FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Material Picker: tool for detecting material properties
Student:	Bc. Anh Viet Tran
Supervisor:	Ing. Tomáš Nováček
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2023/2024

Instructions

Chaos Corona, a CPU-based rendering company, is looking for ways to create physically real materials from images. They want to create Material picker, a tool which would use artificial intelligence to detect the right material properties from the image, like roughness, bump or index of refraction.

The goals of the thesis:
1) Analyze the possibilities of extraction material properties from images.
2) Survey current state-of-the-art approaches and tools that extract material properties from any input source.
3) With the use of artificial intelligence, design a method to detect at least three material properties from an image or, if such methods already exist, choose one and extend/ improve it.
4) Experimentally evaluate your solution in terms of precision and accuracy and demonstrate results using one type of material (e.g. carpets).

References:
Jurčák Filip, Material picker: Material recognition in images using deep learning, Comenius University in Bratislava Faculty of Mathematics, Physics and Informatics (2020)

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 20 January 2023 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Material Picker: tool for detecting material properties

Bc. Viet Anh Tran

Department of Applied Mathematics
Supervisor: Ing. Tomáš Nováček

April 1, 2024

Acknowledgements

me, myself and I

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on April 1, 2024 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Tran, Viet Anh. *Material Picker: tool for detecting material properties.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Abstrakt

TODO CZ

Klíčová slova TODO

Abstract

TODO

Keywords TODO

Contents

List of Figures

Introduction

Computer graphics and 3D modeling is a fundamental field in our constantly increasing digital world. Its application in engineering, science, art, product visualization, and most notably, entertainment has pushed this field with a constant demand for better visual quality and performance optimization.

3D models create the very core of any 3D visualization, bringing objects to computer-generated 3D space. The 3D model defines the shape of an object and can be further improved to closely represent real-world objects by applying textures, lighting, and many more techniques. Models are typically created in 3D modeling software by an artist or by 3D scanning. 3D scanning can often bring too many details in terms of polygon numbers. A shape that could be perfectly recognizable by ten polygons would now be described by thousands. The side effect of things is big hardware requirements.

Lowering the number of polygons is a usual practice done either manually or by many developed tools over the years. These tools do not consider any edge flow, a critical feature particularly important for character modeling and animation. Edge flow allows smooth deformation. In terms of animation, we can look at muscle movement, optimal edge avoids wrinkled defects on the character model, and the skin would stretch and contract in a way as we are used to seeing in the real world.

This thesis aims to introduce the final step in polygon reduction workflow when using tools. We present a neural network taking a 3D model and outputting an improved model in terms of edge flow quality, approximating similar polygon quantity.

In Chapter Neural Networks, we explore basics of neural networks, introduce its terminology and several commonly used network architectures.

In Chapter Implementation, we describe used methods and key implementation points of our work.

In Chapter Experiments

In Conclusion, we evaluate the results of the work and suggest possible improvements for future works.

Chapter 1

Neural Networks

An artificial neural network (ANN), first introduced by Warren McCulloch and Walter Pitts in "A logical calculus of the ideas immanent in nervous activity" published in 1943 [11], is a mathematical model based on biological neural networks. It carries the ability to learn and correct errors from previous experience [12], [13].

The ANN has gained popularity in recent years with still increasing advancements in technology and availability of training data. ANN now becomes a default solutions for complex tasks previously thought to be unsolvable by computers [14].

This chapter will briefly introduce different types of neural units and their activation functions, along with some commonly used network architectures.

1.1 Artificial Neuron

Artificial neurons are units of ANN, mimicking behavior of biological neurons. Just as biological neurons, it can receive as well as pass information to other neurons.

1.1.1 Perceptron

Perceptron, developed by Frank Rosenblatt in 1958 [15], is the simplest class of artificial neurons.

Perceptron takes several binary inputs in form of a vector $\vec{x} = (x_1, x_2, ..., x_n)$, and outputs a single binary number. Perceptron uses real numbers called *weights*, assigned to each edge, vector $\vec{w} = (w_1, w_2, ..., w_n)$, to express the importance of respected input edges,

A *step function* calculates the perceptron's output. The function output is either 0 or 1 determined by whether its weighted sum $\alpha = \sum_i x_i w_i$ is less

or greater than its *threshold* value, a real number, usually represented as an incoming edge with a negative weight -1 [1].

$$output = \begin{cases} 1, & \text{if } \alpha \geq threshold \\ 0, & \text{if } \alpha < threshold \end{cases} \tag{1.1}$$

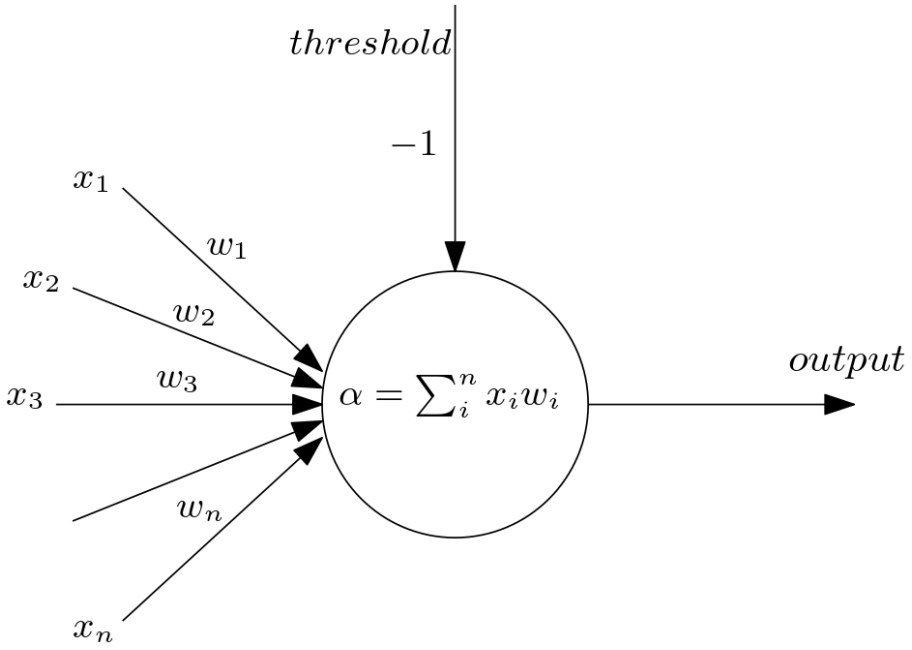


Figure 1.1: Perceptron [1]

1.1.2 Sigmoid Neuron

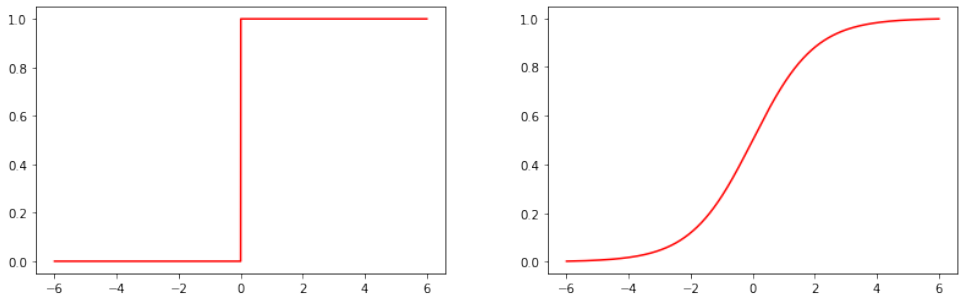
Sigmoid neuron, similarly to perceptron, has vector inputs $\vec{x}$ and weights. The difference is in the output value and its calculation. Instead of perceptron's binary output 0 or 1, a sigmoid neuron outputs a real number between 0 and 1 using a *sigmoid function* [16], [17], [1].

$$\sigma(\alpha) = \frac{1}{1 + e^{-\alpha}} \tag{1.2}$$

As shown in Figure 1.2, the sigmoid function (1.2b) is a smoothed-out step function (1.2a).

1.1.3 Activation Function

An artificial neuron's activation function defines neuron's output value for given inputs, commonly being $f : \mathbb{R} \to \mathbb{R}$ [18]. An important trait of many activation functions is their differentiability, allowing them to be used for



(a) Step function

(b) Sigmoid function

Figure 1.2: Comparison between step function and sigmoid function

Backpropagation, ANN training algorithm. The activation function needs to have a derivative that does not saturate when headed towards 0 or explode when headed towards inf [1].

For such reasons, step function or any linear function are unsuitable for ANN.

1.1.3.1 Sigmoid Function

The sigmoid function is often used in ANN as an alternative to the step function. A popular choice of the sigmoid function is a *logistic sigmoid*, output value ranging between 0 and 1.

$$\sigma(\alpha) = \frac{1}{1 + e^{-\alpha}} = \frac{e^x}{1 + e^x} \tag{1.3}$$

One of the reasons for its popularity is the simplicity of its derivative calculation:

$$\frac{d}{dx}\sigma(\alpha) = \frac{e^x}{(1 + e^x)^2} = \sigma(x)(1 - \sigma(x)) \tag{1.4}$$

Its disadvantages is the *vanishing gradient*. A problem where if given a very high or very low input values, the prediction would stay almost the same. Possibly resulting in training complications or performance issues [19], [1].

1.1.3.2 Hyperbolic Tangent

Hyperbolic tangent is similar to logistic sigmoid function with a key difference in its output, ranging between -1 and 1.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.5}$$

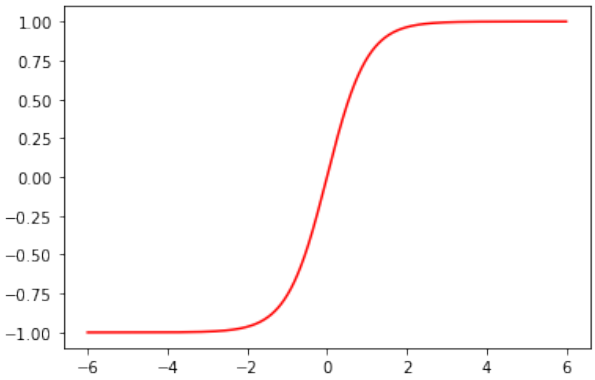


Figure 1.3: Hyperbolic tangent [1]

It shares the sigmoid's simple calculation of its derivative.

$$\frac{d}{dx}\tanh(x) = 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} = 1 - \tanh^2(x) \tag{1.6}$$

By being only moved and scaled version of the sigmoid function, hyperbolic tangent shares not only sigmoid's advantages but also its disadvantages [18], [1].

1.1.3.3 Rectified Linear Unit

The output of the Rectified Linear Unit (ReLU) is defined as:

$$f(x) = max(0, x) \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{1.7}$$

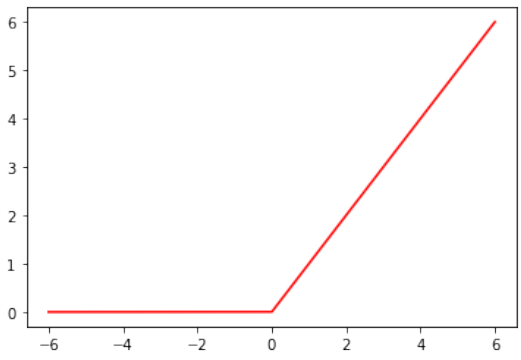


Figure 1.4: Rectified Linear Unit [1]

ReLU's popularity is mainly due to its computational efficiency [19]. Its disadvantages begin to show themselves once inputs approach zero or to a

negative number. Causing the so-called dying ReLu issue, where the network is unable to learn anymore. There are many variations of ReLu to this date, e.g., Leaky ReLU, Parametric ReLU, ELU, ...

1.1.3.4 Softmax

Softmax separates itself from all the previously mentioned functions by its ability to handle multiple input values in the form of a vector $\vec{x} = (x_1, x_2, ..., x_n)$ and output for each x_i defined as:

$$\sigma(x_i) = \frac{e_i^x}{\sum_{j=1}^{n} e_j^x} \tag{1.8}$$

Output is normalized probability distribution, ensuring $\sum_i \sigma(x_i) = 1$ [20]. It is being used as the last activation function of ANN, normalizing the network's output into n probability groups.

1.2 Types of Neural Networks

To this day, there are many types and variations of ANN, each with its structure and use cases. Here we will briefly introduce the most common ones, such as feed-forward networks, convolutional neural networks, or recurrent neural networks.

1.2.1 Feed-forward Networks

Feed-forward network (FFN), the first invented ANN and the simplest variation of an ANN. Its name comes from the way how information flows through the network. The data flows in one direction, oriented from the *input layer* to the *output layer*, without cycles. The input layer takes input data, vector $\vec{x}$, producing $\hat{y}$ at the output layer [21].

FFN contains several hidden layers of various widths but it can also have no hidden layers at all. By having no back-loops, FFN generally minimizes error, computed by *cost function*, in its prediction by using the *backpropagation* algorithm to update its weight values [22], [20].

1.2.1.1 Cost Function

Cost function $C(\vec{w})$ is used in ANN's training process. It takes all weights and biases of an ANN as its input, in the form of a vector $\vec{w}$ and calculates a single real number expressing ANN's error in prediction [23]. Higher number expressing poor prediction and as the number gets lower the ANN's output gets closer to the correct result. The main goal of training is to minimize the cost function.

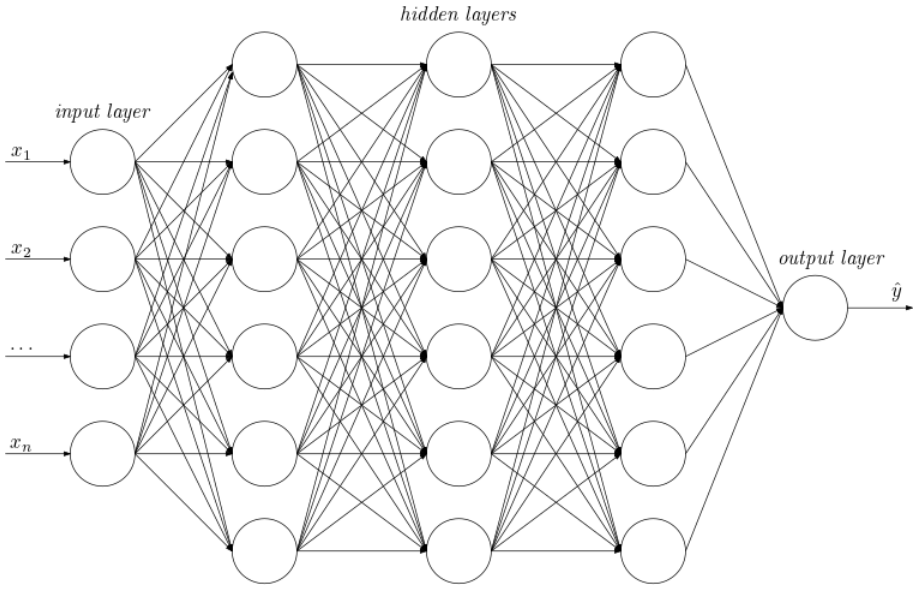


Figure 1.5: Fully connected Feed-forward Neural Network [1]

1.2.1.2 Backpropagation

Backpropagation, short of backward propagation of errors, is a widely used algorithm in training FFN using *gradient descent* to find a local minimum of a cost function and update ANN's weights [24].

The gradient of a multi-variable function provides us with the direction of the gradient ascent, where we should step to rapidly increase the output and find the local maximum. Conversely, the negative of the gradient points towards the local minimum.'

It is common practice to split training samples into small *batches* of size n. For each sample in the batch, we will calculate a gradient descent and use their average gradient descent to update the network's weights. This average gradient descent indicates the adjustments that need to be made to the weights so that the artificial neural network (ANN) moves closer to the correct results [24].

$$-\gamma \nabla C(\vec{w}_i) + \vec{w}_i \rightarrow \vec{w}_{i+1} \tag{1.9}$$

$\vec{w}_i$ is weights of the network at the current state (batch), $\vec{w_{i+1}}$ is updated weights, γ is the learning rate and $-\nabla C(\vec{w}_i)$ is the gradient descent.

1.2.2 Convolutional Neural Networks

The primary objective of a Convolutional Neural Network (CNN) is to enable a computer to identify images and objects, making it ideal for image classification and object recognition tasks.

CNNs are based on the biological processes in the human brain and its connectivity patterns resemble those of the human visual cortex. However, images are perceived differently by the human brain and a computer, with the latter interpreting images as arrays of numbers. As a result, CNNs are designed to work with two-dimensional image arrays, although they can also work with one-dimensional or three-dimensional arrays [25].

CNN is a variation of FNN [23]. It typically consists of an input layer, followed by multiple hidden layers, including several *convolutional layers* and *pooling layers*, and concluding with an output layer.

1.2.2.1 Convolutional Layer

The convolutional layer's goal is to extract key features from the input image by passing a matrix known as a *kernel* over the input image abstracted into a matrix [26].

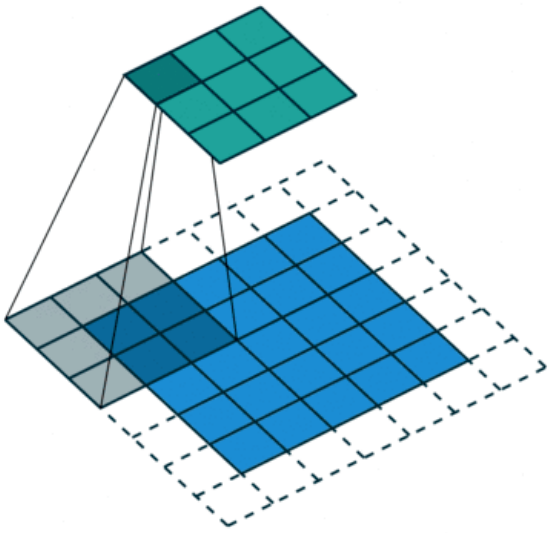


Figure 1.6: Convolution of an 5x5x1 image with 3x3x1 kernel [2]

The outcome of a convolution operation can be either reduced or increased in size. If the size is reduced, it is referred to as *valid padding*. For example, a convolution operation on an 8x8 input image would result in a 6x6 convoluted feature. On the other hand, if the size remains the same or is increased, it is referred to as *same padding* [2].

1.2.2.2 Pooling Layer

Like the convolutional layer, the pooling layer reduces the size of the convolved feature to reduce computational power needed for data processing. It also

extracts dominant features that are invariant to rotation and position, making it beneficial for training the model effectively [2].

There are two types of pooling: max pooling and average pooling. Max pooling returns the maximum value from the portion of the image covered by the kernel, acting as a noise suppressant by removing noisy activations and performing de-noising and dimensionality reduction. Average pooling returns the average of all values in the same portion, reducing dimensions as a noise suppression mechanism. It is worth noting that max pooling performs better [2].

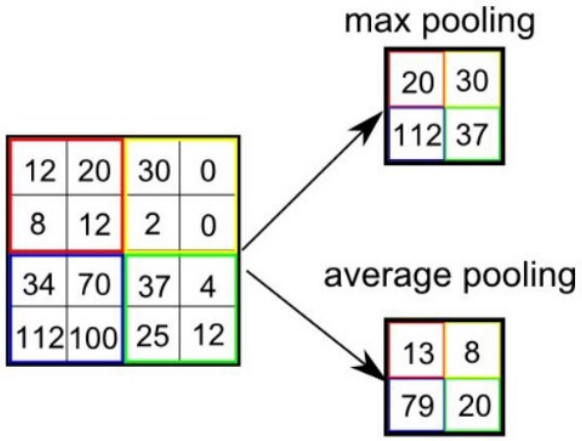


Figure 1.7: Types of pooling [2]

1.2.3 Graph Neural Networks

Graphs are commonly used to describe and analyze entities with relations or interactions. Still, today's deep learning modern toolbox is specialized for simple data types, e.g., grids for images, sequences for text or speech. These data structures have spatial locality, the grid size or sequence length is fixed and can be resized. We can also determine the starting position and ending position. Graph problems are, on the other hand, much more challenging to process as they have arbitrary size and complex topological structure (i.e., no spatial locality like grids). Graphs also have no fixed node ordering or reference point compared to grids or sequences. For such Franco Scarselli et al. introduced graph neural network (GNN) [27].

GNN is designed similarly to CNN. As previously noted, CNN operates on images. Given an image, a rectangular grid, the convolutional layer takes a subpart of the image, applies a function to it, and produces a new part, a new pixel. This is iterated for the whole image. What actually happened is the new pixel resulted in aggregated information from neighbors and itself. This cannot be easily applied to graphs as they have no spatial locality and no fixed

node ordering. As implied, a GNN design stands on passing and aggregating information from neighbors.

1.2.3.1 Node embedding

Graphs require a concept called node embedding. The general idea is to map nodes to a lower dimensional embedding space, where similar nodes in the embedding space approximate similarity in the graph network. For example, we can map a 3D vector to a 2D vector. Node embedding is useful for learning node representations and similarities and can be trained on graphs with arbitrary structures.

Nodes have embeddings at each layer. Taken node v, its layer-0 embedding would be its feature vector x_v. If we want layer-1 embedding, we will explore node v's neighbors. These neighbors are so-called one hop away from our original vector v. We take the feature vector of these nodes and aggregate the information into one single x_v feature vector. Layer-k would get information from nodes that are k hops away.

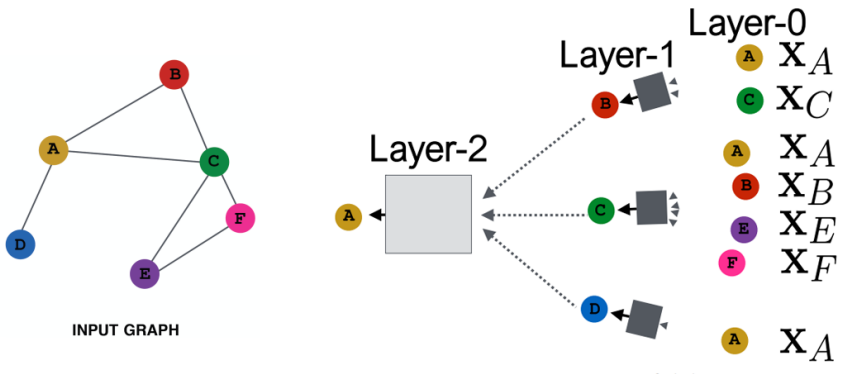


Figure 1.8: Layer-2 embedding applied on node A aggregating infromation from its neighborhood [3]

The described process is called neighborhood aggregation. If we want to predict node v, we need information from its neighborhood, meaning we need a way to propagate the message. Messages are passed and transformed through edges. All received messages are aggregated into a new message and then passed on. This is done systematically for every node in the graph.

The aggregation itself is done by a neural network. This implies the key difference from a typical ANN. Every node gets to define its own neural network and GNN is defined by multiple neural networks.

Using a neural network for each node in the graph, we generate a low-dimensional vector representation, embedding. The network optimizes its parameters to capture important information about the node graphs. The

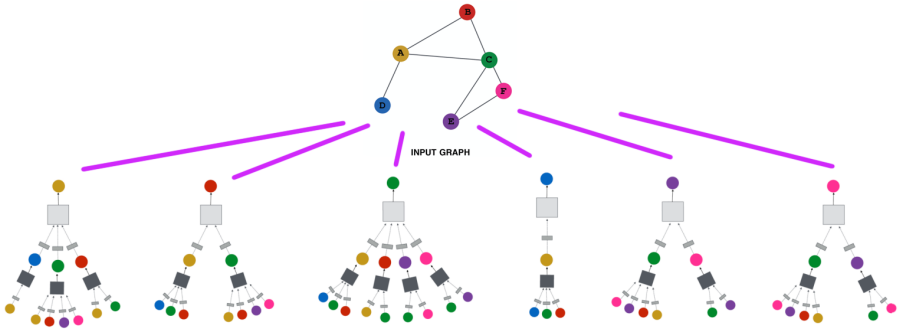


Figure 1.9: Neural network of each node for given input node graph [3]

optimization is typically done by minimizing a loss function expressing dissimilarity between predicted and targeted node embeddings. Similar nodes are close to each other, whereas dissimilar are embedded far apart.

1.2.3.2 Geometric graphs

TODO SECTION Geometric graph is a graph where its nodes represent coordinates in d-dimensional space. Invariant to translation, rotation and scale

... todo more mat...

Chapter 2

Polygon Mesh

In the following chapter, we will discuss shape representation in computer graphics. Explore some of their properties and what tools can be used to modify them. Some of the usual data structures to represent a geometric shape and position within a 3D space are:

- **Mesh** is a set of vertices, edges, and faces defining a 3D object. It is especially used in computer graphics, and it is a simple way to represent complex 3D shapes. A face is a polygon with a minimum of 3 vertices. The most commonly used is a triangle. If a face is made of 4 vertices, it is called a quad, and more than four is called a general polygon. In our case, when we mention faces, we will have triangles in mind. These faces then form a general surface.

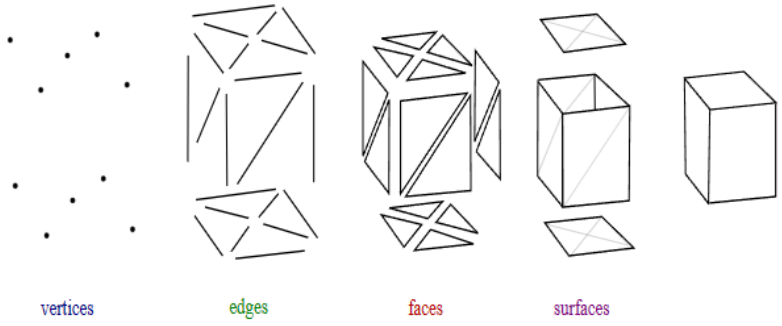


Figure 2.1: Elements of mesh object [3]

A 3D object can have a color. To do so, we assign color values to every vertex of the 3D object. The pixel color of the triangle is determined based on the three vertices by which it is made.

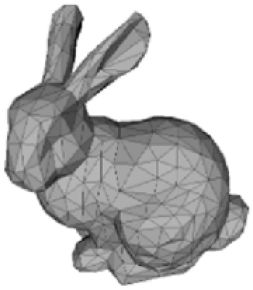


Figure 2.2: Stanford bunny model made of mesh [3]

- **Voxel** objects are in comparison with mesh objects solid. As already said, mesh objects are created from a surface of little triangles, but it is also worth noting that this object is hollow, just like a ballooon. On the other hand, if a model is created from voxels, abbreviation from volumetric pixels, it means that the object was created from cubes and the object itself is solid, its inside also holds information. The working scene is a 3D grid, and its data point holds information about opacity, color, and material information is often also stored.

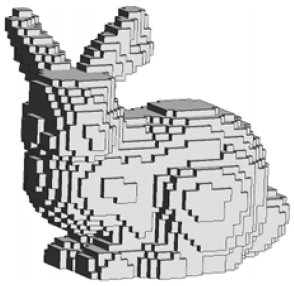


Figure 2.3: Stanford bunny model made of voxels [4]

  Voxels are often used in medicine and terrain representation. Voxel terrain can represent overhangs, caves, arches, and other, which is difficult to represent using heightmaps, which represents only top-layer data, and anything below it would be filled with no option for holes.

  The main disadvantage of voxels is the resolution. If we want to have a highly detailed voxel model, we would have to increase the resolution of the whole scene.

- **Point cloud** is a collection of points plotted in 3D space. Each point contains position coordinates, color values, and luminance values, determining how bright a point is.

Points are usually acquired by a 3D scanner or photogrammetry software. Scanners work by sending out pulses of light to the object and measuring how long each point takes to reflect back and hit the scanner. These measurements are used to determine the exact positions of points on the object, creating a point cloud. Photogrammetry is a process to create measurements from pictures. It uses photos of an object to triangulate points on the object and plot these points to 3D points, resulting in point clouds.

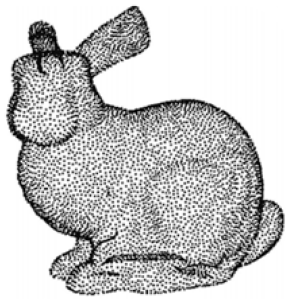


Figure 2.4: Stanford bunny model made of point clouds [**?**]

In terms of the thesis and its goal, we will mainly explore mesh polygons and their edge flow properties while also exploring polygon reduction operations on them.

2.1 Edge flow

Edge flow is a fundamental concept in 3D modeling. The general goal is to ensure that mesh edges follow the curves of an object. A mesh with distinctly different edge flows can represent the same object while preserving the same shape. The key difference and significance of edge flows plays in the world of animation, overall any deformation operation performed on the object. In general, a good edge flow has uniformly distributed points along the 3d model, meaning the length and the area of each primitive is also of similar if not equal sizes.

An optimal edge flow allows smooth and natural deformations. In the case of figure 2.5, if we want to animate various facial expressions, the left example would have distorted wrinkles. In contrast, the right example would allow for natural mimicry, as we are familiar with in real life. A more expressive example would be in 2.6, where we can see a plane of two different topologies having different behavior if bent. The left example has slight bumpy artifacts, while the right preserves smooth surface transition.

Edge flows are manually crafted by 3D artist and require experience to reach their optimal form. They are either being considered from the beginning

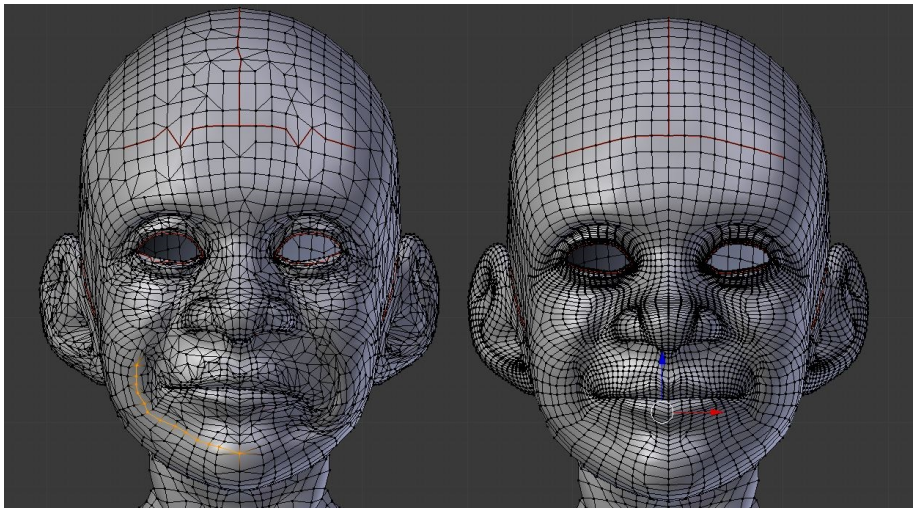


Figure 2.5: A mesh object constructed with 2 different edge flows [5]

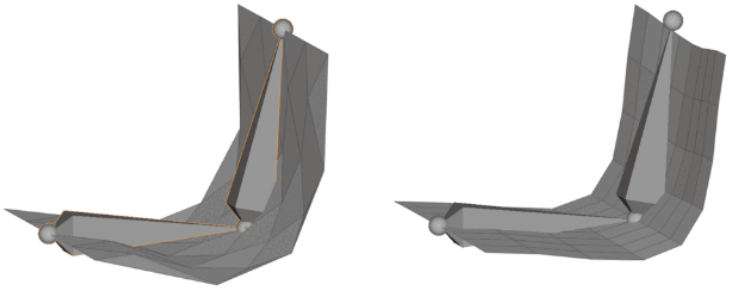


Figure 2.6: Deformation of the same object with different edge flow [6]

while creating the 3D mesh or as a post-correction, where the acquired 3D mesh does not meet the required quality needed for future work. One of the possible reasons for having insufficient quality could be caused by polygon reduction.

2.2 Polygon reduction

Polygon reduction is a common practice for performance and memory optimization as a higher number of polygons may introduce greater details, but they also bring high demand on hardware requirements. A high polygon mesh is usually a byproduct of the 3D scanning process where the resulting 3D mesh can be of enormous memory size. Most importantly, the reduced amount should be considered as the reduction can lose the visual representation of

the object and its key features. The general goal is to balance visual quality and performance optimization. Following, we will list some of the most used techniques for polygon reduction and their effect on the resulting edge flow. Edge flow is a fundamental concept in 3D modeling, playing a pivotal role in natural 3D model deformations mostly used in animation.

2.3 Decimation

Decimation is a commonly used method for quick polygon reduction. It reduces the percentage of vertices and edges uniformly while preserving the overall shape of the reduced object, but it does not handle fine details well. Most 3d modeling software, such as Blender, Maya, 3ds Max, Zbrush, and Cinema4D, support decimation since it is a common technique. The overall control or workflow may differ, but the general concept remains the same.[5]

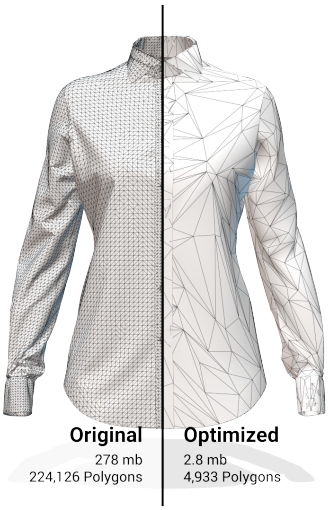


Figure 2.7: Decimation applied on shirt 3D mesh [5]

As seen in 2.7, the optimized 3D mesh may be optimized in the number of polygons, but its edge flow suffered greatly. If the shirt was part of any animated object, its part would deform unnaturally in undesired ways.

2.4 Retopology

Retopology is a semi-automatic method requiring high user input. Retopology requires a user to manually outline the required surface by hand while hinting at its edge flow by following the model's edges before the 3D software follows the outlines and draws a polygon. This gives a user high control over its model but is highly demanding on general experience and skill while identifying the

edges. From a quality perspective, the resulting model will be suitable for animations and smooth 3D model deformations. Still, from a workflow efficiency standpoint, producing one of these reduced models requires a lot of time.

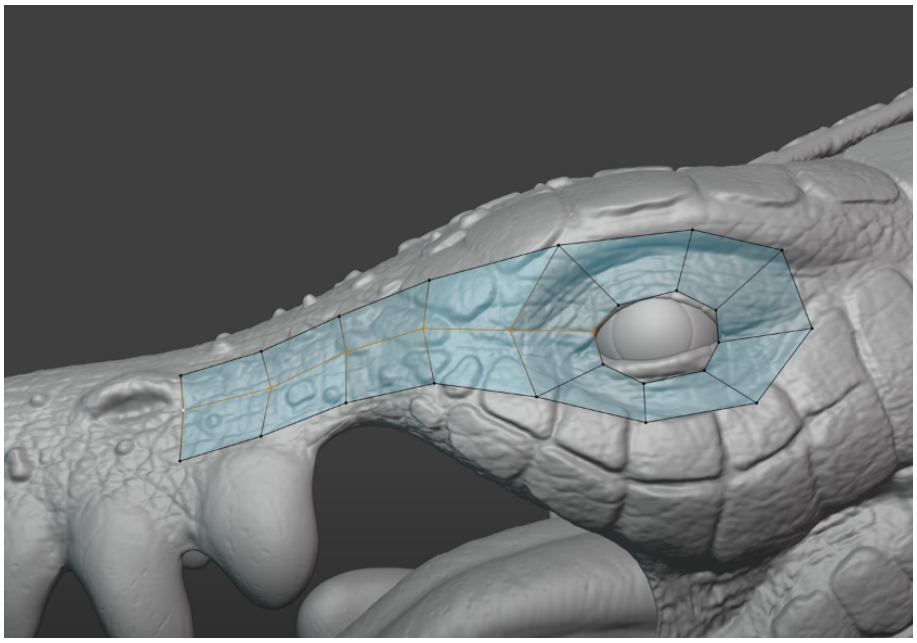


Figure 2.8: Retopology surface manually created lizard 3D mesh [7]

2.5 Quad Remeshing

Instant remesh

Eh...

Decimation -> automatic reduction Decimation does not preserve edge flow

* polygon reduction tools * retopology -> requires manual work Blender * quad remeshing -> often use to preserve nice deformation -> for animation -> main nemesis * edge collapse -> manual work

These reduction techniques don't ensure so called edge flow, a fundamental concept in 3D modeling playing a pivotal role in natural 3D model deformation mostly used in animation.

Chapter 3

Related Work

The main challenge is that there is currently no way to express an optimality of an edge flow by one number, which is critical for learning methods when we want to compute a loss function. We previously mentioned that an optimal flow has uniformly distributed points, edge lenghts and overall area of a primitive. One way to possibly achieve that is to look at the task at hand from different perspective. Instead of doing a correction of the given 3D mesh, we will try to reconstruct the shape of a given 3D mesh with better topology.

In this chapter we will explore researches using neural networks for 3D mesh reconstruction.

3.1 Retrieve and deform a template

Retrieve and deform a template is a two step solution. First a most suitable template is picked and then it is deformed to the target object. The input is processed with neural network which classifies which template is best suited, such as BCNet [8], MultiGarment Net[28] or ShapeFlow [9]. As seen in Figure 3.1 and 3.2, a nearest-neighbor template is retrieved from the embedded space and then a deformation network is applied.

3.2 Deform a primitive

Retrieve and deform solution expects a high quality templates and they were also highly specialized in certain category type, which makes the overall input and output then highly specialized. Other method often used for aproximating to target object from a single image would be deforming a primitive. Popular representation in Pixel2Mesh [10], Pixel2Mesh++ [29], Neural mesh flow [30] was a simple sphere.

Pixel2Mesh had a 2D CNN extracting features from a single image which is then leveraged by a deformation block, progressively deforming a sphere

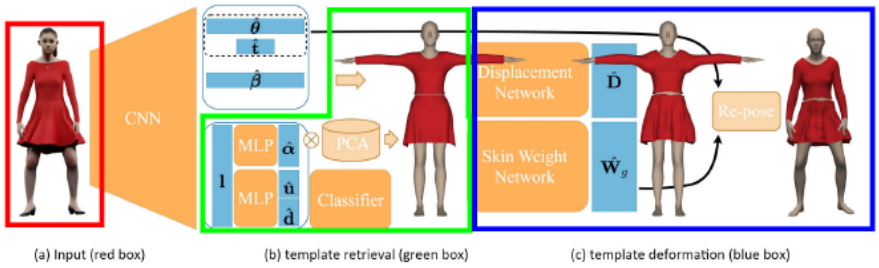


Figure 3.1: Overview of retrieve and deform shape reconstruction using BCNet [8]

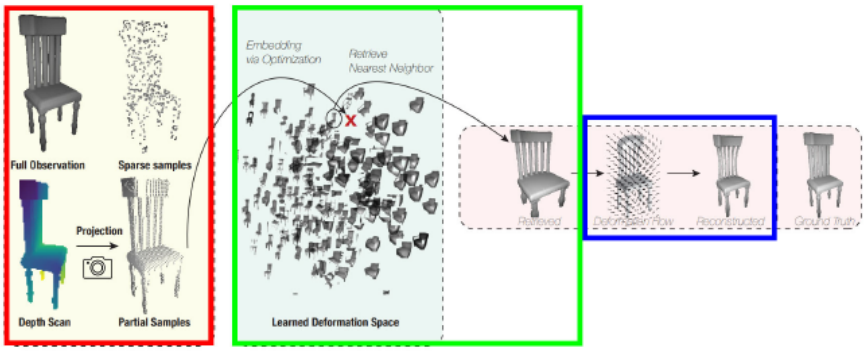


Figure 3.2: Overview of retrieve and deform shape reconstruction using ShapeFlow [9]

into the desired 3D model. The cascaded mesh deformation network is a graph-based convolution network, containing three deformation blocks intersected by two graph unpooling layers, which increases the number of vertices. With given face the points were added to the middle of each connecting edges. Connecting these newly added points then forms a new set of primitive faces, while also ensuring even distribution of vertices and their degrees. The improved Pixel2Mesh is extracts features from multiple view images, increasing the reconstruction accurancy.

The Neural mesh flow shares very similar architecture as Pixel2Mesh in using three deform blocks.

Neural Mesh Flow (NFM) as seen in Figure 3.5, learns to auto-encode 3D shapes. NMF broadly consists of four components. First, the target shape is encoded by uniformly sampling N points from its surface and feeding them to a PointNet [31] encoder to get the global shape embedding. Second, NODE blocks diffeomorphically flow the vertices of template sphere towards target shape conditioned on shape embedding. Third, the instance normalization layer performs non-uniform scaling of NODE output to ease cross-category

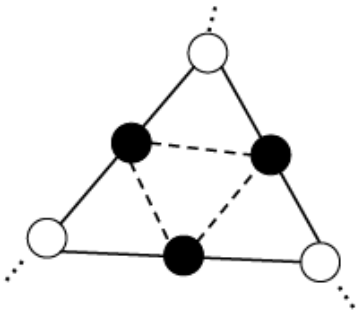


Figure 3.3: Graph unpooling[10]

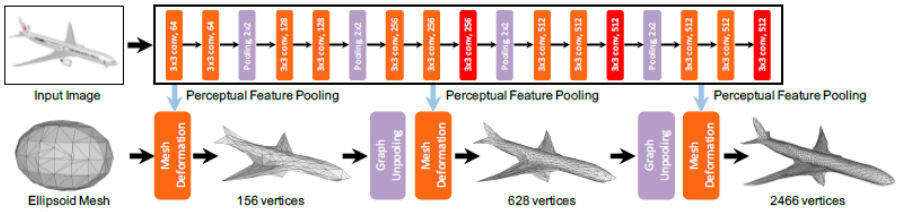


Figure 3.4: Pixel2Mesh architecture overview [10]

training. Finally, refinement flows provide gradual improvement in quality.

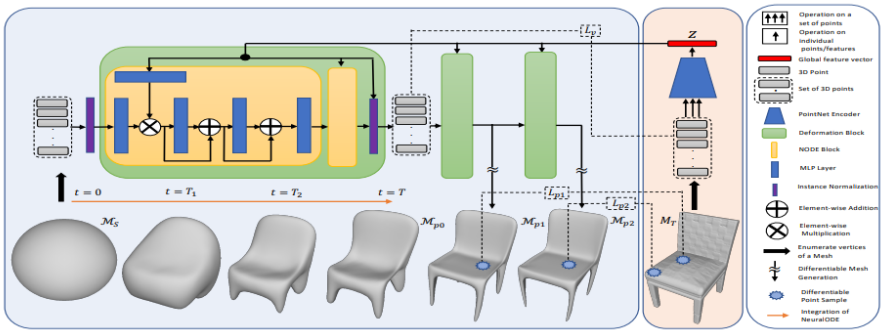


Figure 3.5: Neural Mesh flow architecture overview [30]

The mesh reconstruction is often done from voxels, point clouds, single images, and multi-view images, respectively. There was previously no real purpose of doing a reconstruction from a mesh as in our case.

universal solution, depends on previous topology quality

Chapter 4

Implementation

Chapter 5

Experiments

Chapter 6

Conclusion

Bibliography

[1] Kozák, M. Static malware detection using recurrent neural networks. 2020, [cit. 2020-12-28]. Available from: https://dspace.cvut.cz/bitstream/handle/10467/88342/F8-BP-2020-Kozak-Matous-thesis.pdf?sequence=-1&isAllowed=y

[2] Science, T. D. A Comprehensive Guide to Convolutional Neural Networks-the ELI5 way. Dec 2018, [cit. 2020-12-25]. Available from: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[3] Leskovec, J. Stanford CS224W: Machine Learning with Graphs [online]. Available from: http://web.stanford.edu/class/cs224w/,note={[cit.2022-9-20]}

[4] Holzner, A. LSTM cells in PyTorch [online]. Oct 2017, [cit. 2020-12-28]. Available from: https://medium.com/@andre.holzner/lstm-cells-in-pytorch-fab924a78b1c

[5] Conway, B. What Is Mesh Decimation And Why Is It Vital For AR? [online]. [cit. 2023-10-1]. Available from: https://www.vntana.com/blog/what-is-mesh-decimation-and-why-is-it-vital-for-ar/

[6] Martin, J. Topology Guides [online]. [cit. 2023-10-15]. Available from: https://topologyguides.com/

[7] Reinhardt, Z. Retopology in Blender [online]. [cit. 2023-10-1]. Available from: https://www.youtube.com/watch?v=X2GNyEUvpD4

[8] Jiang, B.; Zhang, J.; et al. BCNet: Learning Body and Cloth Shape from A Single Image. *CoRR*, volume abs/2004.00214, 2020, 2004.00214. Available from: https://arxiv.org/abs/2004.00214

[9] Jiang, C.; Huang, J.; et al. ShapeFlow: Learnable Deformation Flows Among 3D Shapes. In *Advances in Neural Information Processing Systems*, volume 33, edited by H. Larochelle; M. Ranzato; R. Hadsell; M. Balcan; H. Lin, Curran Associates, Inc., 2020, pp. 9745–9757. Available from: https://proceedings.neurips.cc/paper_files/paper/2020/file/6f1d0705c91c2145201df18a1a0c7345-Paper.pdf

[10] Wang, N.; Zhang, Y.; et al. Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images. In *ECCV*, 2018.

[11] McCulloch, W. S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, volume 5, no. 4, 1943: pp. 115–133, ISSN 0007-4985.

[12] Chen, Y.-Y.; Lin, Y.-H.; et al. Design and Implementation of Cloud Analytics-Assisted Smart Power Meters Considering Advanced Artificial Intelligence as Edge Analytics in Demand-Side Management for Smart Homes. *Sensors*, 05 2019, doi:10.3390/s19092047.

[13] Bengio, Y.; Goodfellow, I.; et al. *Deep learning*, volume 1. Citeseer, 2017, ISBN 0262035618, 166–485 pp.

[14] Krishtopa. What Are Neural Networks, Why They Are So Popular And What Problems Can Solve. 2016. Available from: https://steemit.com/academia/@krishtopa/what-are-neural-networks-why-they-are-so-popular-and-what-problems-can-solve

[15] Rosenblatt, F. The Perceptron: A Probabilistic ModelFor Information Storage And Organization In The Brain. *Psychological Re-view*, 1958: p. 2047, doi:0.1037/h0042519.

[16] Nielsen, M. A. *Neural Networks and Deep Learning*. Determination Press, 2015.

[17] Rojas, R. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013, ISBN 9783642610684, 37–99 pp.

[18] Leskovec, J.; Rajaraman, A.; et al. *Mining of massive data sets*. Cambridge university press, 2020, ISBN 9781108476348, 523–569 pp.

[19] Maladkar, A. I. M., Kishan. 6 Types of Artificial Neural Networks Currently Being Used in ML. [cit. 2020-12-25]. Available from: https://analyticsindiamag.com/6-types-of-artificial-neural-networks-currently-being-used-in-todays-technology/

[20] Lipton, Z. C.; Berkowitz, J.; et al. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015:

pp. 5–25, ISSN 2331-8422. Available from: https://arxiv.org/pdf/1506.00019.pdf

[21] Wiki, B. M. . S. Feedforward Neural Networks. [cit. 2020-12-25]. Available from: https://brilliant.org/wiki/feedforward-neural-networks/

[22] Towards AI, M. Main Types of Neural Networks and its Applications-Tutorial. Aug 2020, [cit. 2020-12-25]. Available from: https://medium.com/towards-artificial-intelligence/main-types-of-neural-networks-and-its-applications-tutorial-734480d7ec8e

[23] Goodfellow, I.; Bengio, Y.; et al. *Deep Learning*. MIT Press, 2016, url-http://www.deeplearningbook.org.

[24] Wiki, B. M. . S. Backpropagation. [cit. 2020-12-25]. Available from: https://brilliant.org/wiki/backpropagation/

[25] How Do Convolutional Layers Work in Deep Learning Neural Networks? April 2020, [cit. 2020-12-25]. Available from: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

[26] MathWorks. Convolutional Neural Network. [cit. 2020-12-25]. Available from: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

[27] Scarselli, F.; Gori, M.; et al. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, volume 20, no. 1, 2009: pp. 61–80, doi:10.1109/TNN.2008.2005605.

[28] Bhatnagar, B. L.; Tiwari, G.; et al. Multi-Garment Net: Learning to Dress 3D People from Images. In *IEEE International Conference on Computer Vision (ICCV)*, IEEE, oct 2019.

[29] Wen, C.; Zhang, Y.; et al. Pixel2Mesh++: Multi-View 3D Mesh Generation via Deformation. In *ICCV*, 2019.

[30] Gupta, K.; Chandraker, M. Neural Mesh Flow: 3D Manifold Mesh Generationvia Diffeomorphic Flows. *CoRR*, volume abs/2007.10973, 2020, 2007.10973. Available from: https://arxiv.org/abs/2007.10973

[31] Charles, R. Q.; Su, H.; et al. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 77–85, doi: 10.1109/CVPR.2017.16.

Appendix A

Acronyms

ANN Artificial Neural Network

RNN Recurrent Neural Network

CNN Convolutional Neural Network

LSTM Long Short-Term Memory

ReLU Rectified Linear Unit

Appendix B

Contents of enclosed CD

- `README.md` ..... the Markdown file with description
- `executables` ..... the directory with executables
  - `Dataset` ..... the directory of the original dataset
  - `TrainedModel` ..... the directory of trained model
  - `DataSampler.exe` ..... data sampling application
  - `model_trainig.py` ..... model training Python script
  - `GestureApp.exe` ..... gesture recognition demo application
- `src` ..... the directory of source codes
- `text` ..... the directory of LaTeX source codes of the thesis
- `environment.yml` ..... configuration file for conda environment
- `BP_Viet_Anh_Tran_2021.pdf` ..... the thesis text in PDF format