Achieving geometric transformation of Euclidean representation derived from non-Euclidean data by using Pix2Pix network

Jarvis Ivan Rebello

Master of Science in Data Science
The University of Bath
2021-2022

Achieving geometric transformation of Euclidean representation derived from non-Euclidean data by using Pix2Pix network

Submitted by: Jarvis Ivan Rebello

Copyright

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Master of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Abstract

Data recorded is usually non-Euclidean in nature. Processing such data is not feasible because most of the data processing networks/software are built on handling Euclidean data. Although, due to the recent researches being conducted in the field of neural networks, there are models which are being developed and enhanced to process such non-Euclidean data. Instead of directly implementing such novel neural networks specifically built for non-Euclidean data transformations, this experiment tackles the problem using a different approach. This experiment, however, focuses on two main tasks, viz., converting non-Euclidean data into its Euclidean image representation, and validating if an existing neural network can handle geometric transformation. For converting non-Euclidean data (3-D triangle mesh) into Euclidean representation, the process of obtaining barycentric coordinates is implemented. Validating the neural network for geometric transformation includes creating a Euclidean representations of a 3-D mesh in its original orientation and another Euclidean representation of the same 3-D mesh after the mesh is rotated at a fixed angle. The neural network being validated is a type of conditional adversarial network called Pix2Pix (Isola et al., 2017).

Contents

List of Figures

Acknowledgements

I would like to thank my supervisor, Dr. Nadejda Roubtsova for guiding me throughout the experiment. Her detailed feedback proved beneficial in understanding and writing this experiment. I would like to thank my dissertation peers, Daniel E, Sara F and Juan L for helping out whenever I was in doubt. Lastly, I would like to thank my family, the Rebellos, for all the moral support they have shown me throughout this experiment.

Chapter 1

Introduction

Data is "the new oil of the digital economy" today. It is the backbone of all human organizational activity, like finance, business decision, scientific research, and even improving personal lives. Data plays a vital role in each industry, from exploring patterns and insights from historical records to crafting calculated predictions. It can be defined as information collected in a format that is appropriate for processing and analysis. It is gathered using a query, observation, and measurement methods, which are stored as digits or letters (Wikipedia contributors, 2022a). Data can be divided into two categories: qualitative (categorical) data and quantitative data. As the names suggest, qualitative data deals with organizing information by groups/categories. In contrast, quantitative data are purely based on numbers. Both qualitative and quantitative data are stored in file formats such as CSV (Comma-Separated Values), JSON, plain text, PDF (Portable Document Format), image files, audio files, and many others.

Another approach to studying data would be based on their geometry. Data represented and processed on a flat surface is called Euclidean representation of the data, whereas if the surface is curved, it is non-Euclidean. A simple example to differentiate the two geometric terminologies would be that drawing a line on a piece of flat paper is Euclidean, and drawing a line on a curved surface, like a globe, is non-Euclidean. Graphs, manifolds, point clouds, polygon meshes, triangular networks and molecular proteins are non-Euclidean examples. On the other hand, geometry done on flat surfaces are Euclidean example. Deep learning methods are used to process Euclidean (grid-like) data, which has proved beneficial in solving problems in computer vision, audio analysis, and natural language processing (Bronstein et al., 2017). However, this experiment aims to study how non-Euclidean data can be processed in a way to achieve meaningful geometric transformation by using deep learning neural network. Emphasis is given to 3-dimensional triangular meshes to represent non-Euclidean data.

A set of non-Euclidean mesh data, a subset of CoMA dataset (Ranjan et al., 2018), is made available for usage, which needs to be converted into its corresponding Euclidean representation. In simpler terms, this Euclidean representation means grid-like representation. A pre-requisite task will be to change the orientation or rotation of the 3-D mesh. After the 3-D mesh is rotated, its original and rotated grid-like representations are obtained. The two image representations are fed to an existing neural architecture. The neural network taken into account is a type of conditional Generative Adversarial Network called Pix2Pix (Isola et al., 2017). The main objective of this experiment is to investigate if this neural network is able to produce results corresponding to the ground truth, i.e., 3-D mesh representation in its original orientation. The rectification type is rigid, where the deep neural network is trained to modify

the 3-D mesh representation in a fixed angle.

Chapter 2

Literature and Technology Survey

This review is divided into three key segments. Non-Euclidean geometry in Section 2.0.1 lays the foundation and aids in understanding the central concept of the experiment by getting familiar with the geometric terminologies. Source data in Section 2.0.2 explains the various non-Euclidean data, i.e., different 3-D polygon mesh data-sets available online, and compares them with respect to the configurations. Processing non-Euclidean data (3-D mesh) using neural architecture in Section 2.0.3 covers an explanation of various neural networks developed for processing non-Euclidean data and takes a look at the approach used in this experiment by using a type of conditional Generative Adversarial Networks (GANs). It also states the conclusion which summarises the research.

2.0.1 Non-Euclidean geometry:

The geometric terminologies are based on Euclid's postulates published in 300 BC in his textbook 'Geometry: The Elements'. If geometry satisfies Euclid's postulates, then it is termed Euclidean geometry, and if not, it is non-Euclidean. Euclid was an ancient Greek mathematician known as the 'Father of Geometry'. George D Birkhoff demonstrates these postulates thoroughly by using mathematical notations. Each postulate is purely mathematical, which helps get a clear understanding of the terms even while drawing a geometric figure like a point, straight line, circle, rectangle, or parallel line (Birkhoff, 1932; Singh, 2022). A few encounters with non-Euclidean data include data collected from sensor networks, social networks in computational social sciences (Bronstein et al., 2017), MRI imaging data (Fletcher and Joshi, 2007) and human shape and motion data (Vinué, Simó and Alemany, 2016; Barahona et al., 2018).

(Bronstein et al., 2017) talks about the methods used to process non-Euclidean data (graphs and manifolds). Major problems faced are describing the structure and analysing the functions of the non-Euclidean field. The latter is discussed in detail to streamline Convolutional Neural Networks to resolve such a problem (Bronstein et al., 2017). Gori et al. (Gori, Monfardini and Scarselli, 2005) was the pioneer in streamlining neural networks to graphs. Unfortunately, his contribution was not given attention until deep learning was given importance in recent research. (Masci et al., 2015) introduced Geodesic CNN to generalising CNNs in the field of 3-D manifolds.

Significant improvements were seen to streamline the neural networks in (Monti et al., 2017) as compared to the previous approaches to processing the non-Euclidean paradigm. By introducing

the spatial-domain model, state-of-the-art results were obtained because this model could complete most of the geometric deep learning tasks. (Monti et al., 2017) also solves the problem of training massive datasets and handling high complexity, as noticed in previous papers (Su et al., 2015; Wu et al., 2015; Wei et al., 2016; Defferrard, Bresson and Vandergheynst, 2016) to processing deformable 3-Dimensional shapes. The approach is said to be "intrinsic and deformation-invariant by construction".

2.0.2 Source Data:

Non-Euclidean data is the famous research area for every scientific experiment. (Egger et al., 2020) helps understand all the other 3-D morphable face model research since they were first proposed in 1999. The original paper (Blanz and Vetter, 1999) introduced a method for modelling textured 3-D faces. Using a set of examples of 3-D face models, Blanz and Vetter transformed the shape and texture into a vector space representation to derive a face model which is morphable. Blanz and Vetter were successful in matching morphable models constructed from the first 100 shapes and textures derived from the dataset. Future work of (Blanz and Vetter, 1999) includes optimising the matching algorithm to reduce the time complexity by minimising the cost function using the Newton method. Extending the database by adding children to older people of all races and extending face models since the 3-D model of hair cannot be captured by the laser scanner used during the time of research.

Inspired by the works of Blanz and Vetter (Blanz and Vetter, 1999), the first publicly available morphable 3-D face model was brought by (Paysan et al., 2009) to encourage further research in generative 3-D models. (Paysan et al., 2009) was successful since much other research commenced on the basis of these publicly available 3-D face models. The overview of publicly available 3-D shape from five recent research are given below:

Source Dataset	Format	Area	Number of samples	Scanning method
MMSE (Zhang et al., 2016)	triangle mesh (30K–50K vertices), 1,040 × 1,392 texture image	Inner face	140 individuals × four dynamic sequences	Dimensional Imaging
Headspace (Dai et al., 2017)	triangle mesh (180K vertices), 2,973 × 3,055 UV texture map	Head (face, neck and ears)	1,519 individuals	3dMD
4DFAB (Cheng et al., 2018)	triangle mesh (60K–75K vertices), UV texture map	Head (face, neck and ears)	180 individuals × 4K–16K frames of dynamic sequences	Dimensional Imaging
CoMA (Ranjan et al., 2018)	triangle mesh (80K–140K vertices), texture images (texture images (average resolution resolution 3,700 × 3,200), six raw camera images (each 1,600 × 1,200), alignments in FLAME topology	Head (face, neck and ears)	12 individuals × 12 extreme expression sequences	3dMD
VOCASET (Cudeiro et al., 2019)	triangle mesh (80K–140K vertices), texture images (texture images (average resolution resolution 3,700 × 3,200), six raw camera images (each 1,600 × 1,200), alignments in FLAME topology	Head (face, neck and ears)	12 individuals × 40 dynamic sequences (Speech-4D)	3dMD

The dataset for this experiment is a subset of CoMA (Ranjan et al., 2018) triangle face mesh data made available, specifically focusing on the mouth area.

2.0.3 Processing non-Euclidean data using neural architecture:

To represent 3-D geometry in subject areas like computer graphics and game development, polygon meshes are the most appropriate structure to consider. (Hanocka et al., 2019) analysis uses MeshCNN for processing different 3-D shapes. Because of a mesh's distinctive properties, a convolutional neural network tailored with studied convolution and pooling layer is used to process polygon mesh edges. This MeshCNN is exclusively developed for processing triangular meshes. (Hanocka et al., 2019) deems mesh edges to be the crucial factor for the neural network to function as "the edge set dictates a simple means to define a local, fixed sized neighbourhood for convolutions over irregular structures". The position of Cartesian coordinates of vertices is not being considered, but they are used only for presenting triangular meshes. Future work includes prioritising the edge collapse by adding learned edge features in the pooling operation (Hanocka et al., 2019).

(Nash et al., 2020) presents a way to model a mesh by using a transformer-based architecture, PolyGen – a deep generative model used for processing 3-D models, where vertices and faces

are predicted serially. The model is competent in generating high-quality, reusable meshes and the maximum value of the log of the probability that occurs at the same point as the original probability function, i.e., log-likelihood standards for modelling tasks. This PolyGen model (Nash et al., 2020) is similar to the works of PointGrow (Sun et al., 2020), which "operates on fixed-length point clouds rather than variable vertex sequences, and uses a bespoke self-attention architecture that is relatively shallow in comparison to modern autoregressive models in other domains". Polygon, however, creates high-quality 3-D meshes by using recent advancements in the field of mesh-compatible deep learning architecture.

An interesting neural network called Graph Attention Networks (GATs) processes non-Euclidean graph data is computationally efficient as compared to the other graph processing neural networks which are computationally expensive (Veličković et al., 2017). This method enables parallel computation and matrix operations are not required. However, a couple of major drawbacks with this approach is that the method is only capable of classifying nodes instead of graph classification and the model is yet to contain edge features which could help solve a much larger variety of problems (Veličković et al., 2017).

Understanding the scope of non-Euclidean data space has also been proven to provide a comprehensive study in graph networks (Asif et al., 2021). Graph Neural Networks (GNNs) provide a solution to the problem of finding a pattern in non-Euclidean space by utilising the relationship among data (Asif et al., 2021). The paper portrays a clear understanding of the classification of graphs along with its generation and optimisation techniques used. A streamlined approach to designing GNNs is discussed in (Zhou et al., 2020) to section the applications of graph data. The challenges faced with graph data include its understandability, trainable nature, complicated architecture, and its robustness (Zhou et al., 2020).

Implementing Generative Adversarial Networks (GANs) on non-Euclidean data has been as successful as, or more, compared to the Euclidean domain (Lazcano, Fredes and Creixell, 2021). The two deciding factors for judging the performance of GANs implementation are the curvature (c) and its design. The performance of GANs, Conditional GANs and Wasserstein GANs implemented were recorded, which says GANs had a score of $c = 10^{-3}$, CGANs had $c = 10^{-2}$, and WGANs had $c = 10^{-1}$. Smaller the value of c, greater the performance of the network (Lazcano, Fredes and Creixell, 2021).

Using Graph Neural Networks (GNNs) would be an ideal way to deal with non-Euclidean graph data. It is catered to provide a straightforward, simple method to predict tasks on graph properties. But overcoming its complexities are still a major challenge. These data are dynamic in nature; in other words, the input definition of edges and vertices is ambiguous (Wu et al., 2020). The technique used to record their changes from time to time, results in developing new convolutions to adapt to such changes. GNNs are usually streamlined to enable the processing of uniform graphs. Non-uniform graphs like images and texts make it problematic for GNNs to be implemented. These two factors make GNNs less scalable. Graph properties are being lost in the process of sampling or clustering (Wu et al., 2020).

In this experiment focus is given on non-Euclidean mesh data and trying out a type of CGAN called Pix2Pix. The beauty of this lies in not developing the mapping and loss function from scratch and still being able to accomplish appropriate outcomes (Isola et al., 2017). The experiment is similar to the facade implementation in Pix2Pix architecture. To train this Pix2Pix model, the dataset includes two 256x256 images, where the input is mouth mesh image from the CoMA dataset (Ranjan et al., 2018) and its image (grid-like) representation is

the ground truth. The expected final output should be somewhat similar to the input (mouth mesh image) (Isola et al., 2017).

Chapter 3

Methodology

This experiment focuses on executing a deep neural network given two image representations of a 3-D mesh as an input. The non-Euclidean mesh data derived from the CoMA dataset (Ranjan et al., 2018) is converted into its equivalent Euclidean representation i.e., its image representation. The two images consist of image representation of the mesh in its original orientation and image representation of the same mesh after it is rotated at a fixed angle. Starting from the procedure of obtaining Euclidean representation of non-Euclidean data to understanding the mechanism in the deep learning neural network used viz., Pix2Pix, and how the former is tweaked to fit the network is explained in the sections below.

3.1 Euclidean data representation of non-Euclidean data

As mentioned above, a subset of CoMA dataset (Ranjan et al., 2018) is the source non-Euclidean data available. The subset consists of 3-D mouth meshes which is again a subset of the face meshes in CoMA (Ranjan et al., 2018). (Cosker, Krumhuber and Hilton, 2011) introduced the concept of obtaining a 3-D image from a mesh and the same approach is being used in this experiment. Let us consider a set of meshes $M = [V_1, V_2, \ldots, V_n]$ in mouth dataset and V is a collection of vertices where $V = [A, B, C, \ldots]$. Each vertex will be denoted as $A = [X_A, Y_A, Z_A]$, $B = [X_B, Y_B, Z_B]$, $C = [X_C, Y_C, Z_C]$ and so on $\in R^3$. X, Y and Z are the three axes used for representing the meshes in non-Euclidean space and $[X_A, Y_A, Z_A]$ are the 3-D coordinate values for vertex A, $[X_B, Y_B, Z_B]$ are for B and so on. A fixed set of UV coordinate values are also provided which is denoted by $UV = [a, b, c, \ldots]$ where $a = [u_a, v_a]$, $b = [u_b, v_b]$, $c = [u_c, v_c]$ and so on $\in R^2$. These UV coordinate values are constant across all the 3-D meshes. Same topology is used to represent all 3-D meshes in the form of a grid-like image. All the vertices V correspond to the UV texture map through common set of triangular faces 'f' mentioned in a mesh file (Cosker, Krumhuber and Hilton, 2011; Ranjan et al., 2018). More in-dept information is provided about understanding faces and the file format used in Implementation section.

Non-Euclidean data is transformed into Euclidean representation, i.e., 2-D flat surface, from an implementation perspective. The neural networks are designed for processing images in 2-D. The role of vertices in 3-D meshes is to generate a texture (color) within their corresponding triangles in 2-D space which is defined by the set of faces. This color is created by implementing

Barycentric coordinate mapping (Scratchapixel, 2014). The equation to calculate a point P within a triangle (face) defined by three vertices and Barycentric coordinates is,

$$P = uA + vB + wC$$

Where A, B, C are the vertices of a triangle and u, v, w are the Barycentric coordinates with $u + v + w = 1$. These scalar values also lead to $w = 1\check{}u - v$ and $u + v <= 1$. Alternately, the equation can also be written by using two barycentric coordinates, u and v, as

$$P = A + u * (B - A) + v * (C - A)$$

Similarly, the point P_{uv} is calculated on the flat-surface by following the above equation but by using UV coordinates $(a, b, c, \dots)$ instead of vertices $(A, B, C, \dots)$. Also the role of the scalars (u, v) stay the same.

$$P_{uv} = a + u * (b - a) + v * (c - a)$$

Once the image representation of the 3-D mesh is obtained in its original orientation, the same mesh is rotated and the above-mentioned process of obtaining a 3-D image is commenced.

3.2 Understanding Deep learning neural network, Pix2Pix

3.2.1 Deep learning

Deep learning is an updated version of machine learning which teaches the system to compute data in the same way human brain does (Wikipedia contributors, 2022b). Even though deep learning was first talked-about around the 1940s, the scope has increased ten-folds recently in the last 2 decades. This is because of the advancement in computers which has enabled greater computational speed. Applications of deep learning include virtual assistants (Google, Apple Siri, Amazon Alexa, etc.), driverless cars (Tesla, Porsche, Jaguar, etc.), face-recognition and many more. Deep-learning architectures have been influential and quite successful in the fields of computer vision, speech recognition, natural language processing, board games and many more. Deep-learning architectures includes deep neural networks, deep reinforcement learning, recurrent neural networks, convolutional neural networks, and transformers (Ciregan, Meier and Schmidhuber, 2012; Krizhevsky, Sutskever and Hinton, 2017).

3.2.2 Neural network

Neural network mimics the human brain. It is designed in such a way that it is composed of neurons or nodes. It is a circuit of biological neurons used for solving artificial intelligence related problems. Applications include, regression analysis including time-series prediction, classification including pattern and sequence recognition and data processing category (Wikipedia contributors, 2022d). Because of neural networks being able to solve such complex problems, it has its set of disadvantages like being computationally expensive, requiring more amount of data and spending more time to develop such a solution.

3.2.3 Generative Adversarial Networks (GANs)

The neural network used in this experiment a type of conditional generative adversarial network called Pix2Pix. The mechanism of generative adversarial network focuses on a contest between two neural networks. (Wikipedia contributors, 2022c) says that the contest has probability space (Ω, μ_{ref}) and the two neural networks are the two players, generator and discriminator. The generator's game-plan is $P(\Omega)$, all possible probability measures μ_G on (Ω). The discriminator's game-plan is the set of Markov kernels, $\mu_D : \Omega \to P[0,1]$ where $P[0,1]$ is the set of probability $[0,1]$. The GAN contest is like the zero-sum game, where one agent's loss is another agent's gain, with the objective function

$$"Loss(\mu_G, \mu_D) := E_{x\sim\mu_{ref}, y\sim\mu_D}(x)[logy] + E_{x\sim\mu_G, y\sim\mu_D}[log(1-y)]"$$

The discriminator seeks to maximise the objective, while the generator seeks to decrease it (Wikipedia contributors, 2022c).

3.2.4 Conditional Adversarial Network, Pix2Pix

When it comes to image-to-image problems, conditional adversarial network, a type of GAN, is the go-to solution because these networks learn from a loss function along with the general approach of mapping from input image to output images. Pix2Pix GAN is an implementation of the conditional GAN where the generation of an image is conditional on a given image. The generator in Pix2Pix creates a translated version of a given input image. The discriminator is given a pair of images, an input image and a real or generated image, and it is expected to validate if the paired image is fake or not (Isola et al., 2017).

Finally, the generator is trained to trick both the discriminator and seeks to minimize the loss between the generated image and the expected image.

The generator model, discriminator model, and model optimization process are all carefully specified in the Pix2Pix GAN architecture. Deep convolutional neural networks commonly use Convolution-BatchNormalization-ReLU layers, and the generator and discriminator models in Pix2Pix are no different. Let us take a look at the two neural network architectures and the loss function used to optimise the weights.

U-Net Generator Model

The generator in Pix2Pix uses a U-Net model architecture instead of the usual encoder-decoder model architecture. The encoder-decoder architecture involves down-sampling an input image over a few layers until a choke-point stage and then reversing the process, i.e., the representation is up-sampled again over a few layers before displaying the final image with the expected shape. U-Net architecture is similar in terms of down-sampling and up-sampling to get an image, but links or skip-connections are made between same size layers in the encoder and decoder, allowing the choke-point to be bypassed.

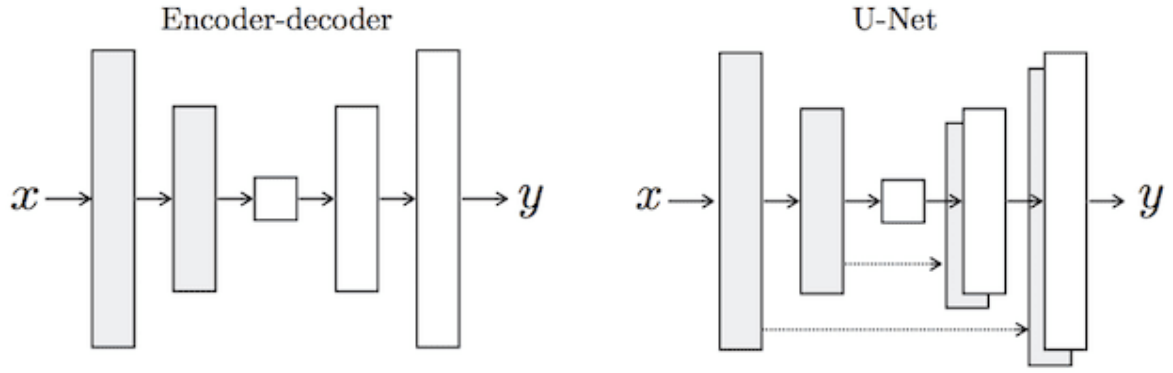


Figure 3.1: Encoder-Decoder Generator and U-Net Generator Models (Isola et al., 2017)

PatchGAN Discriminator Model

PatchGAN is a model takes in a patch of an input image at a time and checks for originality rather than considering the whole image to tells if it is real or fake. "The PatchGAN discriminator model is implemented as a deep convolutional neural network, but the number of layers is configured such that the effective receptive field of each output of the network maps to a specific size in the input image" (Brownlee, 2019).

Composite Adversarial and L1 Loss

Pix2Pix discriminator identifies real and fake images by minimizing the negative log likelihood which is similar to the training mechanism of the traditional GAN model, only difference being conditioned at the source. The discriminator loss is halved because the training time complexity of the discriminator is very less as compared to the generator (Brownlee, 2019).

```
Discriminator_loss = 0.5 * Discriminator_loss
```

The adversarial loss for the discriminator and the L1 or average absolute pixel difference between the translated source image and expected output image together trains the generator. The loss function used to update the generator is a combination of adversarial loss and L1 loss. L2 loss was calculated and resulted in blurred images (Brownlee, 2019).

$$\texttt{Generator\_Loss} = \texttt{Adversarial\_Loss} + (\lambda * \textit{L1\_Loss})$$

Applications

Pix2Pix was implemented on a variety of different image-to-image translation tasks such as,
Semantic labels <-> photo, trained on the Cityscapes dataset
Architectural labels -> photo, trained on Facades
Map <-> aerial photo, trained on data scraped from Google Maps
Black and White -> color photos
Edges -> photo

Sketch -> photo
Day -> night photographs
Thermal -> color photos. Photo with missing pixels -> inpainted photo, trained on Paris StreetView (Brownlee, 2019).

The only problem with this Pix2Pix model is that hand engineering the loss function does not work too well. This experiment with Euclidean representation is closely related to the architectural label implementation in Pix2Pix architecture. The aim is to check if Pix2Pix can be trained to transform a rotated image at a fixed angle back to its ground truth image at its original orientation.

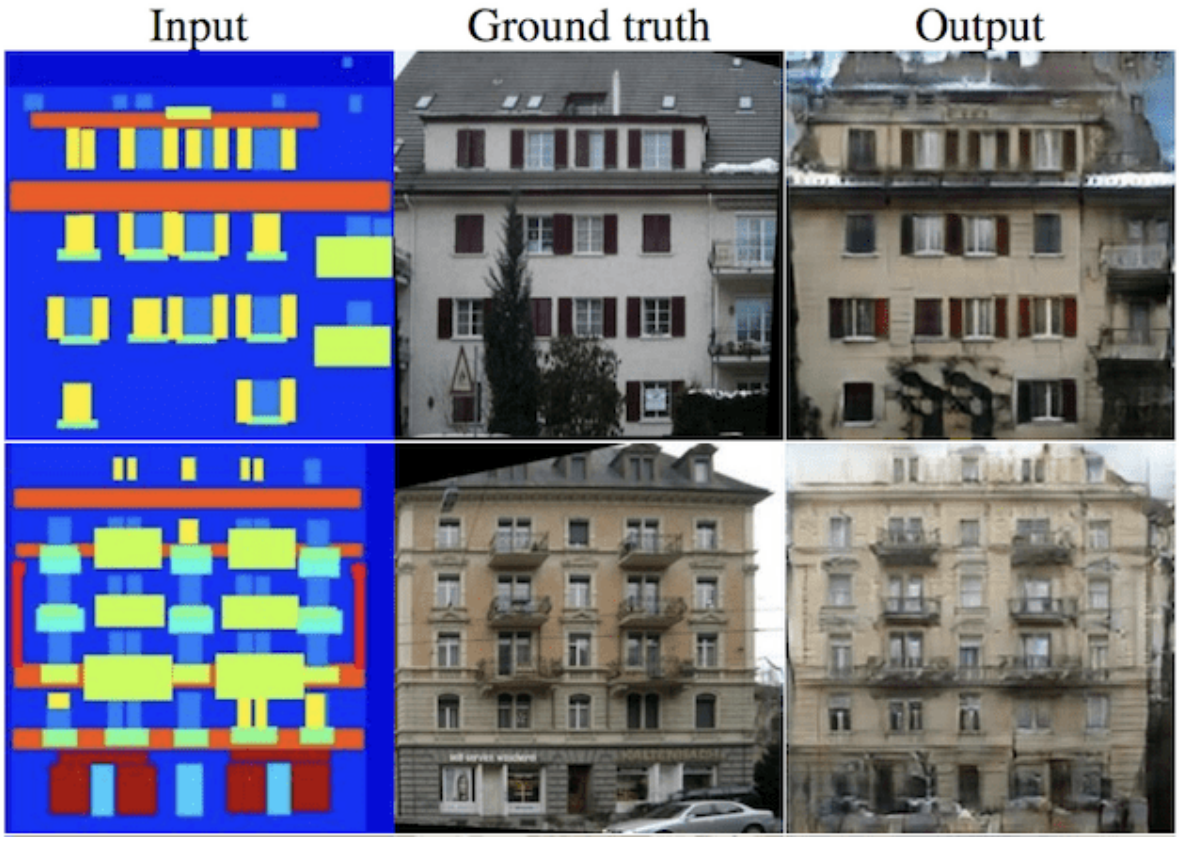


Figure 3.2: Facade Implementation in Pix2Pix

3.3 Validate Pix2Pix model using Euclidean data representation

This experiment validates if the Pix2Pix is able to produce meaningful geometric transformation by using Euclidean data representations. The transformation process of non-Euclidean data into its Euclidean representation is carried out in such a way that no modifications need to be done to the Pix2Pix network to fit the dataset. The dataset consists of a collection two Euclidean representations of a non-Euclidean 3-D mesh each with height and width M where one is an image representation of the mesh is in its original orientation and the other is an image representation of the mesh after it is rotation at a rigid angle. After creating the dataset and splitting it into test, validation and training sets, the dataset is directly imported into the Pix2Pix network. The shape of the two image representations created

is M x $2M$ x 3. The shape implies two M x M images placed horizontally next to each other. The image to the left is the ground-truth or image representation at its original orientation/no rotation and the image to the right is the image representation with a fixed angle rotation. The ideal approach of testing the model is by considering the predicted (output) image and converting this Euclidean grid-like representation back to its non-Euclidean data which is represented by a set of vertices in X, Y and Z-axis. If the vertex values of the predicted image and the original image match, the scope and applications of processing non-Euclidean data using Pix2Pix will increase. However, in this experiment, validation is done by comparing the ground truth with the predicted image generated by the Pix2Pix network.

Chapter 4

Implementation

4.1 Understanding the data, Polygon mesh

Polygon meshes are mainly used in areas such as computer graphics and geometry studies. A polygon mesh comprises of a collection of faces, edges, and vertices which defines a polyhedral object. Polyhedral is a solid 3-D unit with flat faces, straight edges, and corners or vertices. Examples includes a cube, prism, or pyramid. The flat faces could be in a shape of a triangle, quadrilateral and convex or concave polygon. These different types of faces help in the process of creating realistic or non-realistic pictures from a 2-D or 3-D object called rendering or image synthesis (Wikipedia contributors, 2022e).

4.1.1 Elements

As mentioned in the section above, representing a polygon mesh consists of three key elements, viz., vertices, edges, and faces. UV coordinates are used to represent the mesh in Euclidean space and understanding the file format is the base of this experiment. Explanation of each element is provided below.

Vertex

A vertex a point in a three-dimensional space and the smallest component of a polygon model. A vertex in 3-D space can be expressed as $V = [X, Y, Z]$ where X is a coordinate value on the X-axis, Y is a coordinate value on the Y-axis and Z is the coordinate value on the Z-axis. The little yellow dots represent the vertices in a 3-D mesh. Each mouth mesh consists of 534 vertices.

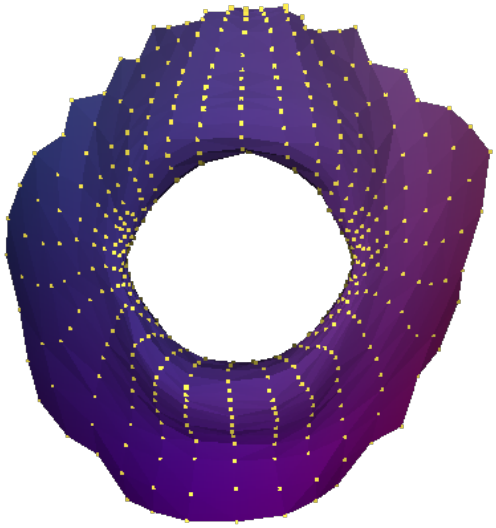


Figure 4.1: Vertices - represented by yellow dots

Edge

Edge helps to define the shape of a 3-D mesh. An edge is a line segment drawn between two vertex points. By connecting many such vertices with edges, a pattern is formed which helps create a polygon. The green lines shown in the figure below are the edges which are formed by connecting the vertices.

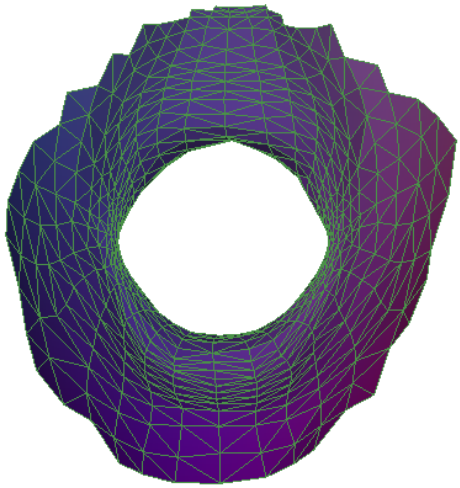


Figure 4.2: Edges - represented by green lines

Face

Face is the most primary element which is made up of vertices and edges. The CoMA dataset (Ranjan et al., 2018) being used (discussed in Literature and Technology Review) in this experiment consists of triangular faces. A triangular face is obtained when exactly three vertices are connected using three edges in 3-D space. Face is what fills in the empty space between the edges and makes up what is visible on the triangular mesh. The figure below displays all the faces in a 3-D mesh, yellow dots are the vertices and the red lines connecting the vertices are the closed set of edges. Each mouth mesh consists of 986 faces.

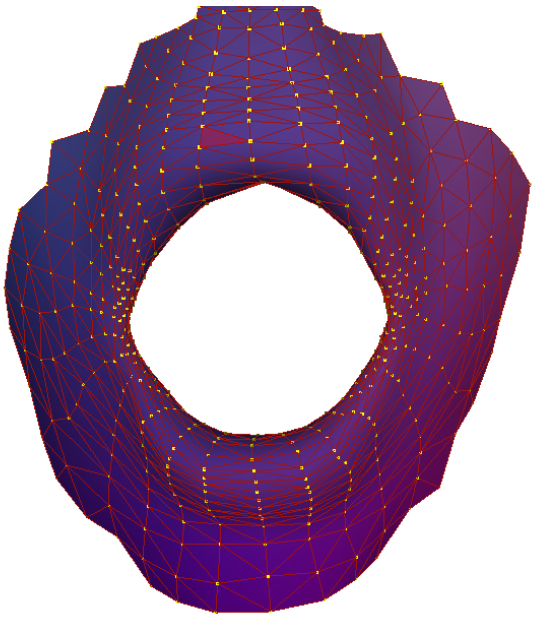


Figure 4.3: Faces - represented by red edges and yellow vertices

UV coordinates

A 3-D triangular mesh is difficult to compute, and much complex as compared to handling 2-D data. Although Graph Neural Network has proved efficient in recent years, representing such 3-D mesh data onto a 2-D grid makes its understanding and processing easier. This conversion into 2-D, grid-like representation is obtained due to the UV coordinates. UV coordinates can also be defined as a flattened representation of the 3-D mesh. The letters "U" and "V" denote the axes of the 2D texture because "X", "Y", and "Z" are already used to denote the axes of the 3D mesh. In other words, UV coordinates helps get the pixel location of the 3-D mesh on a flat canvas. In geometric terms, UV coordinates could be termed as the Euclidean representation of non-Euclidean data. The UV coordinates are the same (fixed values) for all mesh data being processed in this experiment. Hence, there is no need to calculate the UV coordinates in this scenario. Having a fixed topology will enable the Pix2Pix model to train better and produce meaningful results. A constant topology helps streamline the whole transformation process, making the process understandable and less complicated. Experimenting with fixed topology forms the base of further studies in terms of investigating structures with variable blueprints. The figure below is the UV representation of all the 3-D

triangular meshes (refer figures for vertex, edge, and face) shown above. Each mouth mesh has 534 UV coordinates.

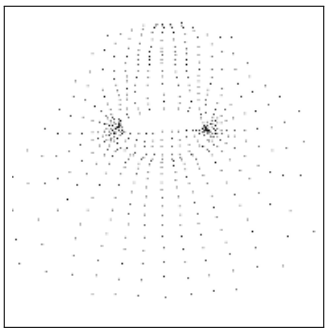


Figure 4.4: UV coordinates

4.1.2 File format

Obj file format is used to support the CoMA dataset. How the vertices, edges, faces and uv coordinates are recorded in an obj file, will be discussed in this section.

Vertices in the obj file are denoted by the letter 'v' (lowercase V) with three floating-point numbers depicting the coordinates of the vertex in 3-D space. The three numbers denote the values on X-axis, Y-axis and Z-axis respectively. Below is a snippet of how the vertices are recorded in obj file.

```
v 0.021340 -0.066697 0.032358
v 0.021342 -0.068211 0.031440
v 0.022450 -0.064421 0.040234
v 0.021449 -0.068080 0.038846
v 0.020236 -0.069652 0.038090
v 0.021235 -0.066677 0.039296
v 0.018850 -0.070166 0.036022
v 0.020036 -0.067679 0.036682
v 0.019657 -0.068525 0.037513
v 0.018343 -0.071507 0.036921
v 0.027620 -0.038890 0.044070
v 0.031860 -0.034190 0.044180
v 0.034100 -0.039940 0.041610
v 0.029710 -0.043880 0.041890
v 0.021798 -0.065240 0.034834
```

Figure 4.5: Vertices recorded in obj file

UV coordinates are denoted by the letters 'vt' (lowercase VT) with two floating-point numbers depicting the UV coordinates of the 3-D mesh in Euclidean space, i.e. on a flat-surface. VT is an acronym for Vertex Texture. The two numbers following 'vt' are the values on U-axis and V-axis. UV coordinates in an obj file look like the snapshot given below.

```
vt 0.884414 0.505306
vt 0.747333 0.498974
vt 0.779113 0.461874
vt 0.537987 0.831977
vt 0.495288 0.848342
vt 0.494855 0.835964
vt 0.562644 0.951471
vt 0.533411 0.938220
vt 0.569259 0.932593
vt 0.548267 0.406778
vt 0.498757 0.374878
vt 0.553227 0.375172
vt 0.530706 0.521719
vt 0.493148 0.502888
vt 0.534078 0.507133
```

Figure 4.6: UV coordinates recorded in obj file

Faces are denoted by the letter 'f' (lowercase F), followed by three sets of two integer separated by a single space. They are recorded as 'f v1/vt1 v2/vt2 v3/vt3 . . . . . .'. Since we are dealing with triangular meshes, exactly three sets of two-integers are recorded in the file. The two integers in a set are separated by a forward-slash (/). The first integer denotes the index number of the vertex in the file and the second integer (after the forward-slash) denotes the index of the UV coordinate. This helps in defining the mesh in non-Euclidean and Euclidean space respectively. The snapshot below helps understand how faces are defined in an obj file.

```
f 480/1 120/2 478/3
f 35/4 440/5 441/6
f 53/7 159/8 52/9
f 179/10 423/11 178/12
f 170/13 420/14 171/15
f 169/16 427/17 170/13
f 80/18 11/19 71/20
f 55/21 64/22 53/7
f 47/23 437/24 46/25
f 529/26 431/27 583/28
f 7/29 9/30 8/31
f 20/32 15/33 19/34
f 11/19 13/35 12/36
f 18/37 16/38 15/33
f 26/39 24/40 23/41
```

Figure 4.7: Faces recorded in obj file

Looking at the face (f) on line 1, the set indexes of vertices and UV coordinates are denoted as A/a B/b C/c. Each vertex has three values $[X, Y, Z]$ and UV coordinate has two values $[U, V]$. To get the vertex values of A, we need to locate index number, 480, amongst the list of vertices and to get the UV coordinate of a, we need to locate index number, 1, amongst the list of UV coordinates. An important information with respect to processing data in Python (or any other programming language) will be to keep in mind that indexing in obj files start from 1 and not 0.

The resulting values will provide the vertex value of

A = [0.036861, -0.052547, 0.041362] and UV coordinate values of a = [0.804414, 0.505306].
Similarly, B = [0.030129, -0.053289, 0.045191], b = [0.747333, 0.490974] and
C = [0.033326, -0.056989, 0.042059] and c = [0.779113, 0.461074]

4.2 Process of converting non-Euclidean data into its Euclidean representation

4.2.1 Fixed Topology of the dataset

The subset of CoMA (Ranjan et al., 2018) dataset made available consists of 700 meshes and each mouth mesh in the dataset has a fixed topology. This means each mesh consists of fixed number of vertices, UV coordinates and faces.

Element	Fixed element values across all meshes?	Total number of elements in each mesh
Vertex	No	534
UV Coordinate	Yes	534
Face	Yes	986

From the above table, the values provided in UV coordinates and faces are constant throughout all mouth meshes. The only variable elements are the vertex values mentioned in each mouth mesh. Experimenting with a fixed topology offers simplicity of operations. It will not also enable the neural network to train well but also aid to test if the neural network can produce results close to ground truth. To widen the scope and applications of adapting non-Euclidean data into a neural architecture, experimenting with a fixed topology will prove to be the starting point towards a new field of research if the results are proven successful.

4.2.2 Pre-Processing

Let us consider a single mesh file for better understanding of the pre-processing task. Elements are read from a single mesh file (obj) and stored in such a way that these different elements are correctly segregated. More details about the process of storing elements is provided below:

Vertex extraction

The X, Y, Z values of 534 vertices are stored as an array ‘vertex’ with shape (534, 3) where 534 are the total number of vertices and 3 denotes the three values on the X, Y, and Z-axis. These values are used to obtain the image representation of the 3-D mesh when no rotation is applied.

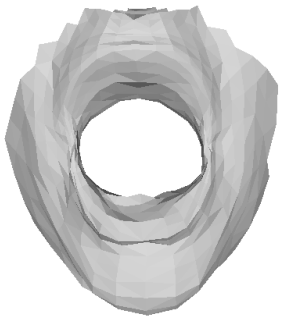


Figure 4.8: Mesh with no rotation

Rotating the vertex

The rotation being tested is at an angle of 90° about the Z-axis in clockwise direction. The simplest way to obtain such a rotation is by creating an array with the same properties as 'vertex',
Step 1: Swapping the values in X-axis with the values in Y-axis
Step 2: Multiplying the new X-axis by -1

So, at 90 degrees, the X-axis values becomes the Y-axis values, and the Y-axis values becomes negative X-axis values which is then stored in an array 'vertex_r'.

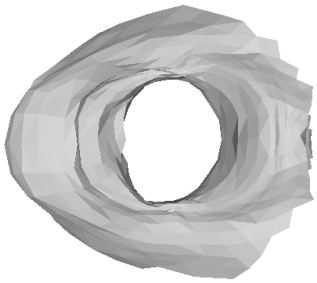


Figure 4.9: Mesh with rotation

UV coordinates extraction

The U, V values of 534 vertices are stored as an array 'uv_coords' with shape (534, 2) where 534 are the total number of vertices and 2 denotes the two values on the U and V axis. These values are used in faces which maps the two sets of vertices (original and rotated vertices) on a flat-surface using these UV coordinate values.

Face extraction

The A, B, C a, b, c values of 986 faces are stored in a list 'face' with shape (986, 3) where 986 are the total number of faces and 3 denotes the three-sets of two-integer values. First integer in each set represents the vertex index stored in an array 'face_vertex_indices' and second integer represent the UV coordinate index stored in an array 'face_vertex_texture_indices'. The integer value in 'face_vertex_indices' and 'face_vertex_texture_indices' starts with '1' that is why the integer index is subtracted by 1 for better understandability and easier processing.

4.2.3 Scaling and Normalization

The vertex values are required to be normalised between 0 and 255 after obtaining the barycentric coordinates for all the faces. This calculation is performed by using the formula,

$$X_{normalised} = (X - X_{minimum}) * 256/(X_{maximum} - X_{minimum})$$

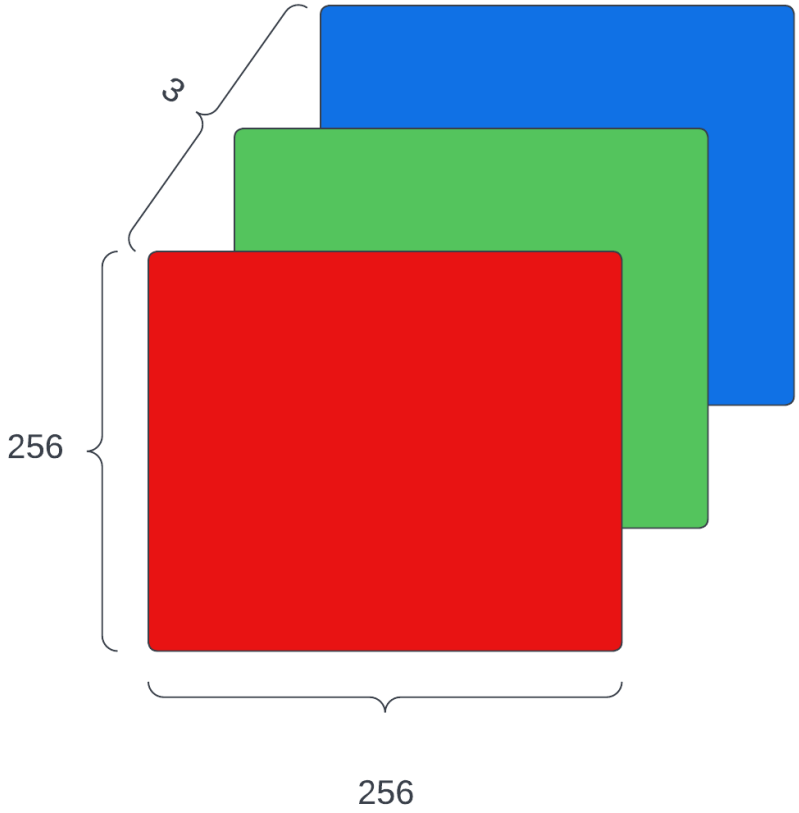
X is the vertex value, $X_{minimum}$ and $X_{maximum}$ is obtained by acquiring the minimum and maximum vertex value from all the mesh data available.

The minimum value of vertex from all the 700 meshes is -0.129 and the maximum is 0.129. In this experiment, these values are rounded-off to two decimal points. Hence, $X_{minimum} = -0.13$ and $X_{maximum} = 0.13$.

UV coordinates are already normalised, hence there is no need to normalise the UV coordinates. All the UV coordinates are scaled by multiplying each coordinate value by 256.

4.2.4 Barycentric coordinates

This is the most crucial process of representing a 3-D mesh into a grid-like image. It is used in shading, providing a smooth gradient across the image. At first an array with shape (256, 256, 3) is defined where 256 x 256 is the height and width of the image and 3 are the number of layers for the colours Red, Blue and Green. Below illustration helps understand the shape better.



The formula for acquiring the Barycentric points of the vertices is defined by,

$$P = A\_v + r * (B\_v - A\_v) + t * (C\_v - A\_v)$$

P is the list barycentric point of the vertices,
r and t return evenly spaced numbers over a specified interval such that the values do not exceed 1, i.e., $r + t <= 1$, in this experiment r and t return 35 evenly sampled spaces calculated over the interval $[0, 1]$
A_v, B_v and C_v are the vertices of the triangle derived from the index integers mentioned in the faces.

The Barycentric points implemented for UV coordinates is defined as,

$$P\_uv = U\_vt + r * (V\_vt - U\_vt) + (W\_vt - U\_vt)$$

P_uv is the list of barycentric points for the UV coordinates, r and t follow the same rule while calculating P, U_vt, V_vt, W_vt are the UV coordinates of the triangle derived from the index integers mentioned in the faces.

4.2.5 Image representation of 3-D mesh (with and without rotation)

This is the final step in acquiring the image representation. A loop is iterated starting from zero till the end of the length of P, P_uv is scaled by multiplying it by 256. A nested loop is iterated which maps the normalised barycentric coordinates of the vertices with the image array of shape $(256, 256, 3)$. This helps in acquiring a smooth gradient.

The entire procedure is repeated twice, once for the mesh with no rotation and second time for the mesh with rotation. Shape of the two image representation generated side-by-side is $256x512x3$. Similarly, the process is repeated 700 times since the CoMA dataset (Ranjan et al., 2018) contains 700 mesh files.

4.2.6 Implementing the image representation into an existing neural architecture, Pix2Pix

The code is directly taken from the paper (Isola et al., 2017). No changes are made to the Pix2Pix model. The dataset generated above needs to be loaded directly to the model. The dataset is split in three folders, viz., train, val and test. The train, val, test split is $60 : 30 : 10$. The train folder contains 420 mesh files, val contains 210 mesh files and test contains 70 mesh files, getting a total of 700 mesh files.

4.2.7 Validating the results

The output image generated by Pix2Pix is of size 1080 x 1080 which contains input image, ground truth and predicted image. The pixel position of each image is constant. By using the values mentioned below, images are cropped for validation.

For cropping ground truth image from the main output image,

Side	Pixels
Left	430
Top	414
Right	676
Bottom	660

For cropping predicted image from the main output image,

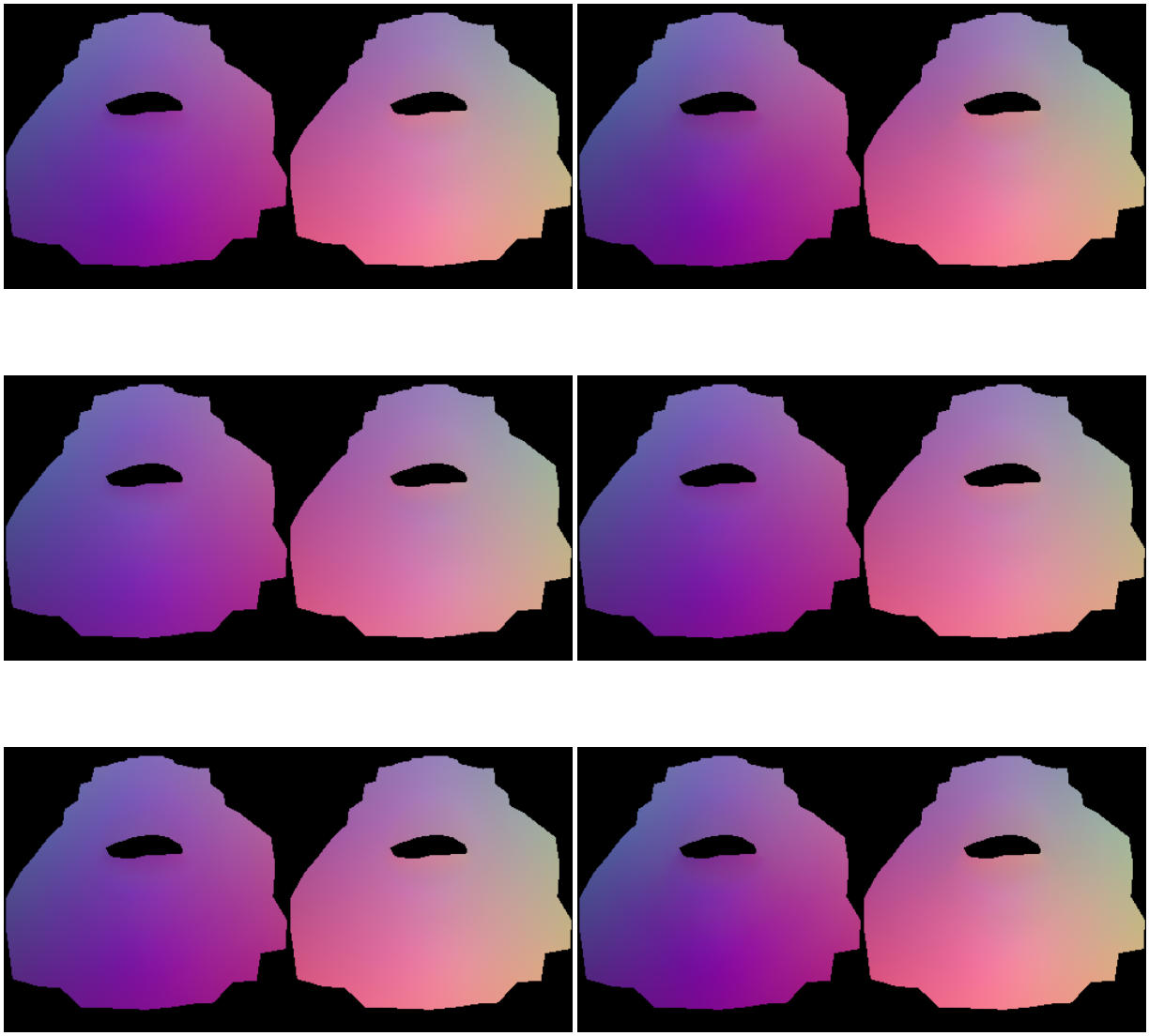
Side	Pixels
Left	726
Top	414
Right	972
Bottom	660

Chapter 5

Results

5.1 Dataset created

Below are a few examples of the image representations created as in input for the experiment.



Shape of each pair of images is equal to the shape of the facade dataset used in Pix2Pix model (Isola et al., 2017), i.e, 256 x 512 x 3.

5.2 Pix2Pix results

After parsing the dataset in Pix2Pix, the input image, ground truth, predicted image is saved at different stages of step completions. The shape of each image has reduced during Pix2Pix processing, resulting in 246 x 246 x 3.

From 0 steps to 40k steps:

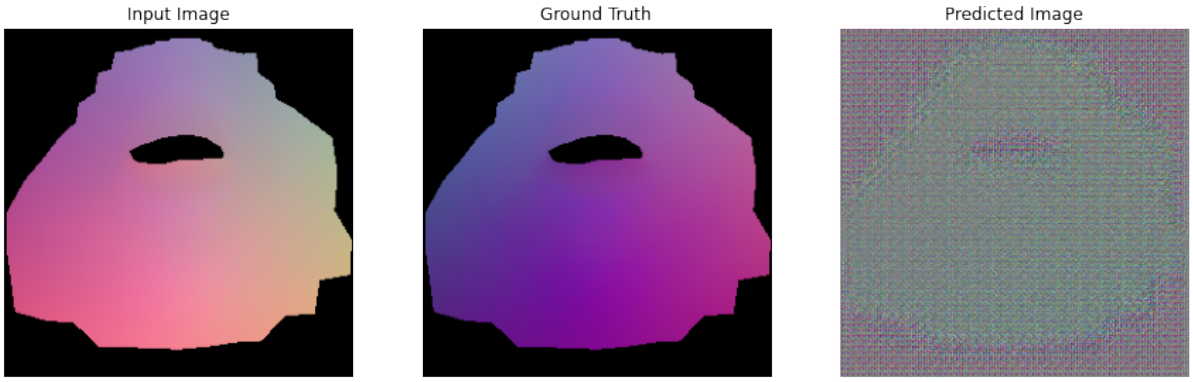


Figure 5.1: Starting point - 0k steps

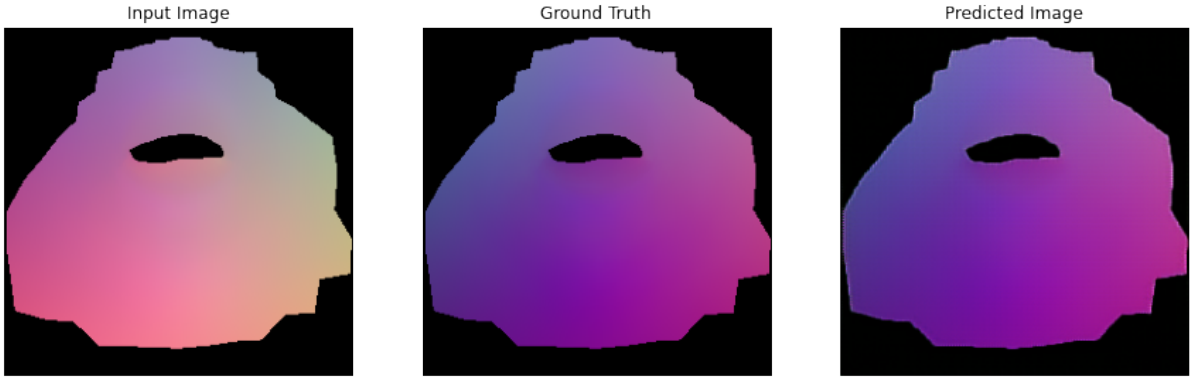


Figure 5.2: After 1k steps

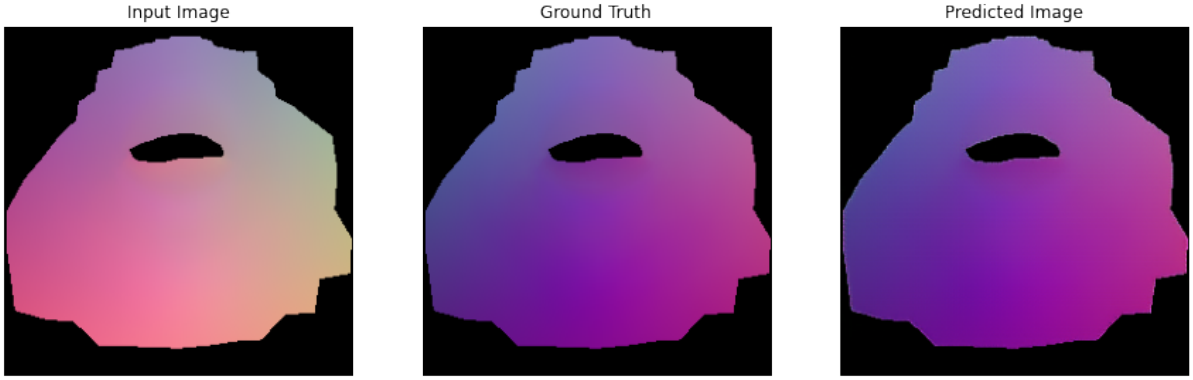


Figure 5.3: After 2k steps

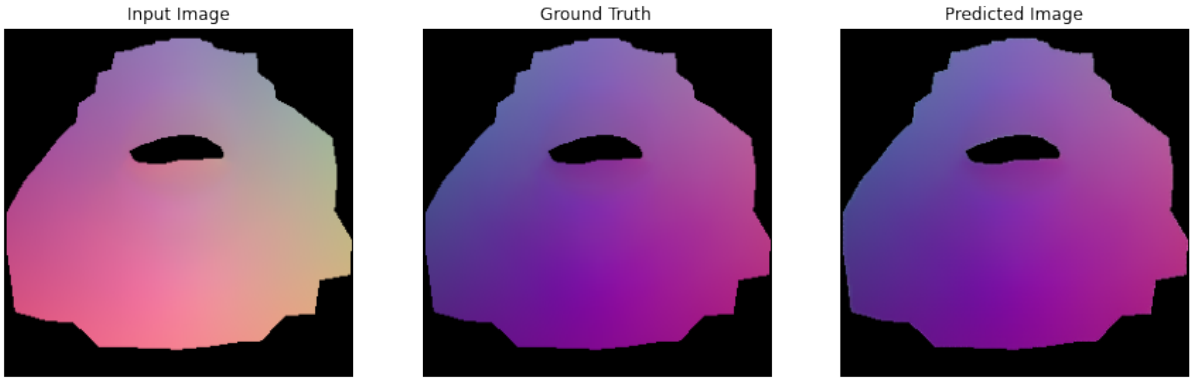


Figure 5.4: After 5k steps

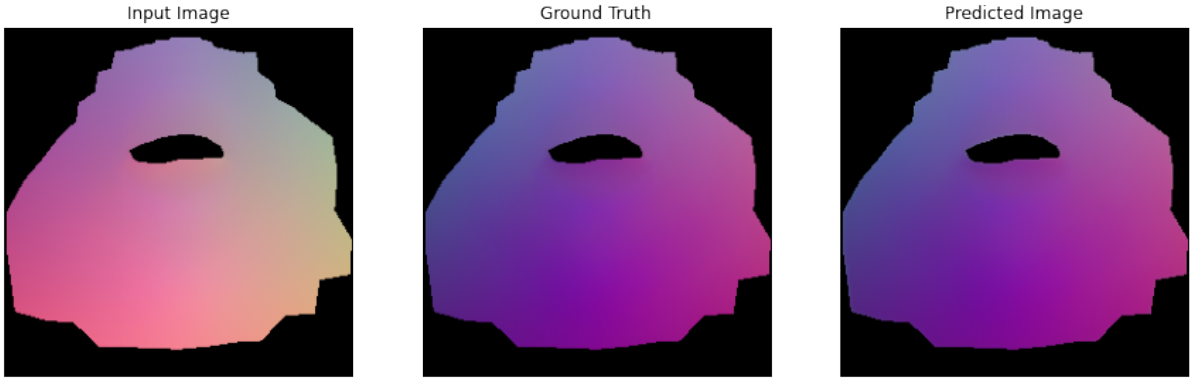


Figure 5.5: After 10k steps

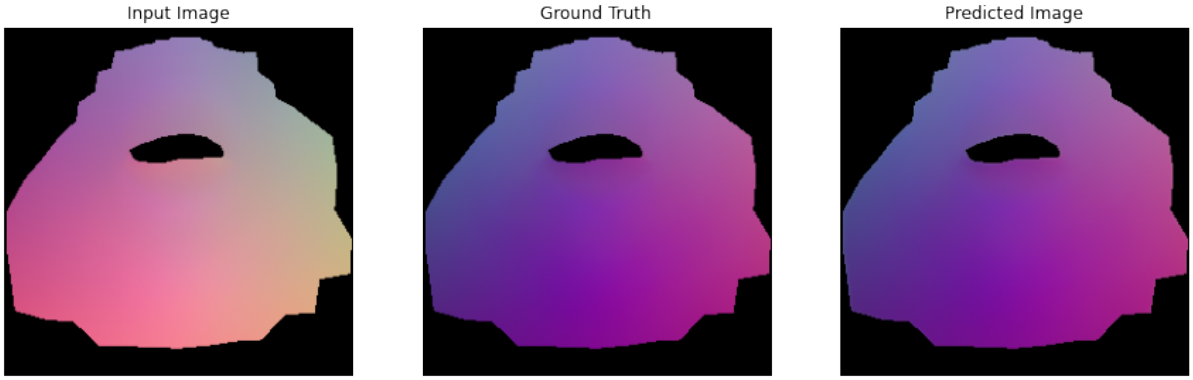


Figure 5.6: After 15k steps

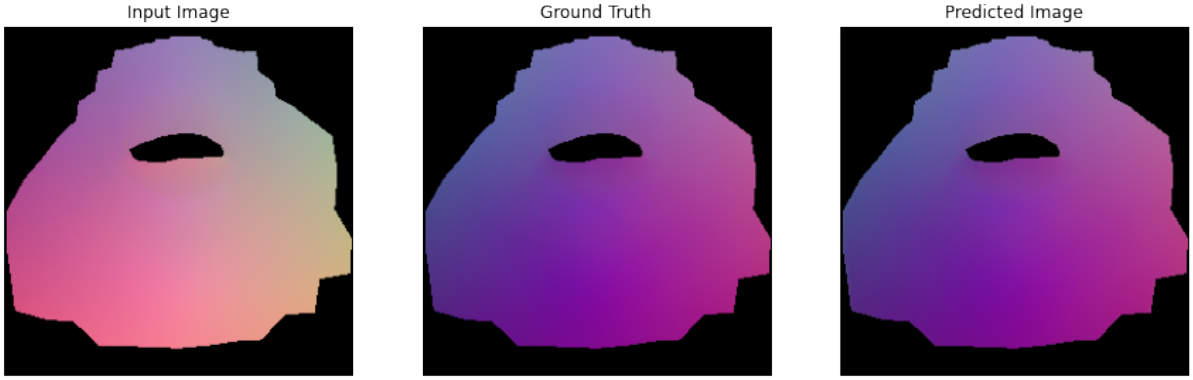


Figure 5.7: After 20k steps

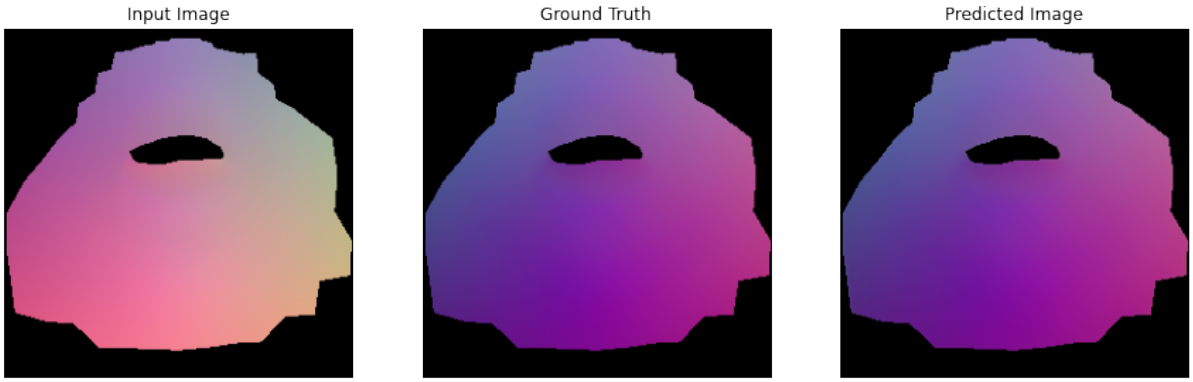


Figure 5.8: After 25k steps

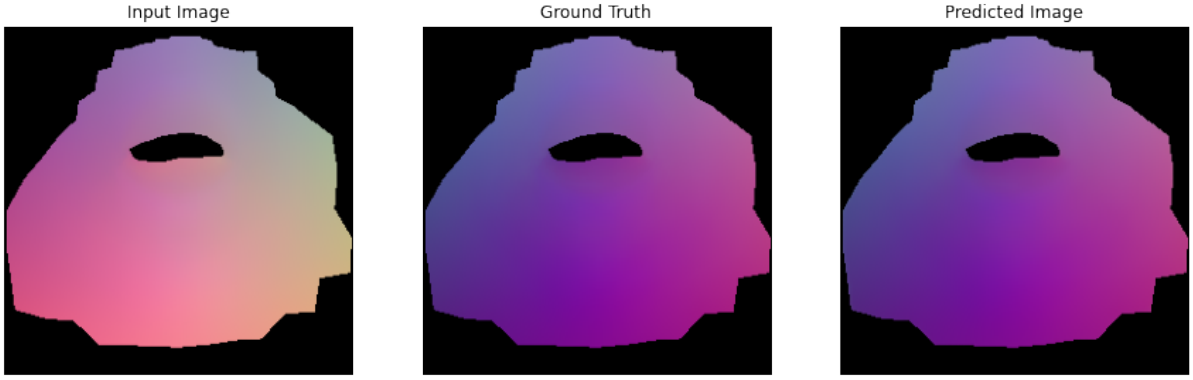


Figure 5.9: After 30k steps

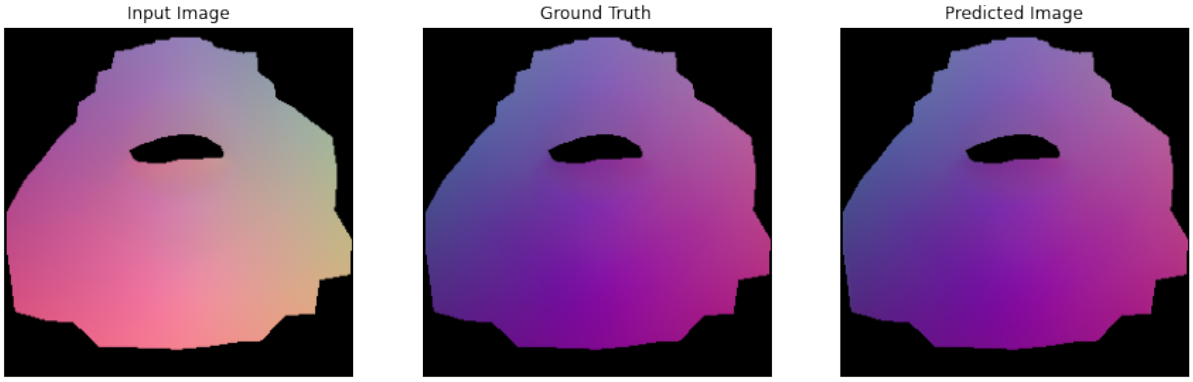


Figure 5.10: Final stage - 40k steps

By looking at the predicted images in the above example, it is fair to presumed that the Pix2Pix model is doing a brilliant job when it comes to transforming geometric figures at a fixed angle. However, instead of obtaining the original vertices back, this experiment is comparing the ground truth and the predicted images to gauge the accuracy of the transformation.

To bring the images mentioned above into perspective, below table helps understand the accuracy of the predicted image at every stage as compared to the ground truth.

Steps	Accuracy in %
0k	0.33%
1k	21.56%
2k	35.20%
5k	61.18%
10k	77.38%
15k	68.78%
20k	81.68%
25k	72.96%
30k	75.59%
40k	78.70%

The Pix2pix model learns the dataset pretty well after the after 1k steps and the accuracy peaks after 20k steps with 81.68% accuracy. At the end of 40k, the model generates a predicted image which matches 78.70% to the ground truth.

Chapter 6

Observation, Discussion and Analysis

Apart from the dimensions of the images getting reduced from inputting 256x256 images to getting 246x246 output images, a key observation would be that the edges of all the image representations, after being processing in Pix2pix network appear to be blurred as compared to the original input provided to the network. Below are a couple of zoomed-in examples of the original input compare to the image representation provided by Pix2Pix.

Figure 6.1: Original input with sharp edges fed to Pix2Pix network

Figure 6.2: Blurred edges generated by Pix2Pix network

Since we are dealing with lesser number of vertices, UV coordinates and faces, this issue is not severe and the outcome is still meaningful. It might cause potential deformation problems when dealing with a larger and more complex non-Euclidean 3-D mesh dataset having a variable topology.

The limitation would be the use of fixed normalization range used $[-0.13, 0.13]$. Only meshes with maximum vertex value 0.13 and minimum vert value -0.13 can be considered in this experiment. If a random test dataset has a vertex value greater than 0.13 or less than -0.13, the image representation will consist of a different vibrant texture. On contrary, considering a wider range would not result in a true image representation as well. In both cases, the

experiment will be unable to achieve a true Euclidean representation from a non-Euclidean dataset.

Another limitation would be fixed topology, only 3-D meshes with 534 vertices, 534 UV coordinates and 986 faces can be executed in this experiment. Any other 3-D mesh with more or less number of elements (vertices, UV coordinates and faces) would either cause an error in the code or a distortion in the image.

This experiment focuses on achieving meaningful geometric transformation limited at a fixed angle only. All the meshes are rotated at an angle of 90^o about the Z-axis. Changing the rotation at different angles will help understand the potential of Pix2Pix network.

If emphasis is given only on achieving geometric transformation, considering the use of Graph Neural Network will be useful (Asif et al., 2021) for working with non-Euclidean data.

Chapter 7

Conclusions

The main aim of this experiment is to validate if the network is able to achieve geometric transformation given a set of Euclidean image representation derived from non-Euclidean dataset. Overall, the Pix2Pix network has performed fairly well. The network generated a predicted images which is 78.70% accurate. The experiment looks like a success, in the sense that it is fit be validated for complex geometric transformation in future. The only caveat being the topology needs to be fixed in order to train the network and compare the results. Also, experimenting with state-of-the-art neural network which are specifically developed for handling non-Euclidean data could be considered for future works.

Testing the model in a fixed angle rotation can be termed as a starting point in experimenting Euclidean representations with neural networks and if successful, will be the steppingstone towards implementing more complex transformations like testing the model with variable angle rotations, tweaking the mesh to change its expression, etc. Further applications of being able to process Euclidean representation of non-Euclidean data using Pix2Pix would include developing gaming models with better 3-D visualizations, generating fake-images, etc.

Bibliography

Asif, N.A., Sarker, Y., Chakrabortty, R.K., Ryan, M.J., Ahamed, M.H., Saha, D.K., Badal, F.R., Das, S.K., Ali, M.F., Moyeen, S.I. et al., 2021. Graph neural network: A comprehensive review on non-euclidean space. *Ieee access*, 9, pp.60588–60606.

Barahona, S., Gual-Arnau, X., Ibáñez, M.V. and Simó, A., 2018. Unsupervised classification of children's bodies using currents. *Advances in data analysis and classification*, 12(2), pp.365–397.

Birkhoff, G.D., 1932. A set of postulates for plane geometry, based on scale and protractor. *Annals of mathematics*, pp.329–345.

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

Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A. and Vandergheynst, P., 2017. Geometric deep learning: going beyond euclidean data. *Ieee signal processing magazine*, 34(4), pp.18–42.

Brownlee, J., 2019. A gentle introduction to pix2pix generative adversarial network. Available from: `https://machinelearningmastery.com/a-gentle-introduction-to-pix2pix-generative-adversarial-network/#:~:text=Pix2Pix%20is%20a%20Generative%20Adversarial,presented%20at%20CVPR%20in%202017.`

Cheng, S., Kotsia, I., Pantic, M. and Zafeiriou, S., 2018. 4dfab: A large scale 4d database for facial expression analysis and biometric applications. *Proceedings of the ieee conference on computer vision and pattern recognition*. pp.5117–5126.

Ciregan, D., Meier, U. and Schmidhuber, J., 2012. Multi-column deep neural networks for image classification. *2012 ieee conference on computer vision and pattern recognition*. IEEE, pp.3642–3649.

Cosker, D., Krumhuber, E. and Hilton, A., 2011. A facs valid 3d dynamic action unit database with applications to 3d dynamic morphable facial modeling. *2011 international conference on computer vision*. IEEE, pp.2296–2303.

Cudeiro, D., Bolkart, T., Laidlaw, C., Ranjan, A. and Black, M.J., 2019. Capture, learning, and synthesis of 3d speaking styles. *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*. pp.10101–10111.

Dai, H., Pears, N., Smith, W.A. and Duncan, C., 2017. A 3d morphable model of craniofacial shape and texture variation. *Proceedings of the ieee international conference on computer vision*. pp.3085–3093.

Defferrard, M., Bresson, X. and Vandergheynst, P., 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29.

Egger, B., Smith, W.A., Tewari, A., Wuhrer, S., Zollhoefer, M., Beeler, T., Bernard, F., Bolkart, T., Kortylewski, A., Romdhani, S. et al., 2020. 3d morphable face models—past, present, and future. *Acm transactions on graphics (tog)*, 39(5), pp.1–38.

Fletcher, P.T. and Joshi, S., 2007. Riemannian geometry for the statistical analysis of diffusion tensor data. *Signal processing*, 87(2), pp.250–262.

Gori, M., Monfardini, G. and Scarselli, F., 2005. A new model for learning in graph domains. *Proceedings. 2005 ieee international joint conference on neural networks*. vol. 2, pp.729–734.

Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S. and Cohen-Or, D., 2019. Meshcnn: a network with an edge. *Acm transactions on graphics (tog)*, 38(4), pp.1–12.

Isola, P., Zhu, J.Y., Zhou, T. and Efros, A.A., 2017. Image-to-image translation with conditional adversarial networks. *Proceedings of the ieee conference on computer vision and pattern recognition*. pp.1125–1134.

Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2017. Imagenet classification with deep convolutional neural networks. *Communications of the acm*, 60(6), pp.84–90.

Lazcano, D., Fredes, N. and Creixell, W., 2021. Hyperbolic generative adversarial network. *arxiv preprint arxiv:2102.05567*.

Masci, J., Boscaini, D., Bronstein, M. and Vandergheynst, P., 2015. Geodesic convolutional neural networks on riemannian manifolds. *Proceedings of the ieee international conference on computer vision workshops*. pp.37–45.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J. and Bronstein, M.M., 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. *Proceedings of the ieee conference on computer vision and pattern recognition*. pp.5115–5124.

Nash, C., Ganin, Y., Eslami, S.A. and Battaglia, P., 2020. Polygen: An autoregressive generative model of 3d meshes. *International conference on machine learning*. PMLR, pp.7220–7229.

Paysan, P., Knothe, R., Amberg, B., Romdhani, S. and Vetter, T., 2009. A 3d face model for pose and illumination invariant face recognition. *2009 sixth ieee international conference on advanced video and signal based surveillance*. Ieee, pp.296–301.

Ranjan, A., Bolkart, T., Sanyal, S. and Black, M.J., 2018. Generating 3d faces using convolutional mesh autoencoders. *Proceedings of the european conference on computer vision (eccv)*. pp.704–720.

Scratchapixel, 2014. Barycentric coordinates. Available from: `https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/barycentric-coordinates`.

Singh, P.K., 2022. Data with non-euclidean geometry and its characterization. *Journal of artificial intelligence and technology*, 2(1), pp.3–8.

Su, H., Maji, S., Kalogerakis, E. and Learned-Miller, E., 2015. Multi-view convolutional neural networks for 3d shape recognition. *Proceedings of the ieee international conference on computer vision*. pp.945–953.

Sun, Y., Wang, Y., Liu, Z., Siegel, J. and Sarma, S., 2020. Pointgrow: Autoregressively learned point cloud generation with self-attention. *Proceedings of the ieee/cvf winter conference on applications of computer vision*. pp.61–70.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P. and Bengio, Y., 2017. Graph attention networks. *arxiv preprint arxiv:1710.10903*.

Vinué, G., Simó, A. and Alemany, S., 2016. The $$ k $$ k-means algorithm for 3d shapes with an application to apparel design. *Advances in data analysis and classification*, 10(1), pp.103–132.

Wei, L., Huang, Q., Ceylan, D., Vouga, E. and Li, H., 2016. Dense human body correspondences using convolutional networks. *Proceedings of the ieee conference on computer vision and pattern recognition*. pp.1544–1553.

Wikipedia contributors, 2022a. Data — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Data&oldid=1103098610`. [Online; accessed 12-August-2022].

Wikipedia contributors, 2022b. Deep learning — Wikipedia, the free encyclopedia [Online]. [Online; accessed 3-September-2022]. Available from: `https://en.wikipedia.org/w/index.php?title=Deep_learning&oldid=1108086973`.

Wikipedia contributors, 2022c. Generative adversarial network — Wikipedia, the free encyclopedia [Online]. [Online; accessed 4-September-2022]. Available from: `https://en.wikipedia.org/w/index.php?title=Generative_adversarial_network&oldid=1107364266`.

Wikipedia contributors, 2022d. Neural network — Wikipedia, the free encyclopedia [Online]. [Online; accessed 3-September-2022]. Available from: `https://en.wikipedia.org/w/index.php?title=Neural_network&oldid=1105137149`.

Wikipedia contributors, 2022e. Polygon mesh — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Polygon_mesh&oldid=1097100057`. [Online; accessed 29-August-2022].

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C. and Philip, S.Y., 2020. A comprehensive survey on graph neural networks. *Ieee transactions on neural networks and learning systems*, 32(1), pp.4–24.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X. and Xiao, J., 2015. 3d shapenets: A deep representation for volumetric shapes. *Proceedings of the ieee conference on computer vision and pattern recognition*. pp.1912–1920.

Zhang, Z., Girard, J.M., Wu, Y., Zhang, X., Liu, P., Ciftci, U., Canavan, S., Reale, M., Horowitz, A., Yang, H. et al., 2016. Multimodal spontaneous emotion corpus for human behavior analysis. *Proceedings of the ieee conference on computer vision and pattern recognition*. pp.3438–3446.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C. and Sun, M., 2020. Graph neural networks: A review of methods and applications. *Ai open*, 1, pp.57–81.