no overall metric to compare all GANs, but there are a few main ones that have been used to compare models with the same applications.

Image-to-image models have been evaluated using *FCN (Fully-Convolutional Network) score*. This method uses a classifier to perform semantic segmentation on the

output images, and then it is compared to a ground truth labelled image. For Image-to-Image translation that performs photo-to-label translation (CycleGAN[ZPIE17]) metrics such as *per pixel accuracy*, *per class accuracy*, and *mean class Intersection-Over-Union (Class IOU)* are commonly used.

In the case of conditional image synthesis, the two most commonly used metrics that are used is *Frechet Inception Distance (FID)*[HRU⁺17] and *Inception Score (IS)*[BS18]. The inception score is an objective metric used for evaluating the quality of generated images. The inception score was introduced as an attempt to remove the subjective human evaluation of images. The inception score leverages a pretrained deep neural network for image classification to classify the generated images. Specifically the inception v3 model has been used. The probability of the generated image belonging to each class is calculated. The predictions are then summarized into inception score. The score represents two properties of the generated images, i.e. image quality and image diversity. The range of inception score is $[1, (\text{number of classes supported by the classification model})]$. The inception v3 model supports 1000 classes and hence IS ranges between $[1, 1000]$. The higher the inception score, the better is the model.

The Frechet Inception Distance or FID calculates the distance between feature vectors calculated for generated and real images. The score represents how close two groups are in terms of statistics on computer vision features that are obtained using the inception v3 model. The FID score is used for the purpose of this thesis and a short description is provided in the *Evaluation* chapter.

3.3 Image Data Modification

Geometric and Photometric transformation

Two very basic augmentations that can be performed on an image is geometric and photometric. Geometric transformation includes zooming, cropping, flipping, etc. And photometric transformation includes altering RGB channels, gray scaling, colour jittering, contrast adjustment, etc. Wu et al. adopted a series of geometric and photometric transformation to prevent overfitting.

Hair Style transfer

Kim et al. used DiscoGAN[KCK⁺17] to transform the hair colour, which was introduced to discover cross-domain relations with unpaired data. The same was achieved by StarGAN[CCK⁺17], whereas it could perform multi-domain translations using a single model.



Figure 3.3: Hair style transfer[KCK⁺17]

Age Progression and Regression

The methods that have been traditionally used includes the prototypes based method and model-based method. The prototype based method creates average faces for different age groups, learns the shape and texture transformation between these groups, and applies them to images for age transfer. The model based method constructs parametric models of biological facial change with age, e.g. muscle, wrinkle, skin, etc. But these models have high complexity and computational cost.

Recent works applied GANs with encoders for age transfer. The input images are first encoded into latent vectors, it is then transformed in the latent space, and re-constructed back into an image with different age. Palsson et al.[PATV18] proposed three aging transfer models based on CycleGAN. Zhang et al.[ZSQ17] proposed a conditional adversarial autoencoder (CAAE) for face age progression and regression. Zhao et al.[ZCPR03] proposed a GAN-like Face Synthesis Subnet (FSN), that learns a synthesis function that can achieve face rejuvenating and aging with photo-realistic and identity preserving properties.

Disentangled Representation

Yujun et al.[SGTZ19] proposed an automatic face attribute modification method. The system was based on latent SVM model that described the relations among facial features, facial attributes and makeup attributes. Xi et al.[PYS⁺17] made the pose variation disentangled from identity representation by inputting a separate pose code to the decoder, and then using the discriminator to validate the generated pose.

3.4 Face Texture Generation

Zhixin et al.[SSG⁺18] disentangled the textures from the deformation of input images by adopting the Intrinsic DAE (Deforming- Autoencoder) model. It transformed the input image to 3 image properties namely shading, albedo and deformation. It then combined the shading and albedo to generate texture image.

Kyle et al.[OLY⁺17] presents a novel method to animate a face from a single RGB image from a video sequence. For the purpose of making the animations more realistic, they needed the dynamic per frame textures to capture the subtle deformations and wrinkles corresponding to the animated facial expressions. For this purpose they had

3. Related Work

proposed a novel framework that can generate realistic dynamic textures using a conditional generative adversarial network to deduce precise deformations in textures such as blinking, mouth open closed, etc.

Ron et al.[SSK18] presents a new approach to generate human faces. They achieved it by constructing a dataset of facial 3D scans. They mapped the facial texture onto a 2D map by the alignment of facial geometries and using an universal parametrization. Then GAN is used to generate more textures using the mapped textures as dataset. The texture dataset formation steps include:

- Producing 3D facial landmark annotations on the scanned faces.
 - Back-projecting the landmarks on a 3D mesh.
 - Performing vertex to vertex correspondence between face template mesh and facial scan.
 - Once the template is aligned with the scan, the texture is transferred from the scan to the template using ray casting technique in Blender.
-

4.1 Dataset Preparation

4.1.1 CelebFaces Attributes Dataset (CelebA)

The facial images dataset that has been used for this thesis is CelebA dataset. CelebFaces Attributes Dataset (CelebA) is a large-scale face attributes dataset with more than 200K celebrity images, each with 40 attributes annotations. The images in this dataset cover large pose variations and background clutter. CelebA has large diversities, large quantities and rich annotations, that includes:

- 10,177 number identities.
- 202,599 face images.
- 5 landmark notations, 40 binary attributes annotations per image.



Figure 4.1: CelebA dataset sample with a few annotations[LLWT15].



Figure 4.2: Downloaded Texture [Gmb20]

Figure 4.1 is a small portion of the dataset showing the different class of faces that are available in the dataset. In the figure only 8 are shown, but there are 32 others that includes bald, male, young, black hair, brown hair, etc. The resolution of the images are 178x218. For the purpose of the thesis, the images had to be square in shape, which means it has to be of equal height and width. All the images were converted to 512x512 resolution, because high resolution image generation is a prerequisite for the thesis, and that is the only augmentation that is performed on the CelebA dataset for the thesis.

4.1.2 Texture Dataset

As the main purpose of the thesis is high quality texture generation, getting texture data was also important. Texture dataset is not available commercially and it was a difficult task to get the images. Only 860 texture images was downloaded for this thesis. The downloaded textures were converted to 512x512 and 1024x1024 resolutions. An example of the downloaded texture is given below:

As the number of images is only 860, it was important to perform different augmentations in order to increase the dataset. All augmentations did not have satisfactory outcomes after the completion of the training that is discussed in the *Results and Discussion* chapter. The different augmentations that has been performed to make the textures fit for training are:

1. The black portions has been removed and images resized to 512x512 or 1024x1024 dimensions, according to the training needs.
2. Gaussian blurring of the images. The example has been shown below.
3. Adding Gaussian noise to the images.
4. Converting the image from RGB to HSV mode and then increasing the V channel.
5. Horizontally flipping the images.
6. Decreasing and increasing the contrast of the images.
7. Modifying the blue, green and red channel of the images.
8. The number of texture images that has been used without augmentation is 860

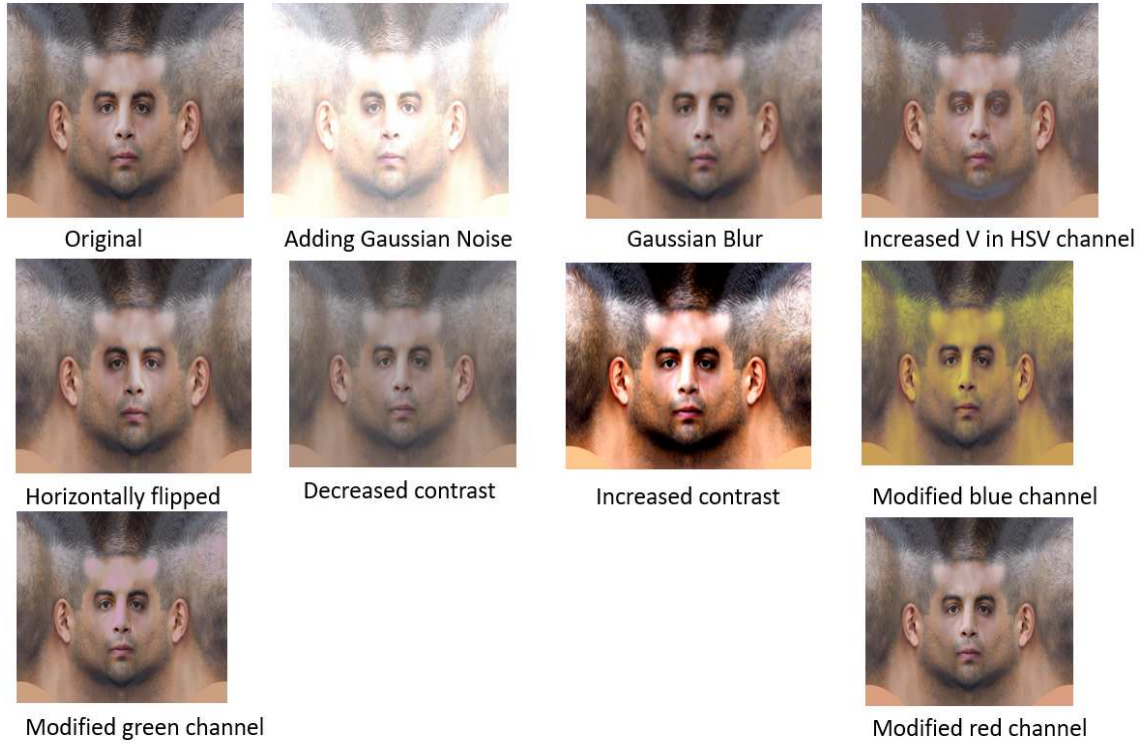


Figure 4.3: Different types of augmentations that has been performed on the images

and with augmentation it is 1600.
 Example of each kind of augmentation is shown in the figure 4.3.

4.2 Training

The official tensorflow version of the StyleGAN published by Nvidia Corporation[KLA18] is used for the thesis. The code was adjusted to suite the available hardware resources and dataset. The training was performed for 13 days on 512x512 resolution images of CelebA dataset. Two P6000 Nvidia Quadro GPUs were used for the training purpose. All weights of the mapping network, generator and discriminator network and affine transformation layers were initialized using $\mathcal{N}(0, 1)$. The constant input of the synthesis network is set to 1 and the biases and the noise scaling factor are set to 0. Adam optimizer is used for optimization. The batch sizes are different for training different resolution layers. The table 4.1 gives the batch size information. The learning rate used was 0.001. 260 epochs were used for training. With the progress of the training, different models were saved for different time steps. **Frechet Inception Distance**, also known as **FID** score, is used as an evaluation metric for the models. The model with the lowest FID score that was achieved was 5.89, after which the training was stopped, because the FID score had reached satisfactory level and the images were visually appealing. Hence quantitatively and qualitatively it was justified to stop the training. The figure 4.4

shows the images generated by the StyleGAN after initial training on CelebA dataset.

Table 4.1: Batch size used for training different resolution layers.

Resolution	4x4	8x8	16x16	32x32	64x64	256x256	512x512
Batch-size	160	140	120	100	80	40	30



Figure 4.4: Generated images after initial training

The next phase of the training is training with the texture dataset with the augmentation and without the augmentations. The evaluation regarding both the trainings is discussed in the *Results and Discussion* chapter. The concept of **transfer learning** is used in this context. Transfer learning is using an already trained model and retraining it with a different related dataset. As texture and facial images are almost similar, hence a model trained on facial images can be successfully retrained on textures also. Transfer learning has several benefits that has made it particularly beneficial in this scenario:

- A lot of data is usually needed to train the neural network which is not always available, which is exactly the situation in this case. This is where transfer learning acts as a saviour because not a lot of data is required for the retraining. With only 860 texture images it was possible to train the entire StyleGAN network, because it was already trained with the huge CelebA dataset.
- The other advantage is, it provides better results in most cases.
- Last but not the least, it is not needed to train a network every time from scratch, and hence as a consequence saves a lot of time.

The image below [Figure: 4.5] shows the generated texture of 512x512 resolution, that was trained with only 860 images of non-augmented textures.



Figure 4.5: Generated texture images after training. Each texture in this collage is 512x512 resolution. The FID score achieved is 51.2264

4.2.1 Effect of image augmentation during training

Augmentation of images has a profound effect in reducing overfitting and better generalization, especially when the data is less. In case of images, significant research has been made to show that image augmentation has helped in a better way to train a model and have achieved significant good results than their without augmentation counterpart[PW17]. Image augmentation involves horizontal and vertical flipping, rotation, cropping, and some other types has been already mentioned in Texture Dataset section above. Image augmentation may have positive impact on classification task, because it prevents the model from overfitting, but in-case of generative models, augmentation may not prove beneficial. The basic idea of a generative model is to mimic the distribution of the dataset, and hence if the model is provided with a dataset that is augmented or to be specific distorted images, then eventually the model will also learn the distortion. The figure 4.6 shows an augmented dataset.

The results that were obtained after the training with the augmented texture dataset are shown in figure 4.7. From the figure it is clear from the qualitative point of view that generated textures are not of desired quality. From the figure we can see that some of the textures has a pink hue, some are more yellowish, some has very high contrast and some has undesirable patches in the image. Hence all these undesired qualities render the generated textures unrealistic and useless.



Figure 4.6: Texture Dataset that is augmented. The augmentations that are done are changing the contrast, brightness, colour channel augmentations and adding different types of noise like Gaussian and salt and pepper noise.

Hence it provides a conclusive proof that augmentation is not a good idea for generative models. After removing all type of augmentations and training with just non augmented images, the model generated textures that are more realistic in nature and without any unusual colour or contrast issues as shown in figure 4.5. Further assessment for generated texture is mentioned in following *Degree of Realism* section and their evaluation is mentioned in the *Results and Discussion* chapter.

4.3 Degree of Realism Assessment

One important aspect of the thesis is the degree of realism. It was important to verify the sanctity of the generated images. FID score is one metric that has been used for a long time. The FID score actually compares the statistics of the generated samples to the real samples. But FID score is not the only one. For the purpose of this thesis some other metrics is considered to verify the degree of realism of the generated images. The other two metrics that are considered are *Face Index*[ZCPR03] and *Intercanthal Index*[ZCPR03]. Figure 4.8 shows the different facial *photometric points* of a face. This can be compared to the 68 facial landmark coordinates from the *IBUG 300-W dataset*[CSP13].



Figure 4.7: Generated textures when trained with augmented texture data. The generated textures in this case are unrealistic. As evident from the image some textures have yellow, pink tinge and some has high contrast, etc.

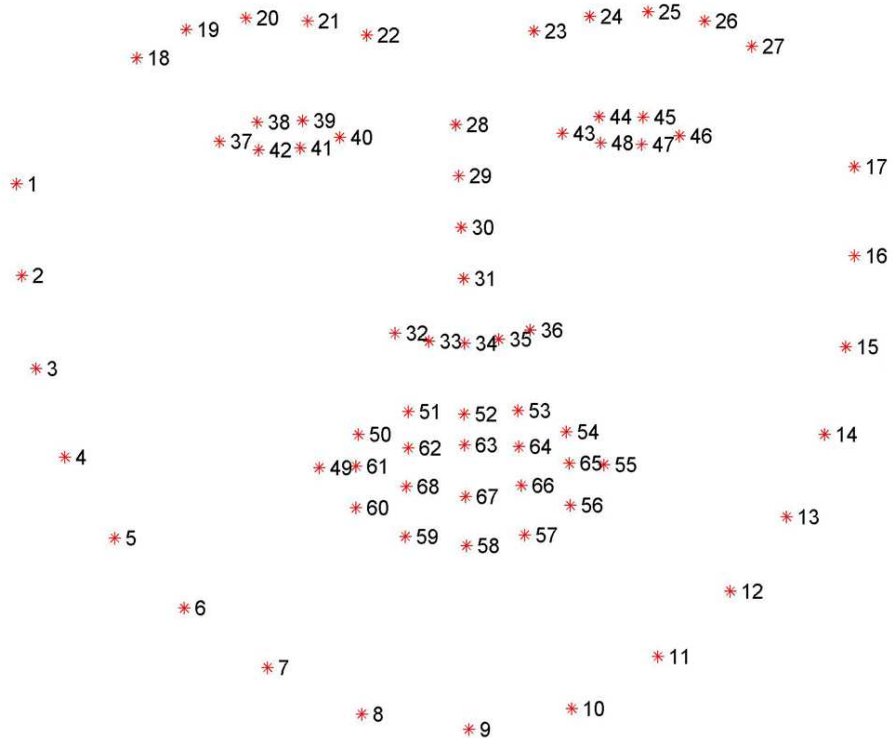


Figure 4.8: Visualizing the 68 landmarks coordinates from the iBUG 300-W dataset[CSP13].

These landmarks can be used to calculate important parameters to assess the degree of realism. Among them Face Index and Interanthal Index is chosen for the thesis.

- Face Index[ZCPR03]: It is the ratio of the facial height(28-9) to the upper facial width(1-17)[Figure: 4.8] . As it is a ratio it does not have any units.

$$Face\ Index = \frac{facial\ height\ (28 - 9) \cdot 100}{upper\ facial\ width\ (1 - 17)} \quad (4.1)$$

- Interanthal Index[ZCPR03]: It is the ratio between the interanthal width(40-43) to the biocular width(37-46). As it is also a ratio it does not have an index.

$$Interanthal\ Index = \frac{Interanthal\ width\ (40 - 43) \cdot 100}{Biocular\ width\ (37 - 46)} \quad (4.2)$$

Table 4.2: Face and Interanthal Index from original medical paper[ZCPR03]

LV Classification	Face Index Range		Face Index Mean		Interanthal Index Mean	
	Male	Female	Male	Female	Male	Female
Narrow	<79.90	<77.90	76.49 ± 2.82	75.39 ± 1.64	36.72 ± 3.65	36.75 ± 4.90
Medium	80-80.99	78-82.90	84.29±2.49	79.76±1.44	36.72±3.65	36.75±4.90
Wide	>89	>83	91.99±2.42	84.93±1.97	36.72±3.65	36.75±4.90

Face and Interanthal Index for different geometry of faces

The calculation for the Face Index and Interanthal Index is achieved using the dlib face detector that is shown in figure 4.9. The following steps are performed to calculate the Degree of Realism for the faces generated using the StyleGAN model :

- Loading the image is the first step.
- Then the face detection algorithm is applied on the image to identify the face and the 68 landmarks [Figure: 4.8].
- Those 68 landmarks or co-ordinates are stored in the form of a list, from which the required co-ordinates are extracted to calculate the indices that are mentioned above (Face and Interanthal Index).

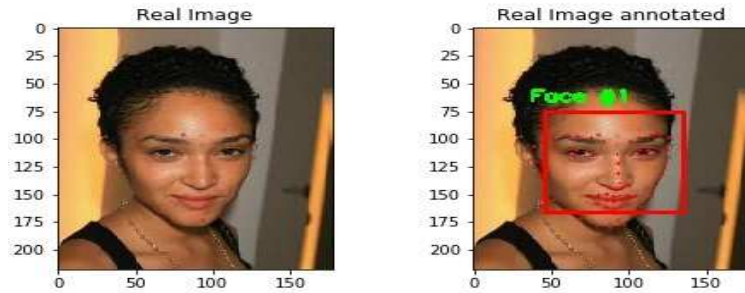


Figure 4.9: Visualizing the 68 landmarks coordinates from the iBUG 300-W dataset on the facial image

Table 4.2 shows the results from the original paper[ZCPR03]. These values are taken as reference to evaluate the degree of realism for the generated images as well as the images from the dataset. After performing the experiments it is observed that the results are quite satisfactory. From table 4.3 it is evident that the Face Index and the InterCanthal Index for CelebA dataset and the generated images are quite close to each other which justifies the accuracy of the GAN model and also provides a conclusive proof about the quality of generated faces. The graphical representation for the results are given in the *Results and Discussion* Chapter.

Table 4.3: Degree of Realism for generated faces

	Face Index (Mean \pm std)		InterCanthal Index (Mean \pm std)	
	CelebA	Generated	CelebA	Generated
Male	72.47 ± 6.08	70.36 ± 4.50	44.04 ± 2.39	44.06 ± 2.14
Female	68.91 ± 5.36	67.09 ± 3.84	43.15 ± 2.17	43.33 ± 1.83

Face and InterCanthal Index comparison between CelebA faces and generated faces for male and female

The Degree of Realism assessment is also done for the generated texture. The comparison is made between the face index values of the generated textures and the face index values for the downloaded textures that is used for the dataset. The table 4.4 shows the face index values for the generated textures and the downloaded textures. The values gives a conclusive proof that the GAN is able to generate similar quality textures as the downloaded ones, which is absolutely desirable.

Table 4.4: Degree of Realism for generated textures

	Face Index (Downloaded)	Face Index (Generated)
Textures	100 ± 9.06	100.13 ± 5.85

Face Index values for generated and downloaded textures. From the table it is clear that the model is able to generate similar quality textures as per the downloaded textures.

4.4 Modification of Generated Images (Faces and Textures)

Generating images is not always the end of the story. Conditional generation is an important topic in present day's research. In conditional GAN, as it has been shown in the Background chapter, the dataset is composed of images along with their labels. But

what if the labels are not known beforehand. In that case there should be some ways to modify the images to ones needs, just like the task that can be performed with photoshop software. Modifying images using machine learning techniques has huge potential and hence it is explored for the purpose of this thesis with great detail.

4.4.1 Style mixing using the StyleGAN model

It is already mentioned in the *Background* chapter, that the StyleGAN architecture makes it possible to control the image synthesis via scale specific modifications to the styles. The mapping network and the affine transformations [Figure: 2.9] can be viewed as a way to draw samples for each style from a learned distribution, and then the synthesis network can be viewed as a way to generate a novel image based on a collection of styles. The effects of each style are localized in the network, i.e. modifying a specific subset of the styles can be expected to affect only certain aspects of the image.

The reason for this localization is the AdaIN operation [Eq: 2.8]. The AdaIN operation first normalizes each channel to zero mean and unit variance and then applies scales and biases based on the style. The new per channel statistics that is dictated by the style, changes the relative importance of the features for the next convolution operation, but are not dependent on the original statistics because of the normalization. Hence it can be concluded that the style controls only one convolution layer before being overridden by the next AdaIN operation.

To further encourage the localization of styles, images are generated using 2 random latent codes instead of one during training. To generate such an image the latent code is changed from one to another at a randomly selected point in the synthesis network. To be more specific, z_1 and z_2 are two latent codes from which w_1 and w_2 are generated by passing through the mapping network and both are run through the entire synthesis network [Figure: 2.9], and w_2 is applied after the crossover point. Using this method, the network can also be prevented from assuming that adjacent styles are correlated.

The block diagram below shows an overview of the style mixing method [Figure: 4.10]. From the block diagram it can be seen that z_1 is used for the first few convolutional layers and z_2 is used for the last few convolutional layers. The crossover point in the diagram can be considered as 64x64 convolutional layer. The figure 4.11 shows an example of the style mixing method. The images are being generated using the model that is self trained. Modifying textures using style mixing is shown in the Evaluation chapter.

4.4.2 Modification using Regression

The purpose of this method is to make the latent space transparent. The basic generation process involves the feeding of noise to the generator and then getting face as an output. Hence, if it can be understood what the latent space represents, then the generation process can be completely controlled. The features of a face are gender, hair colour, etc. which are quite well represented in the CelebA dataset. Table [tab: 4.5] represents a small section. There are 40 such kinds of features mentioned in CelebA dataset. Here 1

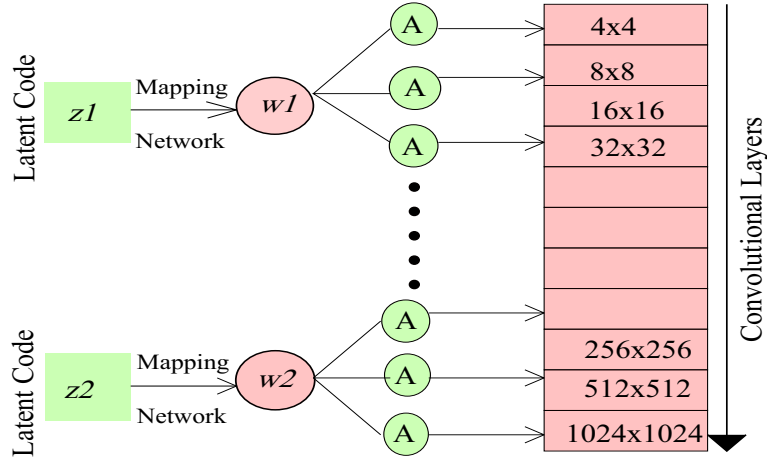


Figure 4.10: Style mixing block diagram

represents that the feature is present for that image and -1 represents that the feature is not present for that particular image.

Table 4.5: CelebA facial attributes[LLWT15]

image_id	5_o_Clock_Shadow	Attractive	Bald	Black_Hair
000001.jpg	-1	1	-1	-1
000002.jpg	-1	-1	-1	-1
000003.jpg	-1	-1	-1	-1
000004.jpg	-1	1	-1	-1

Hence the purpose of this approach is to link the latent space with these 40 features. The simple block diagram [Figure: 4.12] summarizes the entire approach.

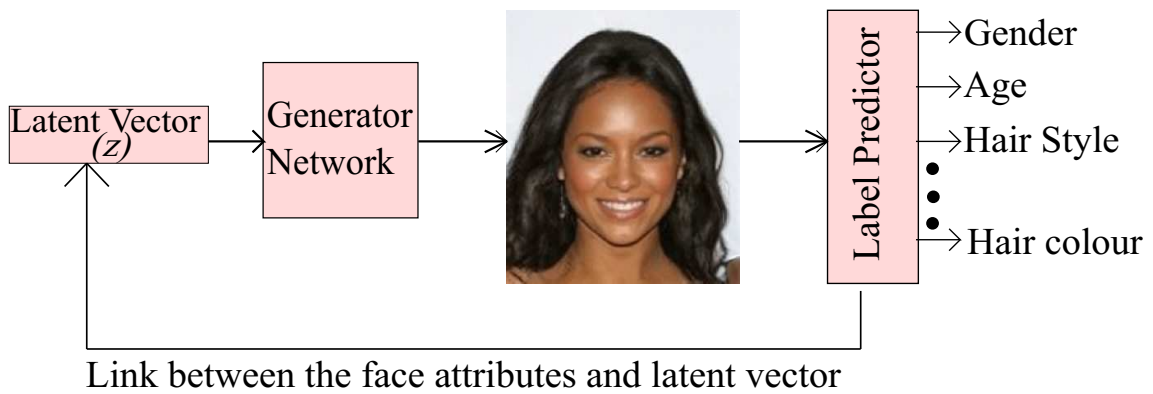


Figure 4.12: A simple block diagram that represents how the regression approach works.

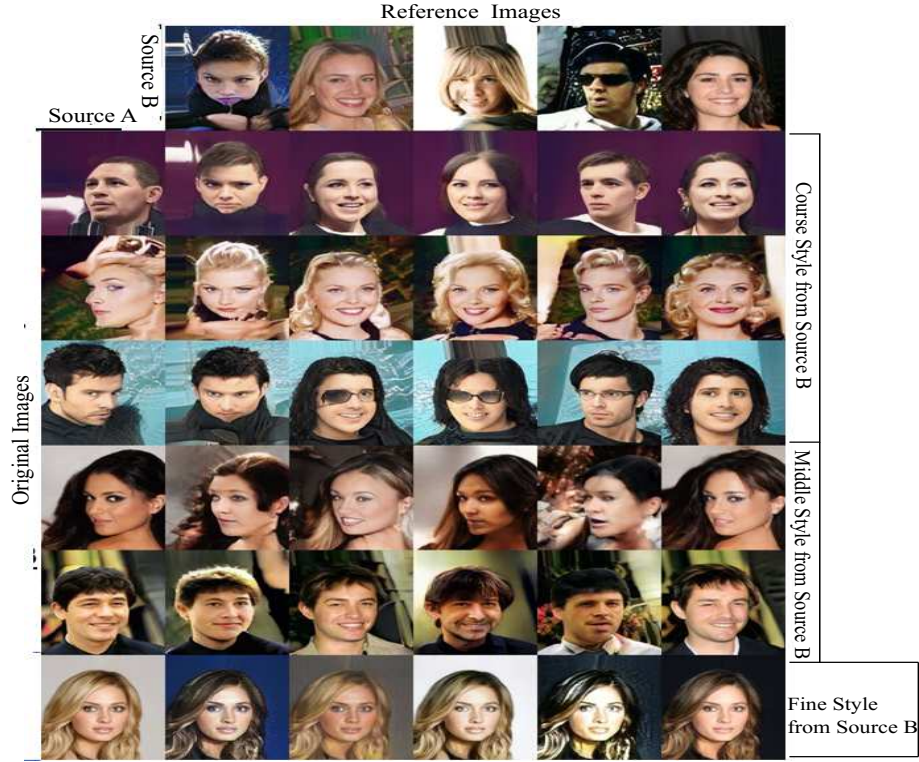


Figure 4.11: Implementation of style-mixing method where the new variations in images are created by combining the features of the original images with the reference images.

Challenges to the above approach:

- Firstly, how the features/labels can be obtained from generated images?
- Secondly, how the linking can be done between the feature vector and the latent vector?

The solution to the first challenge is to train a separate feature extractor network. The feature extractor network is basically a multi-label classifier model for discrete labels. Given a dataset with real labels (like celebA), the network can be trained using the x_{real} (image) and y_{real} (labels). The trained network can then be coupled with a trained generator network to get the features of the generated images. The small subsection below explains the face attribute predictor network that is used for this thesis.

Face Attribute Predictor

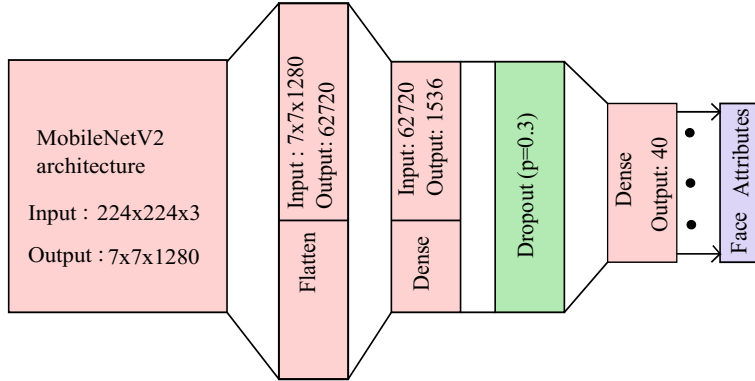


Figure 4.13: Label Predictor network

The above figure [Figure: 4.13] represents the face attribute predictor network. Transfer learning concept is used. MobileNetV2 is used as the base model. Mathematically this feature extractor can be written as $y = F(x)$, where F is the feature extractor, y is the feature vector and x is the generated image. Hence the workflow is to generate $x_{gen} = G(z)$ and then get $y = F(x_{gen})$ and then link y to z . Here x_{gen} is the generated image and z is the noise vector that is given as input to the generator G .

After all the features are obtained, corresponding to the latent vectors, that were passed through the generator, a regressor model $y = A(z)$ is trained to uncover all the feature axes for controlling the image generation process, which is the solution to the second challenge. Here A is the linear regression function.

Steps involving the modification process

- **Learning the distribution:** This step is basically training a generator network. For the purpose of this thesis StyleGAN architecture is used.
- **Classification:** This step involves the feature extractor. The feature extractor that is shown above [Figure: 4.13], is trained for this thesis to extract the features from the generated images. For that the network is trained using the CelebA training dataset. CelebA validation dataset is used for the validation. The training details are in the evaluation section.
- **Generation:** Generate images by passing random latent vectors, and then use the feature extractor network to get the features for the images.
- **Correlation:** Using a regression model to perform regression between the latent vector and features. The **regression slope** becomes the feature axes.
- **Exploration:** Taking one latent vector, the vector is moved along the feature axes, and examined how that affects the generated images.

Here **correlation** refers to the **Pearson's Correlation Coefficient**. It is a test statistics that measures the statistical relationship between two continuous variables.

It is considered to be the one of the best methods of measuring association because it is based on the method of covariance. It provides information about the magnitude of the correlation or association. Pearson's Correlation Coefficient (r) for continuous data ranges from -1 to 1. The formula below represents the correlation coefficient r [Eq: 4.3]:

$$r = \frac{\sum xy}{\sqrt{\sum x^2} \sqrt{\sum y^2}} \quad (4.3)$$

The diagram below [Figure: 4.14] shows the entire workflow for the modification process using the regression technique.

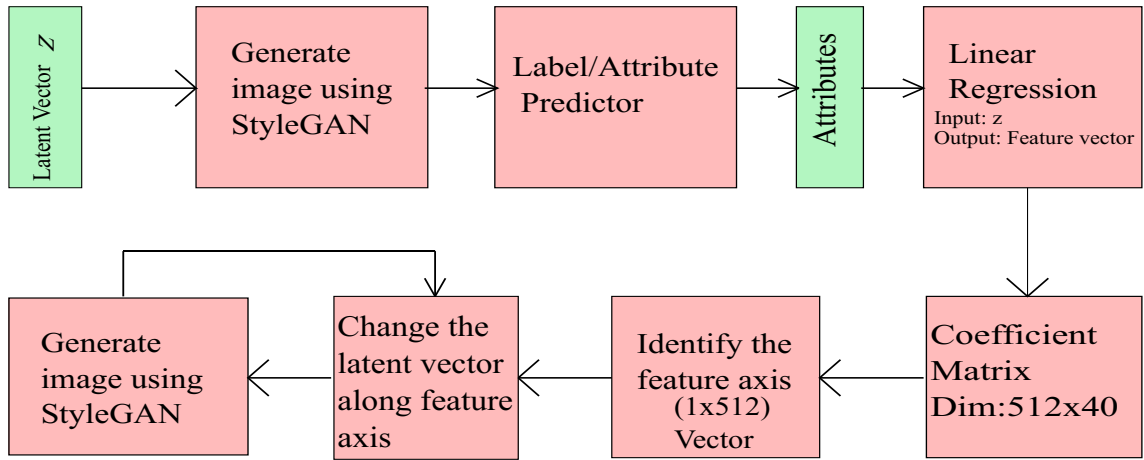


Figure 4.14: Workflow for modification of generated images using regression technique

The correlation matrix in the figure gives the regression slope. Each column in the matrix represents each feature (like gender, age, etc) and the values of each column tells the amount of that particular feature present in the 512D latent vector. One problem with above mentioned process is the entanglement of features. It means that all the features are correlated with one another. Hence, if one feature is changed, some other features are getting changed. The solution to this problem is the **disentanglement of features**. Disentanglement means to make the features uncorrelated to one another. To make the features uncorrelated to one another the feature vectors are made orthogonal to one another. When two vectors a and b are orthogonal to one another then their dot product $||a|| \cdot ||b|| \cdot \cos \theta = 0$ because here $\theta = 90^\circ$. When their dot product is 0 then according to equation 4.3 the numerator is 0 and hence correlation coefficient r is also 0.

4.5 Best Fit Texture Resolution

After generation and modification of textures, another important task is to wrap the textures around the 3D face model. The figure 4.15 shows the 3D model. For this thesis the facial model is taken into consideration.

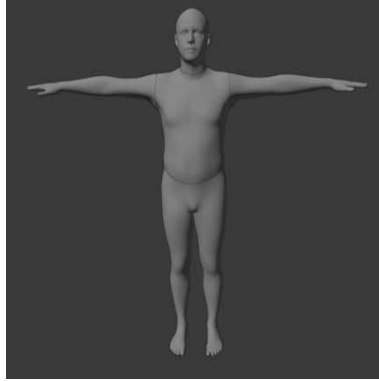


Figure 4.15: 3D model on which the generated textures is wrapped[Gmb20]

Every 3D model is associated with a UV map which is discussed in the *Background* chapter. The figure 4.16 shows the UV map corresponding to the facial 3D model. It is important that the generated textures fits the UV map properly, otherwise the generated faces (after wrapping the texture on 3D model) will be distorted, which is undesirable. Hence, it is important to figure out whether the wrapping is dependent on the resolution of the generated textures or not. Or is it the relative positions of the facial features (eyes, nose, ears, etc.) which is more important than the resolution for the texture to fit properly on the 3D model.

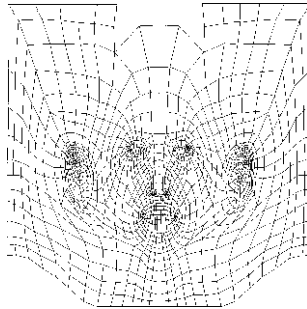


Figure 4.16: UV map of the facial 3D model[Gmb20]

Steps involved in the process of determining the best fit texture dimension for the given UV map are:

- After generating textures for a definite resolution, resize the textures for some different resolutions.
- Write a Blender script to wrap the texture around the 3D model and check qualitatively whether the textures fit properly or not.
- Calculate the face index and intercanthal index to check whether they match the values in table 4.2.
- Compare the values for downloaded textures and generated textures (wrapping both the downloaded and generated textures around the model).
- Evaluate the results and make an inference.

- Repeat these steps for all the resolutions for both downloaded and generated textures.

The figure 4.17 summarizes the steps mentioned above. The results and inferences are detailed in the *Results and Discussion* chapter.

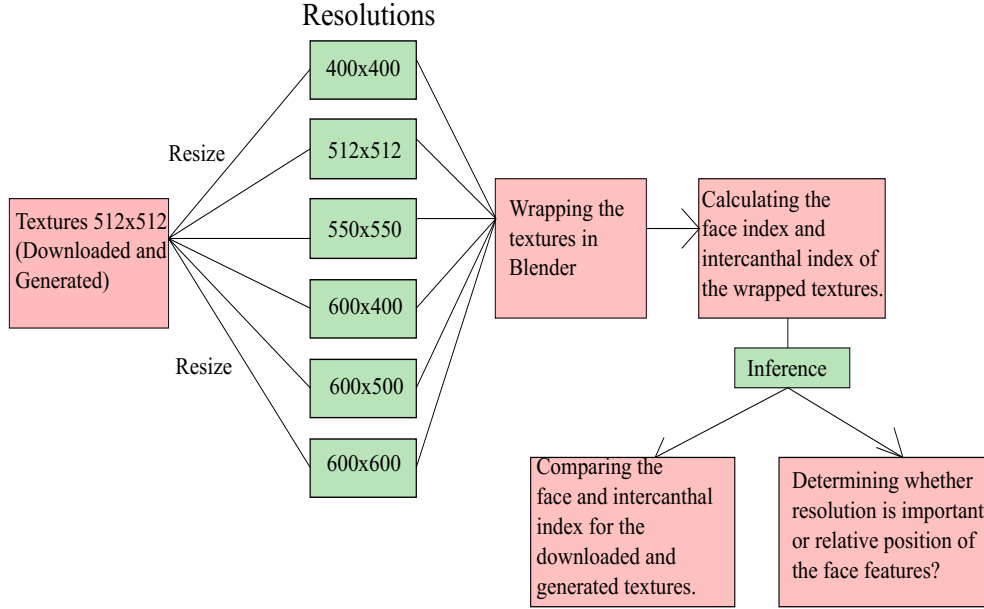


Figure 4.17: Block diagram for finding the best fit resolution of texture for the given UV map

4.6 Rectifying Bad Fit Textures

It is observed that some of the generated textures when fitted on the 3D model have a distorted appearance because the generated texture does not align to the UV map [Figure: 4.16]. As a matter of fact the textures in their raw form (when not wrapped around the 3D model) does not show any distortions until they are being wrapped around the 3D model. An example of such a distorted texture is shown below [Figure: 4.18].

The solution to this problem is two-fold. Firstly, it is important to differentiate between good textures and bad textures. This is important because, determining a bad texture after being wrapped on the 3D model is both time consuming and computationally expensive. Secondly, implementing an efficient method to rectify the bad textures, or to put it in a different way, aligning the textures to the UV map, so that the wrapped model is not distorted.

To determine a good texture (that aligns to the UV map) and bad texture (that does not align), the face landmarks are leveraged. The block diagram below [Figure: 4.19] gives an overview of the process that has been used to differentiate between a good texture and bad texture.

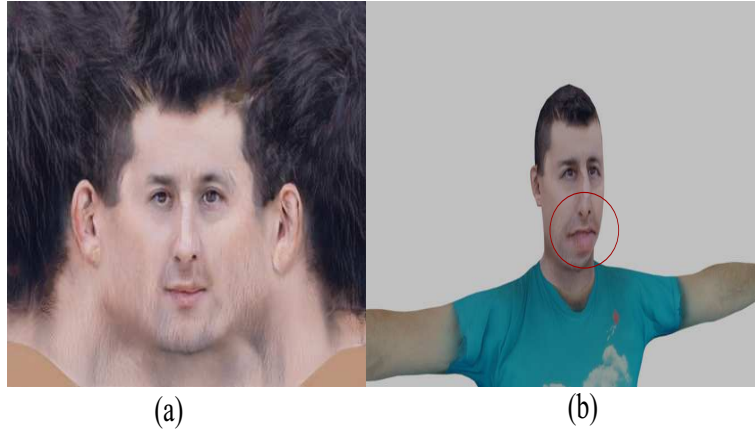


Figure 4.18: (a) Generated texture[Gmb20]; (b)Distorted wrapped texture. The red circled area shows the distortions[Gmb20].

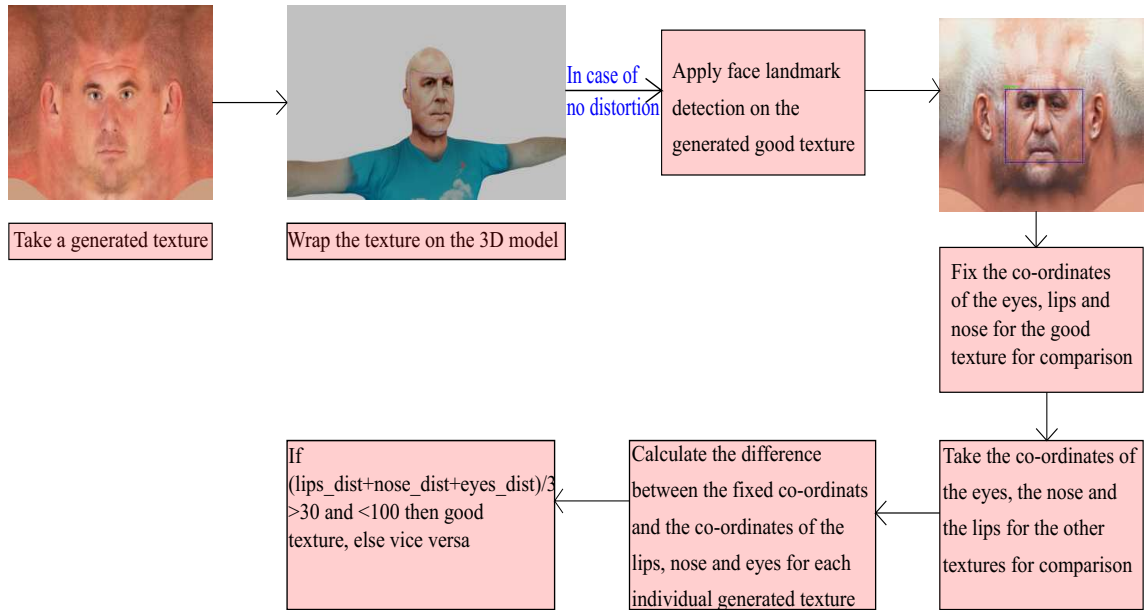


Figure 4.19: Block diagram to differentiate between good texture and bad texture. Here *lips_dist*, *nose_dist* and *eyes_dist* refers to the difference between the fixed co-ordinates and the co-ordinates of the textures that is being compared.

Steps involved in differentiating a good texture from a bad texture:

- Firstly around 1000 random textures are generated using the trained StyleGAN architecture.
- The generated textures are then wrapped around the 3D face model using Blender.
- The good textures (that does not get distorted on wrapping) are identified and dlib's face landmark detector is applied on the good textures.
- The co-ordinates of the lips, nose and eyes are fixed for comparison. From figure 4.8 the landmarks for which the coordinates of the lips, nose and eyes are considered are as follows:
 - Lips: [59, 52, 55, 58]
 - Left Eye: [37, 40]; Right Eye: [43, 46]
 - Nose: [28, 34, 32, 36]
- The average difference between the fixed co-ordinate values and co-ordinate values for each individual texture is calculated. Here the co-ordinate values for lips, nose and eyes are saved as 2D arrays. E.g. for lips the co-ordinate array is a 4x2 dimensional array. Here $Average\ difference = (lips_dist + nose_dist + eyes_dist)/3$. And $lips_dist = lips\ array(fixed) - lips\ array(texture\ under\ test)$. Similarly the $nose_dist$ and $eyes_dist$ are also calculated.
- After a few experiments it is observed that if the average difference is > 30 and < 100 then the texture is a good one, else vice versa.

After differentiating between the good and bad texture, the second part of the solution is to rectify the generated bad textures. For that the concept of Support Vector Machine (SVM) is used. The block diagram [Figure: 4.20] gives an overview of the rectification process leveraging SVM classifier.

This method is based on the fact that for any binary attribute, in this case, good or bad texture, there exists a hyperplane in the latent space (here latent space refers to the 512D latent vectors that are fed to the StyleGAN generator for the texture generation), such that all the textures on one side of the hyperplane has the same attribute, which is evident from the figure 4.20(b), where the red dots are the bad textures and the green dots are the good textures and the red line is the hyperplane that divides them.

The process of rectification involves firstly differentiating between a good and bad texture and at the same time storing their latents (512D vector fed to the StyleGAN generator) that generated those textures [Figure: 4.20(a)]. Then a SVM classifier is trained on 4K good and 4K bad texture latents [Figure: 4.20(b)]. The hyperplane is obtained on training the classifier. Given a hyperplane with unit normal vector $n \in R^d$, the distance from a sample z to the hyperplane is defined as,

$$d(n, z) = n^T z \quad (4.4)$$

here $d(,)$ can be both positive and negative. When z is moved towards and across the hyperplane, the distance and the semantic score (a texture being good or bad) changes. When the distance changes its numeric sign, i.e. it crosses the hyperplane, the semantic

attribute reverses i.e. the bad texture becomes a good texture. According to the above explanation, it is evident that to manipulate the attribute of a synthesized image, it can be done by changing the latent vector using the formula,

$$z_{new} = z_{old} + \alpha \cdot n \quad (4.5)$$

where z_{new} is the new latent vector, z_{old} is original latent vector, n is the unit vector perpendicular to the hyperplane, and α is the step length in the positive or negative direction [Figure: 4.20(c)]. Once a new z_{new} is obtained, it can be determined using the classifier whether the new latent vector lies on the positive or negative side of the hyperplane, and once the latent vector is on the positive side, the texture is generated using the latent vector and it can be observed that it is no longer distorted when wrapped on the 3D model. The justification of the mentioned method is shown in the *Results and Discussion* chapter.

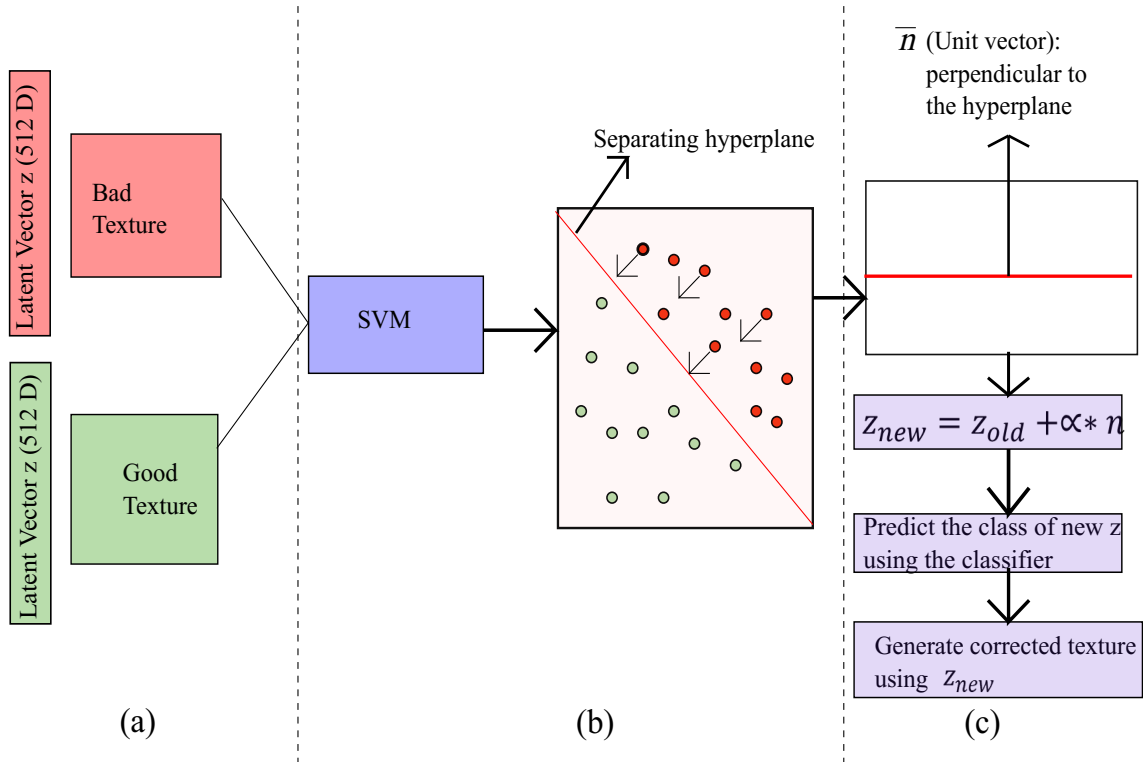


Figure 4.20: Process to rectify generated bad textures leveraging SVM classification technique.

Results and Discussion

The purpose of this chapter is to put all the results of the experiments that has been performed throughout the thesis in one place and making inferences from them.

5.1 Training of the StyleGAN architecture

This section contains the evaluation for the training progress of the StyleGAN model and the evaluation of the realism of the generated images.

5.1.1 Evaluating Training progress using FID score

The basic metric that is used to test the accuracy of the architecture during training is the FID Score[HRU⁺17]. The FID Score is an improvement to the Inception Score[BS18], that actually compares the statistics of the generated samples to the original samples. For FID the Inception network[SLJ⁺14] is used to extract features form an intermediate layer. Then the data distribution is modelled using multivariate Gaussian distribution using mean μ and covariance Σ . The FID between the real images x and generated images is computed as:

$$FID(x, g) = ||\mu_x - \mu_g||_2^2 + Tr(\sum_x + \sum_g - 2(\sum_x \sum_g)^{(1/2)}) \quad (5.1)$$

FID is more robust to noise and better measurement for image diversity. The lower the FID score, the better is the result. Ideally the FID score ranges between $[0, \infty]$. The figure [Figure: 5.1] shows the initial training progress on CelebA dataset w.r.t the FID metric. The lowest FID score achieved for 512x512 resolution images is 5.89.

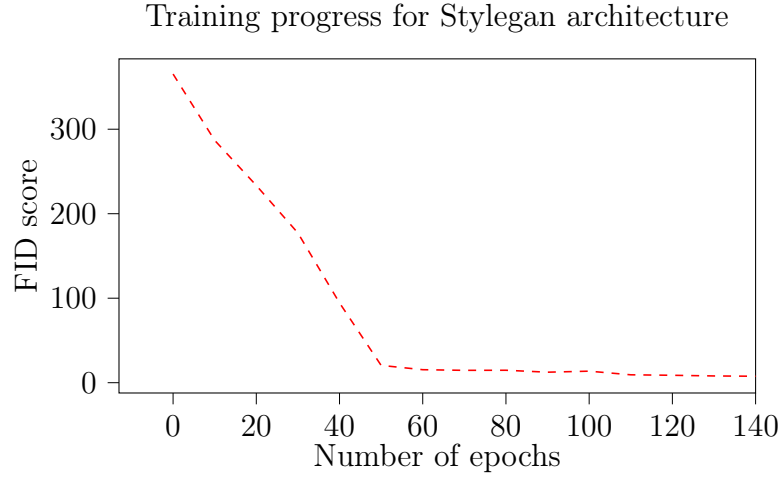


Figure 5.1: Training progress for StyleGAN architecture. From the figure it is evident that FID score decreases as number of epochs increases. The lowest FID score obtained is 5.8928 and the images are 512x512 resolution.

5.1.2 Degree of Realism for generated faces

In this section the degree of realism of the generated faces are being tested using two other metrics namely the *intercanthal* and *face* index that are already explained in the *Methodology* chapter under section *Degree of Realism Assessment* [Section: 4.3]. The evaluation is made by comparing the face and intercanthal index for CelebA dataset faces, that is used for the training and the generated images, after training the model for 260 epochs (11days). As mentioned earlier the generated images are of 512x512 resolution and the FID score obtained by the latest model is 5.89 which is quite impressive. The official paper achieved a FID of 5.06 on 1024x1024 resolution. But to justify for a lower FID, the available resources for the thesis was not up to the mark as mentioned in the official paper[KLA18].

The figures below [Figure: 5.2,5.3] shows the intercanthal and face index distribution for CelebA and generated faces.

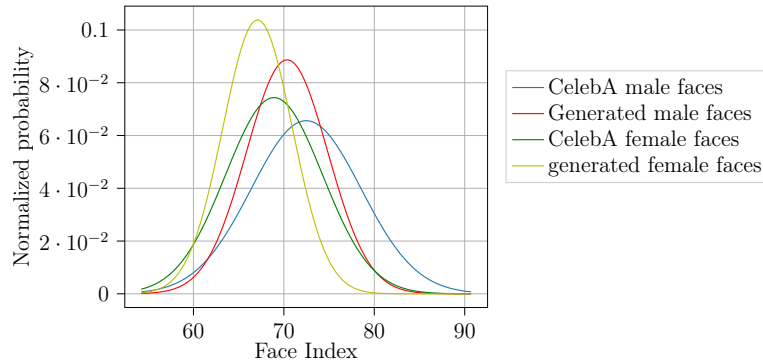


Figure 5.2: Face Index comparison between generated and CelebA faces for female and male.

The following inferences can be made from the figure:

- The distributions are quite similar to each other which means that the generated faces have similar features like the CelebA faces.
- For face indices in figure 5.2 it is evident that the distribution for the generated faces and the faces from the CelebA dataset are almost similar to each other proving the fact that the generated faces are similar in quality to the CelebA faces.
- The same scenario can be observed also for intercanthal index. From table 4.2, medically the range should be between 32 and 42. From table 4.3, it is evident that both CelebA and generated intercanthal indices are quite close to the real figures and also to each other.
- Hence it can be concluded that the generated images are up to the mark and hence the corresponding model could be successfully used for further experiments.

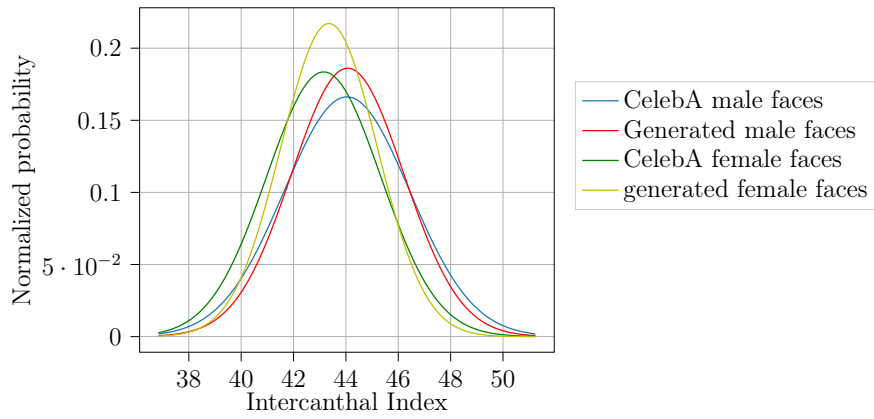


Figure 5.3: Intercanthal index comparison between CelebA and generated faces for male and female.

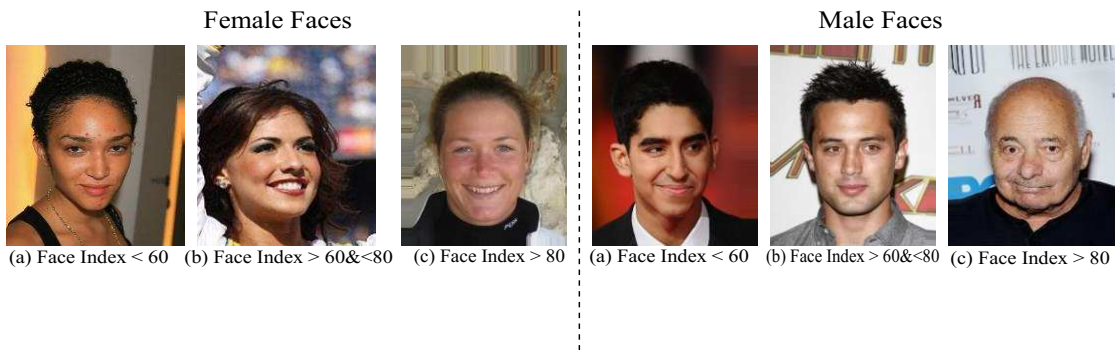


Figure 5.4: Comparison of face index values for male and female faces

5.2 Variational Analysis

The next most important part of the evaluation is the variational analysis. There is a concept called *Mode Collapse*, that actually refers to the variation part. Mode collapse refers to the situation when the GAN starts generating similar kind of images. That is something which is not desirable. Hence to quantitatively analyse the variation part this section is included.

The CelebA dataset comes with a *label.csv* file[LLWT15], that contains 40 labels that corresponds to each image in the dataset [tab: 4.5]. Hence a good way to analyse the variation is to check whether or not the generated images actually has those features.

For that reason a face attribute predictor network is trained that would tell what features are actually present in an image. The face attribute predictor network is explained in details in the *Methodology* chapter [Section: 4.4.2].

To check how accurately the face attribute predictor is able to predict the features a simple approach has been taken. Features w.r.t 10K images are taken from the CelebA dataset. Same 10K images were passed through face attribute predictor and the features that were obtained from the network were matched with original labels to find how accurate is the predictor network.

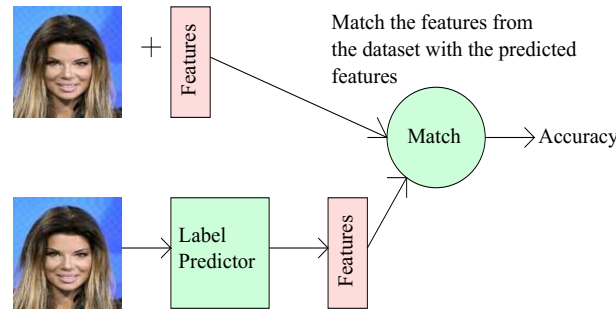


Figure 5.5: Block diagram to find the accuracy of the face attribute predictor network

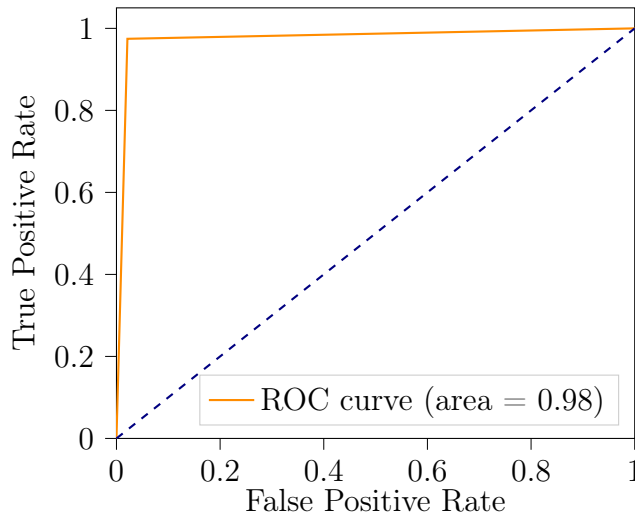
The accuracy is determined using the ROC (Receiver Operating Characteristics) curve. The ROC curve is created by plotting the True Positive Rate (TPR)[Eq: 5.2] against the False Positive Rate (FPR)[Eq: 5.3]. The TPR is also known as *sensitivity*, in terms of machine learning.

$$TPR/sensitivity = True\ Positive / (True\ Positive + False\ Negative) \quad (5.2)$$

$$FPR = False\ Positive / (True\ Negative + False\ Positive) \quad (5.3)$$

Below is an image [Figure: 5.6] of the ROC curve plot for male/female classification made by the face attribute predictor network. From the figure it is evident that the network can identify efficiently between male and female with 98% accuracy.

Receiver operating characteristic for gender identification

**Figure 5.6:** ROC curve for gender identification by the face attribute predictor network

Next, it is important to determine the variations in the Generated images. It is observed that the trained StyleGAN model is able to generate images with all kinds of labels that are present in the CelebA dataset. This is a good sign because variation is one of the most important thing that this thesis is aimed at. These experiments that are mentioned above proves that the model does not suffer from mode collapse. To determine the variations the face attribute predictor is used to predict the attributes of the generated images.

The following figure [Figure: 5.7] shows the variations in the images. E.g. Figure 5.7 tells that *black hair females* are present in almost equal proportions in both CelebA and generated images, i.e around 18%. Similarly face with and without beard are also present in almost equal amounts in both the sets of images.

5.3 Finetuning with Texture

After training on the CelebA dataset and assuring that the trained StyleGAN on CelebA can generate diverse and realistic images, the next step was to train the model to generate textures. For textures there were only 860 images to train on and hence the concept of transfer learning was considered, where the model was 1st trained on CelebA images where dataset is not an issue and then trained on texture dataset, where the dataset is too limited. It is already shown in *Methodology* chapter [Section: 4.2.1] that image augmentation is definitely not a solution to increase the number of images because that gives undesirable texture quality like in figure 4.7. Hence training the network with textures was a little tricky. The main hyperparameter that was considered for changing was the **learning rate** for both the generator and the discriminator network. Ideally, according to the official paper, the learning rate for both the networks is 0.001. In case

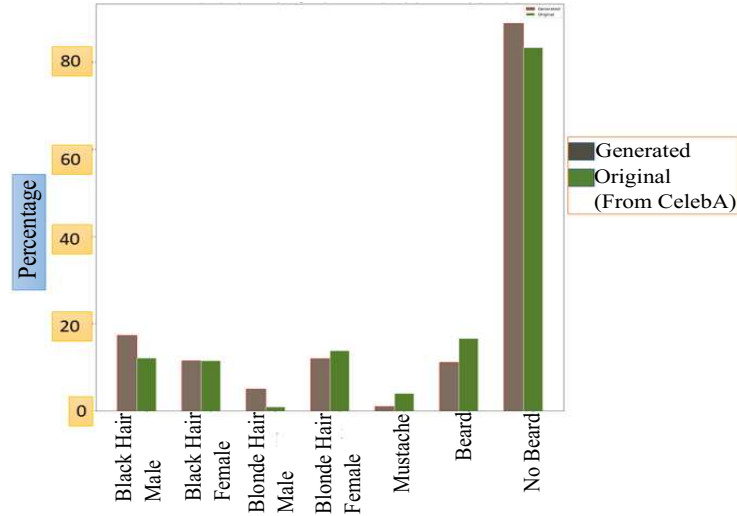


Figure 5.7: Variations present in the generated images and comparing their corresponding percentage with the CelebA dataset. The results have been obtained after testing on 1000 images from both CelebA and generated images. Hence it may not be exact but it gives a clear picture about the different variations present. The gray bar represents percentage of generated images and the green bar represents celebA images.

of textures that proved to be too high. Hence the learning rate was decreased step by step to get desirable texture quality.

Another comparison is made in this section. Textures are generated with the best model that is mentioned in the official paper[KLA18] and textures are also generated using the model that is self trained using the CelebA dataset. The table below [tab: 5.1] gives an overview of the training process for the textures.

Table 5.1: Training details of the StyleGAN model for Textures

Model Number	FID Score	Learning Rate	Dataset
11642	5.89	0.001	CelebA
11672	49.05	0.001	Textures
11702	41.66	0.0001	Textures
11942	97.99	0.0001	Textures
12242	105.12	0.0001	Textures

From the table it is evident that the best model 11702 has the lowest FID score of 41 and beyond that the FID score increases even though the learning rate is decreased. The quality of texture that is obtained with this model is quite satisfactory. The image below shows the generated images using the self trained StyleGAN model.



Figure 5.8: Textures generated using StyleGAN model 11702. None of the textures in the dataset is augmented

The table below [tab: 5.2] shows the training details for the StyleGAN model that is provided by Nvidia that is trained on CelebA dataset.

Table 5.2: Training details for the best model provided by Nvidia. The starting model number is imaginary and it has nothing to do with the training process. The subsequent models are saved after 20 epochs

Model Number	FID Score	Learning Rate	Dataset
14382	94.27	0.0001	Textures
14402	70.07	0.00002	Textures
14422	48.09	0.00002	Textures

From the above table [tab: 5.2] it is evident that the best model is 14422. The learning rate for the case of the best model was decreased further and beyond the model 14422 it was observed that the FID score increases and hence is excluded from the table.

The image below shows the textures generated by the best model.



Figure 5.9: Textures generated using StyleGAN model 14422. None of the textures in the dataset has been augmented

As it is evident from both the training details in table 5.1 and 5.2 that the FID scores differs a little, but quality wise there is no such difference. This is also justified using the face index parameter. The face indices of the textures obtained by both the models are almost the same. Hence for the purpose of the thesis, the model 11702, which is self trained, is used for further experiments. The face index for the generated and the downloaded textures are shown in table 4.4.

5.4 Modification of Generated Textures

After successful random generation of textures, the next important task is the modification of the textures.

5.4.1 Style mixing implementation for textures

The first method that is implemented is the Style mixing method which is explained in the *Methodology* [Section: 4.4.1] chapter. The figure below [Figure: 5.10] shows the style mixing method implemented using textures.



Figure 5.10: Style mixing implementation using textures

In the above figure, the 1st 3 rows, the images inside the yellow border, gives the result of copying the styles from source B corresponding to coarse spatial resolutions (4^2 - 8^2) that brings high level aspects such as pose, hair style, a face shape from source B, while all colours and finer facial features represents source A. Similarly copying the styles from the middle resolutions (16^2 - 32^2) from B, small scale features like hair style, eye open/close, are inherited from B, whereas higher level features like face shape, pose, etc are preserved from A. The last row represents copying fine styles (64^2 - 512^2) from B that brings finer changes like colour scheme and micro-structures.

This method has already been implemented in the official StyleGAN paper[KLA18] using face images. For the purpose of the thesis the same method is implemented using the trained model for textures. It can be concluded that the number of variations that can be achieved using this method is commendable. It is evident from the figure

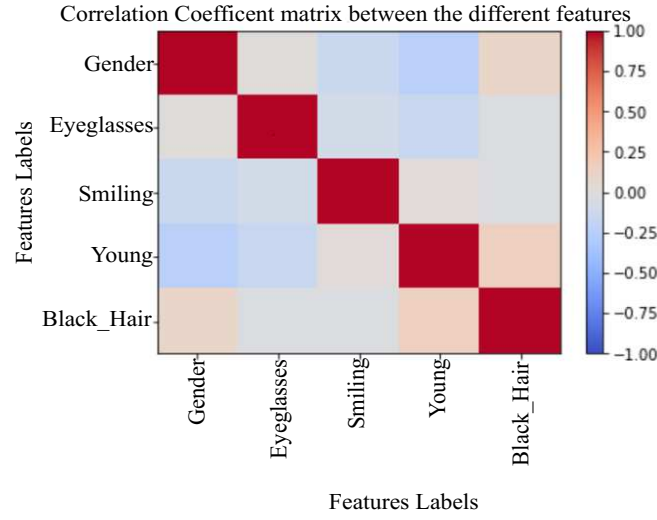


Figure 5.11: Feature entanglement representation using 5 features.

that on increasing the number of reference images the number of variations can also be increased. E.g. here with 5 reference images, 5 variations could be created. In figure 5.10, 30 images are created from 11 images which is almost 3 times and all are different from each other.

Even in this method it is not possible to exclusively change each and every feature. The next method has made it possible to identify the features and modify accordingly.

5.4.2 Modification of Images using Regression Analysis

Modification of images using regression method is explained in the *Methodology* chapter [Section: 4.4.2]. Using this method it is possible to identify the features individually and modify them accordingly. The most important part of the regression technique is the **correlation matrix**. The correlation matrix tells how the different facial features are interrelated by computing the correlation coefficient between the different features. In the regression technique selective modification is performed. Theoretically if 5 features are considered as shown in figure 5.11, then the number of variations that could be possible using this technique is $5! = 120$ variations. Hence, it is evident that the number of variations that could be achieved is commendable.

Application of regression technique

First, the experiments are performed on faces. Two features are considered to perform the experiments and for the correlation matrix 5 best predicted features are considered. Figure: 5.11 shows the entangled correlation matrix.

From the correlation matrix 5.11 it is evident that the features *smiling* and *eyeglasses*

have an inverse relationship. Similarly *age (young)* and *gender* also have an inverse relationship. Hence changing either of the features will affect the other features also. The image 5.12 shows the entanglement property. On trying to reduce the smile property, eyeglasses starts appearing. In addition to that some other features have also changed a little bit, as it is already shown that the features are entangled.



Figure 5.12: (Left to right) Feature entanglement representation. On trying to reduce the *Smile* property, *Eyeglasses* starts appearing. 30 intermediate images have been generated, but not all the intermediate stages have been shown to keep it clean.

Similarly age and gender are also entangled. The image 5.13 shows that trying to change the age, the gender of the subject changes.

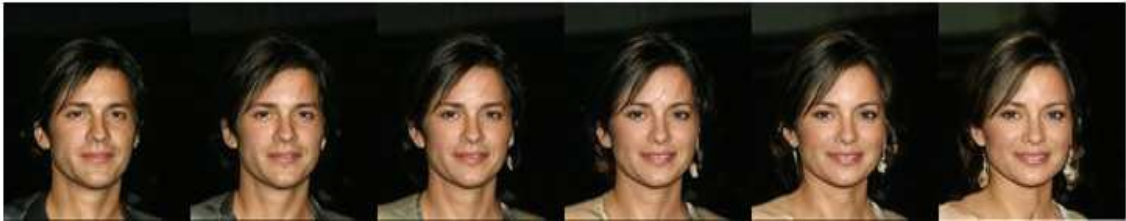


Figure 5.13: (Left to right) Feature entanglement representation. On trying to increase the *Age* property, *Gender* changes. 30 intermediate images were generated, but not all the intermediate stages are shown to keep it clean.

As it is already discussed in the *Methodology* chapter [Section: 4.4.2], to reduce the entanglement, the features have to be made orthogonal to each other. This means that the feature vectors of the regression coefficient matrix have to be made orthogonal to each other, so that the correlation coefficient is 0.

The figure 5.14 demonstrates the outcome of making features orthogonal to each other. In figure *smiling* and *eyeglasses* are made orthogonal to each other and hence it can be observed that, changing the smile attribute does not change anything w.r.t the eyeglasses attribute. It has to be kept in mind that, only these two attributes are orthogonal but these attributes are still correlated to the other attributes and hence changing *smile* will affect the other features except *eyeglasses*.



Figure 5.14: (Left to right) Feature disentanglement representation. On trying to reduce the *Smile* property, *Eyeglasses* does not change. It is evident that some other features like dress, background, changes a little bit as they are still entangled. 30 intermediate images have been generated, but not all the intermediate stages are shown to keep it clean.

Similarly the *young* and *gender* attribute have been made orthogonal to each other and consequently changing the age, doesn't change the gender. The figure 5.15 represents that.



Figure 5.15: (Left to right) Feature disentanglement representation. On trying to reduce the *Age* property, *Gender* does not change. It is evident that some other features like dress, background, hair style changes a little bit as they are still entangled. 30 intermediate images have been generated, but not all the intermediate stages are shown to keep it clean.

After testing on the faces, the next set of tests are conducted on textures. Similar to faces, for textures also the correlation matrix is also plotted.

From the figure 5.16 it is evident that *gender* and *beard* has a strong negative correlation. Which means if we try to change the *beard* attribute, e.g. reducing the amount of beard, the gender will also change with that. The next figure [figure: 5.17] is an example of such a transformation.



Figure 5.17: (Left to right) Feature entanglement representation. On trying to reduce the *Beard* property, *Gender* starts changing. 30 intermediate images have been generated, but not all the intermediate stages are shown to keep it clean.

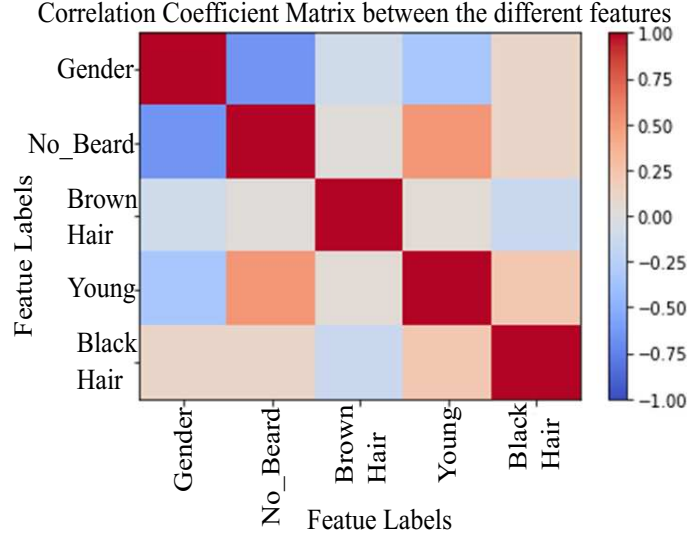


Figure 5.16: Correlation Matrix w.r.t 5 best features that can be predicted by the Label Predictor w.r.t the textures.



Figure 5.18: (Left to right) Feature entanglement representation. On trying to increase the *Beard* property, *Gender* starts changing. 30 intermediate images have been generated, but not all the intermediate stages have been shown to keep it clean.

From the figure 5.17 it is evident that from left to right, the subject changes to a more feminine looking subject which is not desirable. The vice versa can be observed for the figure 5.18, where on trying to increase the beard the subject changes from female to male.

It can be also observed from figure 5.16 that *beard* and *young* has a positive correlation, which means increasing the beard should also increase the age of the subject. The figure 5.19 represents such a transformation.



Figure 5.19: (Left to right) Feature entanglement representation. On trying to increase the *Beard* property, *Age* starts increasing. 30 intermediate images have been generated, but not all the intermediate stages are shown to keep it clean.

To avoid the entanglement, first the beard and age is made orthogonal to each other. The correlation matrix obtained after the orthogonalization is shown in the figure 5.20.

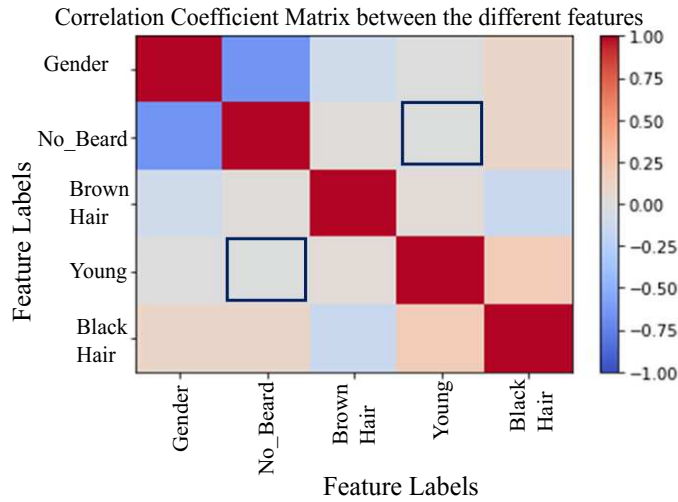


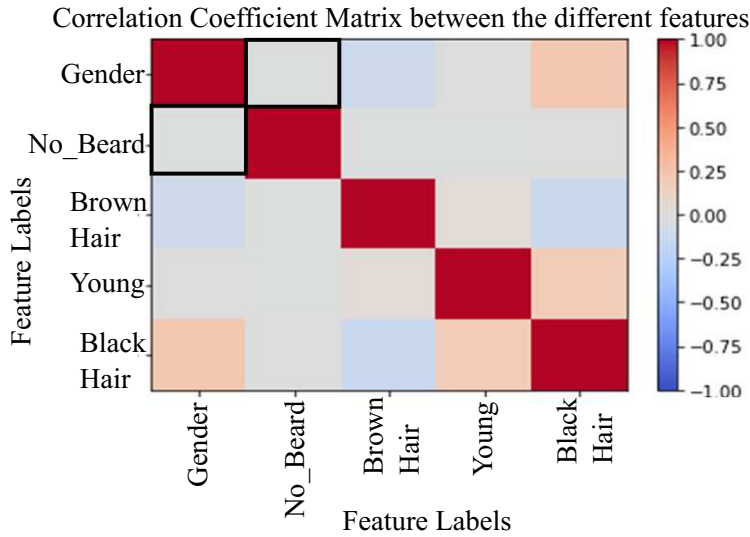
Figure 5.20: *Beard* and *Young* made orthogonal to each other. The 2 black boxes are gray in colour which means that the correlation is 0 between these 2 attributes

The figure 5.21 below shows the disentanglement property between *age* and *beard*. When the beard property is increased the subject's age does not increase. Here it should be noted that in CelebA there is only 2 distinctions w.r.t age, i.e young and old. Hence the face on the extreme right of figure 5.21 can be considered to be a young subject.

Next the *Beard* and *Gender* is made orthogonal to each other. The figure 5.22(a) shows the orthogonalized correlation matrix and figure 5.22(b) shows a example of the disentanglement.



Figure 5.21: (Left to right) Feature disentanglement representation. On trying to increase the *Beard* property, *Young* doesn't change. This means the age of the subject does not change. 30 intermediate images have been generated, but not all the intermediate stages are shown to keep it clean.



(a) Figure 1



(b)
Figure 2

Figure 5.22: Beard and gender disentangled. (a) Corelation matrix showing the disentangle-ment; (b) (Top to bottom) Faces showing the disentanglement between beard and gender.

5.5 Determining the best fit texture for the 3D face model

The steps to determine the best fit resolution is already mentioned in the *Methodology* chapter [Section: 4.5]. The figure 5.23 shows a generated texture wrapped on the 3D model and the face detection algorithm applied on it to calculate the face and intercanthal index.



Figure 5.23: Generated texture wrapped on a 3D model and face detection algorithm applied on it.

The tables 5.3 and 5.4 shows the mean face and intercanthal indices for different resolutions of the textures wrapped on the 3D model.

Table 5.3: Mean face index for different resolutions of downloaded and generated textures wrapped on the 3D face model.

Resolution	Face Index (Mean \pm Standard Deviation)	
	Downloaded Texture	Generated Texture
400x400	92.32 \pm 10.97	91.14 \pm 26.52
512x512	93.23 \pm 5.88	96.78 \pm 4.46
550x550	93.22 \pm 5.57	97.02 \pm 3.86
600x400	93.44 \pm 5.77	96.80 \pm 4.26
600x500	93 \pm 5.67	96.75 \pm 4.27
600x600	93.68 \pm 5.68	96.68 \pm 4.61

Table 5.4: Mean intercanthal index for different resolutions of downloaded and generated textures wrapped on the 3D face model.

Resolution	Intercanthal Index (Mean \pm Standard Deviation)	
	Downloaded Texture	Generated Texture
400x400	40.74 \pm 4.89	39.15 \pm 7.94
512x512	41.29 \pm 2.66	40.21 \pm 2.25
550x550	41.29 \pm 2.64	40.24 \pm 2.34
600x400	41.12 \pm 2.63	40.41 \pm 2.02
600x500	41.18 \pm 2.56	40.31 \pm 2.09
600x600	41.20 \pm 2.60	40.36 \pm 2.15

It is evident from the tables that all the resolutions have almost similar values for both the downloaded and generated textures. Figure 5.24(a) and 5.24(b) shows the distribution for face index and intercanthal index for different resolutions and it is evident

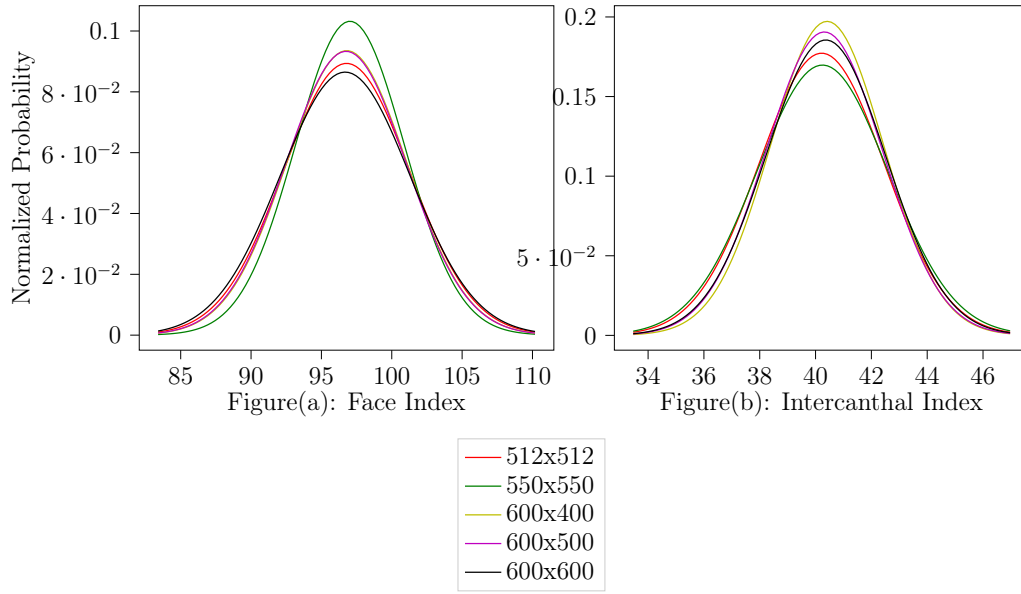


Figure 5.24: Face and Intercanthal Index plots for diferent resolutions of textures. The legend represents different resolutions.

that the indices are almost equal to each other and figure 5.25 shows the textures for different values of face indices. Hence it can be inferred that the UV map fitting is not dependent on the resolution of the textures. If that would have been the case, then the indices would not have been the same for all the resolutions. Hence, it can be inferred that the relative position of the nose, ears, eyes and lips in the texture with respect to the available 3D model is important.

Last but not the least it is important to justify that the relative position of the features of the face is important to fit the texture properly on the UV map, and not the resolution. The figure 5.26 gives a clear picture of the statement made above.



Figure 5.25: Face textures for different face indices

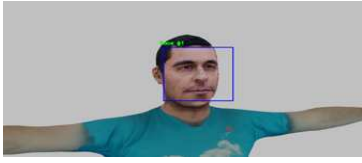


Resolution	Good Texture	Bad Texture
512x512		
550x550		
600x600		

Figure 5.26: Comparison between fitting a good texture and a bad texture on the given UV map. (The textures are generated)

It is evident from figure 5.26 that in spite of the resolution, the good quality texture on the left, fits perfectly on the model, whereas the bad quality texture, on the right side, when wrapped around the 3D model, has a distorted appearance, irrespective of the resolution. When face detection algorithm is applied on it, the algorithm can successfully identify the face for both the cases. But if a closer look is taken [Figure: 5.27], it can be observed that the 68 face landmarks can be properly identified in case of the good texture, but not for the bad texture, which also justifies that the facial features like the lips, eyes, etc are not in position w.r.t the UV map.



Figure 5.27: Face detector applied on generated textures. (Left) is a bad quality texture, (right) is a good quality texture. The red encircled area on the left image shows the failure of the face detector to detect the lips and the nose properly.

5.6 Rectification of distorted texture

The process of rectifying bad fit textures is already explained in details in section 4.6. Here a justification is provided in order to prove how appropriate the method is and how well it works. Here both the scenarios of positive α and negative α values is shown in order to establish what is the consequence of moving a bad texture towards the hyperplane (when α is positive) and away from the hyperplane (when α is negative). The figure 5.28 shows how the textures get rectified when the latent vector z is changed based on equation 4.5. Here the value of α is $+0.1$. The z is changed for different values of α like 3α [Figure: 5.28(b)], 5α [Figure: 5.28(c)], 9α [Figure: 5.28(d)] and 13α [Figure: 5.28(e)]. Figure 5.28(a) is the original distorted texture. Considering the hyperplane

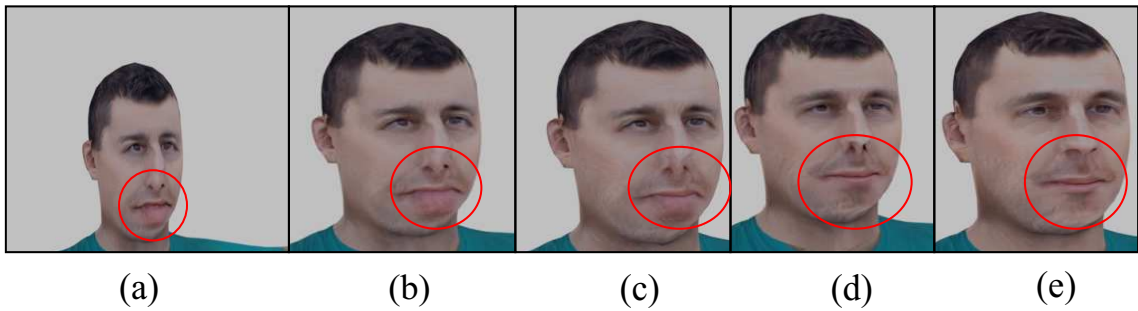


Figure 5.28: (Left to right) Rectification of distorted texture leveraging SVM concept. The red circled areas shows the distortion and how the rectification process minimizes the distortion.

to be at the origin, table 5.5 shows the distance between the hyperplane and the latent vector z which is used to generate the texture. It is evident that originally the texture lies on the negative side of the hyperplane and as it is moved towards the hyperplane following the equation 4.5, the distance [Eq: 4.4] from the hyperplane reduces and consequently distortion also reduces. As the distance becomes positive at $i = 9$, i.e. a step length of 0.9 is considered, the semantic attribute of the texture reverses which means it becomes a good texture because finally the latent vector z is now on the positive side of the hyperplane.

Table 5.5: Change in distance of latent vector z [Eq: 4.4] from the hyperplane for different multiples of $+\alpha$. i here is 0, 3, 5, 9 and 13

$(\alpha=0.1)*i$	0	0.3	0.5	0.9	0.13
Distance from the hyperplane (d)	-0.59	-0.29	-0.09	0.1	0.3

Similar to positive α , experiments are also performed using negative values of α to show that farther the texture is from the hyperplane the more distorted it appears. As a consequence it also justifies that the method of SVM is feasible and works as expected.

The figure 5.29 shows how the textures get rectified when the latent vector z is changed based on equation 4.5. Here the value of α is -0.1 . The z is changed for

different values of α like 3α [Figure: 5.29(b)], 5α [Figure: 5.29(c)], 9α [Figure: 5.29(d)] and 13α [Figure: 5.29(e)]. Figure 5.29(a) is the original distorted texture.



Figure 5.29: (*Left to right*) Distortion of distorted texture leveraging SVM concept. The red circled areas shows the distortion and how the face gets more distorted as the texture is moved further away from the hyperplane.

Table 5.6 shows that as the latent z is moved further away from the hyperplane by considering α to be negative, the distance increases more towards the negative direction and consequently the texture becomes more distorted which is evident from the figure 5.29.

Table 5.6: Change in distance of latent vector z [Eq: 4.4] from the hyperplane for different multiples of $-\alpha$. i here is 0, 3, 5, 9 and 13

$(\alpha=-0.1)*i$	0	0.3	0.5	0.9	0.13
Distance from the hyperplane (d)	-0.59	-0.89	-1.09	-1.29	-1.49

Conclusion and Outlook

6.1 Conclusion

In this thesis it was shown how the state of the art StyleGAN architecture could be used for the generation of high quality realistic textures. It was also shown how variations could be achieved and how the proper fitting of the textures could be automated. It starts with the leveraging of the StyleGAN architecture to generate 2D face textures, which is technically a more efficient way than the already existing methods mentioned in the *Related Work* chapter. A conclusive proof was also shown that augmenting texture data to train StyleGAN architecture might not be a good idea. Two new metrics were introduced in this thesis namely the *face index* and *intercanthal index* that can be used to assess the degree of realism of generated faces or textures. These two metrics are geometrically more informative because they take into consideration the geometry of the face and will provide a more intuitive understanding of the quality of the generated faces and textures apart from the FID score.

After generation the topic of modification of textures was also dealt with, in this thesis. It was shown how the already existing method of style-mixing can be used to efficiently modify the generated textures, the same way as it can be done for faces. To have a more controlled modification process an efficient technique, the regression technique, that actually helps in exploring the latent space of the GAN, was discussed in this thesis. After generation and modification the next important topic that was discussed was the fitting of the 2D texture on the 3D model. Blender has been used to fit the textures to the 3D model i.e. the process of UV mapping has been done using Blender. It was shown through extensive experiments that the proper fitting of a 2D texture onto a 3D model is independent of the resolution of the texture. It depends on the relative positions of the eyes, nose, ears and lips with respect to the face and on the geometry of the 3D model that is available. Through extensive experimentation it was shown how support vector machine (SVM) can be leveraged to rectify those textures that appear distorted after wrapping them on the 3D model. It was also shown through experiments about how to identify a possible bad texture that will appear distorted post wrapping, before actually fitting the texture onto the 3D model.

6.2 Future Work

As a part of the future work, the first and the foremost thing that could be done is to leverage the GAN architecture to generate textures for the other parts of the body and try to optimize the fitting process and in that way an entire 3D model could be generated and with just one 3D model we can generate as many textures as we want and we can have different 3D models.

Another important topic would be to generate face textures based on some predefined criteria. Until now we have been modifying the images after they are being produced but an important topic for research will be to find an optimal network that would generate a latent vector (512D), that would be fed to the StyleGAN generator, from a criteria vector similar to the annotations that we find in the CelebA dataset. In that way we can generate textures or faces with the required features that we want.

Bibliography

- [ACB17] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *ArXiv*, abs/1701.07875, 2017.
- [Aga18] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018.
- [BDS18] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. *CoRR*, abs/1809.11096, 2018.
- [BS18] Shane T. Barratt and Rishi Sharma. A note on the inception score. *ArXiv*, abs/1801.01973, 2018.
- [CCK⁺17] Yunjey Choi, Min-Je Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. *CoRR*, abs/1711.09020, 2017.
- [CSP13] S. Zafeiriou C. Sagonas, G. Tzimiropoulos and M. Pantic. 300 faces in-the-wild challenge: The first facial landmark localization challenge. *2013 IEEE International Conference on Computer Vision Workshops, Sydney, NSW*, pages 397–403, 2013.
- [DCSF15] Emily L. Denton, Soumith Chintala, Arthur Szlam, and Robert Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. *CoRR*, abs/1506.05751, 2015.
- [Gmb20] BIT Technology Solutions GmbH, 2020.
- [GPAM⁺14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

- [HB17] Xun Huang and Serge J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *CoRR*, abs/1703.06868, 2017.
- [HLS07] Kai Hormann, Bruno Lévy, and Alla Sheffer. Mesh parameterization: Theory and practice. In *ACM SIGGRAPH Course Notes*, 2007.
- [HRU⁺17] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017.
- [KALL17] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017.
- [KCK⁺17] Taeksoo Kim, Moon-su Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. *CoRR*, abs/1703.05192, 2017.
- [KLA18] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018.
- [KW19] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *CoRR*, abs/1906.02691, 2019.
- [LBBH98] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.
- [LLWT15] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.
- [OLY⁺17] Kyle Olszewski, Zimo Li, Chao Yang, Yi Zhou, Ronald Yu, Zeng Huang, Sitao Xiang, Shunsuke Saito, Pushmeet Kohli, and Hao Li. Realistic dynamic facial textures from a single image using gans. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5439–5448, 2017.
- [PATV18] S. Palsson, E. Agustsson, R. Timofte, and L. Van Gool. Generative adversarial style transfer networks for face aging. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2165–21658, June 2018.
- [PW17] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017.
- [PYS⁺17] Xi Peng, Xiang Yu, Kihyuk Sohn, Dimitris N. Metaxas, and Manmohan Chandraker. Reconstruction for feature disentanglement in pose-invariant face recognition. *CoRR*, abs/1702.03041, 2017.

[rad]

- [RMC] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*.
- [SGTZ19] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. *CoRR*, abs/1907.10786, 2019.
- [SLJ⁺14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.
- [SSG⁺18] Zhixin Shu, Mihir Sahasrabudhe, Riza Alp Güler, Dimitris Samaras, Nikos Paragios, and Iasonas Kokkinos. Deforming autoencoders: Unsupervised disentangling of shape and appearance. *CoRR*, abs/1806.06503, 2018.
- [SSK18] Ron Slossberg, Gil Shamaï, and Ron Kimmel. High quality facial surface and texture synthesis via generative adversarial networks. *CoRR*, abs/1808.08281, 2018.
- [vdOKK16] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *CoRR*, abs/1601.06759, 2016.
- [ZCPR03] Wen-Yi Zhao, Rama Chellappa, P. Jonathon Phillips, and Azriel Rosenfeld. Face recognition: A literature survey. *ACM Comput. Surv.*, 35:399–458, 12 2003.
- [ZPIE17] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017.
- [ZSQ17] Zhifei Zhang, Yang Song, and Hairong Qi. Age progression/regression by conditional adversarial autoencoder. *CoRR*, abs/1702.08423, 2017.
-