Future of Clothing, A technological approach

Dinesh Devkota, Maharshi Bhusal, Rajat Parajuli, Sushant Thapa

May 3, 2020

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 ones 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 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 crucial in developing our project to the stage it has reached. We would also like to extend our gratitude to the staff of Alternative Technology who have supported us throughout our project timeline.

Abstract

The project The Future of Clothing: A Technological Approach is a project that aims to use existing technology to improve the experience of custom designed clothing. Here, a program is written for the design of the clothes and generation of a human body model, both on the basis of user input and preference. The program further wraps the generated clothing on the body model and enables the end-user to view this model in a 3-D environment.

Table of Contents

List of Figures

1 Introduction

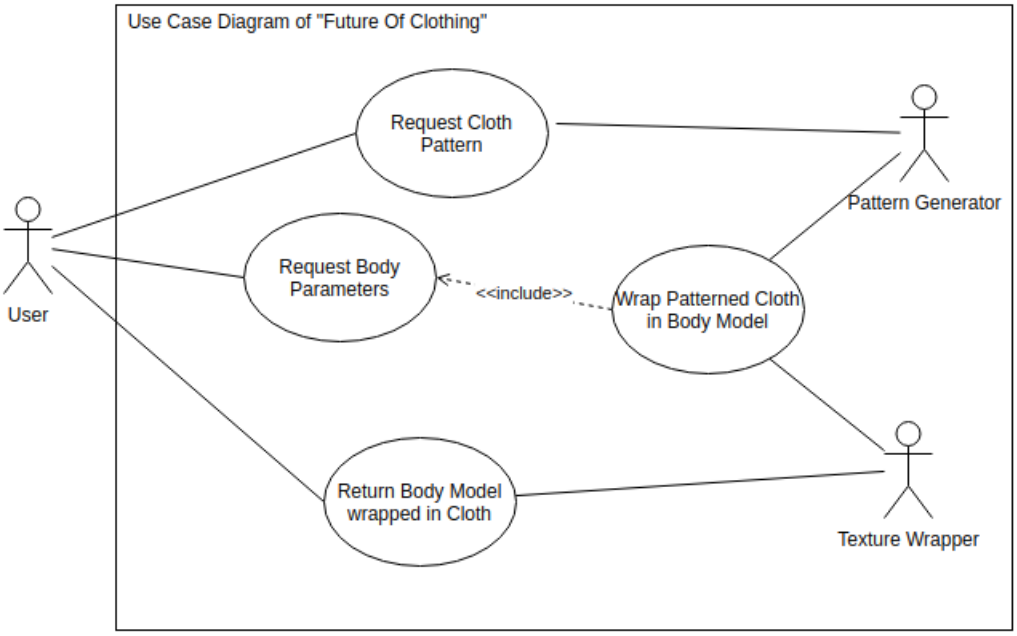

Figure 1: Use Case Diagram of the system

Technology in the sector of image processing has matured enough for developers to create interesting applications. With the increase in computation power, software development tools, developer communities, etc. and decrease in the cost of computer hardware it has become easier for technology creators and consumers to create and consume technology for a better quality of life. In this project, the attention is focused towards using technology in the field of image processing and applying the said technology towards clothing. 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 body of the subject is based on the description provided 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 generative adversarial network (GAN) which will synthesize brand new designs and patterns based on the users preference. Such a GAN can be trained by supplying training data according to the user preference.

2 Literature Review

2.1 Generative Adversarial Networks

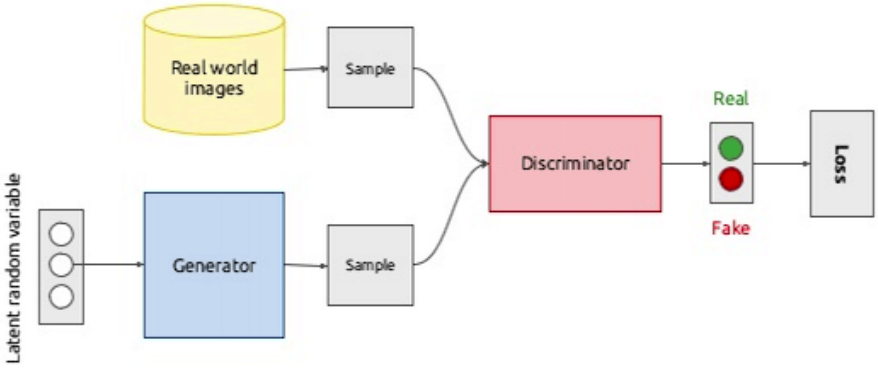

Figure 2: Basic GAN

Generative Adversarial Networks (GANs) are a class of the deep neural networks which works by competing two networks namely Generative and Adversarial networks.This network was defined and introduced to the world by Ian Goodfellow in 2014 in a now legendary paper. Generative Adversarial Network consist of two networks. A generative network and an Adversarial network. The generative network is a simple network which takes in gaussian noise as input and then scales and modifies the noise to create a new generated image. This generated image is the result of the GAN we need. New examples from the GANS are generated on the basis of this noise and the network. The adversarial network on the other hand is generally a classifier based on Convolutional networks. The architecture of the generative and adversarial network is not defined anywhere but generally a modified Convolutional network for adversarial and Transpose Convolutional network for generative network. However the original paper introducing GANs. It was a fully connected network for both generative and adversarial networks. The Generative and Adversarial network compete in a game to fool one another, sort of like a race. On each epochs,the generative network or generator generates new images based on the input noise the adversarial network learns from the training data and then uses that information to verify image produced by the generative network. The real image is classified as true for the discriminator while the generated image is classified as false. Now the loss is calculated and is used to update both discriminator and the generator. If all goes well, after a few epochs the generator is able to fool the discriminator with a generated image and we get a very lifelike and real image.

Our architecture aims to provide a high quality geometric shapes as output. So vanilla GANs with fully connected networks cannot work as the one required as per our project. Thus we have looked into several other variation of GANs which we hope to mix in our architectures.

2.1.1 Deep Convolutional GAN

Deep Convolutinal GAN is a variation of GAN which is based on Deep Convolutional architecture. It uses Deep Convolutinal architecture for the dicriminator 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 benifit of usig DCGAN over classical GAN is that they are 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).

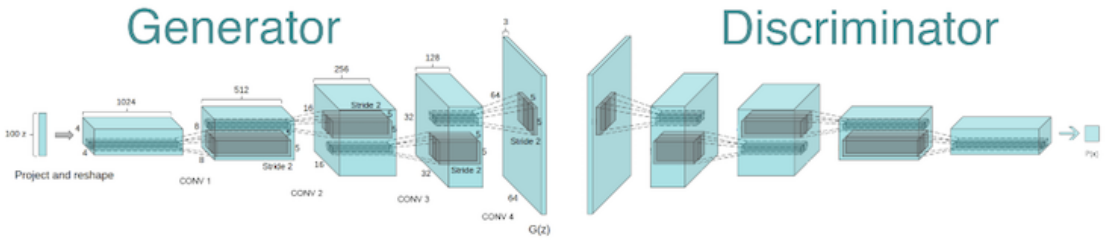

Figure 3: Caption

2.1.2 StackGAN

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 proposed to increase the size and quality of the image produced by the GAN. In layman terms this paper proposes to stack the output of smaller GANs to the larger GANs and increase the size of the output in two times increment.

2.1.3 StyleGAN

GANs are very successful to provide high quality image of the natural environments and natural structures. However, vanilla and most other types of GAN fail to provide satisfactory output for a more geometric and cartoony images. StyleGAN aims to remove this problem through using a random noise in the middle of a generator architecture which will help with the sharper ends of the images. As per the website, generation of animation faces, which is similar in structure to motifs and designs are difficult to create using simple GAN architecture including the vanilla GAN architecture. The author also moves on to explain that the creation of the faces is faster and more accurate using a variant of GAN called StyleGAN.Style GAN is a Style-Based Generator Architecture for Generative Adversarial Networks is an alternative generator architecture for generative adversarial networks, borrowing from style transfer literature. The new architecture leads to an automatically learned, unsupervised separation of high-level attributes.

2.1.4 ProGAN

GANs have always been very hard and large to train. A group of researchers of NVIDIA corporation did extensive research and have proposed a better way to train GANs. Large GANs which have very large output donot seem to scale well with complexity.

2.1.5 WarrensteinGAN

GANs are very difficult to train. This difficulty is in part thanks to having no good metric to provide a way to measure the success of the GANs and other generating networks. Thus the researcher are pretty much eyeballing the learning rates and other mechanism of the design. WarrensteinGAN tries to make the learning method more easy by proposing a newer and better metric to calculate the loss of the gan and provide a way to quantify their progress. WarrensteinGAN uses earth distance as a progress bar to measure the advancement of the training and provides pretty good result.

2.1.6 Least Square GAN

WGANs proposed to address the problem by using the EMD or Wasserstein loss function which has a smooth differentiable function even when there is little or no overlap between the two distributions. However, WGAN is not concerned with the generated image quality. Apart from stability issues, there are still areas of improvement in terms of perceptive quality in the generated images of the original GAN. LSGAN theorizes that the twin problems can be solved simultaneously. LSGAN proposes the least squares loss.

Ideally, the fake samples distribution should be as close as possible to the true samples' distribution. However, for GANs, once the fake samples are already on the correct side of the decision boundary, the gradients vanish.

Using the least squares loss function, the gradients do not vanish as long as the fake samples distribution is far from the real samples' distribution. The generator will strive to improve its estimate of real density distribution even if the fake samples are already on the correct side of the decision boundary.

2.2 3D Human Model Construction

There has been a lot of work done in the construction, use and manipulation of three-dimensional human body models for various purposes. Many researchers have used 3D full body scanners, 3D cameras and even the Xbox Kinect camera to capture and map human models. Similarly, a lot of research has be done to manipulate and generate new human models from the assets available by training regression functions to correlate semantically significant values or by employing the use of principal component analysis on feature curves and segment patches drawn onto the model and thus modified. Our project will employ a mix of principal component analysis and linear regression models to generate parameters on the human body model and allow general modifications.

2.3 UV Mapping

UV mapping is the process of mapping textures from an image into the surfaces of a 3D object.

When a model is created as a polygon mesh using a 3D modeller, UV coordinates (also known as texture coordinates) can be generated for each vertex in the mesh. One way is for the 3D modeller to unfold the triangle mesh at the seams, automatically laying out the triangles on a flat page. If the mesh is a UV sphere, for example, the modeller might transform it into an equirectangular projection. Once the model is unwrapped, the artist can paint a texture on each triangle individually, using the unwrapped mesh as a template. When the scene is rendered, each triangle will map to the appropriate texture from the "decal sheet".

A UV map can either be generated automatically by the software application, made manually by the artist, or some combination of both. Often a UV map will be generated, and then the artist will adjust and optimize it to minimize seams and overlaps. If the model is symmetric, the artist might overlap opposite triangles to allow painting both sides simultaneously.

UV coordinates are optionally applied per face. This means a shared spatial vertex position can have different UV coordinates for each of its triangles, so adjacent triangles can be cut apart and positioned on different areas of the texture map.

The UV mapping process at its simplest requires three steps: unwrapping the mesh, creating the texture, and applying the texture to a respective face of polygon.

UV mapping may use repeating textures, or an injective 'unique' mapping as a prerequisite for baking.

Among the many datasets available, the MPII Human Shape dataset is a readily available free dataset that was used which was based on the widely used statistical body representation and learned from the CAESAR dataset, the largest commercially available scan database to date. The Polygon File Format (.ply) files obtained from the dataset formed the generic point cloud shape of the human body which was further processed in a commercially available software MeshLab for the calculation of normal and triangulation. Further work could employ Delaunay triangulation and also spherical parameterization of 3D meshes for more optimized mesh construction, manipulation and also for better texture mapping.

3 Methodology

3.1 Basic Structure of System

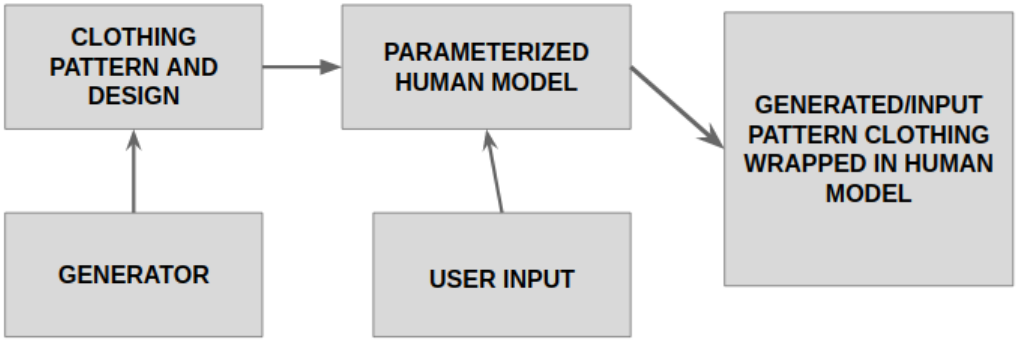

Figure 4: basic architecture of system

The system is composed of various components, with the major ones shown in figure 2. The generator while generate the patterns which are wrapped in the human model, parameterized by the user input.

3.2 Image Synthesis using GAN

On the next step of our experiment we tried to generate a dandelion image from the standard flower dataset which consisted of about 1000 images of dandelion. The kernel from the dog generator was modified and used to fit our new dataset. The new generated image was also 64-by-64 px in size. On our next experiment we tried to make a classed generator which takes input a number and outputs the image of different classes on different numbers inputted. The kernel used was a modified version of the TensorFlow tutorial kernel which was taken from a GitHub repository. However, we did not achieve much success with this kernel and thus we used a different architecture which was made by modifying the dog generator kernel. The latest experiment involves generation of monochrome motif designs. The motifs are converted to monochrome and then fed to the TensorFlow kernel which did not give us the expected result.

The motif designs were a complex hurdle to complete. The data was few in number and was very high quality which made it very hard to use in the online platform like colab. Furthermore using the data directly to train the network gave very bad results. So, the data needed to be artificially enlarged to give more images. This was done easily using normal data synthesis techniques. A simple proposed pipeline for the GAN training is given below.

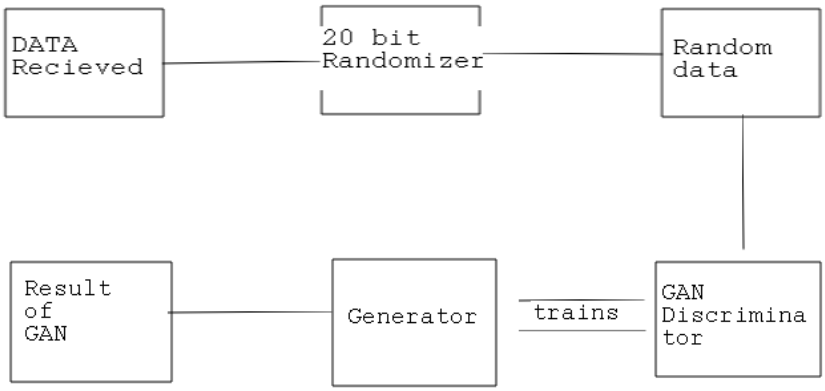

Figure 5: Proposed GAN pipeline

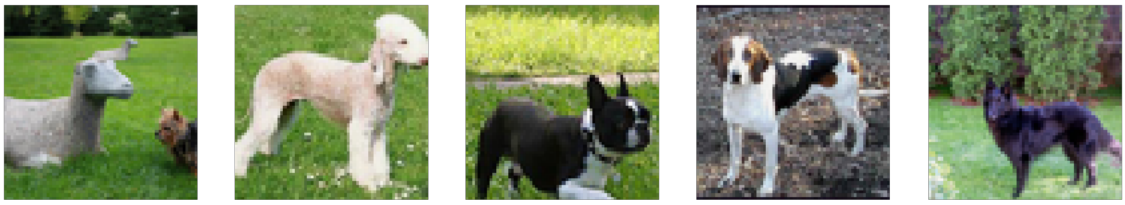

Figure 6: Images generated by GAN

On the next step of our experiment we tried to generate a dandelion image from the standard flower dataset which consisted of about 1000 images of dandelion. The kernel from the dog generator was modified and used to fit our new dataset. The new generated image was also 64-by-64 px in size. On our next experiment we tried to make a classed generator which takes input a number and outputs the image of different classes on different numbers inputted. The kernel used was a modified version of the TensorFlow tutorial kernel which was taken from the GitHub repository [**CITATION Lam ł1033**]. However, we did not achieve much success with this kernel and thus we used a different architecture which was made by modifying the dog generator kernel. The latest experiment involves generation of monochrome motif designs. The motifs are converted to monochrome and then fed to the TensorFlow kernel which did not give us the expected result.

3.2.1 Data Acquisition

Data acquisition being the first and foremost job in any machine learning project, we also got the data for our experiments early on. Data from various online sources including Kaggle and Google dataset. However we could not find proper and usable data for the motifs and shapes. These data were recieved from alternative technology, which deals with the research in field of ai and image processing. The dataset used for Dog and Flower generation were both taken from the Stanford university dataset. These datasets were used to give

proper results in the earlier experiments and consisted of about 2000 flowers and 25000 dogs. The dataset received from the company was extensive and had about 8000 pictures of varying degrees. Out of those, we used a 300 image rich folder consisting of zigzag lines and blew the size up in the next stage.The data acquired are unclean and mostly consist of 512*512 pixels sized images in png format. The data however seems very varied and may not work in complete perfection. So, the next step will be cleaning the data.

3.2.2 Data Cleaning

There are 77 folders in the data received. The folders were not only populated with images but also consisted of other unwanted files. These unwanted files were scraped and deleted manually. The incoherent directory structure was made coherent by taking the image folders to upper level manually. Most of these folders consist of about ¿100 images in consistent sizes. These images and folders can be used for image generation. However some consisted of about ¡30 images. These images can produce noise during training. So, they are kept separately. The incoherent size of images are to be cropped or filled in to provide a coherent size of the image, which will be 512*512 px.

3.2.3 Synthetic Data Production

The remaining data becomes very few in number and it becomes impossible to train the delicately precise GAN with so little data no matter how many epochs it is trained for. So, synthetic data needs to be produced from the present data. Simple procedures are used to make new data from the existing data. The synthetic data production uses four basic transformations for changing data. A basic 6 bit random binary number was generated and used to provide the switch for the application of different transformations. The total variation becomes about 23-bits variations which is quite good considering the limitations of the data. We however use only about 200 samples to increase our data. For a geometric model data like ours, this becomes quite good and gives us very good result. They are as follows:

- Stretching and Squashing

  The image was stretched and/or squashed by a random pixels from 0 to 15% of the total size of the image in a step of 3. for a 512 px image it provides 25 different random conditions.

- Cropping

  The image was cropped upto 10% of the total width of the image randomly in step of 1. This Gives 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 image can be rotated whole 360 degrees without losing details. Thus we rotate image from 0 to 90 degrees in a step of 1 degree. Thus we have 360 different random conditions for the data

3.2.4 GAN Training

The GAN is then trained to provide new styles. Up until now we have trained the GAN in 28 px square images. However, this will be improved using the stack GAN which will help us with higher size images GAN training. The training took about 11 second per cycle for 28*28 px images which numbered about 56000 in size of the dataset. This GAN was trained for 50 epochs but the best result was seen in the 40th epoch, which was thus used as the final result. Further on the road we aim to get 512*512 size of image using stack GAN. Higher order GANs require a lot more time to train and thus are quite lengthy to train.

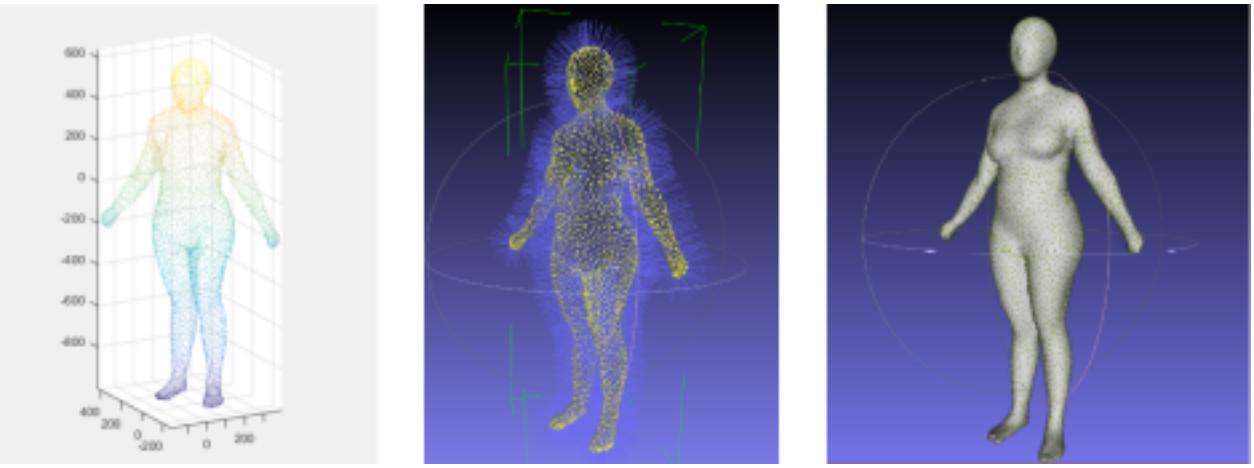

Figure 7: (Left) the extracted point cloud. (Middle) The calculated point normal in MeshLab. (Right) The mesh resulting from the screened Poisson surface reconstruction

3.3 Generation of Human Body

From the MPII Human Shape dataset, 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 Design Wrapping

The process of mapping texture to a 3d object is called UV unwrapping. The process is illustrated in figure 5. When building an object from scratch, the textures also need to be built from scratch. However, we already have the textures in the colored photo. Since we also have the relationship between the surfaces and pixels, we also have the map of textures to the surfaces. The corresponding textures are now rendered in their appropriate surfaces.

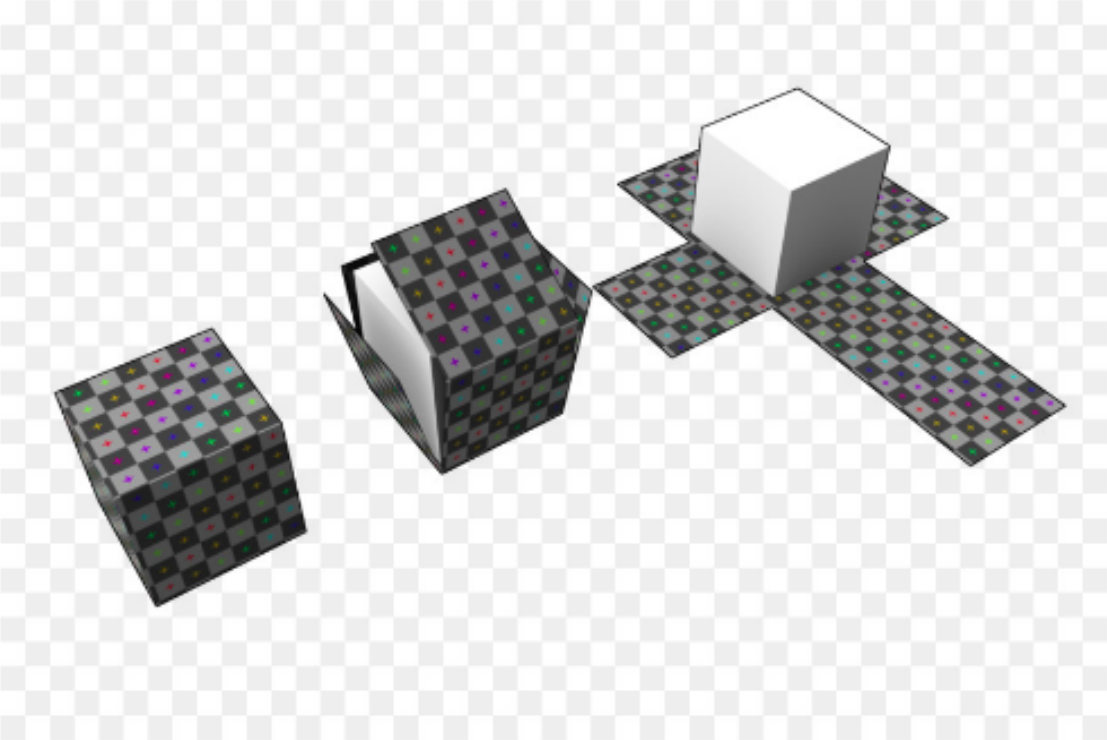

Figure 8: UV unwrapping illustration

4 Output

4.1 GAN outputs

We have until now trained different and modified versions of opensource, available and stable variation of GANs.Out of different types of GANs these four stand out the most. The type of GANs we have trained includes vanilla GAN with modified structure and DCGAN with pure structure. The vanilla GAN used trained the discriminator first and Generator last to stabilise the training of the GAN. While this provides low quality but stable output, it suffers greatly from mode collapse and dynamic training i.e. training discriminator and generator different times and stopping prematurely. The Vanilla GAN being stable was trained on 64*64 images while the DCGAN was trained on 28*28 images since the GAN was unstable and needed quick experimentation. Previous efforts to port DCGAN was unsuccessful due to the architectural inadequacy. We now however can attempt to do that again with greater chance of success.

4.1.1 Shapes with Vanilla GAN with Local Dataset

In this we trained the same GAN structure with about 500 images of floral designs. However, we were unable to get any result due to inherent limitation of the vanilla GAN. From this, we shifted our focus from simple vanilla GAN to advanced variations of GANs, which includes DCGAN and other GANs. The image shown here were trained here with vanilla GAN at 64*64 pixels.

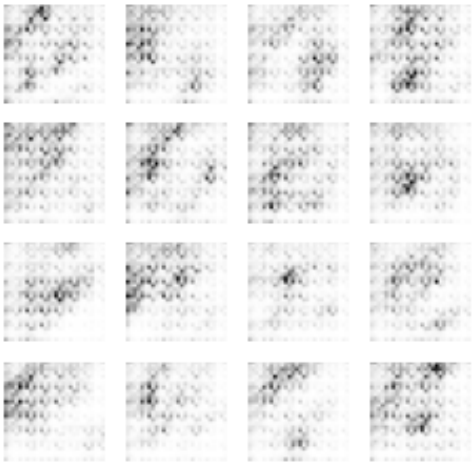

Figure 9: Vanilla GAN with Local Unmodified data

4.1.2 ZigZag line with DCGAN with local and synthesized dataset

In this experiment we conducted two different scenarios. We were trying to experiment with overtraining the GAN with few data on one hand, while we were trying out the synthesized dataset variation on the other hand. The first scenario was however not successful as it didnt produce anything remotely like the dataset provided, no matter how hard we tried. The synthetic dataset however shows promise for the future. The results are shown below.

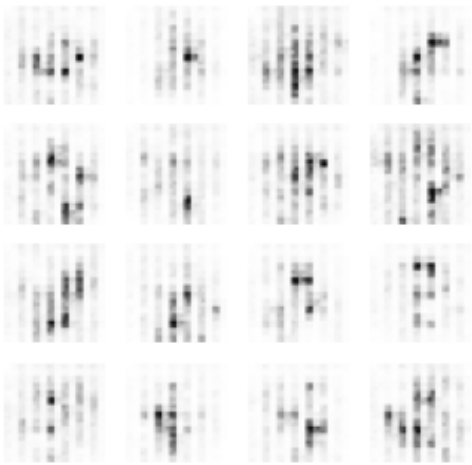

Figure 10: DCGAN result with Local Unmodified data

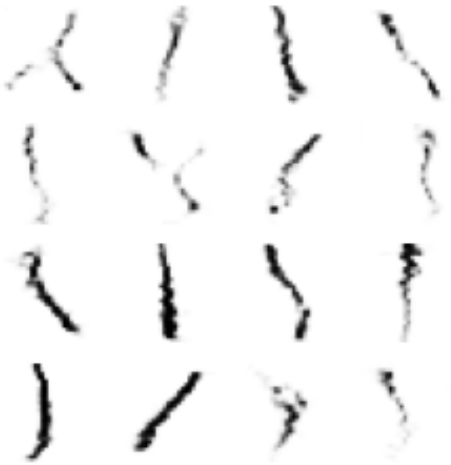

Figure 11: DCGAN result with Synthetic data

5 Performance and Analysis

6 Conclusion