– Independent Work Report Spring, 2023 –

Using a Neural Network to Determine the Maximum Simplification of a 3D Mesh

Lauren Gardner

Advisor: Dr. Xiaoyan Li

Abstract

The recent growth of the computer graphics industry, specifically in the realm of gaming and other applications of 3D model use and rendering, has had a huge impact on the detail and number of 3D meshes available for use. Video games using meshes with high levels of detail have certain GPU constraints that affect their render time and processing speeds. This introduces the question of portability – what is the render time of these meshes, and how can this be reduced across all devices and platforms? This is the core question that inspired this project, the goal of which is to aid in the portability of complex meshes and advanced computer graphics by evaluating differing levels of mesh simplifications using machine learning. This project goes in depth about the creation of a dataset used for 3D object classification, as well as, the implementation of a machine learning model to decipher the 'recognizability' of a mesh: basically, does the simplification render the object unrecognizable?

In this application, a convolutional neural network (CNN) is used in the place of a human to make these decisions about the 'recognizability' of a mesh. Basically, the CNN is used as a human proxy to answer the question of if a mesh is recognizable in its simplified state. With the goal of a non-biased machine learning model, attempts are made to equalize the ModelNet10 [1] dataset to make it free of bias and to achieve the most accurate classification for analysis. This includes a thorough analysis of the dataset and the synthesis of a new dataset that amends these issues.

1. Introduction / Motivation

In recent years, the computing power and display capabilities of technology have improved along with the capabilities of computer graphics. This improvement has benefited a multitude of industries such as computer graphics, animation, and especially the video gaming industry with increasingly detailed games being released each year. But since graphics are becoming more complex, 3D meshes and textures are also becoming increasingly detailed and complex: more vertex information is required and the number of faces on 3D models have increased. Rendering these detailed games heavily relies on the GPU power of a device. This is especially a problem for devices that lack sufficient GPU power to render these graphics. In many cases, games will stall or slow, inhibiting the experience of the player; therefore, it is important to balance both visual appeal and performance when dealing with complex meshes. This balance has been made possible through the introduction of dynamic rendering — specifically progressive meshes.

Progressive meshes contain both a *base mesh* and sequence of *vertex splits*. In this paper, I use machine learning to find the most suitable *base mesh*, which is the most simplified version of a mesh, for progressive rendering. I describe my process of translating the ModelNet10 dataset [1] into a 2-dimensional version I call 'ModelNet2D'. This new dataset contains training information for 4,000 meshes of varying object types and four testing sets of increasing simplification for the remaining 1,000 meshes. I then implement a CNN to identify the increasingly simplified meshes and analyze the results for differences between the test cases. This paper will go step by step into the process, first discussing related work in this field and how that work relates to the current project, and finishing with the creation of ModelNet2D, implementation of a 3D object classifier, and an analysis of the results from this project.

2. Related Work

In terms of my project, there are two sections of related work: that which is related to dynamic rendering and that which is related to 3D object classification itself. There could also possibly be a third section on 3D model simplification, but since I used a precoded library for this I will be omitting that section. Starting with the first section, dynamic rendering, specifically with *progressive meshes*.

2.1 Progressive mesh

First introduced by Hugues Hoppe in 1996, progressive streaming is an efficient method to improve the render speed of 3D meshes on any platform. Object meshes are loaded piece-wise, slowly increasing in detail as the information is received. These meshes are called *progressive meshes* and they are able to represent a complex 3D object with just a *base mesh* and *vertex splits*. The *base mesh* – which is the first low-resolution mesh representation of an object – contains the minimum amount of vertices needed to represent that object; basically, the most simplified mesh possible. This mesh is sent to the client first, and if the client's connection remains steady and is able to render the image, more data is sent in the form of *vertex splits*; these instructions progressively add detail to the mesh by adding more vertices and thus, increasing the *level of detail* (LOD) of the object. These vertex splits are continuously applied to the base mesh until it reaches its full complexity, at which point there are no more vertex splits to account for. Because of these progressive meshes, in the case of low bandwidth, an object can be represented by its base mesh or whichever stage the mesh was in during the progression from base mesh to final mesh. Using this method, highly detailed 3D meshes can be used and rendered

on a variety of devices including mobile devices and other devices with insufficient GPU power [3].

2.2 Point Cloud

Moving from the topic of dynamic rendering and shifting towards object classification, 3D object classification is not only important in the application of video games and computer graphics, but also in computer vision, specifically in cases where 3D object classification is necessary for decision making — consider self-driving cars an example. They must be able to analyze objects from its environment quickly and accurately. In these cases, 3D object classification must be precise since the chance of human injury is increased. Using technology such as light detection and ranging (LiDAR) sensors gives point cloud information that depicts objects in the highest precision possible. Current point cloud classifiers use this information and deep learning applications to classify models. For instance, researchers Song et al. used a CNN in order to implement a point cloud based 3D object classifier. Their approach begins by filtering out 'noise', or anomalies, from the input point cloud before the training or classification steps. They then identify clusters in a scene, excluding the ground, and classify these clusters as different objects. The training model is trained on large amounts of data to reduce the error amount, and in the case of a four classification dataset, the model performed extremely well at 96.7% - 99% accuracy depending on the object [5]. Although this method results in good classification accuracy, this project requires a method computationally simpler as well as more visually based. Thus, there is another variety of 3D object classification that should be considered.

2.3 Image Classification

It has already been established that as the quality and use of computer graphics has increased, the demand for 3D models has also increased. With the increase of open source 3D models online, the question of categorization and searchability need to be considered: how would a search engine for 3D models work? One possible solution is to implement a 3D object classifier to group similar meshes and associate them with certain keywords related to their object. Chen et al. was interested in implementing this method which requires a classification model that can identify a mesh from its visual appearance and match it to a specific set of keywords. Wanting their searching mechanism to be primarily based on visual rather than geometric similarities, they implemented a machine learning model that identified models based on images. The logic is as such, if two models are similar, they should look similar at most viewed angles – so classification is based off of images taken from multiple angles. In this research paper, Chen et al. chose a variety of viewing angles from a spherical position around the center object to base classification on. These images were then classified using 2D image classification and were considered all together for the final object classification [2].

3. Methods

One pitfall of progressive meshes is that an appropriate base mesh needs to be identified in order to implement them. My proposed machine learning model will be used to determine that base mesh by classifying increasingly simplified versions of 3D objects. This classification will help identify if a simplified mesh is ‘recognizable’, at least in the sense of a computer’s capability to recognize.

In terms of the classifier itself, I considered each of the options described above. In the beginning, I considered using the point cloud approach to object classification since extracting vertex information from my dataset seemed the most straightforward approach [5]. Eventually, I switched to the 2D image classification method for a couple of reasons (1) classifying based on thousands of vertices per mesh would be too computationally heavy for the size of the dataset and the abilities of my computer, and (2) the motivation behind my machine learning model is that it would be able to identify a simplified mesh based on how it looks, therefore, the 2D image classification suits this motive the best since it is a visually based classification model [2].

When implementing the image classifier, I created a CNN with three hidden ReLU activation layers and an Adam optimizer. Twenty percent of the training set was set aside as a validation set used during training. Model accuracy was measured across all classifications and was reported after the training processes along with the loss, including the accuracy and loss for the validation set.

4. Implementation

4.1 ModelNet10

I obtained the ModelNet10 dataset [1] online from a website called kaggle. This dataset contains information for ~5,000 3D meshes. Columns containing 'object_id', 'class', 'split', and 'object_path' were provided and gave each object a name, classification, test / train split, and a path to the .off file respectively. The .off file type is used to define a 3D object, each vertex is defined in (x, y , z) coordinates and the faces are defined next [4]. Using the python libraries open3D and matplotlib a 3D object can be displayed from an .off file. Since my chosen classifier

requires 2D data, I had to process the ModelNet10 dataset [1] into a 2D format. Thus, I created the ModelNet2D dataset by snapping images of each model at five different angles, which resulted in a total of 25,000 images. ModelNet10 [1] contains a total of 10 classification types, all of which are household furniture: bathtubs, beds, chairs, desks, dressers, monitors, night stands, sofas, tables, and toilets. Although the dataset contains a total of 5,000 meshes, each object does not have equal representation as seen in the chart below:

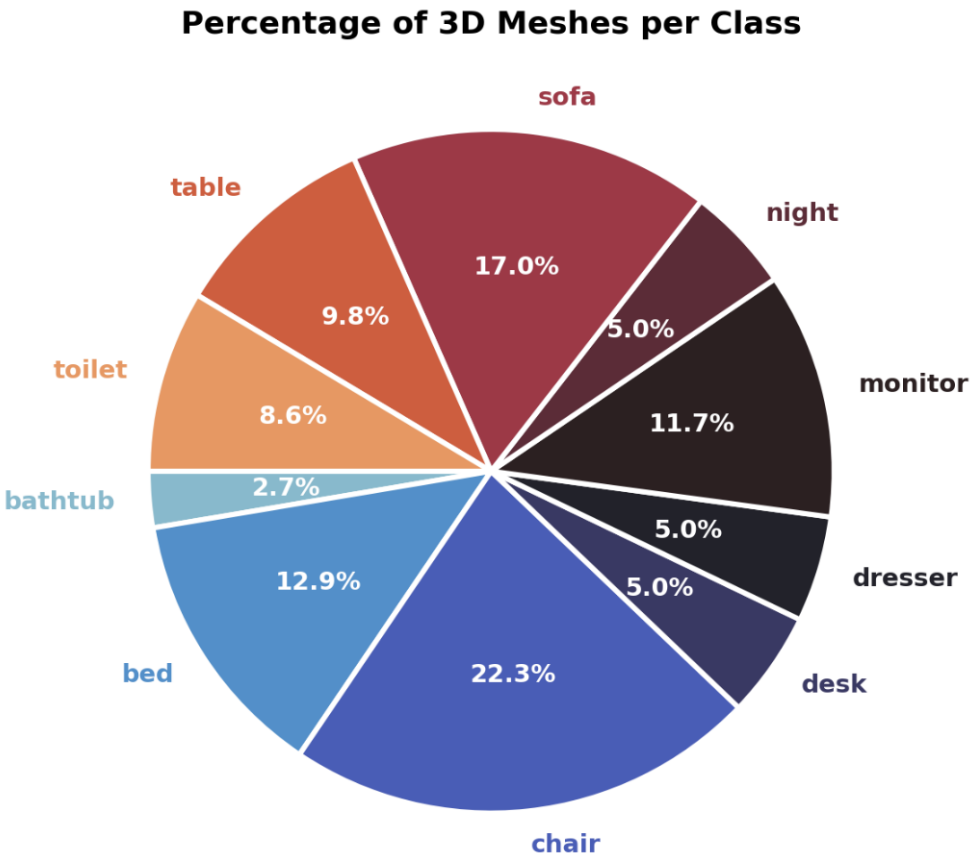

Figure 1: ModelNet10 classification type distribution

As seen in fig. 1, the difference in representation between objects is rather drastic. Chairs make up the largest percentage of that dataset at 22.3%, whereas bathtubs make up the smallest

percentage at just 3%. This means that during the creation of ModelNet2D, I needed to take this difference into account when training my model.

Along with the uneven class distribution, ModelNet10 [1] also has an uneven test / train split within each class. After importing ModelNet10 [1] and processing it using the pandas library has a 20/80 test train across all object classifications based on the split column they provide, but when the split is analyzed per class individually, the distribution is not alway 20/80.

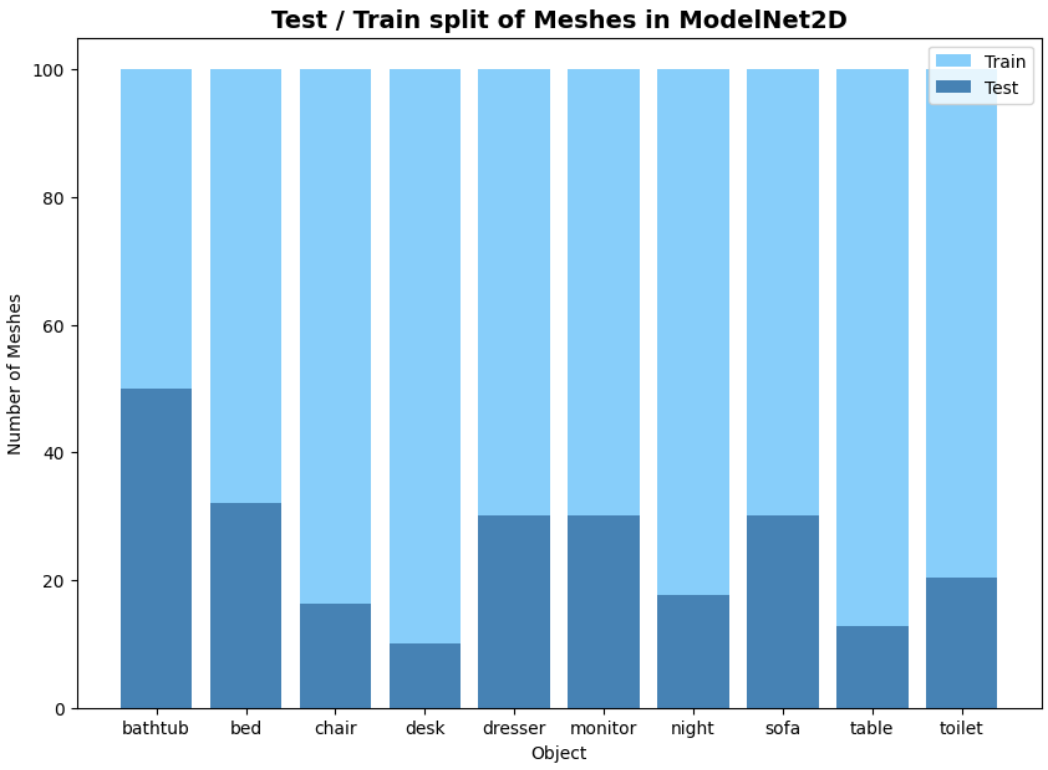

Figure 2: ModelNet10 percentage of meshes labeled test / train per each object

Some objects including bathtubs, beds, dressers, monitors, and sofas had rather small training sets in comparison to the other objects like desks, tables, and chairs which have large training sets. At first, I assumed this uneven test / train delegation was to account for the uneven amount

of meshes per classification, but this isn't the case. Bathtubs – the most underrepresented mesh in the dataset – also have the lowest percentage of meshes in their training set whereas chairs – which make up 20% of the overall dataset – have a fairly large training set distribution. Therefore, it does not make sense that this uneven distribution between the test / training sets is to account for the uneven amount of meshes.

My first amendment to the dataset was redistributing the test / train splits to properly divide each class into a 20/80 split. I wrote code to erase the split column from ModelNet10 [1] and replace it with a new redistributed split column for ModelNet2D that split every class into a 20/80 test / train split. The meshes were then sorted into their new testing and training sets according to this column.

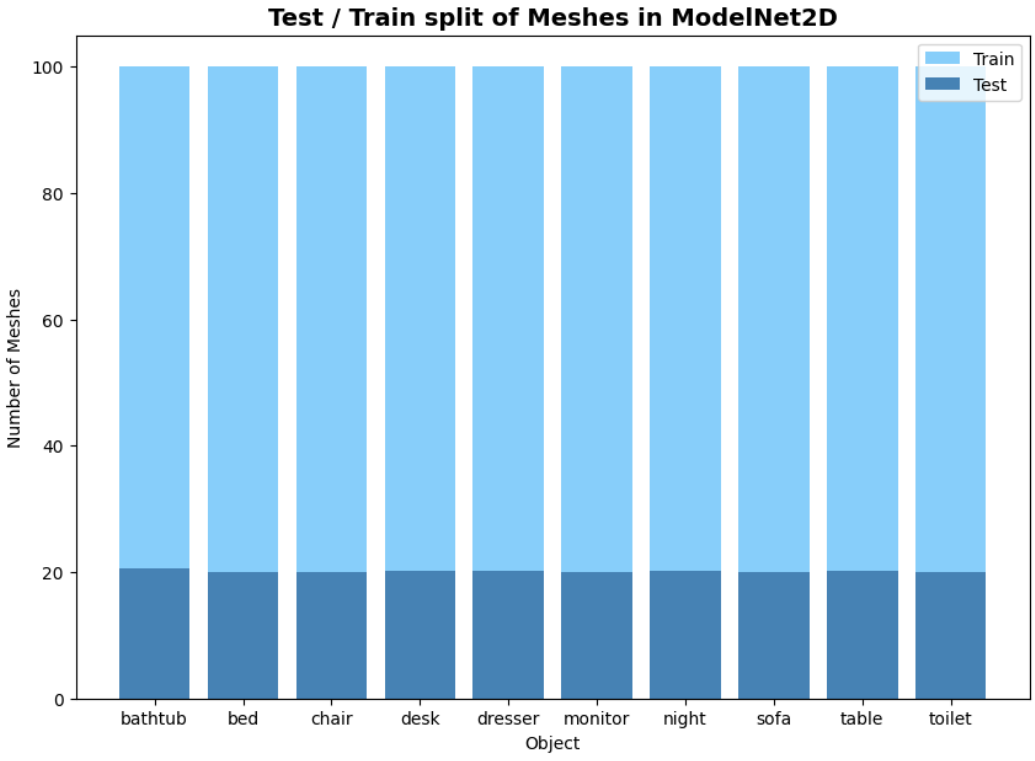

Figure 3: ModelNet2D percentage of meshes labeled test / train per each object

4.2 ModelNet2D

I created ModelNet2D to be a 2D representation of ModelNet10 [1]; as seen already, some features of this dataset are also improvements made to the ModelNet10 [1] structure. To start with, I created a program that iterates through every object in ModelNet10 [1], reads and displays the .off file as a 3D object using the python libraries open3d and matplotlib, and records the image at five different camera angles into the respective directory based on object type and viewpoint. Since image classification was the route chosen for object classification, a set number of images and viewpoints needed to be defined. The five viewpoints chosen were all taken with a frontal view of the object. If it helps, one can imagine the object in the center of a cube, each camera angle was set on a single face of that cube. The 3D object was displayed at coordinates [0, 0, 0] and the five camera angles were as follows: **view 0:** [0, -2, 0], **view 1**: [2, -2, -2], **view 2**: [2, -2, 2], **view 3**: [-2, -2, 2], and **view 4**: [-2, -2, -2] (these equate to the corners and center of the front face of the cube previously defined). At each position, the camera angle was changed to point towards the object in the center.

Figure 4: The five viewpoints in order left to right views 0-4

Although the above camera positions were used, the resulting photos contained a fair amount of white space and required follow-up code to zoom in each image. Also, the original images recorded were too high quality and required a fair amount of processing time when input to the

CNN, so I also wrote code to lower the quality of the image to improve the speed of the model training processes. After all necessary edits, each image was sorted into its respective folder corresponding to its object type and view angle. Ex. ‘dresser_2’ refers to dresser objects from view 2 defined above.

```
lauren@dynamic-oit-vapornet100-10-9-70-19 Train_Small % ls
bathtub_0       chair_0         dresser_0       night_0         table_0
bathtub_1       chair_1         dresser_1       night_1         table_1
bathtub_2       chair_2         dresser_2       night_2         table_2
bathtub_3       chair_3         dresser_3       night_3         table_3
bathtub_4       chair_4         dresser_4       night_4         table_4
bed_0           desk_0          monitor_0       sofa_0          toilet_0
bed_1           desk_1          monitor_1       sofa_1          toilet_1
bed_2           desk_2          monitor_2       sofa_2          toilet_2
bed_3           desk_3          monitor_3       sofa_3          toilet_3
bed_4           desk_4          monitor_4       sofa_4          toilet_4
lauren@dynamic-oit-vapornet100-10-9-70-19 Train_Small %
```

Figure 5: Example file structure for ModelNet2D training set

4.3 Testing Sets of Simplified Meshes

After the training set for ModelNet2D was completed, the next challenge was to create the testing sets. Each test set would contain the same meshes as the previous set, but a certain percentage of the faces are removed to simplify the mesh. This gives the most realistic results when classifying since each set contains the same meshes as the other test cases, just at different simplifications rather than a new variety of meshes that might alter classification accuracy in unrelated ways. These testing sets are designed to be judged by the same overall trained model which is trained on a unique set of these objects. That is, the testing and training sets contain a different sample of meshes, but only the level of detail of the testing set is reduced by a predefined percentage of faces. Creating these reduced detail testing sets was relatively easy considering open3d had a built in simplification method for meshes. Using this function, I ran

separate code that processed the testing set meshes four times: once for 0% simplification and again for 30%, 50%, and 70%. In the end four testing sets were created named ‘Test0’, ‘Test30’, ‘Test50’, and ‘Test70’ respectively.

4.4 Machine Learning Model

I chose to use a convolutional neural network because it is commonly used in image classification. I adapted a CNN from the source code provided by the TensorFlow authors [6]. Although my project is an image processing model at the core, I needed it to identify objects considering all dimensions in order to classify 3D objects. So, I altered the code to account for 50 different classifications, one for each of the ten objects and their five viewpoints. When recording my presentation of this project, this step had not yet been implemented. Thus, I had five separate trained models for each viewing angle with only ten classifications each. During the first round of testing, these models performed extremely well (accuracy of ~90-92%), which compared to other 3D model classifiers available online, this percentage was well above average and was more akin to an image classifier’s accuracy score. At this stage, I had basically implemented 2D image classification for each viewing angle, but not a 3D classifier that considers all of the angles at once. My solution was to combine all of the viewpoints into one trained model and use that for classification – the complication was that this large volume of meshes would not successfully process on colab and often timed out during training. The solution to this issue comes after another discovery I made in the process.

After running the preliminary tests on my data, not only did I discover the overall accuracy was too high for a standard 3D object classifier, I also discovered that each object had a differing

identification accuracy, some of which were rather drastic with a ~15% accuracy difference. After comparing these accuracy scores to the data in fig. 1 that depicts the percentage of each classification in ModelNet10 [1], I concluded that the reason was most likely due to the uneven distribution of the number objects in the dataset. After quite a bit of thought, I decided to create 'Small' variations of each dataset in ModelNet2D. These 'Small' datasets included all of the information present in the previous ones, but equalized the data to the level of the least represented mesh. In this case, the bathtub mesh was the most underrepresented with just 3% of meshes being bathtubs, so all other meshes were equalized to have the same number of sample meshes as the Bathtubs had. In the end, this data set contained ~1,550 meshes and ~7,750 images. The introduction of the 'Small' dataset not only solved the problem of equal representation between objects, but also the size of the dataset was reduced to a level that colab could handle. I trained a new model based on this dataset and saw more realistic classification results.

5. Evaluation / Results

The first step in training the model was to normalize the RGB values of the data between [0, 1] rather than [0, 255]. After normalization, the model was trained using 80% of the dataset with 20% of that acting as a validation set. The CNN has three hidden layers and achieved the following results in the last epoch. loss: ~0.4, accuracy: ~0.86, val_loss: ~1.2, val_accuracy: 0.67. These results were suspicious and hinted at a case of overfitting since the training accuracy skyrocketed above the validation accuracy in just one epoch, and the difference between the accuracies was fairly large after just five epochs. A standard solution to this problem is to implement data augmentation which produces new data by flipping, stretching, and rotating

existing images in the dataset. In the case of ModelNet2D, where classification is dependent on the viewpoints of objects, data augmentation caused the model to perform worse. As a solution, I implemented data augmentation to a lesser degree: removing augmentations that rotated or flipped images since these could be mistaken for another viewpoint during classification.

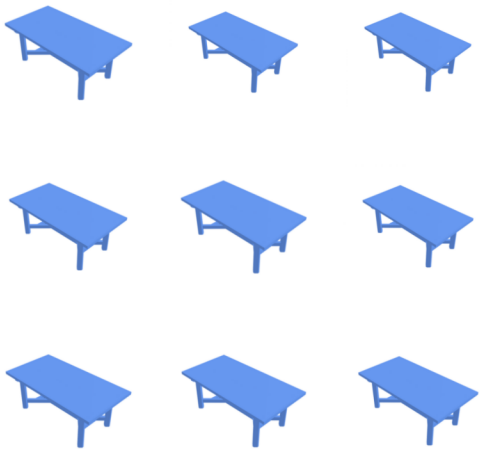

Figure 6: Example images created using a variation of data augmentation

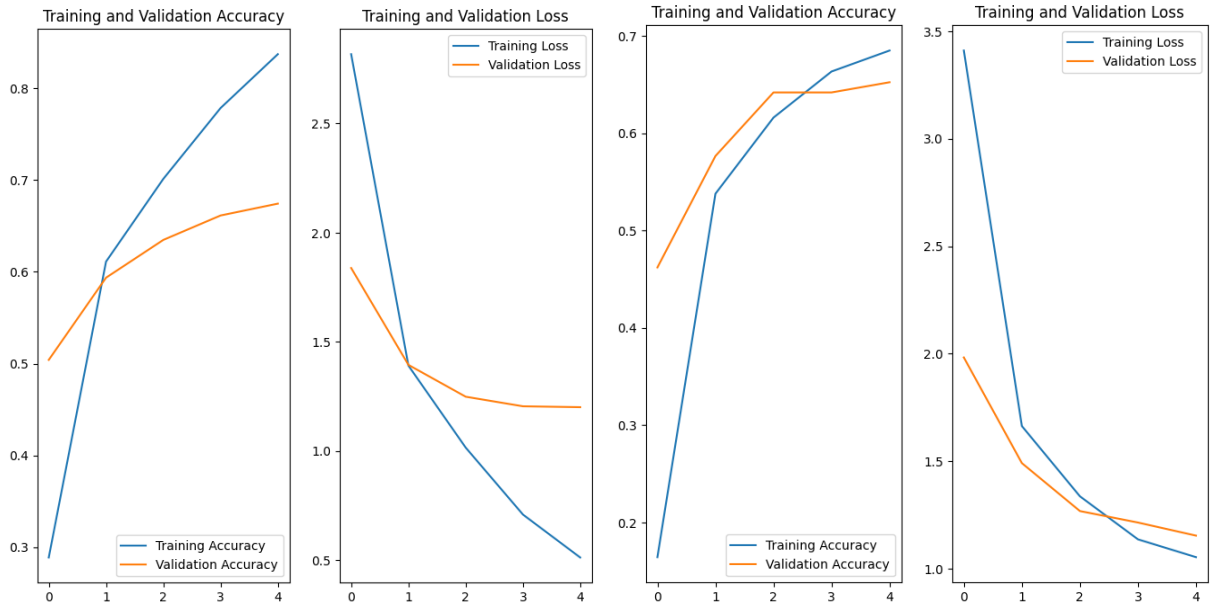

Figure 7: [Left] Classification model accuracy and loss functions without data augmentation and five epochs. [Right] With data augmentation and five epochs

This data was gathered over a total of five epochs, regardless, the difference between the two sides is substantial. The final model was trained over a total of ten epochs.

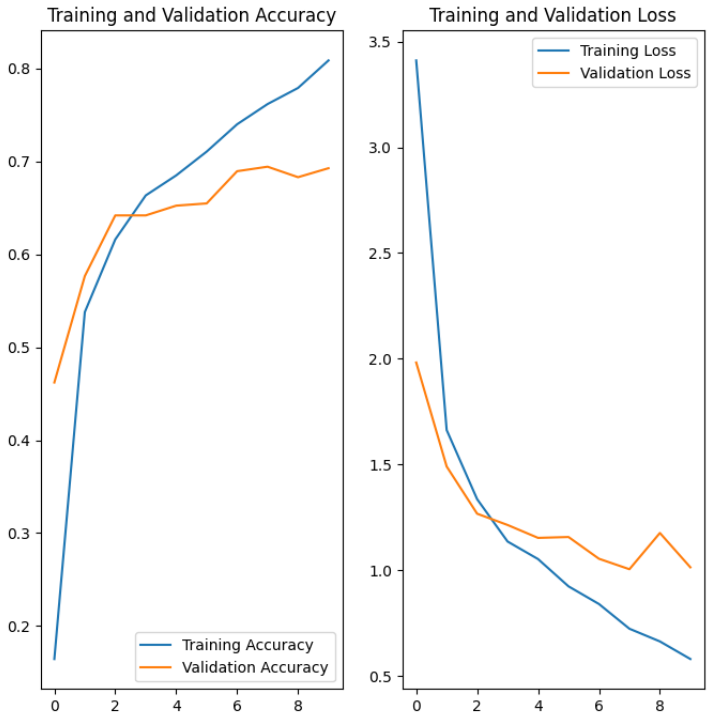

Figure 8: Classification model accuracy and loss functions with data augmentation over a total of ten epochs

After implementing my version of data augmentation, the problem of overfitting was greatly reduced. Looking at fig. 7, the difference between the training set's accuracy and loss and the validation set's accuracy and loss was quite large before data augmentation. After, the difference between the accuracy and loss was greatly reduced. In fig. 8, it is clear that after the total number of epochs was reached, the model still remained fairly even in its training / validation accuracy and loss. Therefore, my variation of data augmentation which only added variety by scaling and stretching, was still beneficial to the final classification model.

Following the training process, each test set was evaluated using the final trained model. For each testing set, the accuracy was calculated based on how many meshes were correctly identified based on their viewpoints. This was repeated for ‘Test_0’, ‘Test_30’, ‘Test_50’, and ‘Test_70’ respectively, and the following charts show a variety of results based on the accuracy report.

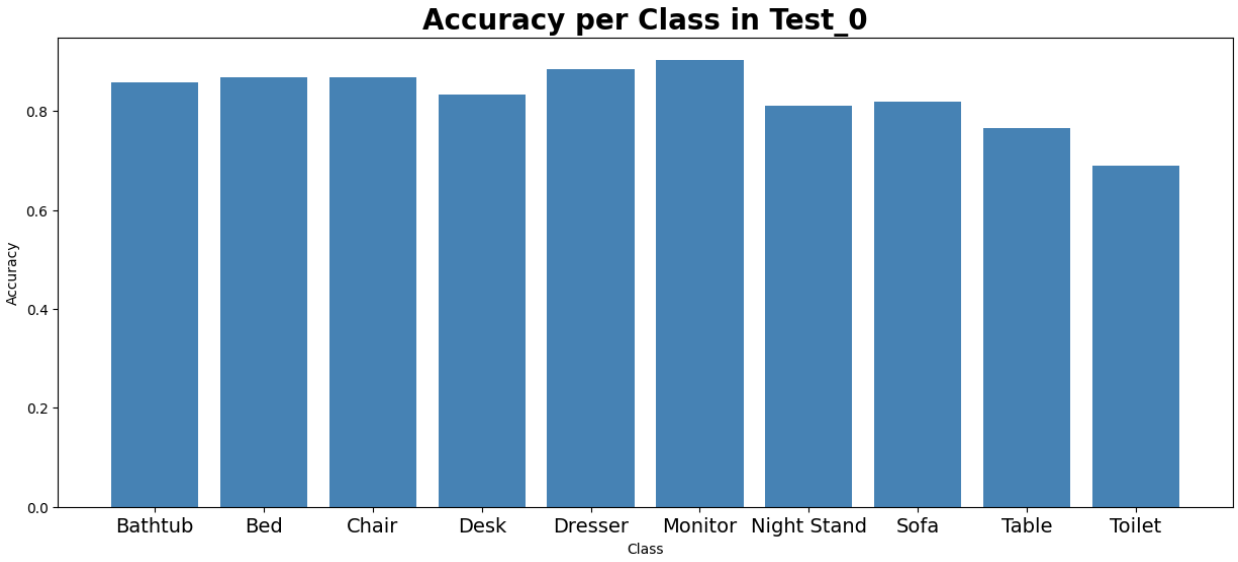

Figure 9: [Based on Test_0] Accuracy based on object type

As seen in fig. 9, the accuracy among the ten object types remained fairly similar. Notably, the object with the lowest accuracy percentage was the toilet, a highly detailed object with arguably the most curves out of all objects in this list. In comparison, we see that object types such as the dresser, chair, monitor, and bed have the highest accuracies. This could be because the meshes represent fairly angular objects with few curves. This difference in the meshes might be the cause for the drop in accuracy.

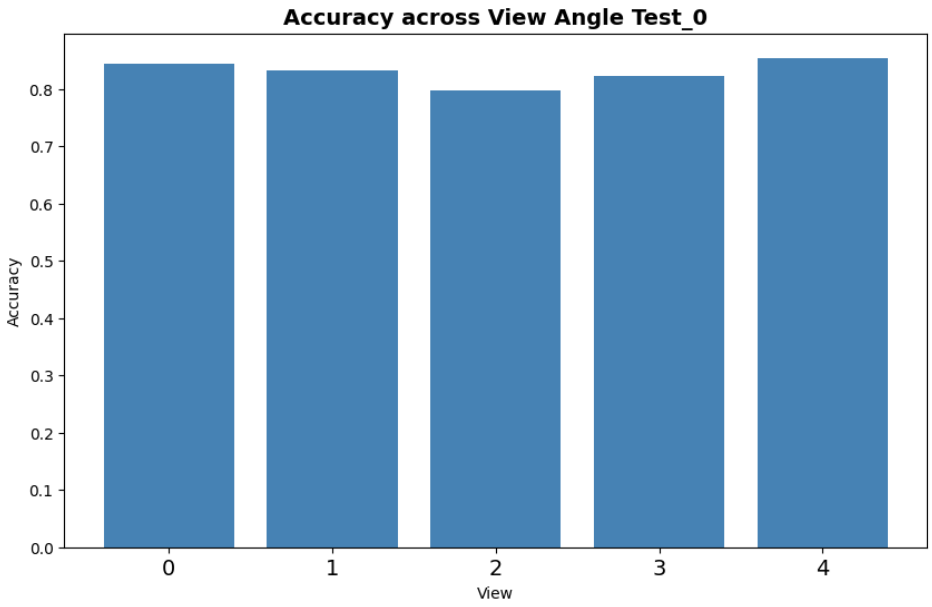

Figure 10: [Based on Test_0] Accuracy based on viewing angle [ref. fig. 4 for visual representation of views]

Looking at figure 10, there doesn't seem to be a drastic change in the accuracy between viewing angles since differences do not exceed ~4%. This was also true for the accuracy results for each of the testing sets.

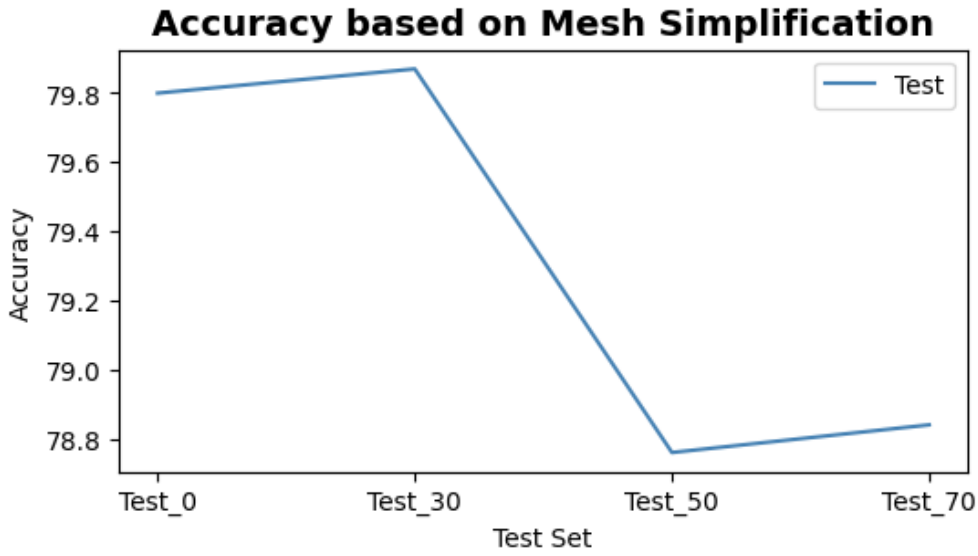

Figure 11: Accuracy results for Test_0, Test_30, Test_50, and Test_70

Note that the difference in percentage between ‘Test_0’ and ‘Test_30’ is 30% whereas the difference between the other sets is 20%. Regardless, differences in accuracy don’t exceed ~1% which is insignificant. There are a couple of hypotheses of why this was the case, one in particular stands out to me. Since ModelNet10 [1] is composed of fairly angular and simple meshes, a simplification of these meshes – even if by 70% – often yields results that are similar to the original mesh. If this test were to be run on a different set of meshes, perhaps ones with more organic shapes or curvatures, the results would no doubt be different.

6. Conclusion

3D Models are becoming increasingly complex, and with this rise in complexity adds another layer of computability as was discussed earlier. Innovative designs and objects, such as progressive meshes, have made it possible for devices with insufficient computing power to still display highly detailed meshes. This project sought to identify the minimum level of detail required for the base of a progressive mesh, and although a solid percentage was not identified, the results found while processing ModelNet10 [1] were important in other ways. For instance, the discovery that that test / train split was not the same for all of the objects is important for future users of this dataset to know. Therefore, this project was successful in that it gave an insight into some of the potential flaws of ModelNet10 [1] while also showing that a dataset with more complex meshes might be useful in future iterations of this project.

7. Future Work

There are quite a few possible extensions and improvements to this project. Of course, in future iterations of this project, a dataset with more complex meshes should be used since this is key to

the reduction of level of detail and its correlation to an object's recognizability. In addition, access to a more powerful CPU and processing capabilities is required if the project is to be scaled up at all. That is, in future work, I would like to use a larger dataset, but will also need to request access to cloud computing in the first stages of the project. Another area of future work, which is rather ambitious, is to create a mesh synthesizer, which synthesizes similar meshes based on the input dataset and keyword associated with a specific object.

8. Acknowledgments

I would like to thank Dr. Xiaoyan Li who, in this seminar, helped me to improve my knowledge and confidence in machine learning. Without this seminar, I would be lacking in valuable first hand experience with ML. I would also like to thank the other students in COSIW03 who not only enthusiastically shared the progress of their projects weekly, but also gave me feedback at critical moments of this project.

9. Honor Code

This paper represents my own work in accordance with University regulations.

- Lauren Gardner

10. References

[1] Ashwath, Balraj. “ModelNet10 - Princeton 3D Object Dataset.” Kaggle, 28 Oct. 2020, https://www.kaggle.com/datasets/balraj98/modelnet10-princeton-3d-object-dataset.

[2] Chen, Ding-Yun, et al. “On Visual Similarity Based 3D Model Retrieval.” Computer Graphics Forum, vol. 22, no. 3, 2003, pp. 223–232., https://doi.org/10.1111/1467-8659.00669.

[3] Hoppe, Hugues. “Progressive Meshes.” Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, 1996, https://doi.org/10.1145/237170.237216. https://hhoppe.com/pm.pdf

[4] “Object File Format (.off).” Princeton University, The Trustees of Princeton University, https://shape.cs.princeton.edu/benchmark/documentation/off_format.html.

[5] Song, Wei, et al. “CNN-Based 3D Object Classification Using Hough Space of Lidar Point Clouds.” Human-Centric Computing and Information Sciences, vol. 10, no. 1, 2020, https://doi.org/10.1186/s13673-020-00228-8.

[6] TensorFlow. “Image Classification.” Google Colab, Google, 2018, https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/classification.ipynb.