TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

Future of Clothing, A Technological Approach

By:
Dinesh Devkota (073/BCT/516)
Maharshi Bhusal (073/BCT/522)
Rajat Parajuli (073/BCT/532)
Sushant Thapa (073/BCT/547)

A PROJECT WAS SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE BACHELOR'S DEGREE IN COMPUTER ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
LALITPUR, NEPAL

APRIL, 2021

APPROVAL LETTER

The undersigned certify that they have read, and recommended to the Institute of Engineering for acceptance, a project report entitled **"Future of Clothing, A Technological Approach"** submitted by Dinesh Devkota, Maharshi Bhusal, Rajat Parajuli and Sushant Thapa in partial fulfilment of the requirements for the Bachelor's degree in Electronics and Computer Engineering.

Supervisor: **Dr. Jyoti Tandukar**, Associate Professor
Department of Electronics and Computer Engineering
Institute of Engineering, Pulchowk Campus

Internal Examiner: **Daya Sagar Baral**, Lecturer
Department of Electronics and Computer Engineering
Institute of Engineering, Pulchowk Campus

External Examiner: **Dr. Pradip Paudyal**, Deputy Director
Nepal Telecommunication Authority

DATE OF APPROVAL: 15th April 2021

COPYRIGHT

ACKNOWLEDGEMENT

We would like to express our gratitude to the Department of Electronics and Computer Engineering of the Institute of Engineering, Pulchowk Campus for providing a platform for exchanging knowledge and developing one's personal creativity. All guidance and resources provided by the college has been crucial in our vision that we present today. By assigning a major project as part of the fulfilment of the Bachelors' Degree in Computer Engineering, the Department has helped us develop technical skills and convey the necessities in handling real life projects in the future.

We are greatly indebted to Dr Jyoti Tandukar for being our supervisor and guiding us towards a feasible project roadmap. His guidance, experiences and expertise have been a boon for our group and have crucial in developing our project to the stage it has reached.

We would also like to extend our deep and heartfelt gratitude to Mr. Shreeniwas Sharma, CEO of Alternative Technology and everyone else at the organization who helped us by providing resources and assistance during the entirety of this project.

ABSTRACT

Technology has been harnessed in many ways to make the lives of people easier and more convenient. Advances in tools of 3D manipulation and abundance of research in the field of Machine Learning has allowed us to built the application “The Future of Clothing: A Technological Approach” . We aim to use technology to improve the experience of custom designed clothing. Users of this application can manipulate a simple 3D human model to approximate a figure of choice and apply various designs of the clothes. Alternatively, we have provision of applying and testing brand new designs generated solely by a Generative Adversarial Network (GAN).

TABLE OF CONTENTS

List of Figures

List of Abbreviations

CNN Convolutional Neural Network

DCGAN Deep Convolution Generative Adversarial Network

GAN Generative Adversarial Network

JPG Joint Photography Expert Group

LSGAN Least Square Generative Adversarial Network

PGGAN Progressively Growing Generative Adversarial Network

PNG Portable Network Graphics

ReLU Rectified Linear Unit

StackGAN Stacked Generative Adversarial Network

StyleGAN Style-based Generative Adversarial Network

TIFF Tagged Image File Format

WGAN Wasserstein Generative Adversarial Network

1. INTRODUCTION

The main theme of this project is to provide an inexpensive and user-friendly solution to the prospect of customized clothing. For the layperson, the most customization that one can do is supply a design to be printed on a T-shirt, and try the T-shirt on when T-shirt is made. Such custom printing can be expensive, if there are only a few pieces of garments printed. And even then, there is a good chance that the person may not be satisfied with the design.

To understand the problem that we are looking to solve, we need to briefly look at the steps in making a T-shirt.

1.1. Steps involved in making an apparel

Without diving into great detail, the basic steps involved in manufacturing an apparel are listed below. An example of T-shirt production is as follows [1]

1. **T-Shirt Design**

   The first step in making a t-shirt involves creating a design(usually done on paper). A sketch is created that depicts the angles and details one would like the final shirt to have. It does not need to be accurate, as long as one can show what is the needed design is. Designing a shirt with the end goal in mind helps the pattern maker know what is intended. This step also involves deciding what kind of aesthetics the shirt will have.

2. **Pattern Making**

   Once the designs are completed, it is sent over to the pattern maker. The pattern maker will take the designs and create patterns. Patterns in this context mean the pieces that will be cut and sewn together. This pattern maker will take the sketch and create patterns based on the provided instructions.

3. **Sample Cutting**

   Once the patterns are completed, they are printed on a plotter that creates the outline for the sample cutter. The patterns ,i.e. the pieces are cut to the right specifications.

4. **Sewing The T-Shirt**

After the fabric is cut out, it is ready to be sewn. The cut pieces are now the shape of the patterns and multiple sewing machines are used to finish the design. The process is repeated until the size, fit and shape is satisfactory.

5. **Mass Production Cutting**

   As soon as the created shirt or apparel design is approved, the patterns are finalized and sent into the production process. The patterns of finalized dimensions are produced in large volumes.

6. **Mass Production Sewing**

   Once the pieces are all cut out, they are sent to the sewing factory. As long as the patterns and cutting was accomplished correctly, the sewing process should be straightforward.

7. **Order Fulfillment/ Clothing Line Distribution**

   The last step in the operation is order fulfillment and distribution of the finished clothing. This includes the actual delivery of the product.

1.2. Inconvenience for the average user

A user who is not skilled in design and sewing cannot hope to obtain a desired garment unless he/she has access to resources only available in a factory. Even then, the prospect of customization for a single user or a small group is an expensive affair. The specifics of the aforementioned inconveniences can be enumerated as following.

1. **Tshirt Designing**: It is not an easy task to design a garment, especially the aesthetic part. It usually takes years of practice and experience to come up with designs that can be printed onto a garment. Even though a person may have good ideas, he/she can be run out of ideas after a while.

2. **Customising**: If the end goal of creating customized tshirt or apparel is trying on oneself, then all the steps in the previous sub section, are very impractical. One cannot hope to make a design, take a measurement of oneself, sew an apparel and then decide on whether it is a desirable choice. The problem compounds exponentially when the user wants to try different patterns.

   For argument's sake, let us consider that we are able to obtain the measurements for a user, have a blank apparel where we can try various aesthetics and design. The solution is very short sighted as it does not address the case when

(a) A different user wants to try custom clothing

(b) The user undergoes a considerable physical change such that the old measurements are not valid

(c) The user wants to try custom clothing and design on an apparel of a different style. Whether that difference is subtle or noteworthy, custom clothing in the traditional sense does not help to capture all the nauances.

1.3. Solution proposed by our System

The main aim of the project is to create a system whereby a user can virtually wrap a computer generated cloth to a computer generated body of the end user. The physical body of the subject is based on the customization by the users themselves. Not only is the clothing wrapped around a user defined body wireframe, but the clothing and the pattern of clothing is also user defined providing a high degree of customization from the perspective of the user. Patterns and designs used in the clothing can be generated by using a GAN which will synthesize brand new designs and patterns based on the user's preference. Such a GAN can be trained by supplying training data accordingly.

1. **Aid in Design**:

   Our system helps in choosing a design for the user by

   (a) Selecting from a wide range of already available designs. This could have designs sorted by popularity based on a rating system.

   (b) Uploading a user preferred design. The system can integrate designs supplied by the user. The designs would be required to meet a few basic guidelines to be compatible with the system.

   (c) Generating completely new designs using Artificial Intelligence. This aspect of the system creates entirely new designs with the technology known as Generative Adversial Networks.

2. **Aid in Modelling**

   Our system aids in customizing a 3D model to reflect the physique of the user. The appeal of custom design is the ability to try a design on oneself. So, by virtually presenting a model that accurately models the user, the user can focus on the design aspect. Virtually testing apparels before having them manufactured also has the additional benefit of making sure that a user is fully satisfied with the design. This is not only frugal in terms of finances and resources, but also a pleasant experience for the user.

2. LITERATURE REVIEW

Generative Adversarial Nets(GANs) are a class of machine learning frameworks designed by Ian Goodfellow and his colleagues in 2014[2]. Two neural networks contest with each other in a game (in the form of a zero-sum game, where one agent's gain is another agent's loss). The two networks are a Generative network (Generator) and an Adversarial network (Discriminator).

The generative network is a simple feed forward network which takes in a simple vector input (generally Gaussian Noise hereby simple called noise) and then operates on it to produce a vector. The introduction of the noise gives us the element of randomness in the vector output. The feedback for the Generator is obtained from the result of both Generative network and Adversarial network.

The adversarial network is the second network in GAN system. It is a classifier generally based on Convolutional Neural Networks (CNNs). [3]. The feedback for training the adversarial network is obtained from the network itself and the likelihood of correctly identifying the fake created by the Generator.

The Generative and Adversarial networks compete to fool one another. For each epoch, the Generator generates new images based on the input noise. The Discriminator learns from the training data. It then uses that information to check the images produced by the Generator. The real image is classified as true for the discriminator while the generated image is classified as false. Here, we consider the image from the data set as real or true, and image generated by generated by Generator as fake or false. Loss is then calculated. The calculated Loss is used to update both the discriminator and the generator. If all goes well, then after a few epochs, the generator is able to fool the discriminator with a generated image. The generator then generates lifelike and real images.

Vanilla GANs with their fully connected networks were not equipped with the needed complexity to create the desired architecture for high quality geometric shapes as output. Several other variations of GANs, each of which had different strengths with regard to the type of output required were studied.

Article on generation of animation faces, which is similar in structure to motifs and designs[4] claims that such images are difficult to create using simple GAN architectures. The article goes on to explain that the creation of faces is faster and more accurate using a variant of GAN called StyleGAN. StyleGAN[5] is a Style-Based Generator Architecture, which pro-

vides an alternative architecture for generative adversarial networks, borrowing from style transfer literature. This architecture leads to an automatically learned, unsupervised separation of high-level attributes. However,for styleGAN to be effective, the data set needs to have feature sets. Since the extraction of feature sets could be a project on its own, it was beyond our current scope and we did not settle with StyleGAN.

PGGAN was developed by a group of researchers at NVIDIA Corporation[6]. Althought, it scales well for outputs of 512*512 pixels, for relatively lower resolution outputs, it is slightly more complex and takes longer to train.

DCGAN is a variation of GAN which is based on Deep Convolutional Architecture[7]. It uses Deep Convolutional architecture for the discriminator as well as Transpose Convolutional to scale the image from the noise in the generator. The paper on DCGAN presents it as a topologically constrained variant of conditional GAN. The main benefit of using DCGAN over classical GAN is that it is more stable to train and very useful in training unsupervised image representations. Furthermore, GANs are difficult to scale using CNN. The paper proposes to replace pooling layers with strided convolution i.e. no fully connected layers and replace it with convolutional counterparts. It further proposes to use LeakyReLU and batch normalization in all layers of discriminator while using ReLU and batch normalization in all layers of the generator(except output).

WGAN[8] and LSGAN[9] provided similar but since their outputs were similar to DCGAN while being more complex, we went with the latter as the choice of our architecture.

A very innate problem of both DCGAN and vanilla GAN is that they are unable to be scaled to higher degree. GAN is structurally an unstable architecture and it needs a lot of work to achieve momentary stability. Furthermore the complexity of any image based neural network increases exponentially with the increase in the size of the input and output. Thus the GANs which consist of large image input and output are very unstable and require a lot of processing power along with other complications in the training method. A new system was required to increase the size and quality of the image produced by the GAN. The paper on StackGAN[10] proposes to stack the output of smaller GANs to the larger GANs to increase the dimension of the output by two times.

A multitude of research papers and projects exist that involve the construction of three dimensional human body meshes for various purposes. Greater commercial availability of affordable depth sensors and cameras have led to more and more contributions from people in this field. From this, lots of people have undertaken projects concerning the parameterization of human body models. One such paper by Chih-Hsing Chu et al. [11] describes training regression functions to correlate semantically significant values to manipulate and generate

new human models from available assets by employing the use of principal component analysis on feature curves and segment patches drawn onto the model and thus modified [11]. For just one semantic such as the waist size, however, a simple linear regression between waist-variant models was used.

A range of datasets are available that describe the human body shape in three dimensional space. One of them was the K3D-hub Dataset [12], a freely available 3D human body dataset taken from Microsoft XBOX 360. However, due to it having a lot of imperfections in data such as missing fields, noise in point positions and holes, it did not fit mesh modeling and manipulation work. The dataset used in the project instead came from the MPII Human Shape dataset [13] [14], a freely available resource that is used for human shape space building, manipulation and evaluation. After handling the pre-processing of the dataset using MATLAB programming platform, a median of the outputs were used to construct the base model to apply parameterization.

3. SYSTEM IMPLEMENTATION

3.1. Basic Structure of System

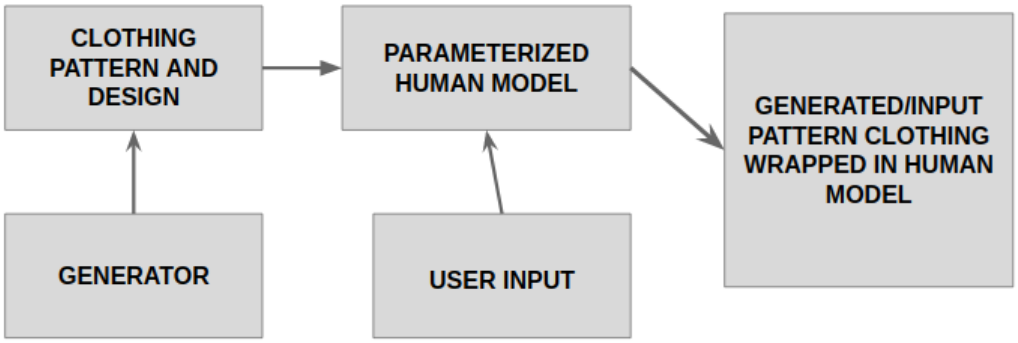

Figure 3.1: Basic Architecture of System

Our system is composed of various components, with the major ones shown in figure 3.1 . The generator will generate the patterns which are wrapped in the human model, parameterized by the user input.

The use case diagram shows how the user interacts overall with the system and what external entities are present. The generator and wrapper are shown as part of the system, although being actors. The Data Flow Diagram shows how the data flows in the system.

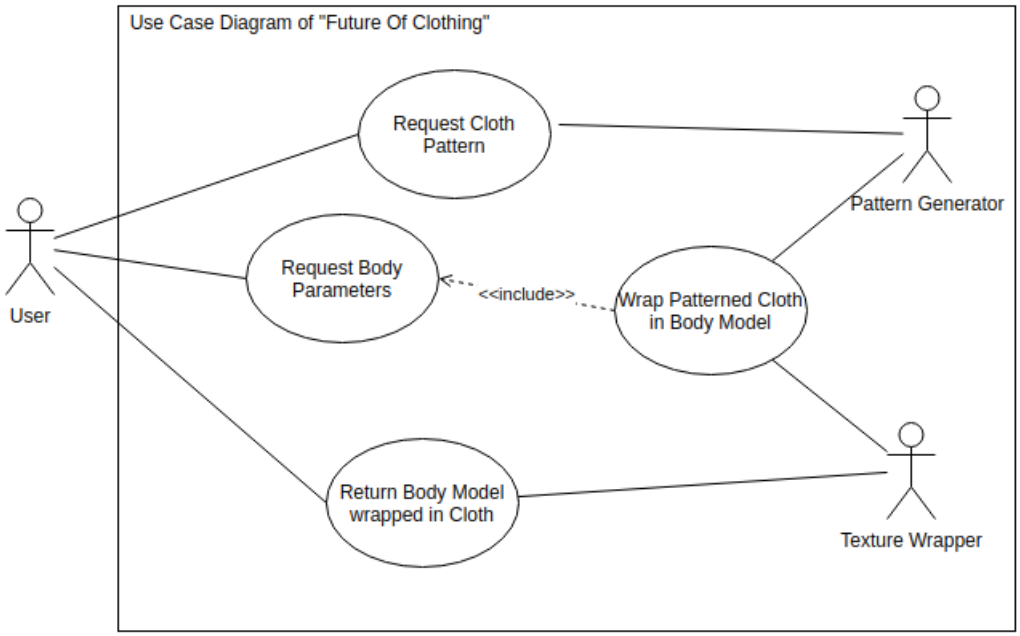

Figure 3.2: Use Case Diagram of the system

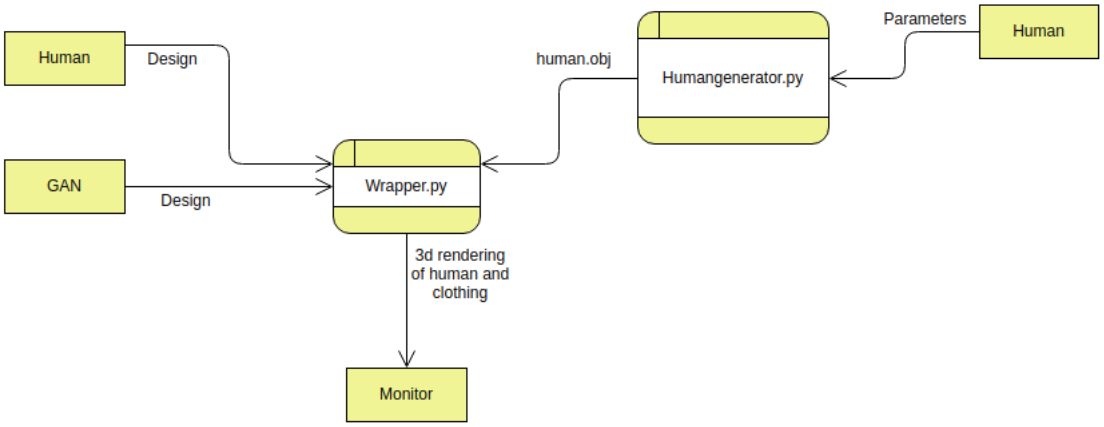

Figure 3.3: Data Flow Diagram of the system

3.2. Motifs Generation using GAN

We used a modified form of a DCGAN[15] to generate images. Keras with Tensorflow was used as the framework of choice for development. The model takes in a 150 bits long noise vector as input and outputs a 32*32 px motif image. The second stage of GAN takes in the 32*32 px image generated by the first model and a 150 bit long noise vector as input and outputs a 128*128 px image. The implementation is discussed in details in the following sub-section.

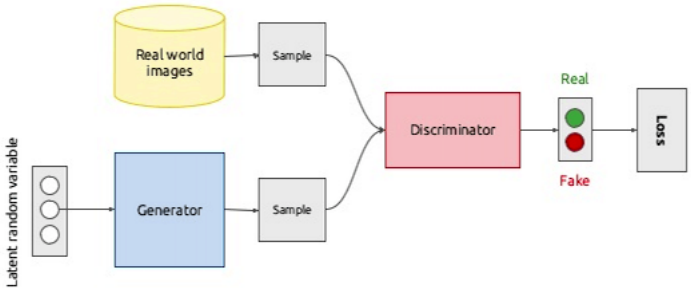

Figure 3.4: Basic Structure of GAN

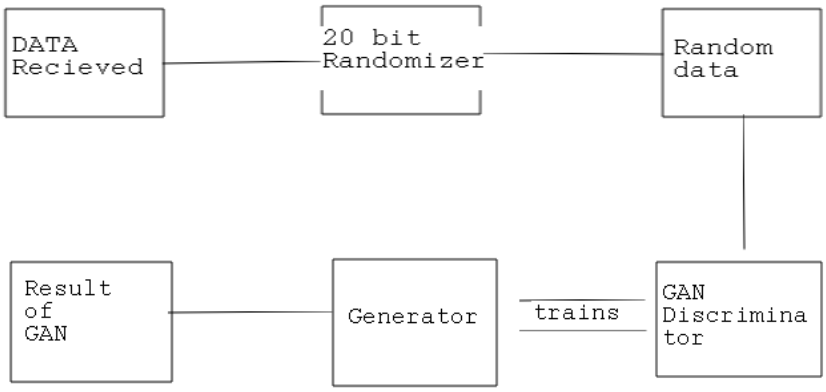

Figure 3.5: Proposed GAN pipeline

3.2.1. Data Acquisition

Data acquisition process was a relatively lengthy process in the project timeline. Due to the absence of pre-made and pre-cleaned data sets online.

Data for training was provided to us by Alternative Technology[16]. Alternative Technology is a private company that specializes in software for carpet designs,customization and photo-realistic rendering. The received data consisted of about 8000 images of motifs, spread in a directory, with 77 further sub-directories. The directories also consisted of unrelated files (such as text files), which needed to be identified and removed. The data was unclean, disorderly and small in size, so they required some operations before it was fit for training. Most of our target image were of size 512*512 pixels in PNG file format.

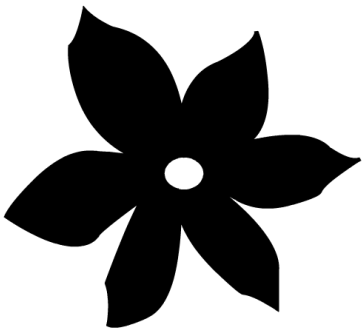

Figure 3.6: A sample of input data

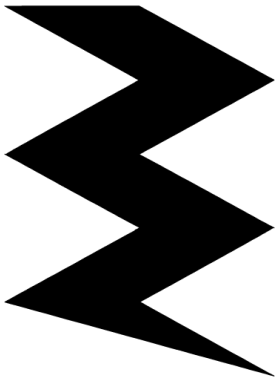

Figure 3.7: A sample of “zigzag line” input data

The data set had the largest number of images in “zigzag lines” motifs which numbered around 800 and thus was used for the preliminary experimentation.

The data set prepared in this manner is hereby referred to as “local” data set, as it is prepared for training locally.

3.2.2. Data Cleaning

The received data was fist cleaned of unwanted files. Since, the directory system was incoherent (with images being in different levels with respect to the root directory), the data was organized to make later processes easier. The images in the deeper level of directory were moved upwards to the directory directly below the root.The cleaning of the data could be then automated using scripts. Simple bash scripts were used to delete text files from the data. Next, the directory structure was reconfirmed. The directories below the second level of root directory were cleaned.

Some of the obtained images were in JPG or TIFF file formats and the size of the images was also incoherent ranging from very wide images to very tall images.Thus JPG and TIFF were converted to PNG file format and finally, the images were checked for the size. The relatively tall and wide images were cleaned

The duplicate images were then detected and deleted. The remaining images were up-scaled or down-scaled to 512*512 pixels in size.

3.2.3. Synthetic Data

After the cleaning process, the size of the data decreased and around 4000 individual images remained. Considering the complexity of the generator and the time to train it, the data was deemed insufficient. Thus the data was processed and synthetic data were added to the data obtained after cleaning. Here on wards, the datset without synthetic data set is called unmodified data set.

Simple procedures were used to synthesize new data from the existing data. Four basic transformations were used. A six bit random binary number was generated and used as switch for applying different transformations. The total variation becomes equivalent to 23-bits noise vector. The variation was considered to be satisfactory for synthesis. Only about 60 samples were synthesized from each image using the aforementioned procedure. The acquired result was a collection of about 92000 image with small variation among them. This ensured that the data was not entirely random without any features.

They are as follows:

- Stretching and Squashing

  The image was stretched and/or squashed by a random pixels from 0 to 15 per cent of the total size of the image in steps of three. for a 512 px image it provides 25 different random conditions.

- Cropping

  The image was cropped up to 10 per cent of the total width of the image randomly in steps of one. This gave us about 50 different random conditions in each side, totalling to 100 conditions.

- Transpose

  Transpose is possible in two ways, horizontal and vertical. Horizontal transpose flips the image horizontally, while vertical transpose flips image vertically.

- Rotate

  The images were rotated clockwise from 0 to 90 degrees in steps of one. The choice of step count and total rotation depended on the volume of synthetic data required for the current scope of the project.

3.2.4. Stage I: Generator

The GAN was implemented in two stages. The first model is a DCGAN. This is a simple DCGAN which takes in 150-bit noise vector as input and converts the noise vector to a 32*32 px image.

The 150-bit noise vector is fed into a fully connected layer which outputs a 8*8*256 bit vector. This is then reshaped to shape (8,8,256). It is successively fed into two deconvolutional sections. Each deconvolutional section consists of three layer. A Conv2Dtranspose is used as the deconvolutional layer. Each of the sections has the same padding and no bias in this layer. The first Section has a stride of 1 while the second has a stride of two in this layer. In each section, this layer is followed by a Batch Normalization layer, a ReLU layer and a dropout layer with dropout rate of 0.3. The result is then passed through a deconvolutional layer with no bias, same padding, stride two and tanh activation function. The obtained output from this layer is the output of generator or generated image of size 32*32px. The architecture is discussed in the diagram.

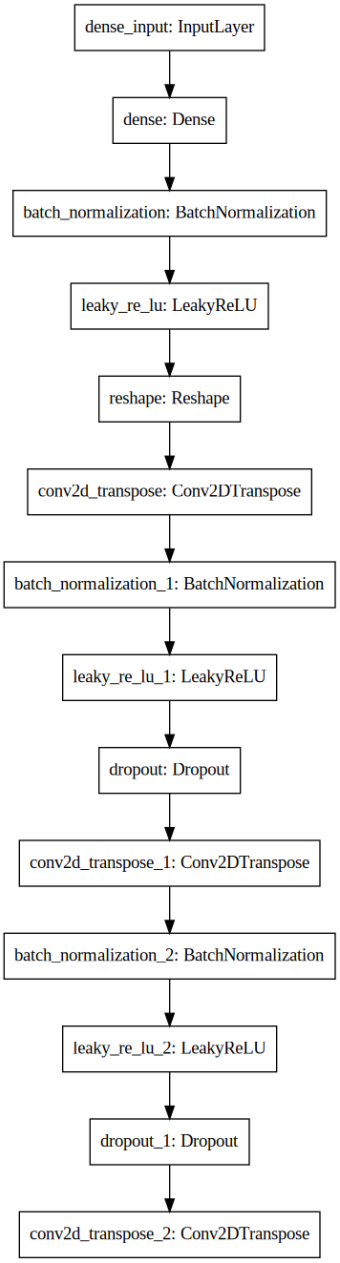

Figure 3.8: Architecture of Stage I Generator

3.2.5. Stage II: Generator

The second stage of the GAN is a modified form of DCGAN. It is based on the Generator from StackGAN. The basic design of the GAN is a DCGAN with residual blocks to reinforce the design from the previous result in each step. With exception of addition of the residual blocks, other parts are similar to stage I DCGAN. The second Stage DCGAN first takes the 150 bit long noise vector and output of Stage I Generator as input. Since the dimensions of the inputs are not same, they cannot be added directly. Thus we convert the noise vector

into a (32,32,1) vector using fully connected layers to make its dimension same as that of the Generator output. These two vectors i.e the converted noise and the output of stage I generators is then added. It is then passed through layers of Residual blocks. The output of 128*128 pixels resolution is then received. The architecture is shown in the diagram.

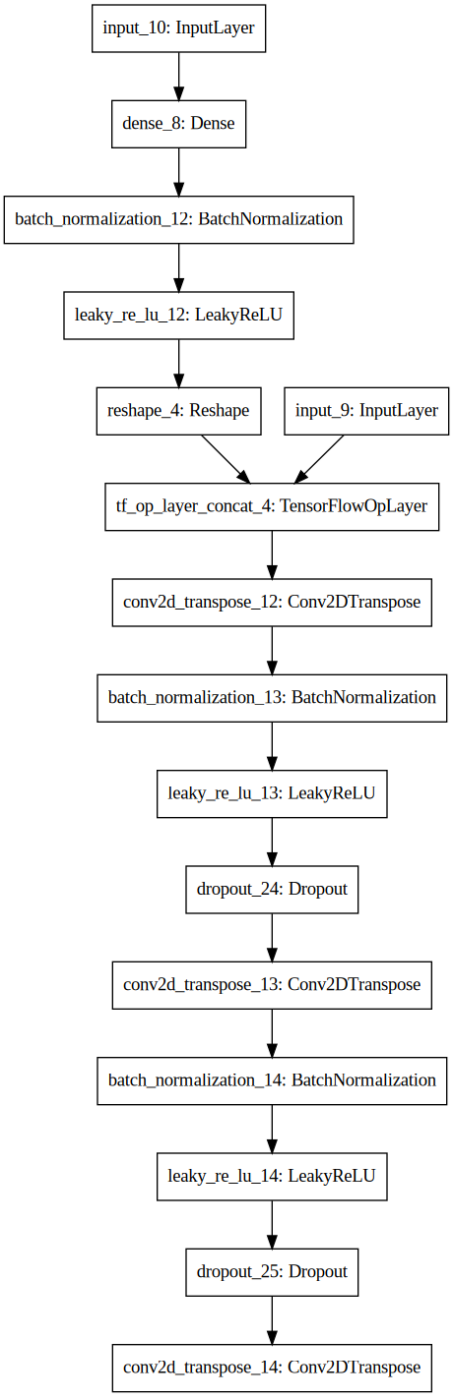

Figure 3.9: Architecture of Stage II Generator

3.2.6. Discriminator

The discriminator of both stages are similar in design and architecture. The discriminators used are convolutional classifiers. The image is obtained as an input. The image is then passed through several layers of convolutional section to get a vector with shape (8,8,128) for stage I or (8,8,256) for Stage II. A detailed architecture for both discriminators is in the diagram. Stage I discriminator has three convolutional section. Each section except the last has a convolutional layer,a Leaky ReLU layer, a Batch normalization layer and a Dropout layer for processing. The result is squeezed to a one dimensional vector. It is then passed to a fully connected layer with one output. The architecture is discussed in the diagram.

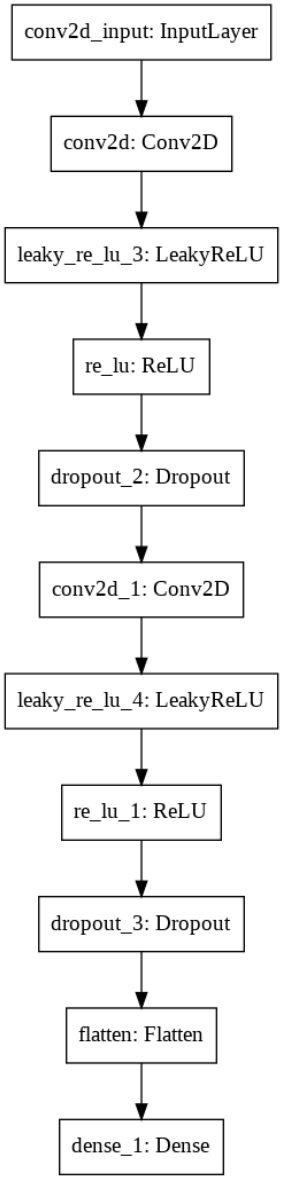

Figure 3.10: Architecture of Stage I Discriminator

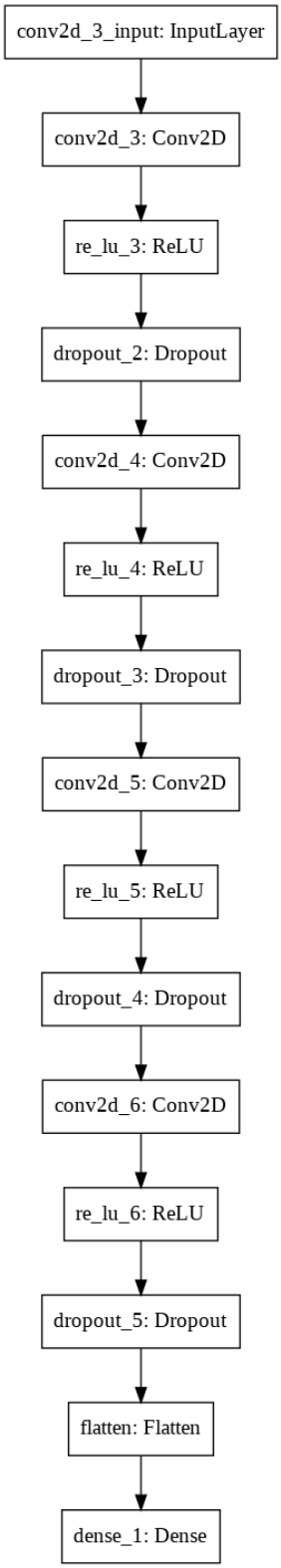

Figure 3.11: Architecture of Stage II Discriminator

3.2.7. Loss

Loss generation was done by a custom function. Binary crossentropy was used to calculate the loss from each generated image. The binary crossentropy for images generated by the generator as well as the image from database was used to calculate loss for the Discriminator.

$$discriminator\ loss = crossentropy(zero\_like, fake) + crossentropy(one\_like, real)$$

Similarly binary crossentropy was used to calculate loss for the generator.

$$generator\ loss = crossentropy(ones\_like, fake)$$

3.2.8. Optimizer

Adam[17] was used as optimizer for both stages of GAN. For training the models, the performance was observed after a few epochs and the value was adjusted accordingly. This was done for better stability and variation from the model. The value for adam optimizer was adjusted from 1e-4 to 5e-6 over the range of 15 epochs in successive three epochs. Each epoch took roughly two minutes to complete for 32*32 px image generation. For the training of the second stage Generator, the optimizer value was adjusted from 1e-5 to 7e-6 over the range of eight epochs in successive two epochs. For 128*128 px motif generation each epoch took about 15-20 mins.

3.3. Generation of Human Body

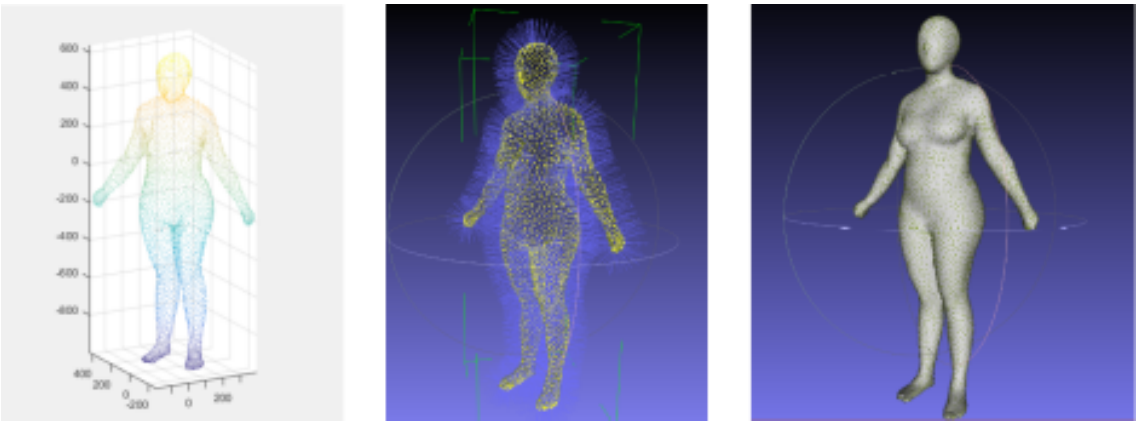

Figure 3.12: (Left) the extracted point cloud. (Middle) The calculated point normal in MeshLab. (Right) The mesh resulting from the screened Poisson surface reconstruction

From the MPII Human Shape data set, the collection of 4308 models were converted from a .mat format (binary data containers that are used to include variables, functions, arrays, and other information) which contained a 6449x3 matrix of vertices in three-dimensional Cartesian coordinates into a .ply standard polygonal format for the representation of point clouds. These point clouds contained only the basic positional information of the vertices that defined the human body shape. The generated .ply files were imported into a commercially available tool called MeshLab where the point normals were generated. Then the surface was constructed using Screened Poisson Surface Construction method, both of which are available tools in the program. The model was then exported as a Wavefront object (.obj) and ready for further editing.

3.4. Displaying Models

Figure 3.13: Human Model rendered in the webapp

Any model that we can view must be lit by a light and be viewed through a camera.The human model is stored as a wavefront object (.obj), which is loaded via one of the functions provided by three.js.

3.4.1. Lighting

The model is lit by a hemisphere light. A hemisphere light is positioned directly above the scene, with color fading from the sky color to the ground color.This light cannot be used to cast shadows. The sky color has a value of 0xB1E1FF(light blue), a ground color of 0xB97A20 (brownish orange) and an intensity of 1.

3.4.2. Camera

The most common camera in three.js and the one used in our application is the Perspective Camera. It gives a 3d view where things in the distance appear smaller than things up close, so it is useful for realistic renderings.

The Perspective Camera defines a frustum, which is a solid pyramid shape with the tip cut off.

A Perspective Camera defines its frustum based on four properties,

1. **near** : It defines where the front of the frustum starts.
2. **far**: It defines where it ends.
3. **fov** : The field of view, defines how tall the front and back of the frustum are by computing the correct height to get the specified field of view at near units from the camera. This is measured in degrees.
4. **aspect** : It defines how wide the front and back of the frustum are. The width of the frustum is just the height multiplied by the aspect.

3.5. Customizing the Model

The model can be customized using a slider. The model changes from some minimum value of body size to a maximum value according to the value of the slider. In the current version of the project, the only body parameter that changes is the weight. The key assumption made here was that the height does not change with the increase in the size of the model.

From the implementation perspective, the main idea here is that the model itself does not change in the number of vertices, or the way those vertices are joined to make faces. The

model having the greatest weight is same model having the least weight and as the slider is moved, the vertices of the 3D model are appropriately displaced. The relationships between the vertices to form the faces do not change, but since their position in the 3D coordinate system has been altered, the physical appearance can be seen differently.

For each vertex, v, we can consider the minimum value to be v_{min} and maximum to be v_{max}.

The value of the slider then varies from 1 to s_{max}.

Thus, for any intermediate value of the slider (say, at s) we can calculate the value of the vertex using the formula,

$$v = \frac{v_{max} - v_{min}}{s_{max} - 1} * (s - 1) + v_{min}$$

3.6. Clothing The Model

After the Model is lit with proper lighting and viewed with appropriate camera, the model should be clothed so that the pattern can be visualized in the model.

The user selected pattern or a pattern generated from the GAN was be wrapped in an object resembling a T-shirt. Since the shirt is not a primitive shape, and consists of many polygons, we cannot simply map a a texture onto a multi-polygon 3D object. Every single surface has to be mapped onto it. Tools like Cinema4D and Blender have a few tools that can help us with the process.

Blender is a free and open source 3D creation suite which supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, video editing and 2D animation pipeline.[18] Blender provides various mapping algorithms.[19] The mapping algorithms available are

1. **Unwrap**: The mesh surface is flattened by cutting along seams. This is useful for organic shapes. The algorithm begins by selecting all faces to be unwrapped in the 3D View. With the faces selected, it can be then unwrapped. In this method, the faces of the object are unwrapped to provide the “best fit” scenario based on how the faces are connected and will fit within the image. The algorithm takes into account any seams within the selected faces. If possible, each selected face gets its own different area of the image and does not overlap with any other UV faces. If all faces of an object are selected, then each face is mapped to some portion of the image. The unwrapping

can be calculated by Angle Based method, which gives a good 2D representation of a mesh or Conformal method which uses Least Squared Conformal Mapping(LSCM). LSCM usually gives a less accurate UV mapping than Angle Based, but works better for simpler objects. This process requires manual intervention to define the seams for each new case, so we opted for an automated algorithm.

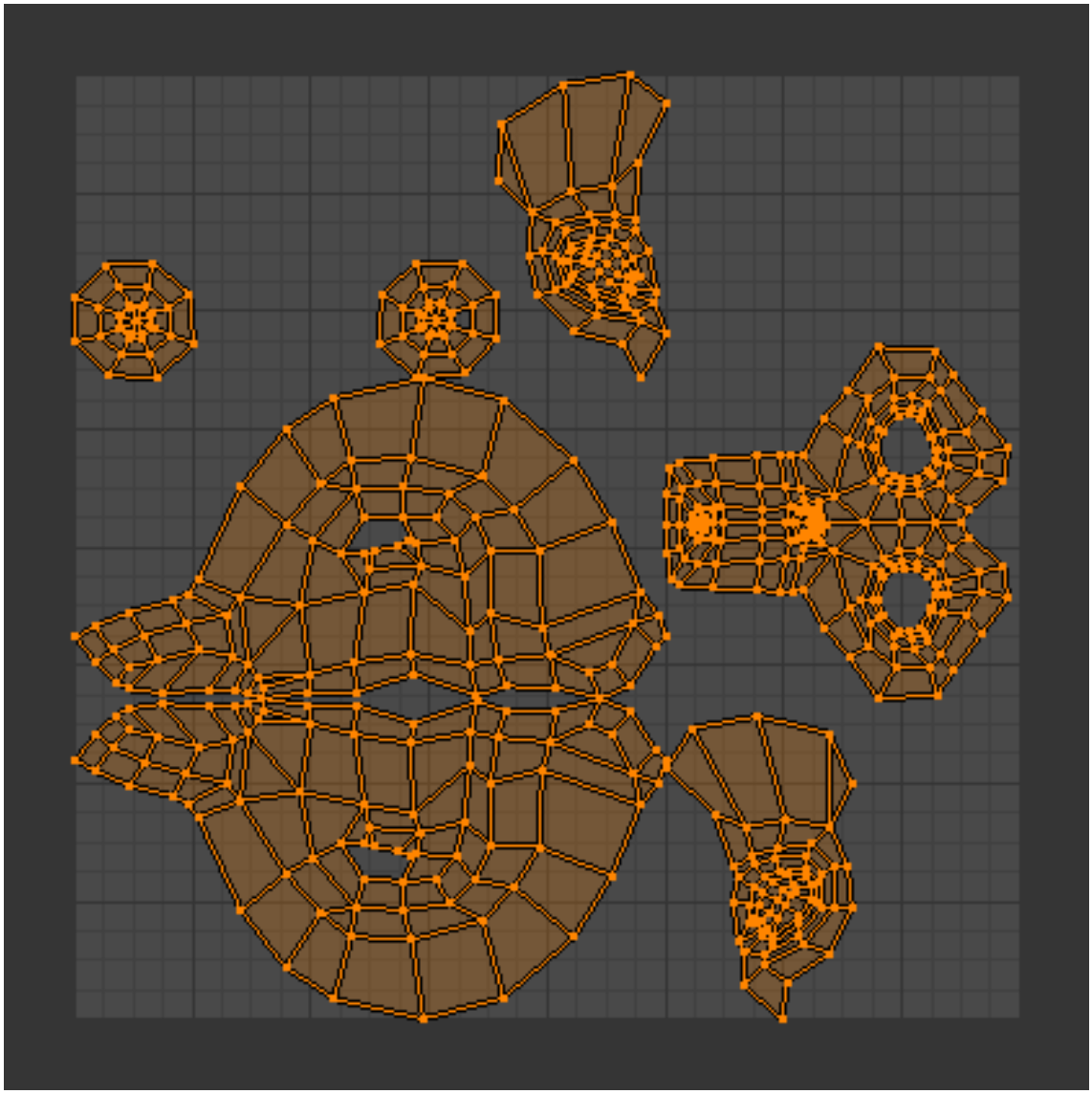

Figure 3.14: Result of Unwrap Mapping Algorithm

2. **Cube Projection**:Cube Projection algorithm maps the mesh onto the faces of a cube, which is then unfolded. It projects the mesh onto six separate planes, creating six UV islands. The Cube Size parameter in the algorithm sets the size of the cube to be projected onto.Any UVs that lie outside the (0 to 1) range will be clipped to that range by being moved to the UV space border it is closest to.If the UV map is larger than the (0 to 1) range, the entire map will be scaled to fit inside.

3. **Smart UV Project**: Smart UV Project algorithm cuts the mesh based on an angle threshold (angular changes in the mesh). This gives the user fine control over how automatic seams are be created. It is good method for simple and complex geometric forms, such as mechanical objects or architecture. This algorithm examines the shape of the object, the faces selected and their relation to one another, and creates a UV map based on this information and settings that are supplied.

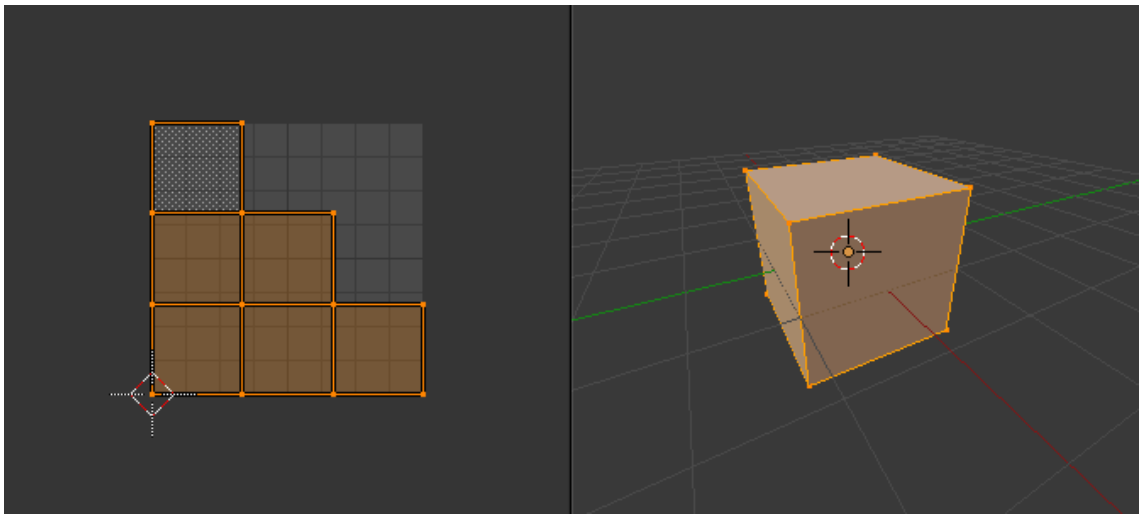

Figure 3.15: Result of Smart UV Project Mapping Algorithm

4. **Cylindrical and Spherical mapping**: Cylindrical mapping projects the UVs on a plan toward the cylinder shape, while a spherical map takes into account the sphere's curvature, and each latitude line becomes evenly spaced. It is useful for spherical shapes, like eyes, planets, etc. To unwrap a cylinder (tube) as if to slit it lengthwise and folded it flat, the view must be vertical, with the tube standing "up". Different views will project the tube onto the UV map differently, skewing the image if used. The axis on which the calculation is done manually and the same idea works for the sphere mapping as well. To unwrap a sphere, the sphere is viewed with the poles at the top and bottom. After unwrapping,an equirectangular projection is obtained. The point at the equator facing the viewer will be in the middle of the image.

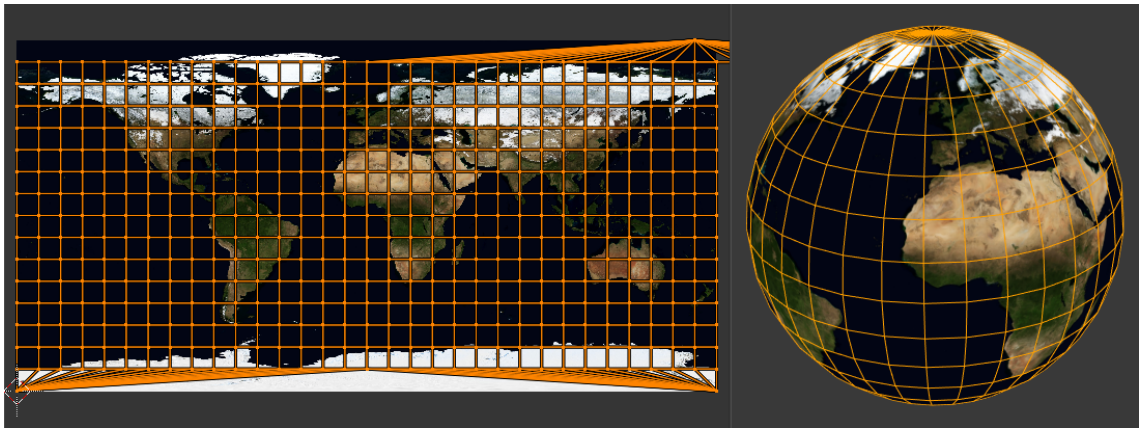

Figure 3.16: Demonstrating the use of an equirectangular image with a Sphere Projection

5. **Project from View**: ”Project from View” algorithm takes the current view in the 3D View and flattens the mesh as it appears. This option is recommended by Blender if a picture of a real object is being used as a UV Texture for an object the user has modeled. Some stretching in areas will be observable where the model recedes away from the viewer.Here, the images aspect ratio is into consideration.
   ”Project from View (Bounds)” is a special case of ”Project from View” where if the UV map is larger than the specified range, the entire map will be scaled to fit inside. Additionally, the UVs will take the images aspect ratio into consideration. If an image has already been mapped to the texture space that is non-square, the projection will take this into account and distort the mapping to appear correct.

“Project from View (Bounds)” algorithm displayed the best result which is implemented. The results of various mapping algorithms on ”shirt” object with clothes pattern are shown in the next section of the report.

4. OUTPUT

4.1. GAN outputs

Since image synthesis is an unsupervised learning process, from the perspective of the trainer, there is no objective way to determine the correctness of output. The resolution of the output is measured in pixels, even though they are not explicitly mentioned here onward. The main criteria that we have considered for evaluation of outputs are as follows:

1. **The subjective quality of Output**, where we compare the likeness of the output to that of input. Quality in this context, is how ”good” a result is to the human observer.
2. **The variation of Output** i.e. the absence of mode collapse. Generally, if the output images are very similar to each another, then the output is not preferable.
3. **The stability of training** is the capacity of the model to converge without mode collapse and frequent manual intervention.

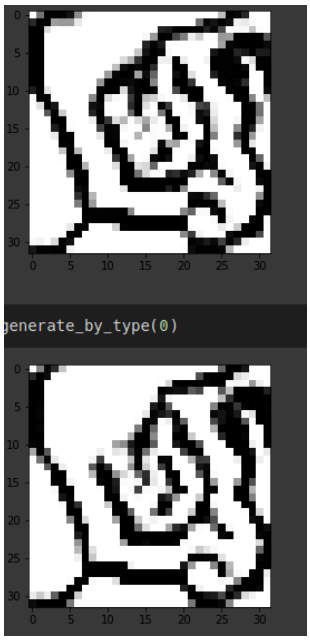

Figure 4.1: Example of mode collapse in a DCGAN experiment

Initially, we trained pure and modified versions of open source and freely available variations of GANs from various projects on Github. Out of the different types of available GANs, we trained

1. **Vanilla GAN**, as it was relatively stable and easy to experiment with during the initial stages of the project.

2. **DCGAN**, although unstable for higher resolution in the available architecture, as it provided good quality outputs for low resolution of 32*32.

Outputs were observed for architecture of Vanilla GAN with 64*64 resolution [20] and the architecture was altered accordingly to fit the single channel input in our project. A DCGAN architecture with 28*28 resolution output [15] was experimented with various architectural changes, most notably with output resolution of 32*32 and 64*64. The output for 64*64 variation of DCGAN was relatively poor and the 32*32 variation of it was accepted as the final Stage I GAN. The DCGAN architecture for 32*32 , although less stable and harder to train, had outputs of higher quality than those from the Vanilla GAN architecture of 32*32.

4.1.1. Outputs with unmodified input data

We observed the output of the the GAN structure with the zigzag patterns, ie. without any synthetic data for both vanilla GAN as well as DCGAN.

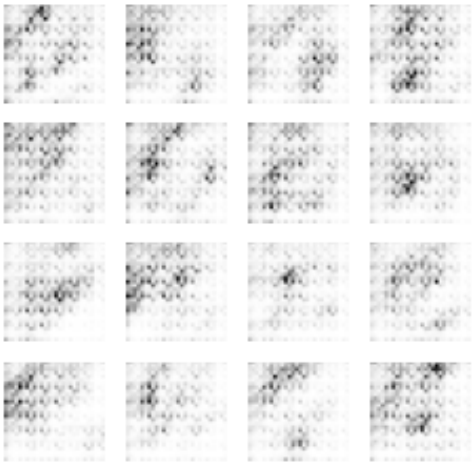

Figure 4.2: Output of Vanilla GAN with resolution of 64*64 to unmodified “zigzag” input data

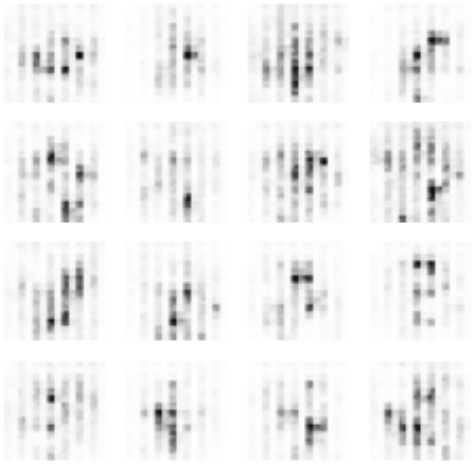

Figure 4.3: Output of DCGAN with resolution of 64*64 to unmodified ”zigzag” input data

4.1.2. Output with synthetic input data

Outputs were observed for both Vanilla GAN and DCGAN with modified inputs for the zigzag patterns in the data set. Synthetic input data were added artificially by performing various transformations on the unmodified data.

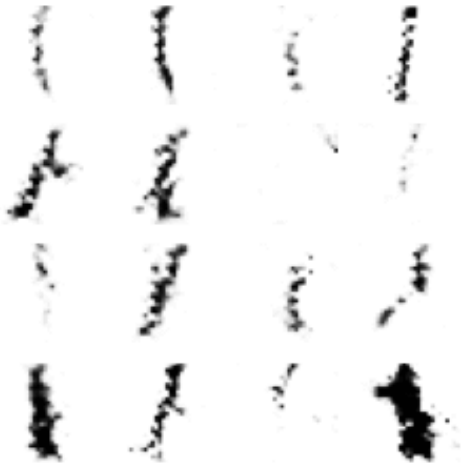

Figure 4.4: Output of Vanilla GAN with resolution of 64*64 to synthetic ”zigzag” pattern input

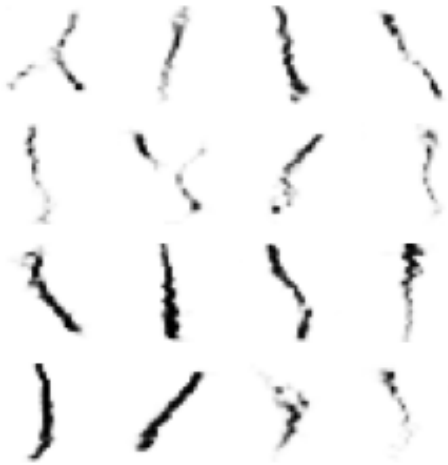

Figure 4.5: Output of DCGAN with resolution of 64*64 to synthetic ”zigzag” pattern input

4.1.3. Improved GAN Outputs

As the output of GANs for synthetic zigzag input motifs was of good quality, synthetic data was added in the entire input data set and there was an attempt to obtain result for 64*64 resolution. However, as it can be seen in the figure, the output was not of good quality, although it had good variation and had low noise.

The architecture [15] was used as a reference and modified to obtain the output for 32*32 resolution.This is the Stage I generator in StackGan approach. In this approach, the output of stage I is scaled to obtain an output of higher resolution. In our case, the output from the first generator model was of resolution 32*32 and it was scaled to 128*128. The second stage generator was a modified version of the StackGAN stage II given in the paper [10].

Figure 4.6: Output of DCGAN with resolution of 64*64 to synthetically modified input set

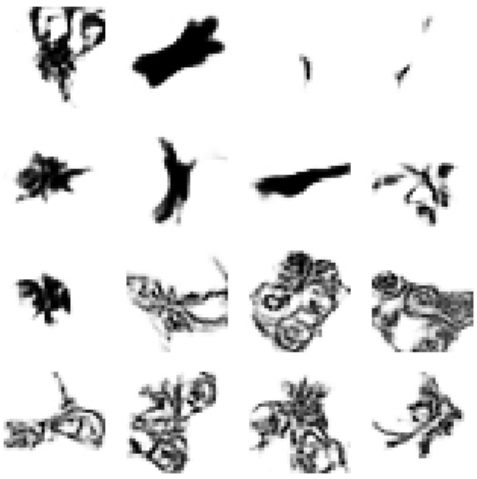

Figure 4.7: Output of Stage I DCGAN Generator with resolution of 32*32 to synthetically modified input set

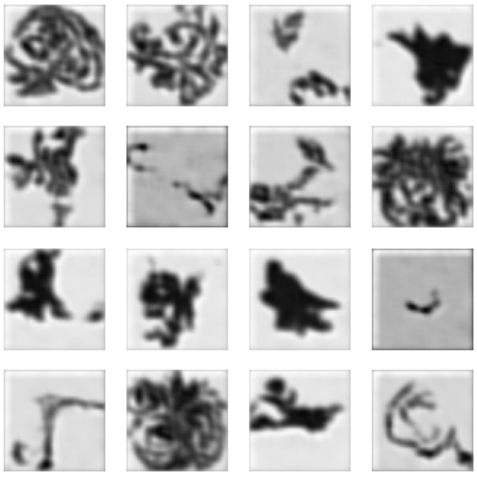

Figure 4.8: Output of Stage II Generator with resolution 128*128

4.2. Output of human body model under various size parameters

Here we can visually observe how the human model changes with respect to the “size” parameter

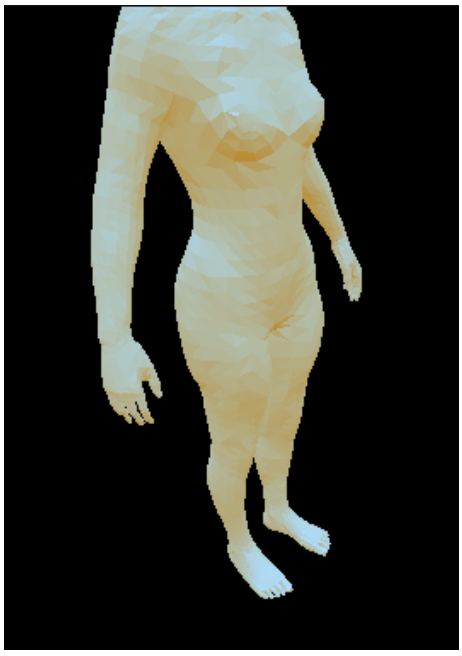

Figure 4.9: Output of human body model under minimum value of size parameter

Figure 4.10: Output of human body model under maximum value of size parameter

Figure 4.11: Output of human body model under an intermediate value of size parameter

4.3. Output of various mapping algorithm

The shirt object was wrapped with clothes texture using various algorithms offered by Blender, whose results are as follows.

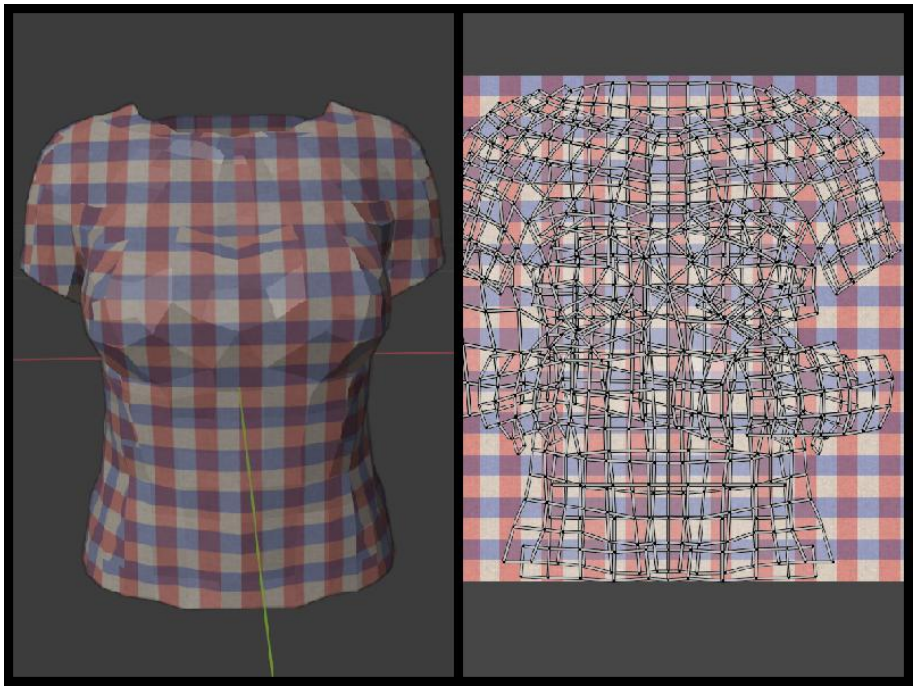

Figure 4.12: “Cube Projection” Mapping Algorithm

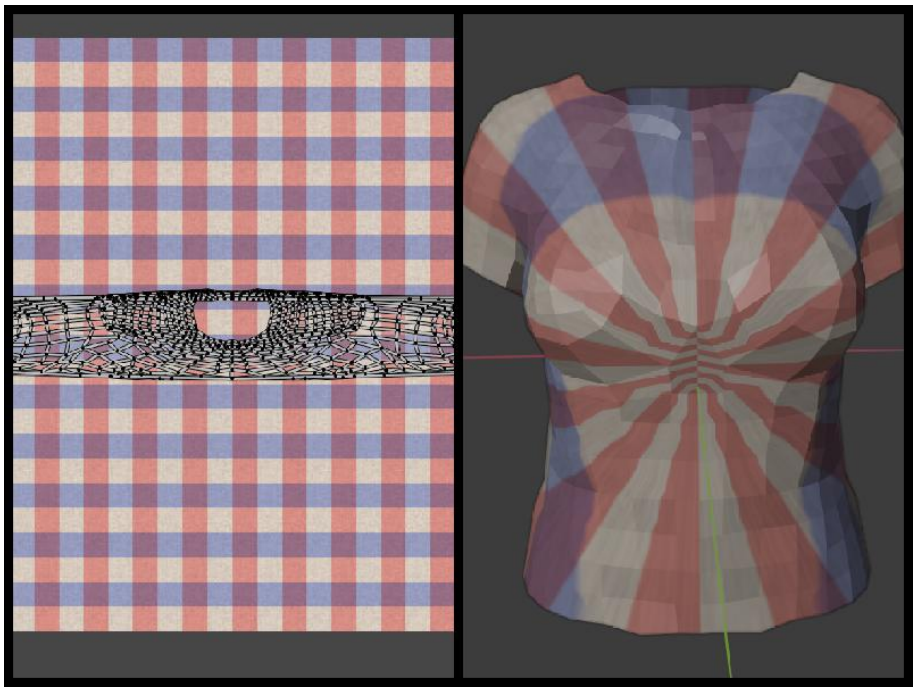

Figure 4.13: “Cylindrical Projection” Mapping Algorithm

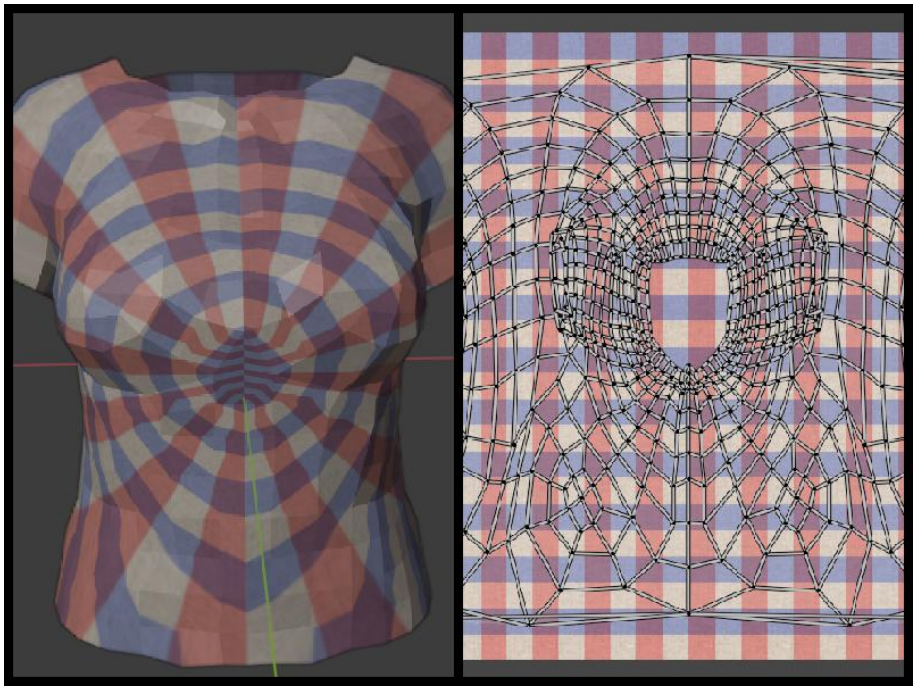

Figure 4.14: “Spherical Projection” Mapping Algorithm

Figure 4.15: “Smart UV unwrap” Mapping Algorithm

Figure 4.16: “Project From View (Bounds)” Mapping Algorithm

The ”Project From View (Bounds)” UV mapping algorithm is used in this project as it gave the best subjective results out of the available algorithms.

4.4. Clothed Model in various figures

Figure 4.17: Red T-shirt model with a pattern generated by GAN

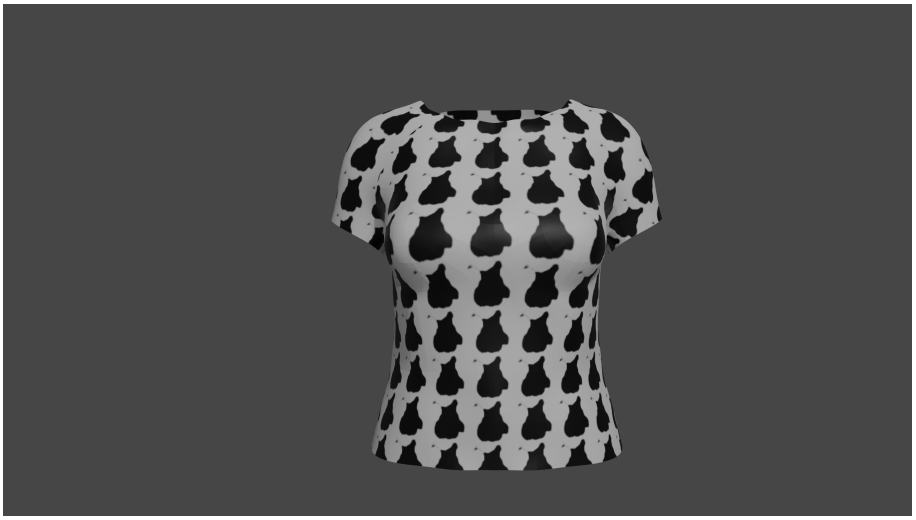

Figure 4.18: White T-shirt model with a pattern generated by GAN

Figure 4.19: Model with a pattern generated by GAN

In these figures, the humanoid model can be seen wearing a T-shirt with patterns that are generated by the GAN as well as pattern supplied by the user.

Figure 4.20: Model with a checked pattern wrapped

4.5. User Interface

The figure shows the user interface of the application. The humanoid model is shown in the centre, with a clothed object. The panel on the left is the “Design” panel, where user can select the design to be wrapped in the shirt of the humanoid model. The “AI” button causes the texture to be randomly generated by our machine learning model. The “Waist” parameter can be used to customize the size of the model in the centre.

Figure 4.21: User Interface of the system

5. LIMITATIONS OF THE PROJECT AND FUTURE ENHANCEMENTS

1. **User Review:** An improvement to the system can be made by incorporating user review and ratings in the generated patterns. This way, users can make more informed decisions based on previous opinions and reviews of other users.

2. **Recommendation System:** Techniques of recommendation systems like Content-based recommendation, Collaborative filtering, and Hybrid approaches can be used to recommend patterns more suited to the taste of the users.

3. **Finer Parameterization:** Currently, there is only one parameter that can be tweaked by the user to change the model. Additional parameters to separately change the proportions of the body can be added so as to finely tune the body based on the users' preferences.

4. **Automatic detection of user size:** Techniques to isolate the user body from a photograph and converting 2D model to 3D can be used to attempt to approximate a model. This process would automatically detect the user size and give a more realistic reflection of the user body.

5. **Improvement of GAN for higher resolution:** More variants of GAN can be experimented with and tested to attempt to increase the resolution of the output images. Experiments may also involve using a different dataset, or increasing number of data, and separately training the GAN for each category of image in the data set.

6. **Curated dataset for generator:** To better train the GAN, a more curated dataset would be needed such that the GAN may provide better results based on the dataset.

7. **Improved Cleaning Algorithm:** The cleaning algorithm can be improved so that while pasting the generated designs onto the clothing texture, there are no edge artefacts.

8. **Improved Generator Algorithm:** The generator algorithm for the GAN can be improved so that the designs generated are more consistently coherent patterns.

6. CONCLUSION

The Major Project “Future of Clothing, A Technological Approach” was completed with the wrapping of texture on a customizable 3D humanoid model. The textures that were used to wrap the 3D model were both user-provided as well as generated by GAN.

The 3D model was displayed using a Javascript framework called ”Three.js”. After supplying two different models, one for a lean humanoid and one for a slightly obese humanoid, intermediate body figures were obtained by linearly interpolating between the two extremities.

The technology that we used to generate our own design patterns for clothing texture was GAN. To be more precise, we used DCGAN because it was complex enough for us to achieve our result, yet comparatively easier to implement with lesser training time compared to other GAN variants.

We trained the network with the data provided to us by Alternative Technology. Since we needed a large volume of data, we did various operations like stretching, cropping and rotating on the available data set to generate more synthetic data for training. We trained our network on about 92,000 images. The result, which is shown in the output section of this report, is of resolution 32*32, 64*64 and 128*128 pixels. The network was trained at approximately 15-25 epochs for the lower resolution output and 8 epochs for the higher resolution output.

7. REFERENCES

References

[1] (), [Online]. Available: http://startabrandasap.com/2017/08/14/how-to-make-a-t-shirt-step-by-step-guide/ (cit. on p. 1).

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27, Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf (cit. on p. 4).

[3] K. O'Shea and R. Nash, *An introduction to convolutional neural networks*, 2015. arXiv: 1511.08458 [cs.NE] (cit. on p. 4).

[4] (), [Online]. Available: https://www.gwern.net/Faces (cit. on p. 4).

[5] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019 (cit. on p. 4).

[6] (), [Online]. Available: https://machinelearningmastery.com/introduction-to-progressive-growing-generative-adversarial-networks/ (cit. on p. 5).

[7] A. Radford, L. Metz, and S. Chintala, *Unsupervised representation learning with deep convolutional generative adversarial networks*, 2016. arXiv: 1511.06434 [cs.LG] (cit. on p. 5).

[8] L. Weng, *From gan to wgan*, 2019. arXiv: 1904.08994 [cs.LG] (cit. on p. 5).

[9] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley, *Least squares generative adversarial networks*, 2017. arXiv: 1611.04076 [cs.CV] (cit. on p. 5).

[10] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. Metaxas, *Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks*, 2017. arXiv: 1612.03242 [cs.CV] (cit. on pp. 5, 27).

[11] C. C. W. Chih-Hsing Chu Ya-Tien Tsai, "Exemplar-based statistical model for semantic parametric design of human body," 2010 (cit. on pp. 5, 6).

[12] S. C. Zongyi Xu Qianni Zhang, "Multilevel active registration for kinect human body scans:from low quality to high quality," pp. 1–14, 2017 (cit. on p. 6).

[13] T. H. Leonid Pishchulin Stefanie Wuhrer, "Building statistical shape spaces for 3d human modeling," (cit. on p. 6).

[14] H. H. Michael Kazhdan, "Screened poisson surface reconstruction," 2013 (cit. on p. 6).

[15] (), [Online]. Available: https://www.tensorflow.org/tutorials/generative/dcgan (cit. on pp. 8, 25, 27).

[16] (), [Online]. Available: http://www.alternative.com.np/ (cit. on p. 9).

[17] (), [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam (cit. on p. 17).

[18] (), [Online]. Available: https://www.blender.org/ (cit. on p. 20).

[19] (), [Online]. Available: https://docs.blender.org/manual/en/2.79/editors/uv_image/uv/editing/unwrapping/mapping_types.html#project-from-view (cit. on p. 20).

[20] (), [Online]. Available: https://github.com/javiermzll/GAN-Dog-Generator (cit. on p. 25).