H3DNET: A DEEP LEARNING FRAMEWORK FOR HIERARCHICAL 3D OBJECT CLASSIFICATION

A THESIS IN
Computer Science

Presented to the Faculty of the University
Of Missouri-Kansas City in partial fulfillment
Of the requirements for the degree

MASTER OF SCIENCE

By
MARMIKKUMAR PATEL

B.E., Gujarat Technological University – Ahmedabad, India, 2015

Kansas City, Missouri
2017

H3DNET: A DEEP LEARNING FRAMEWORK FOR HIERARCHICAL 3D OBJECT CLASSIFICATION

Marmikkumar Patel, Candidate for the Master of Science Degree

University of Missouri-Kansas City, 2017

ABSTRACT

Deep learning has received a lot of attention in the fields such as speech recognition and image classification because of the ability to learn multiple levels of features from raw data. However, 3D deep learning is relatively new but in high demand with their great research values. Current research and usage of deep learning for 3D data suffer from the limited ability to process large volumes of data as well as low performance, especially in increasing the number of classes in the image classification task. One of the open questions is whether an efficient as well as an accurate 3D Deep Learning model can be built with large-scale 3D data.

In this thesis, we aim to design a hierarchical framework for 3D Deep Learning, called H3DNET, which can build a DL 3D model in a distributed and scalable manner. In the H3DNET framework, a learning problem is composed of two stages: divide and conquer. At the divide learning stage, a learning problem is divided into several smaller problems. At the conquer learning stage, an optimized solution is used to solve these smaller subproblems for a better learning performance. This involves training of models and optimizing them with refined division for a better performance. The inferencing can achieve the efficiency and high accuracy with fuzzy classification using such a two-step approach in a hierarchical manner.

The H3DNET framework was implemented in TensorFlow which is capable of using GPU computations in parallel to build 3D neural network. We evaluated the H3DNET framework on a 3D object classification with MODELNET10 and MODELNET40 datasets to check the efficiency of the framework. The evaluation results verified that the H3DNET framework supports hierarchical

3D Deep Learning with 3D images in a scalable manner. The classification accuracy is higher than the state-of-the-art, VOXNET[7] and POINTNET.

APPROVAL PAGE

The faculty listed below, appointed by the Dean of the School of Computing and Engineering, have examined a thesis titled "H3DNET: A Deep Learning Framework for Hierarchical 3D Object Classification" presented by Marmikkumar Patel, candidate for the Master of Science degree, and hereby certify that in their opinion, it is worthy of acceptance.

Supervisory Committee

Yugyung Lee, Ph.D., Committee Chair
Department of Computer Science Electrical Engineering

Zhu Li, Ph.D.
Department of Computer Science Electrical Engineering

ZhiQang Chen, Ph.D.
Department of Civil and Mechanical Engineering

Contents

ILLUSTRATIONS

Figure Page

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor Dr. Yugyung Lee for all the innovative ideas, insights, advice and challenging deadlines that have helped me achieve this thesis. She has been a constant source of motivation and zeal, not only during my thesis but also during my entire Master program. She was always welcoming for all the help I needed throughout my work, it has always amazed me for the kind of support and inspiring suggestions she has given me for the development of this thesis.

Secondly, I would like to thank the University of Missouri-Kansas City, without which this research would not be possible. The school provided me with good opportunities to support myself and a Lab for my research on a GPU machine. I would also like to thank my lab mates and roommates for their generous support.

Finally, I would like to express my heartfelt gratitude to my family and friends for providing me with constant support and encouragement. This accomplishment would not be possible without them.

CHAPTER 1
INTRODUCTION

1.1 Motivation

Machine learning is the area of computer science which allows computers to act without being explicitly programmed or without giving any command. The study of pattern recognition and computational learning theory are the areas from where machine learning has been grown. Machine learning investigates the development of algorithms that can make data-driven predictions to overcome the conventional software's following strictly designed programs. Machine Learning model is developed from training data to answer certain questions. For example, we can build model using lot of images of "cat" and "dog" and then use that model to check if a given image is either dog or cat. The areas such as self-driving cars, language translation, computer vision, smart search, data security, speech recognition, autonomous robots extensively use the Machine Learning algorithms [10]. The "Deep Learning"[1] is the extended and most powerful form of machine learning which builds the neural network, an advanced and complex mathematical structure which work similar to human brain. To build the deep neural network which powers such new applications, it requires having a massive amount of computing resources which is really a huge challenge. The building of neural networks may take too much time ranging from days to months for even the supercomputers. Recent the advancements in GPU computations have speedier the neural network training 10 -20 times than the normal CPUs. The number of deep learning SDKs are developed which are based on NVIDIA GPUs[4] to make the neural network training faster. There are so many organizations which are developing and using such frameworks for building the neural network and this number is continuously increasing. With the increment in neural network applications, the number of models developed are also increasing and even there is a need to combine or use part of some models to serve the purpose.

With the advancement in machine learning and computation power, there is also advancement in the processing and usage of 3D objects. The fields like gaming, virtual reality, autonomous robotics extensively use the 3D data or 3D point cloud. 3D Data is growing enormously each day and because of this, there is a need for distributed and scalable deep learning network. This research presents one such approach, which is distributed and scalable.

1.2 Problem Statement

Various approached are developed using the deep learning framework to solve the supervised machine learning problems such as natural language processing, text classification, and image classification but the 3D object classification is the relatively underrated topic. 3D object classification is the crucial task for many applications such as autonomous robots operating in unstructured environments, autonomous vehicles. 3D gaming etc. There are also some basic neural network models are available for 3D object classification which either used 2D neural network or single neural network.

Few open research questions for distributed 3D object classification are: can we distribute learning tasks to multiple CPU/GPU machines while minimizing loss? Data parallelism, task parallelism, and model parallelism are a different approach for distributed learning and predictions. Data parallelism uses the different data with the same model in the entire cluster. Model parallelism splits the model into the entire cluster for same data. Task parallelism is using different machines of the cluster for different tasks.

The existing deep learning libraries such as Tensorflow, caffe etc support the distributed model training, i.e. distributed computing is available to them. But they do not allow to do distributed learning or prediction with conventional approaches. The inference using the complex neural network needs millions of operations so distributed model inference is must for better performance.

The model needs to learn continuously about the new data as the data grows. For an instance, the recommendation and user behavior systems require updated model based on the preferences of the users which changes time-to-time. In conventional approaches, the existing model is replaced with the new model once the model is retrained for the new data. It becomes even worse when we need to add new class or category for the existing model as it requires to redesign the model and train it from the scratch. Existing approaches do not provide support for distributed deep learning for building models that can evolve efficiently new requirements and data from the users.

To increase the model's performance, more complex models are being developed. These often involve using multiple models i.e. also use visual and audio for image classification. Such implementations require more distributed approaches. As machine learning algorithms advances, user expectation also advances which leads to building model to perform multiple tasks by combining the existing models.

1.3 Proposed Solution

This thesis presents a scalable and distributed deep learning framework. The presented approach aims to solve the 3d object classification problem but can be applied to many machine classification problems which are supervised in nature.

The deep learning problem is divided into two portions hierarchically. The two parts are called divide learning and conquer learning. At the divide learning level, the classes are divided into smaller subsets so large problem is divided into smaller supervised classification subproblems. At conquer learning, the one neural network is trained for each of these subproblems and this neural network is less complex and can be heterogeneous as per the requirements of subproblems. After training each of the models, if we are getting the desired accuracy then higher layer model is trained otherwise optimization is performed by redistributing the confusing class which includes identifying the confusing class by checking the confusion matrix, move the confusing class to another subproblem

and retrains the lower layer model. At the higher level, lower layer's subproblems are considered as the classes and another neural network is prepared. This network can be same as lower level models or can be more or less complex.

The approach makes use of Convolutional Neural Networks [5, 12, 13, 14]. TensorFlow [15], open source library for deep learning, numerical computation and machine intelligence is used. The proposed approach can be extended to solve for most other classification problems like speech recognition, image classification, natural language processing. The evaluation of several case studies is presented to verify that the H3DNET framework can achieve a high rate of accuracy while supporting the distributed deep learning.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we will learn about the terminology that has been used in the thesis and discusses the background technologies. The work related to our problem and available solutions will also discuss at the end of the chapter.

2.1 Terminology and Technology

2.1.1 Machine Learning

Machine learning is the area of computer science which allows computers to act without being explicitly programmed or without giving any command. The roots of the machine learning can be found in the field of computational learning theory in artificial intelligence and pattern recognition. It aims to overcome the strict static program instruction by developing and studying algorithms which can make the data-driven predictions or decisions. The methods, theory and application domains are delivered by the mathematical optimization which is the base of the Machine Learning.

It is strongly dependent on the mathematical optimization, which delivers methods, theory and application domains to it.

“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks the in T, as measured by P, improves with experience E.” [11]

Machine learning tasks are typically classified into three broad categories and this categorization is based on the available learning system’s nature of the learning. These categories are

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

The labeled training data is used for learning in a supervised learning [18]. It designs inferred function by analyzing the training data. The new examples are mapped using this produced inferred function. The training dataset consists of input data and response values which are used by the algorithm to train itself and then the model is validated using another set of data. The accuracy of model largely depends on the size of training data so larger training datasets often results in higher accuracy of the supervised learning algorithm.

Supervised learning algorithms can be further divided into two sub-categories:

- Classification: In this supervised problem, the algorithm takes inputs of two or more classes and predicts the class of the inputs after training.
- Regression: In this problem, algorithm tried to predict the continuous-response values for input.

An unsupervised learning is the branch of machine learning in which algorithm uses the unlabeled dataset to draw the inferences. The commonly used approach for unsupervised algorithms is clustering in which clusters are modeled using metrics which measure the similarity between them and such measures are Euclidean or probabilistic distance.

In reinforcement learning, a computer program must perform certain goal by interacting with a dynamic environment. The goals are like playing a game against an opponent or driving a vehicle. As program explores problem space, it is rewarded or punished to provide feedback.

2.1.2 Neural Network

An Artificial Neural Network (ANN) is a mathematical model which process the information in a way that is inspired by the way biological nervous systems, such as the brain (Figure1 and Figure2). It consists of millions of processing elements which are highly interconnected and aims to solve the specific problem. These elements are called neuron which takes several inputs and generates a one or multiple real-valued outputs. It is estimated that the human brain is the densely interconnected

network of approximately 10^{11} neurons in which on an average 10^{4} other neurons are connected to the single neuron.

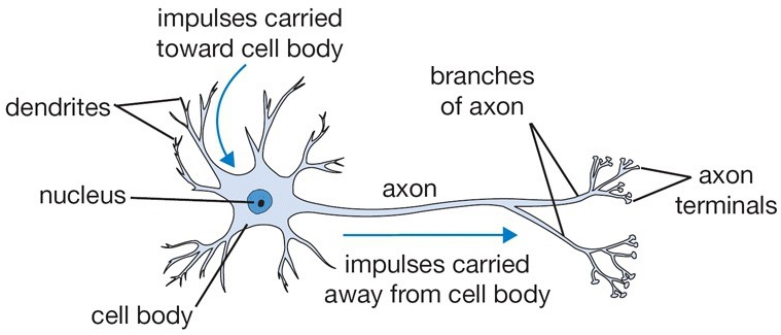


Figure 1: Biological Neuron's Cartoon Drawing [1]

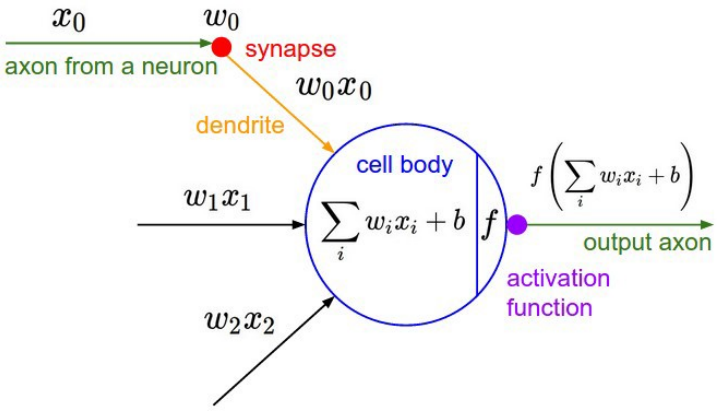


Figure 1. Biological Neuron's Mathematical Model [1]

ANN system is designed using the component called a perceptron (Figure 3). A perceptron is an artificial neuron which takes a vector of real-valued inputs and computes a linear combination of these vectors to generates binary result based on certain threshold.

$$\text{Output} = \begin{cases} 0 \; if \; \sum_j w_j x_j \leq threshold \\ 1 \; if \; \sum_j w_j x_j > threshold \end{cases}$$

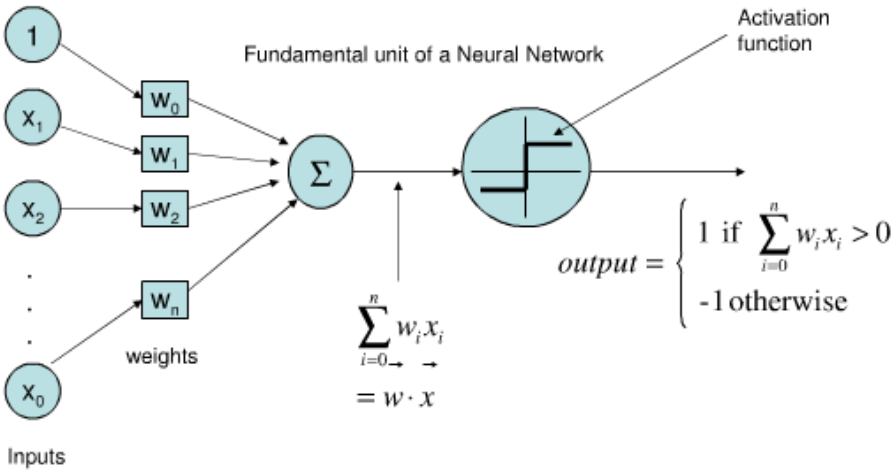


Figure 2. Artificial Neuron's Mathematical Model [1]

This function which generates output is called "activation function". Various activation functions are:

Sigmoid: The Sigmoid function has the following equation

$$y = \sigma(x) = \frac{1}{1 + exp^{-x}}$$

It takes a real value and outputs between 0 and 1. However, the gradient at the tail of 0 or 1 is almost zero because of the saturation.

Hyperbolic Tangent: The mathematical form of TanH is non-linearity is as follows

$$y = 2\sigma(2x) - 1$$

It outputs a real-valued number int the range of -1 and 1.

Rectified Linear Unit: The mathematical form of RELU is

$$y = \max(0, x)$$

2.1.3 Convolutional Neural Networks

Convolutional Neural Network(CNN)s are designed by the number of neurons that have weights and biases which are learnable. Each neuron takes some inputs then performs a mathematical function and optionally applies it a non-linearity. The whole network still expresses a single differentiable score function. On the one end of the neural network, there are raw image pixels while on the other end of the neural network there are scores for classes. In between these ends, there is loss function on the last layer and other configurations parameters.

For the images of size 32x32x3 (32 wide, 32 high, 3 color channels) and if we design a regular neural network using fully-connected then a single fully-connected neuron in a first fully-connected layer of a Network would have 3072 (32 * 32 * 3) weights. This amount is still manageable, but clearly, this fully-connected network does not scale as the images become larger. For example, if an image has the size of 200x200x3 then the first layer would have 120,000 (200*200*3) weights. Moreover, the only single layer may not serve the purpose, so we would like to have several such layers, so the situation becomes worse! Clearly, this full connectivity with a large number of parameters would become the lead cause of overfitting.

3D volumes of neurons: The inputs are images or 3D objects which constrain the architecture in the more effective way. The Convolutional Neural Network takes advantage of this fact for designing the network. In contrast to a regular Neural Network one dimensional structure, the layers of Convolutional Neural networks have the width, height, and depth where neurons are arranged. Here is a visualization:

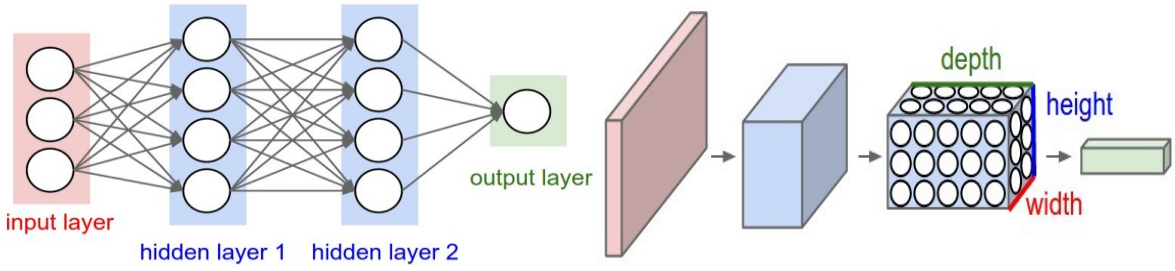


Figure 3. Left - 3-layer Neural Network. Right - Convolutional Neural networks

2.1.4 Model Training

The model training involves providing a training data to a designed algorithm for learning. The out of the training process is the artifact which is referred to the model.

The training data must have a target or target attribute which is the correct answer. The aim of the learning algorithm is to find the patterns in the training data which can be mapped between the input data attributes and the target (the answer that you want to predict), and it generates an ML model which captures these patterns.

The generated ML model can be used to make predictions of new data for which the target is unknown. For example, to train an ML model to classify the email as spam or not spam, you should input the training data that has emails and labels for that email which is either spam or not spam. Once the training model is completed using these data, the generated model will attempt to classify the new email whether it will be spam or not spam.

2.2 TensorFlow

TensorFlow is an open source library which is used for numerical computation using data flow graphs. The graph contains the nodes and the edges. The node represents the mathematical operations and the edges represent the multidimensional data arrays which are communicated to them. Because of the flexibility of architecture, one or more GPUs or CPUs can be used for deployment

of computations. Researcher and engineers of Google Brain Team have developed the Tensorflow for the purposes of conducting machine learning and deep neural networks research.

TensorFlow can be used for designing the large-scale distributed machine learning training and inference. As it is a cross-platform library, can be used across desktop, mobile etc. The architecture of Tensorflow is shown in Figure 4. TensorFlow has a layered architecture. The core runtime kernels which are implemented in different languages are separated from user level code using C API. On the top of Networking kernel implementation resides and the master and dataflow executors on the top of that. The parameter of Tensorflow is from disbelief [3].

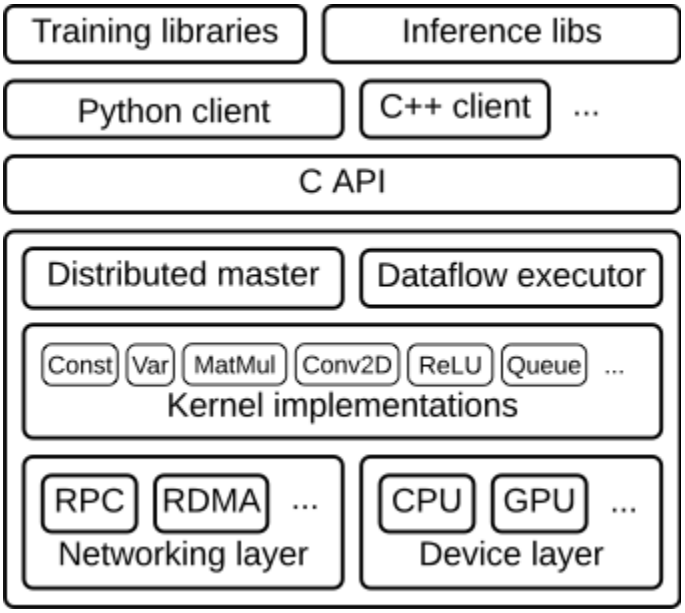


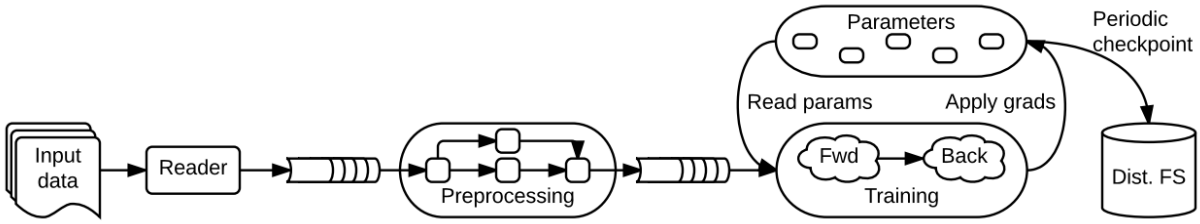
Figure 4. TensorFlow Architecture [15]

The main abstraction of Tensorflow is data flow graph which is used for describing the mathematical computation using the nodes and edges.

- Mathematical operations represented by the nodes
- The i/o relationships between several operations described using the edges of the graph

- Dynamically-sized multidimensional arrays called tensors are carried by the edges of the graph

TensorFlow gets its name from the flow of tensors. Once incoming edges get all the tensors, different computational devices get the nodes and asynchronous operations are performed from the all the nodes. The model becomes more distributed with this design.



A schematic TensorFlow dataflow graph for a training pipeline contains subgraphs for reading input data, preprocessing, training, and checkpointing state.

Figure 5. Example TensorFlow dataflow graph [15]

2.3 Point Cloud

A point cloud is a collection of data points represented in the coordinate system. In a three-dimensional system, X, Y, and Z coordinates define these points and often are aimed to represent the object's external surface.

3D Scanners are used to create the point cloud. These compute the number of points on the surface of an object and generate a data file which contains a point cloud. The generated point cloud has the points which are computed by 3D Scanner which are used in so many applications which includes creating 3D CAD models as a process of manufacturing and quality inspection. The applications like rendering, visualization, mass customization applications and animations also use the point clouds.

Point clouds can be inspected and rendered directly but usually, 3D applications do not use point clouds directly. Therefore, polygon mesh, NURBS models, triangle mesh or CAD Models are created from the point cloud using the surface reconstruction.

Industrial metrology can directly use the point clouds for inspection. In the manufacturing process, a CAD model is aligned with the point cloud of manufacture product for comparing and checking quality and differences of products. The color maps can be created from these differences which give the indicator about the deviation using color visualization. The point cloud can also use to extract the geometric tolerances and dimensions.

2.4 Related Work

The research focuses on building deep learning models and predicting the class of 3D objects. The easiest comparison one would make with this approach is pre-processing the 3D data into voxels and then feed them into neural network for training and prediction. All these approaches build CNN or RNN model and trained them with 3D datasets converted into voxels or point clouds. Our approach uses the same voxelization approach but also converts processed volumetric data into binary format. After performing preprocessing of the 3D dataset uses the hierarchical neural network build using the top-down approach and predict the class by combining the prediction from all models.

VOXNET[7][8] discusses a basic 3D convolutional neural network architecture to perform the fast and accurate object classification. It takes a point cloud segment as input and performs classification. This approach consists of two components: a volumetric grid or voxel grid and a 3D CNN. Occupancy Grids represents the 3D objects as a 3D grid of variables and maintain a probabilistic estimate of their occupancy. This approach creates three types of occupancy grids: the first one is binary occupancy grid, second one is density occupancy grid, and third is hit occupancy grid. In binary occupancy grid, occupied and unoccupied are two states which can voxel choose. In density occupancy grid, voxel state is based on the continuous probability. The 3D CNN model takes this

occupancy grid and predicts the labels. With the various combinations of filters and hyperparameters, this approach allows creating countless architectures for 3D object classification. The overall loss of neural network is optimized using Scholastic Gradient Descent (SGD) with momentum optimizer. The only similarity between this approach and our approach is the use of occupancy grid for converting 3D objects into a 3D lattice. Our approach uses binary occupancy grid for preprocessing of input data.

3D ShapeNet[7] proposes a Deep Belief Network which aims for object class recognition and also aims for shape completion. In this approach, a occupancy grid is used to represent a 3D geometric shape. It uses a ModelNet, 3D object dataset with large number of object for training and testing of the neural network. To optimize the neural network, Next-Best-View-Prediction method was used which try to predict the shape from multiple views. This approach also uses the voxel grid to represent the binary variables of 3D shape and then provide them to the neural network for prediction of shape. We also used same voxel grid approach to represent the 3D shape, but the grid size for their approach was $30 \times 30 \times 30$ compared to $40 \times 40 \times 40$ in our approach. The neural network used in this is a single convolutional deep neural network for comparing all possible view while we have used multiple convolutional neural networks with fewer numbers of layers. This makes our approach significantly different when compared by number of layers, complexity of network and number of networks.

PointNet: Deep Learning on Point Sets [9] is another deep learning neural network for 3D object classification and segmentation. This is approach eliminates the requirements of creating voxel grid from the 3D object and uses point cloud or 3D point sets as an input to the neural network for classification. It also achieves the objective of segmentation of 3D objects. To perform classification, it follows the three steps process inside the neural network. The first step is aggregating the information from the points, the second step prepare the local and global aggregation information generated by max pooling which combined in the third step for final prediction. This model uses the CNN for performing the object classification which is the only similarity between our approach and

this approach. Although its uses the concept of local and global information, it does not use the multiple models for prediction and the neural network used by this approach is typically complex than us.

Figure 6 shows the qualitative comparison between the VoxNet[7], PointNet, ShapeNet and our approach.

	SHAPENET [8]	**POINTNET [9]**	**VOXNET [7]**	**H3DNET (our work)**
Voxelization	YES	NO	YES	YES
3D CNN	NO	NO	YES	YES
Classification	YES	YES	YES	YES
Segmentation	NO	YES	NO	NO
Heirarchical	NO	NO	NO	YES

Figure 6. Qualitative Comparison between SHAPENET, POINTNET, VOXNET[7] and H3DNET

CHAPTER 3

The Hierarchical 3D Net for 3D Object Classification

3.1 Overview

The proposed solution is the generalized approach for any type of supervised learning algorithm. We have used this framework for 3D object classification with the convolutional neural network. The overall architecture which used in 3D CNN model is shown in Figure 7. The problem in our approach consists of two stages, divide learning and conquer learning. In divide learning phase, this set of classes are divided into smaller subsets using some class hierarchy or some clustering algorithm. Conquer learning is to solve these subproblems using the suitable neural networks approaches i.e. softmax regression, multilayer perceptron, feed-forward neural network or convolutional neural networks. As we are dividing the problem into small subsets, the neural network we need to solve these problems will be less complex with fewer parameters. We called these models conquer model which corresponds to each subset. The training and optimizing conquer model includes training neural network for each subset, check the confusion matrix, shuffle the confusing the classes and again train the neural network until the desired threshold is achieved.

Once the training is completed for subproblems, we will use each subset as a class for training the upper-level model. These include training upper level to conquer model with each possible hyperparameters, shuffle or move the subset problems to upper level if they are affecting the performance. For higher level, it may be possible that we need the more complex neural network with the higher number of parameters. After the completion of training for all models, we first load the higher-level model to make the prediction about the generalized class for input and after that, we load the lower conquer model based on the output of the higher-layer model to make the final prediction.

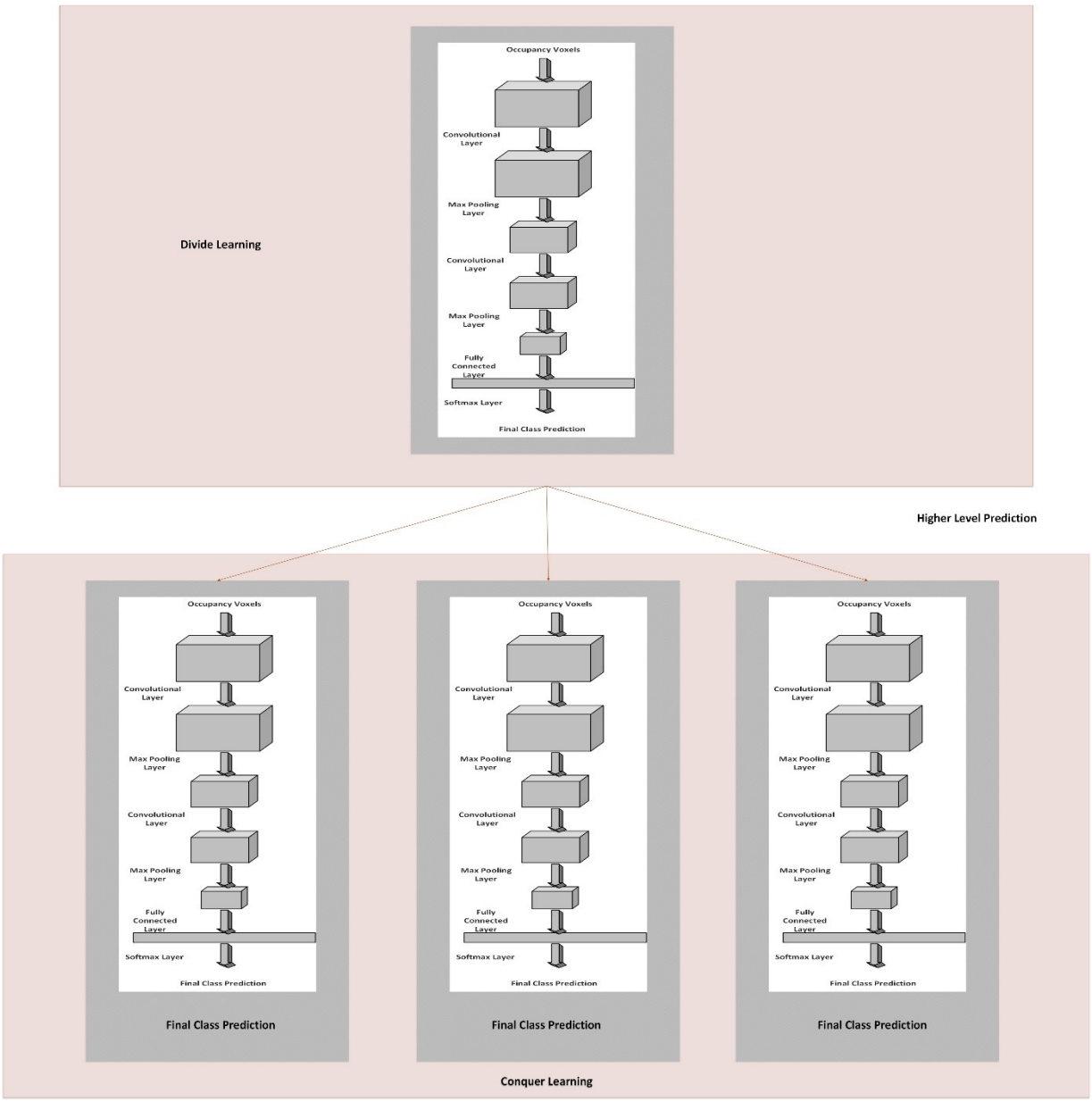


Figure 7. Proposed H3DNET Framework for 3D Object Classification

To solve the problem of 3D object classification using this approach with Tensorflow framework, we followed the three steps which includes preprocessing 3D datasets to convert into Tensorflow standard format, then divide the classes into subproblems and prepare the conquer learning model for sub-problems and last prepare the higher level conquers learning model using sub-problems as a class.

3.2 Data Preprocessing

We have addressed the problem of 3D object classification using the above approach. To use this approach, we must need to preprocess the 3D objects. The structure of the 3D object is not similar to 2D images as 2D images are regular in terms of size i.e. length and width, but it is not same with 3D objects. 3D objects might differ in terms of width, length, and depth so it is necessary to convert them into regular size 3D objects. Figure 8 shows steps of data preprocessing. The steps include reading 3D data, voxelized them and then create the binary record of them.

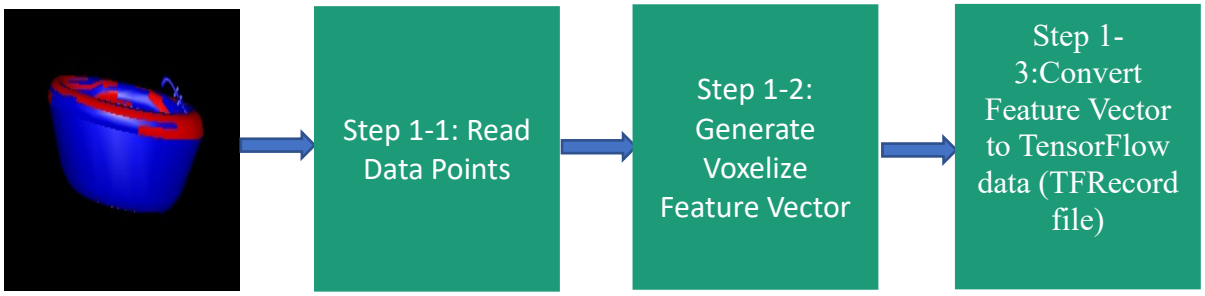


Figure 8. 3D Object before and after voxelization

- **Read 3D Data**

3D data comes in various formats and the structure used in these formats differ from format to format. The 3D format includes OFF (Autocad Format), MAT (MatLab Format), NPY (Numpy Array Format), PLY (Point cloud). To read different format, we need to apply different techniques. The numpy, pandas and matplot libraries allow reading 3D data by providing the various method which can read the as 3D or 2D dimensional array. We have used OFF format for the evaluation of our approach.

- **Voxelization**

The dimensions of 3D objects are not fixed. They come in various formats with heterogeneous length, width, and height. It is difficult to fit them into the fixed dimension of the neural network. Voxelization converts 3D objects into the fixed size voxel or occupancy grid using sampling and clipping of 3D objects. We have used binary voxel grid which has only two states: occupied state and

unoccupied state. Voxelization fits different sized 3D object into fixed size homogenous grid without losing the spatial information of 3D objects. Numpy library provides the various methods which allow doing voxelization by clipping and sampling of points.

3.3 Divide Learning

In this step, smaller subproblems are created from the set of classes which are solved in conquer learning phase using the neural network. There are several ways to select the size of the smaller sub-problems, the number of classes in smaller sub-problems. As we discussed, one way is based on the future needs of incremental and distributed model. If there is no way to decide the sub-problem size based on future needs, there is the more practical method of clustering which identifies patterns in features. Multi-class Discrimination Distribution Model [2] is such algorithm which can be used to find the appropriate clusters. The classes are distributed based on the evidence from the confusion matrix. The Euclidean distances can be used to calculate the degree of heterogeneity of classes.

The algorithm of Hierarchical Distribution:

Step 1: Original dataset classification

Step 2: Several measurements such as Euclidean distance (ED) or normalized ED for Confusion matrices can be used to compute the heterogeneity

Step 3: K-means clustering with the matrices

Step 4: The k matrices classification (distributed classification)

Repeat until the accuracy < threshold or $\sqrt{n} < 4$

The MCDD model's visual representation is shown in Figure 9.

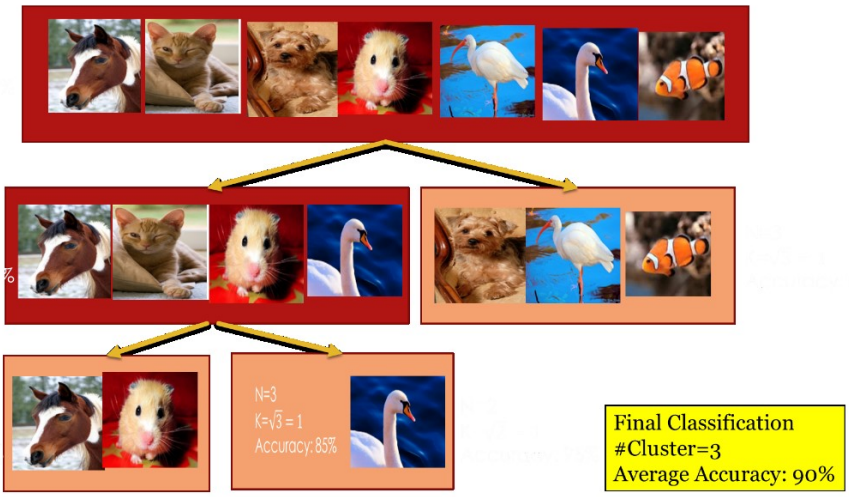


Figure 9. Multi-Class Discrimination Distribution Model (MCDD) for Group Selection [2]

As the number of classes in sub-problems is less, we need the less complex neural network with the lesser number of parameters. The accuracy and the performance of conquering learning are also better than the single large problem.

Consider the classification ModelNet10 [16] dataset of 3D objects. It consists of 3D CAD models of 10 object classes namely Bathtub, Bed, Chair, Desk, Dresser, Monitor, Nightstand, Sofa, Table, and Toilet. Before using these 3D objects for our models, we have preprocessed them to convert them into binary format. Initially, the classes are grouped using the ImageNet [21] hierarchy and divided into 3 subproblems with the name as the bathroom, bedroom, and hall. Bathroom subproblem contains bathtub, toilet, and dresser while bedroom contains classes like the bed, desk, and nightstand. The remaining classes are grouped into subproblem calls Hall. Figure 10 shows the initial class distribution of ModelNet10 using the ImageNet class hierarchy.

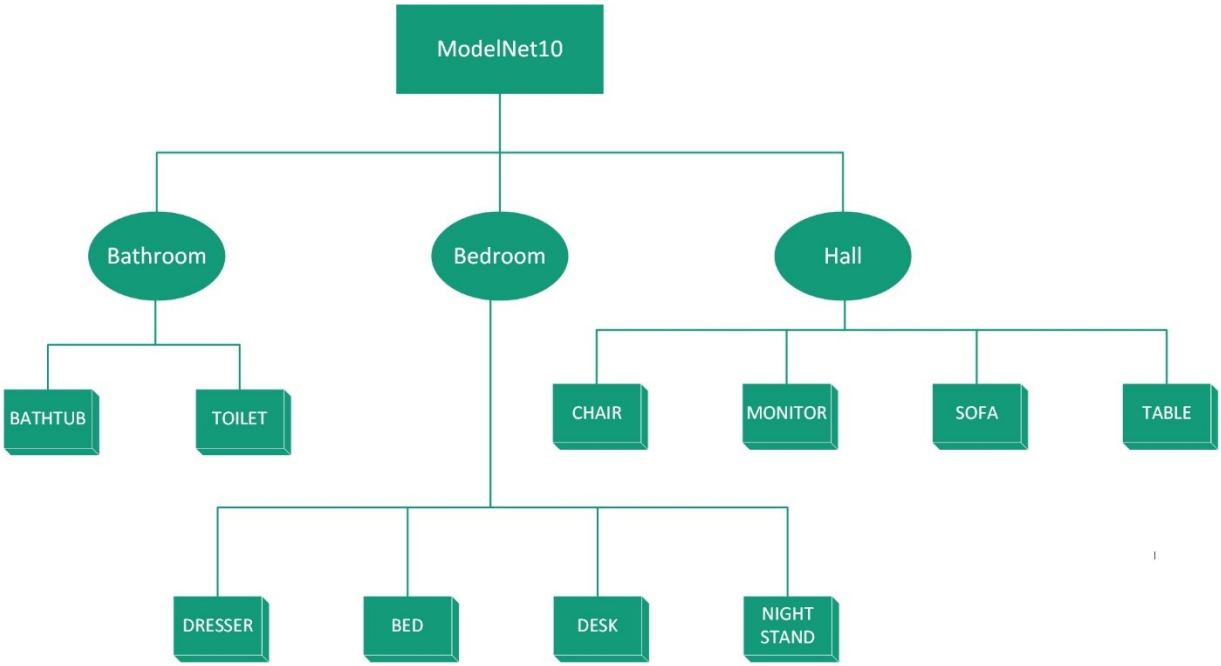


Figure 10. Initial Distribution of ModelNET10 Classes

IMAGENET:

IMAGENET is a large visual dataset designed for the visual object recognition. The WordNet hierarchy is used to organize the IMAGENET. There are hundreds and thousands of images on the each node of the hierarchy. In ImageNet there are over 10 million of images and more than 100 classes of images. Figure 11 shows the hierarchy and images in the ImageNet.

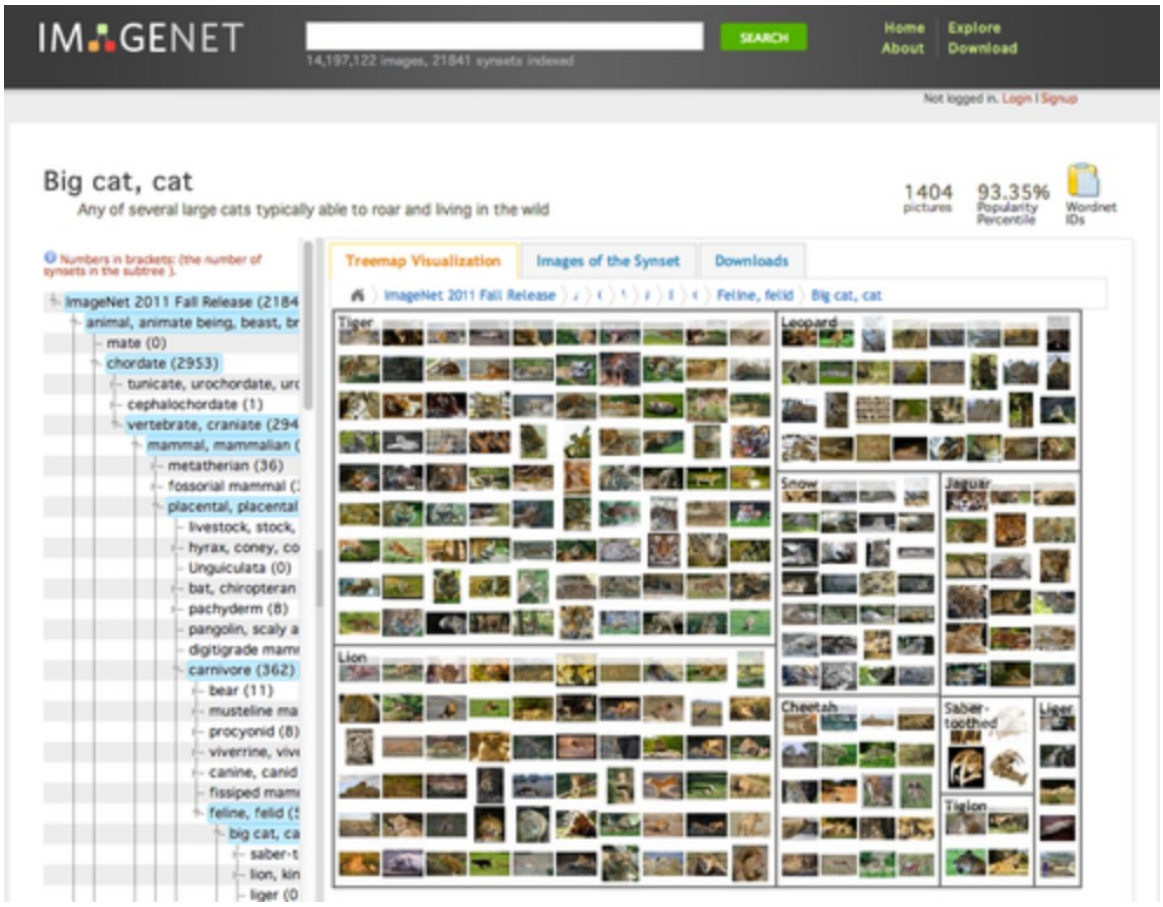


Figure 11. ImageNet Image Hierarchy

3.4 Conquer Learning

Once the classes are divided into smaller subproblems in divide learning, all the subproblems are conquered at this stage. In this stage, all the subproblems are solved using the optimized neural network. This includes designing of the neural network, choosing optimal parameters and training of neural network using the chosen hyperparameter. It also includes optimization of the neural network. In optimization phase, classes are redistributed to subproblems to get the optimal distribution of classes.

The neural network we have used is the 3D convolutional neural network has 3 convolutional layers, 2 max-pooling layers and 1 RELU activation layer with softmax as the output layer. The

hyperparameter we have used are tweaked as per the requirements to get the optimal performance of subproblem.

3.5 Optimization

After divide and conquer learning, if we are not getting the desired accuracy then it is required to do optimization by redistributing the classes. First, we need to redistribute classes to get the desired accuracy for lower level learning model. The steps to do optimization at the lower:

- Train the neural network at in divide learning step for each subproblems
- If any network is not getting desired performance, check the confusion matrix for that model
- If there are any classes which confusing with each other, then initially redistribute them among subproblems
- Train once again the subproblems which are redistributed
- If redistribution among subproblem does not work, then try to move confusing class to upper level
- Follow the same divide and conquer learning until you get optimal solutions

Figure 12 shows the basic optimization strategy algorithm.

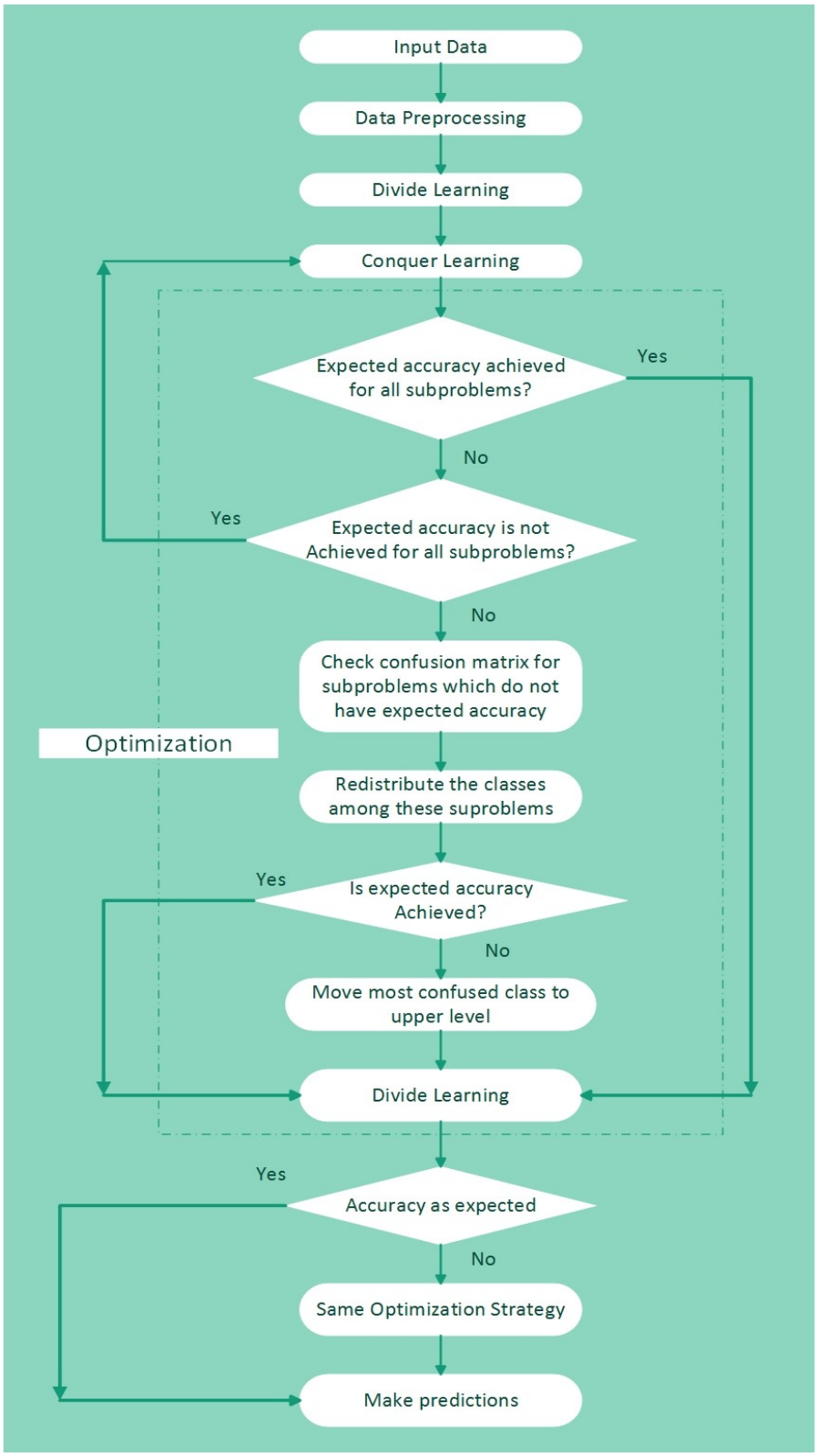


Figure 12. Optimization Algorithm

Once optimization is completed at the lower learning level, train the model for the higher level. Also follows the same optimization strategy at higher level. It is also possible to add more layers or parameters to make the model more complex for higher layer learning. For the higher layer learning, we have tried all the possible hyperparameters and redistribution criteria to get the desired performance. The aim is to get the highest accuracy, we can achieve by tweaking all the possible hyperparameters.

3.6 Fuzzy Classification

Many 3D objects have similar spatial features and hence they have similar shapes and dimensions. The examples of such classes are desk and table, nightstand and dresser, cup and bowl. These classes always got confused with each other when we train the neural network to classify them and affect the accuracy and performance of classification model. It is better to predict the range of label instead of the single label, so we can make the better prediction about the possible label of class. This is we called the fuzzy classification and the number of labels predicted is decided using the fuzzy parameter k. If k = 1 then we will predict one label if k = 2 for the first layer, then we will predict 2 labels at first layer and one layer at the lower level. Now, the question is how to decide the value of k? The strategy show in Figure 13 can be used to decide the value of k.

```
Start with k = 1
Number of subproblems s
Decide threshold (can be overall accuracy, can be accuracy at higher level or any other parameter)
While (performance < threshold)
        Train the models
        Predict labels
        if (performance = threshold)
                choose k as final fuzzy parameter
                Start making prediction
    k++
```

Figure 13. Fuzzy Parameter Selection

The performance can be overall accuracy, accuracy at higher level or any parameter. For our case, we have used the combination of accuracy at higher level and overall accuracy to decide the k. We have used confused classes to finalize the value of k. For the confused classes, first check the high-level accuracy for different value of k. If it is increasing, then check the overall accuracy for that k. When there are minor changes in the value of accuracy at high level and overall accuracy, we are choosing that k. Figure 14 shows such procedure using the nightstand class example.

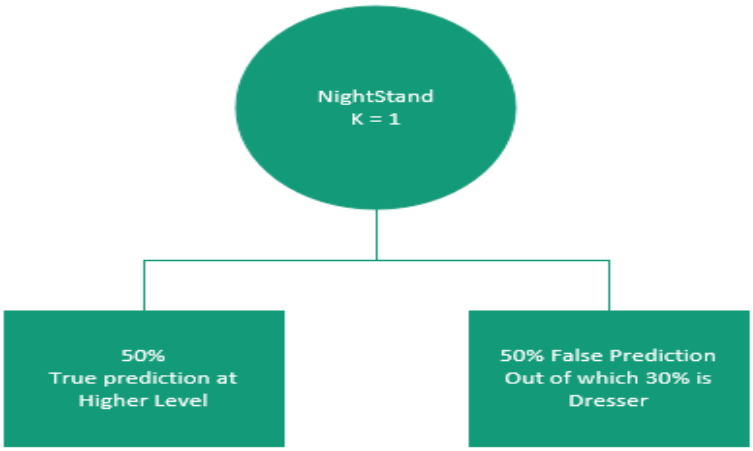


Figure 14. Fuzzy Classification Example (k=1)

When we start with k = 1, when we make prediction there are 50% prediction are true at higher level and we further proceed to next level the predictions are correct for Nightstand. But for the other 50% false predictions, around 30% predictions are predicted for Dresser and 20% prediction are others. This is shown in the Figure 14.

Figure 15. Fuzzy Classification Example (k=2)

Figure 15 shows the prediction sample for k = 2. If we take k = 2, then those 30% prediction which are predicted as Dresser will be predicted as Nightstand and out of 20% false prediction, some of the predictions will be predicted true. Now for k = 3, at the higher level almost 96% predictions are true, so it does not make any sense to increase the value of k and final value of k = 2.

CHAPTER 4

RESULTS AND EVALUATION

4.1 Introduction

The evaluation conducted on the H3DNEt framework is described in this chapter. The testing environment is also discussed here. The factors like training time taken compared to existing state of the are deep learning algorithms, the accuracy and the hyperparameters are also verified in this chapter.

4.2 Hardware Configuration

H3DNet Framework for 3D object classification is implemented using the following system configuration:

- Memory: 31.3 GiB
- Processor: Intel® Xeon(R) CPU E5-2630 v4 @ 2.20GHz × 15
- Operating System: Ubuntu 14.04 LTS
- OS Type: 64 bit
- Disk: 1.9TB
- Graphics: TITAN X (Pascal)/PCIe/SSE2 (12 GiB)

4.3 Software Configuration

To implement the H3DNET, following software and library have been used:

- TensorFlow-gpu 1.0.1
- Numpy Library
- Scipy Library
- MatplotLib Liberary

4.4 CNN Configuration

The solution presented by VOXNET [7] for 3D object classification contains three convolutional layers (one convolution and one pooling in each convolution layer), one fully-connected layer followed by one dropout layer and one softmax layer. The 3D CNN model used by our framework has the three convolutional layers, 2 max pooling layers, one RELU activation layer, one fully connected layer, one dropout layer and one softmax or readout layer. Figure 16 shows the configuration parameters of the neural network.

Property	**Value**
Conv3D1	Filter: [5, 5, 5, 1, 16], Strides: [1, 1, 1, 1, 1]
Conv3D2	Filter: [5, 5, 5, 16, 32], Strides: [1, 1, 1, 1, 1]
Maxpool3D1	Kernel: [1, 2, 2, 2, 1], Strides: [1, 2, 2, 2, 1]
Activation1	RELU
Conv3D3	Filter: [5, 5, 5, 32, 64], Strides: [1, 1, 1, 1, 1]
Maxpool3D1	Kernel: [1, 2, 2, 2, 1], Strides: [1, 2, 2, 2, 1]
Densely Connected	Shape: [10 × 8 × 8 × 64, 1024]
Dropout1	Dropout probability – 0.7
Output	Softmax

Figure 16. CNN Model Configuration

The hyperparameter used in this neural network is shown in Figure 17. For the selection of hyperparameter is done using the brute force approach. We have tried all the possible hyperparameter and selected those parameters which give the best performance.

Property	Value
Batch Size	64
Grid Size	48 × 48 × 48
Number of Iteration	150K
Dropout Probability	0.7
Initial Learning Rate	0.1
Optimizer	AdaGradOptimizer
Cross Entropy (Loss)	Reduce Mean
Regularizer	L2 Regularizer

Figure 17. CNN Model Hyper Parameters

4.5 Datasets

In this section, different datasets that are used for Evaluation are discussed. They are ModelNet10 and ModelNet40. Both are used for common machine learning benchmarks.

4.5.1 ModelNet10

The ModelNet10 contains CAD models from the 10 categories used to train the deep network in our 3D deep learning project. It is a subset of a larger set of Princeton University dataset. The classes in this dataset are bathtub, bed, chair, desk, dresser, monitor, nightstand, sofa, table, and toilet. The models are in OFF format. The Figure 18 shows the sample for each class.

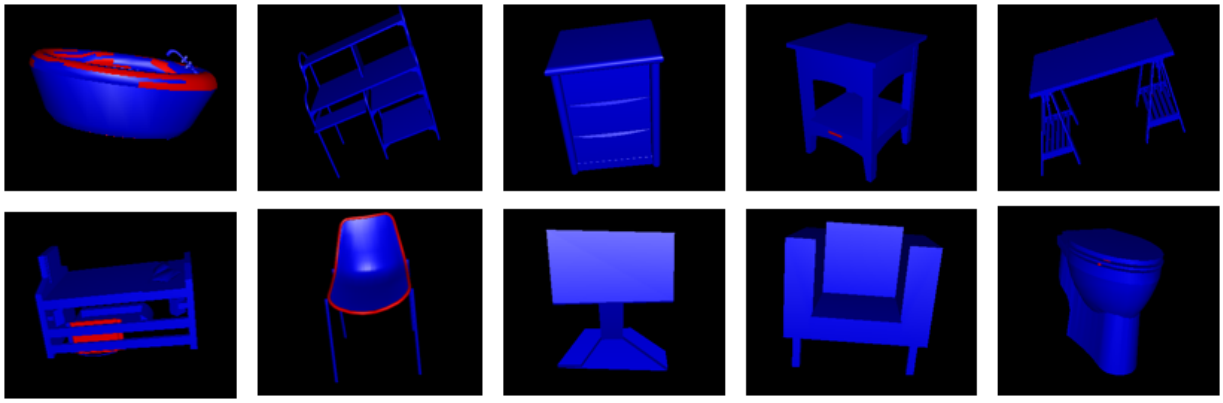


Figure 18. Classes in ModelNet10

Figure 19 shows the statistics about the number of object in each class of ModelNet10.

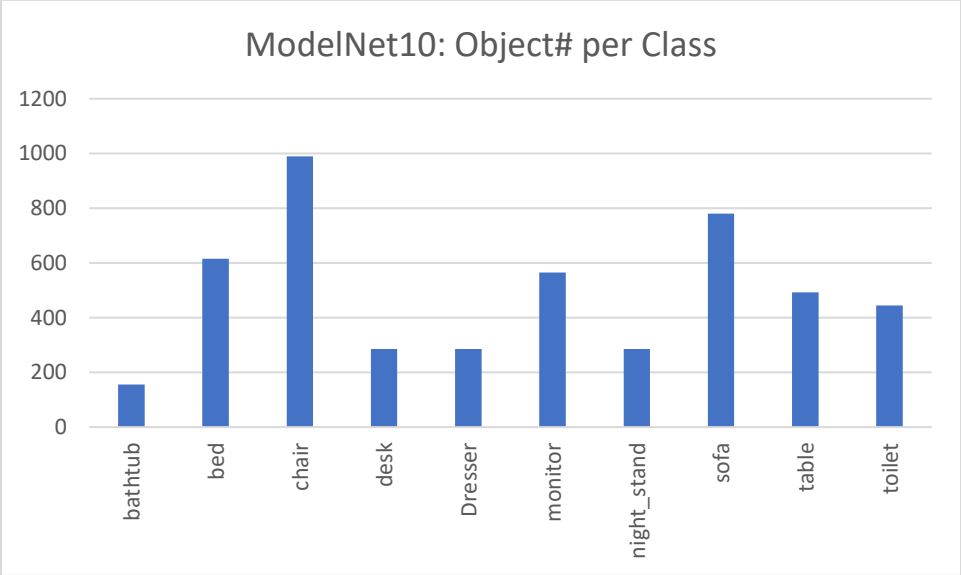


Figure 19. Number of object per Class in ModelNet10

4.5.2 ModelNet40

The ModelNet40 contains CAD models from the 40 categories used to train the deep network in our 3D deep learning project. It is a subset of a larger set of Princeton University dataset. The models are in OFF format. The classes in this dataset are shown in Figure 20:

Airplane	Chair	Glass Box	Night Stand	Table
Bathtub	Cone	Guitar	Piano	Stool
Bed	Curtain	Keyboard	Plant	Tent
Bench	Cup	Lamp	Radio	Toilet
Book Shelf	Desk	Laptop	Range Hood	TV Stand
Bottle	Door	Mantel	Sink	Vase
Bowl	Dresser	Monitor	Sofa	Wardrobe
Car	Flower Pot	Person	Stairs	Xbox

Figure 20. Classes in ModelNet40

Figure 21 shows the statistics about the number of object in each class.

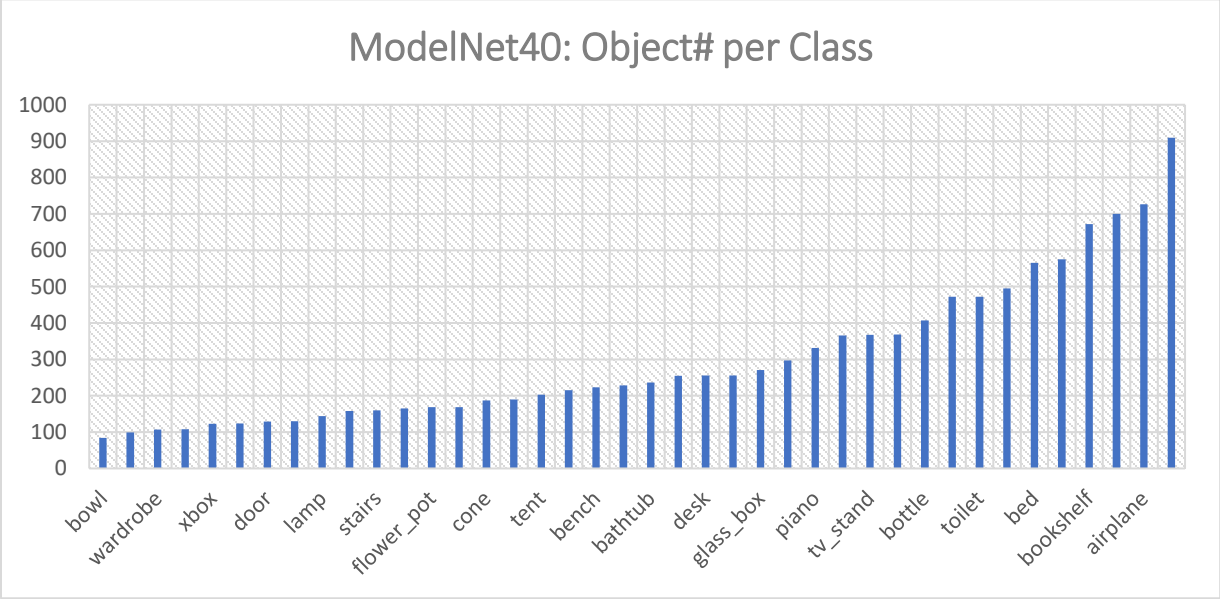


Figure 21. Number of objects in each class of ModelNet40

4.6 Performance Evaluation

The datasets we have used to evaluate the performance are ModelNet10 and ModelNet40. For both dataset we have followed the preprocessing, divide learning, model learning, and optimization stages to get the optimal neural network.

4.6.1 ModelNet10 Performance Evaluation

Objective: Verify the performance of the framework using ImagNet Image Hierarchy

Divide Learning:

In this case, the classes are divided into subproblems using ImageNet image hierarchy. We have divided the classes into the three subproblems, namely bathroom, bedroom, and hall. Figure 22 shows the initial distribution of ModelNet10 classes.

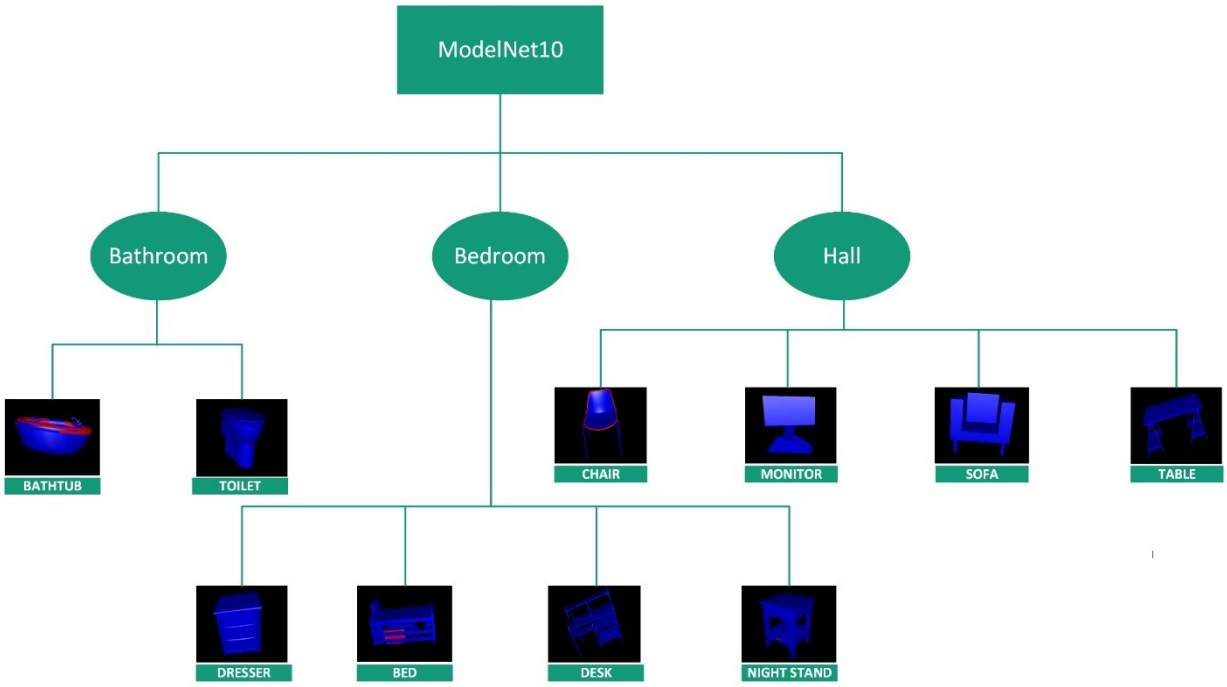


Figure 22. ModelNet10 Initial Class Distribution using ImageNet class hierarchy

From the image, we can see that the bathroom subproblems includes the objects which can be seen in the bathroom such as bathtub, toilet. Similarly, the bedroom subproblems contain the Dresser, Bed, Desk and Night Stand classes while the Hall subproblem contains the Chair, Monitor, Sofa and table classes.

Conquer Learning:

After dividing classes into three subproblems in divide learning phase, three convolutional neural network models are built for 3 groups with configuration discussed above. The

hyperparameters for this model is also shown above. We have used the grid size of 32 × 32 × 32 for voxelized 3D objects. The initialization of weights and biases are done using the random normal initialization method and we have tried all the possible optimizer algorithms. We were able to get the best performance using the Adam Gradient Descent optimizer algorithm. We have trained model for the 100K iteration with the 64-batch size of 64 and 0.01 initial learning rate. For the all the graphs in the figure, x axis represents the number of iterations while y axis represents the accuracy in the scale of 0 to 1.

CNN Model for Bathroom Subproblems:

We can see the accuracy and cross entropy for this model in Figure 23 and 24.

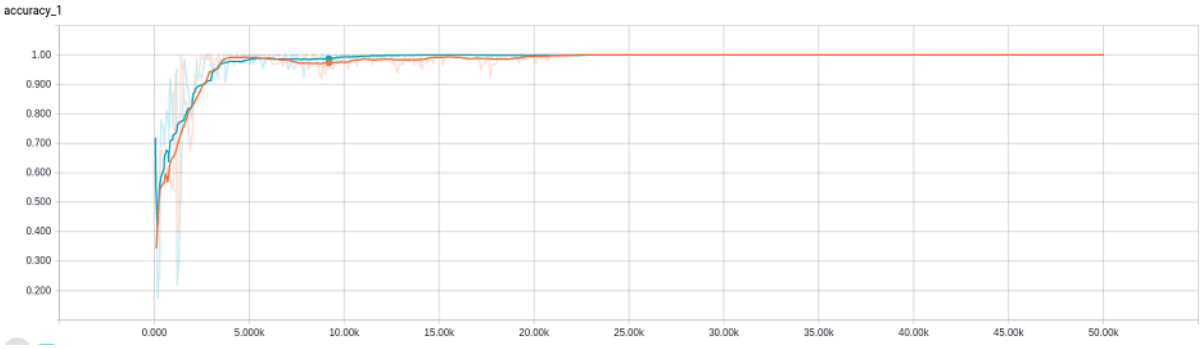


Figure 23. Accuracy of ModelNet10 Conquer Learning for Bathroom

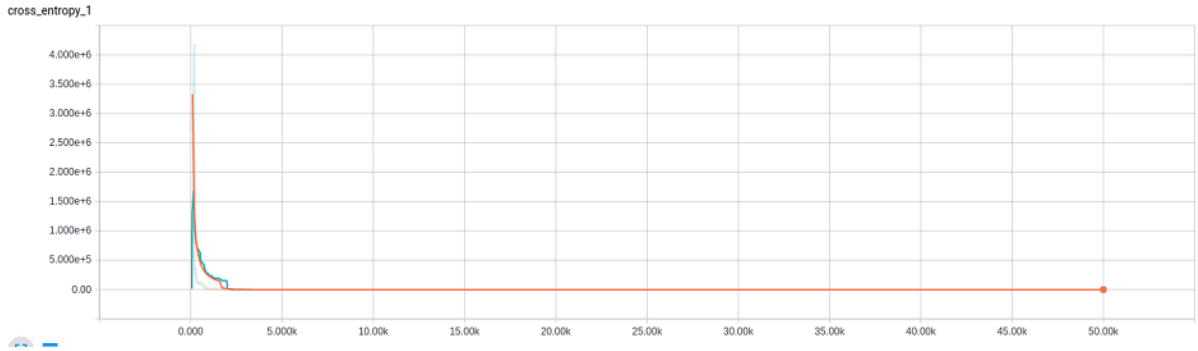


Figure 24. Antropy of ModelNet10 Conquer Learning for Bathroom

CNN Model for Bedroom Subproblems:

The graph of accuracy and cross entropy is shown in Figure 25 and 26.

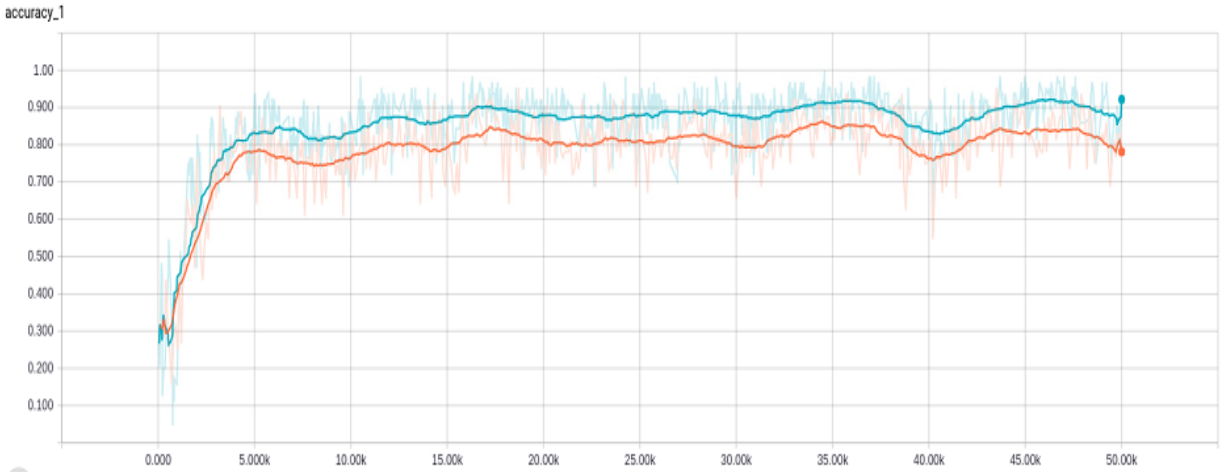


Figure 25. Accuracy of ModelNet10 Conquer Learning for Bedroom

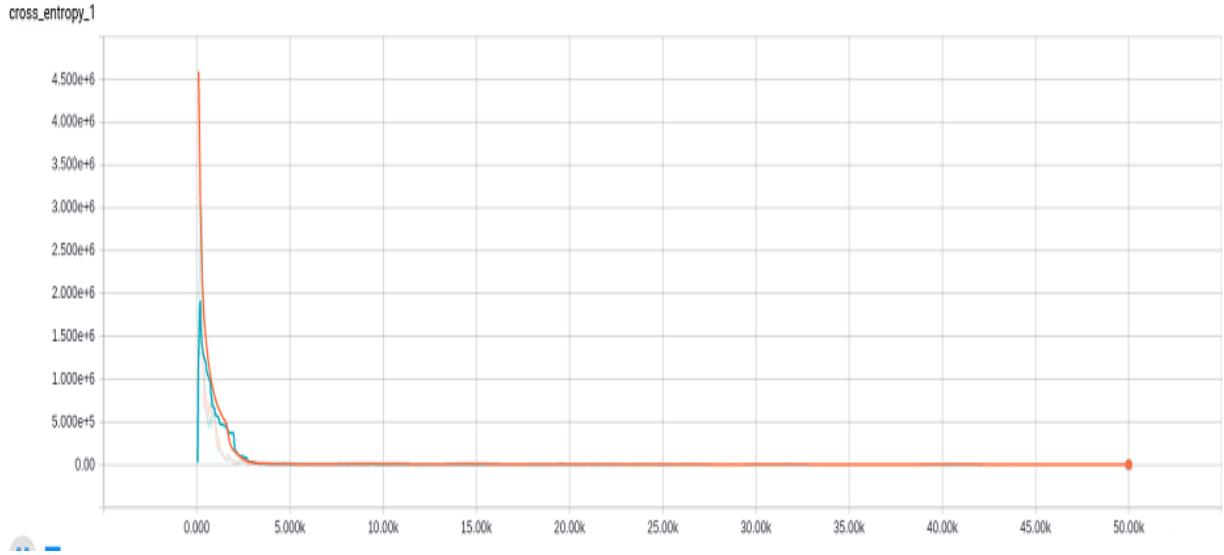


Figure 26. Cross Antropy of ModelNet10 Conquer Learning for Bedroom

CNN Model for Hall Subproblem:

The accuracy and cross entropy throughout the training and testing is shown in Figure 27 and 28.

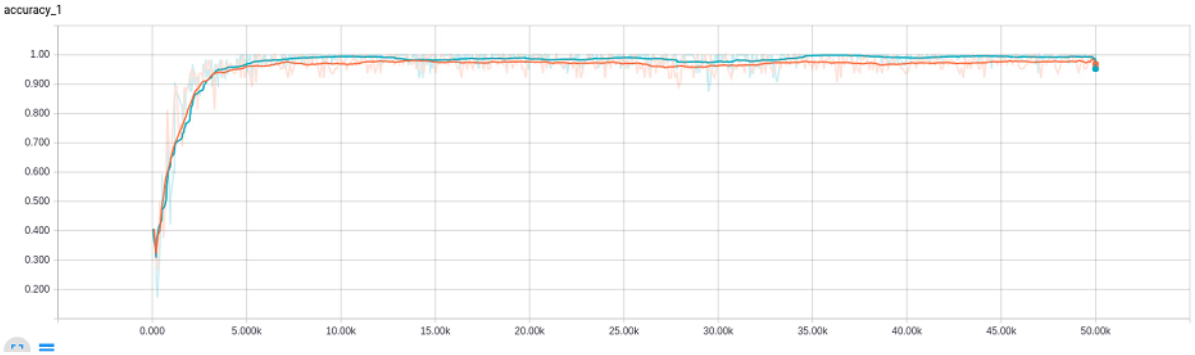


Figure 27. Accuracy of ModelNet10 Conquer Learning for Hall

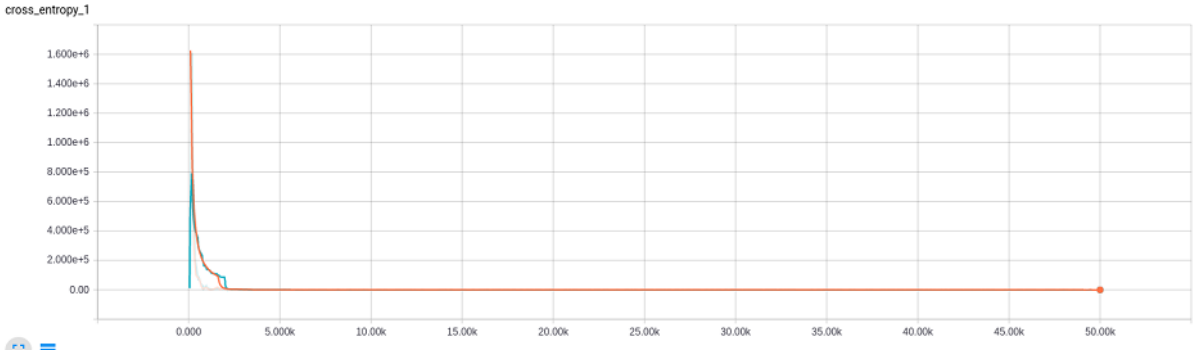


Figure 28. Antropy of ModelNet10 Conquer Learning for Hall

Overall Performance of ModelNet10 with initial distribution:

With this distribution, the highest accuracy we able to get for the training at the upper layer is 0.9687 and for the testing is 0.8925. The graph in Figure 29 shows the accuracy for all subproblems and upper layer.

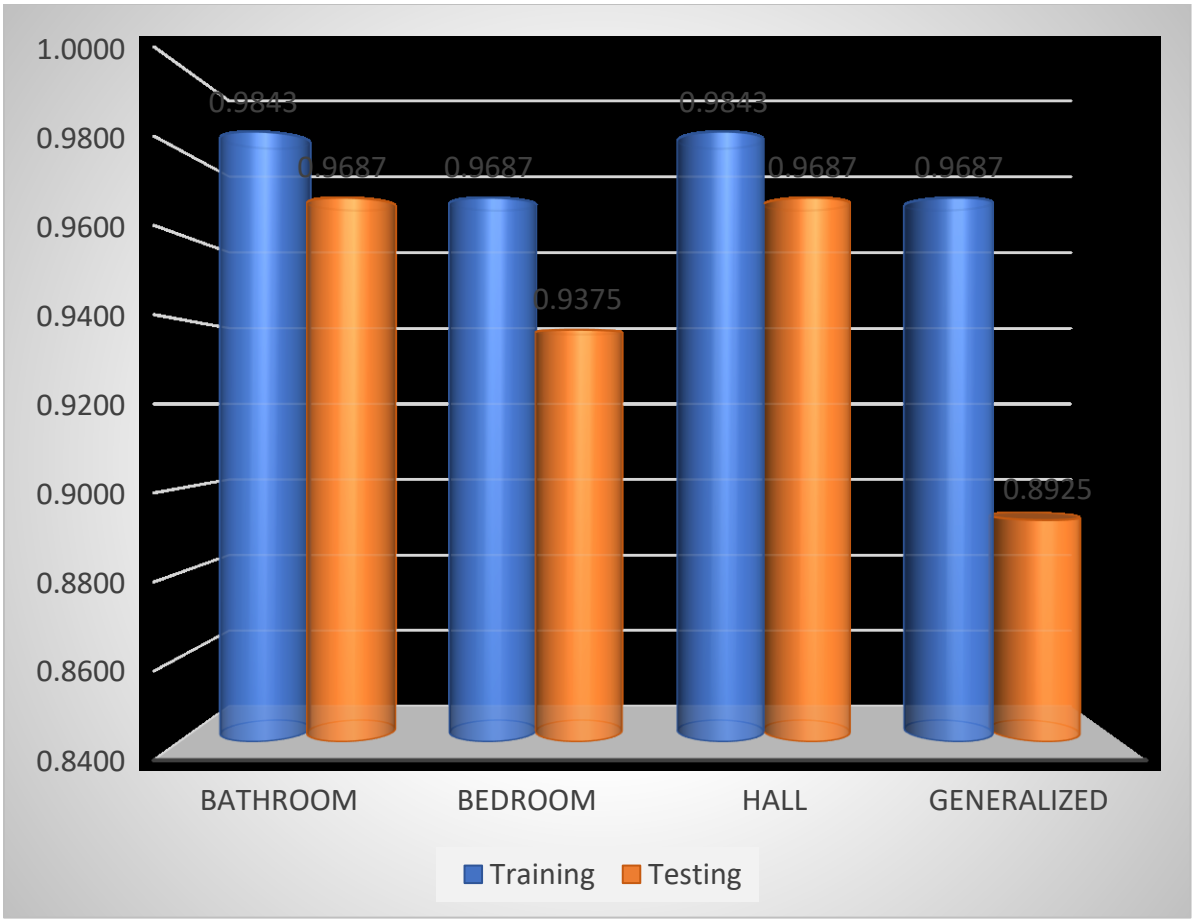


Figure 29. Independent Accuracy of Subproblems and Higher Layer

The overall accuracy of the hierarchical model is always less than a topmost layer. The mathematical accuracy for all subclasses of subproblems is given in Figure 30.

Subproblem	Training	Testing
Bathroom	0.9534	0.8645
Bedroom	0.9383	0.8367
Hall	0.9534	0.8645

Figure 30. Mathematical Accuracy of subproblems for initial distribution

From the Figure 30, the accuracy with initial distribution is not even comparable with the state of art accuracy so it is necessary to redistribute the classes to get the optimal performance.

ModelNet10 Performance After Optimization

With the initial distribution of classes, the classes which are confused were in the bedroom subproblem. We apply redistribution for this subproblem and moved dresser class to the bathroom and trained the model again for bathroom and bedroom. After redistribution, the performance of bathroom was not affected but the accuracy of the bedroom was not close to the threshold. So we again redistributed the classes and move all the classes of the bedroom to the upper layer and treated them as separate subproblems. After several rounds of optimization, the final class hierarchy which gives the optimal performance was in Figure 31.

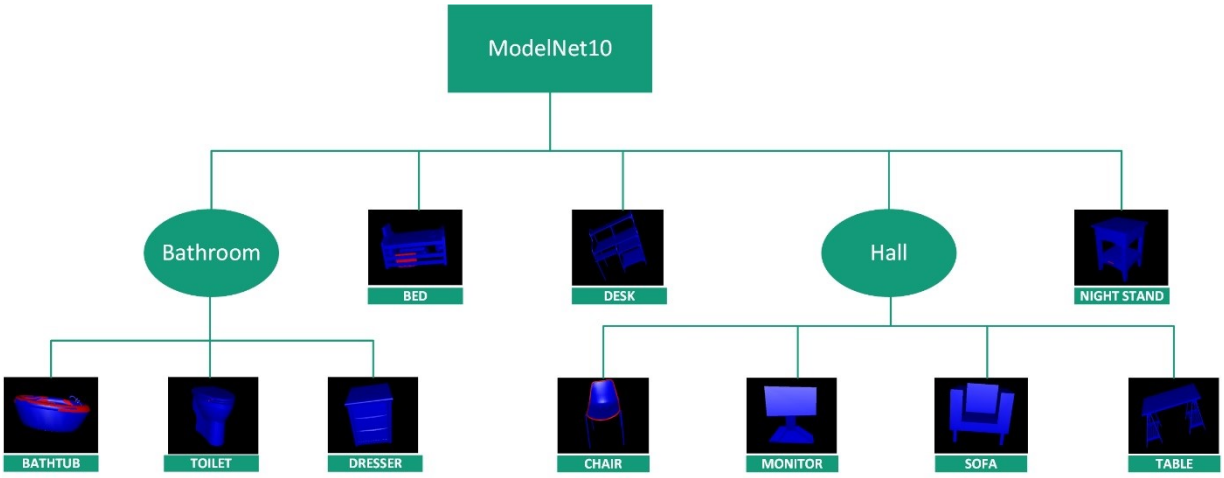


Figure 31. Class Hierarchy After redistribution and optimization

CNN Model for Hall and Bathroom subproblems:

No classes are added or removed from the Hall subproblem, so there is no need to retrain the model for hall subproblem. We have added dresser class the bathroom subproblem, so we have retrained the model for bathroom subproblem. The accuracy and cross entropy throughout the training and testing of bathroom subproblem is shown in Figure 32 and 33.

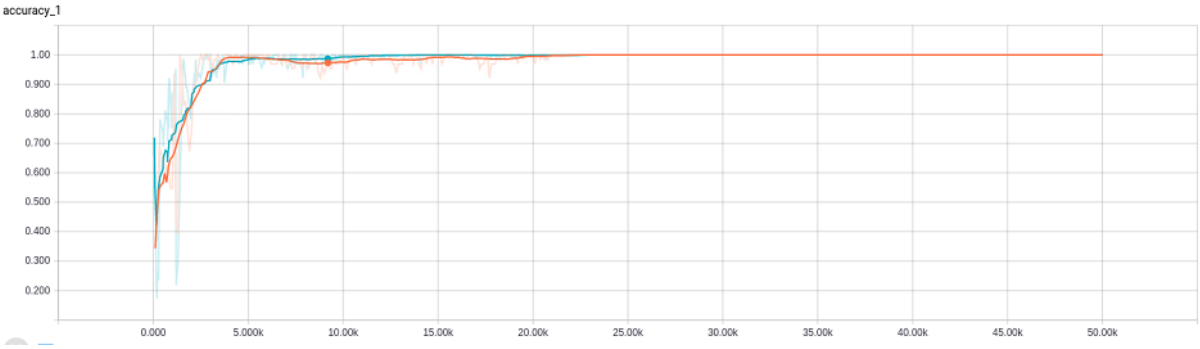


Figure 32. Accuracy of ModelNet10 Conquer Learning for Bathroom

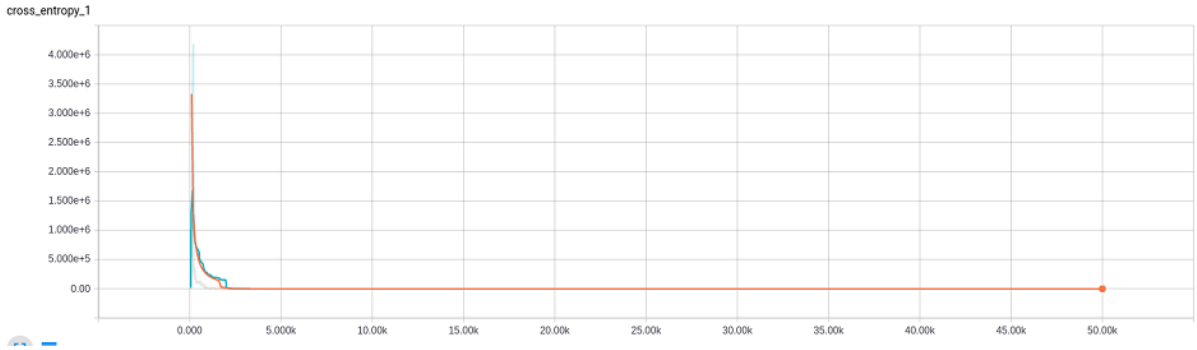


Figure 33. Antropy of ModelNet10 Conquer Learning for Bathroom

After completing training for lower layer subproblems, the conquer learning is applied to the upper layer classes which are bathroom, bed, nightstand, desk, and hall. The 3D CNN used to train this model is same as the lower level model. There are the operations for the different layers, gradient update, accuracy calculation. Figure 34 to Figure 50 shows the graph for accuracy, cross-entropy, summaries, activations, and histograms for all the layers in the model. The weights and parameters are getting updates in each iterations of mini-batches which can be verified by looking at pre-activations and activations which get stabilized with consistency of accuracy and cross-entropy.

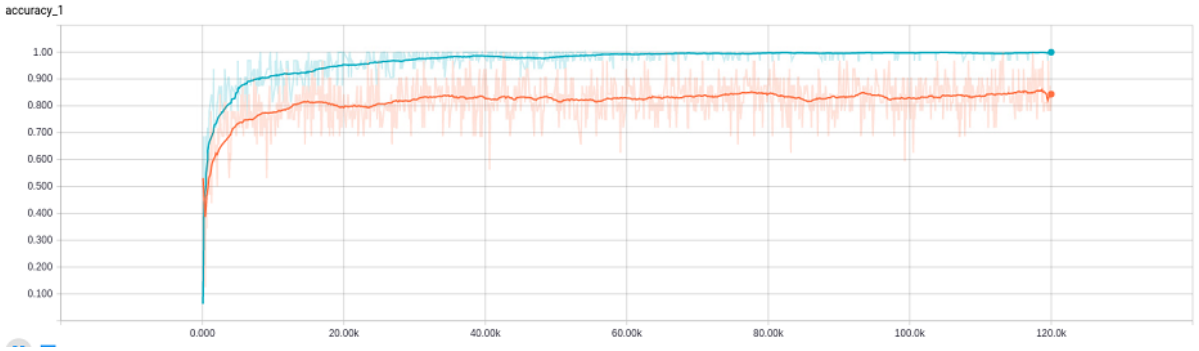


Figure 34. Accuracy of ModelNet10 Upper Layer Model

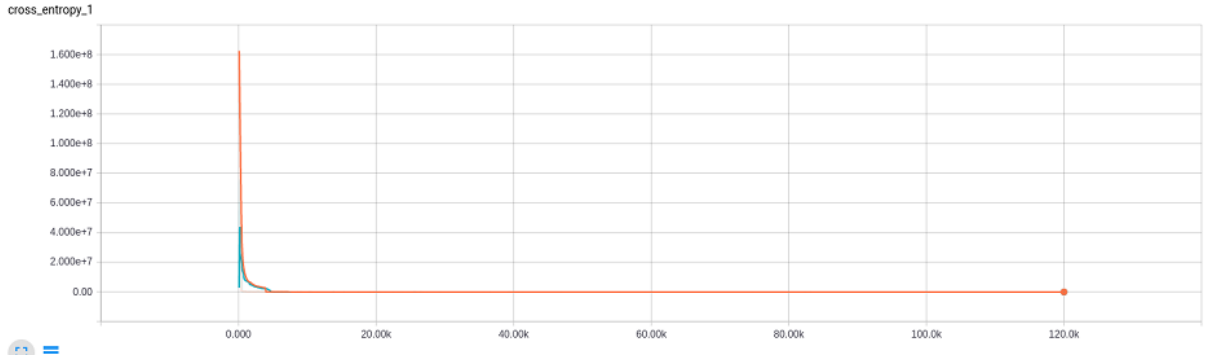


Figure 35. Cross Antropy of ModelNet10 Upper Layer Model

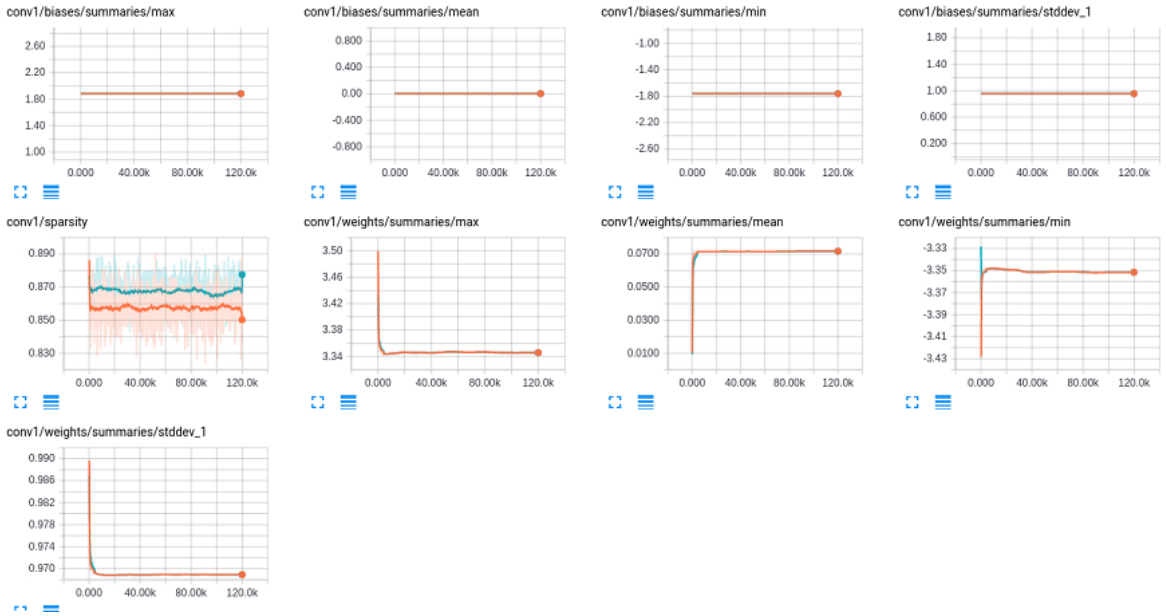


Figure 36. Conv1 Layer Summaries of ModelNet10 Upper Layer Model

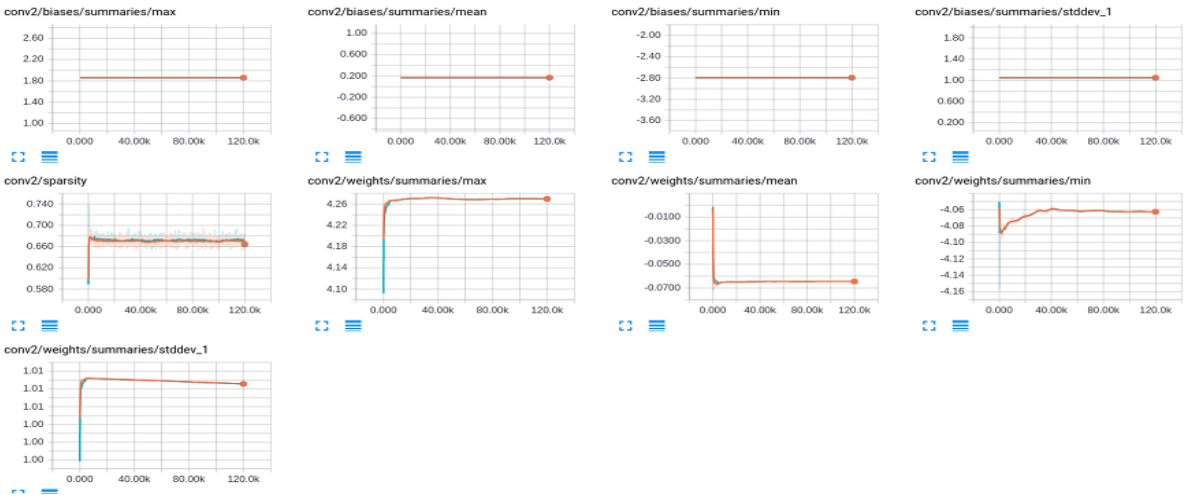


Figure 37. Conv2 Layer Summaries of ModelNet10 Specialized Learning for Top Layer Model

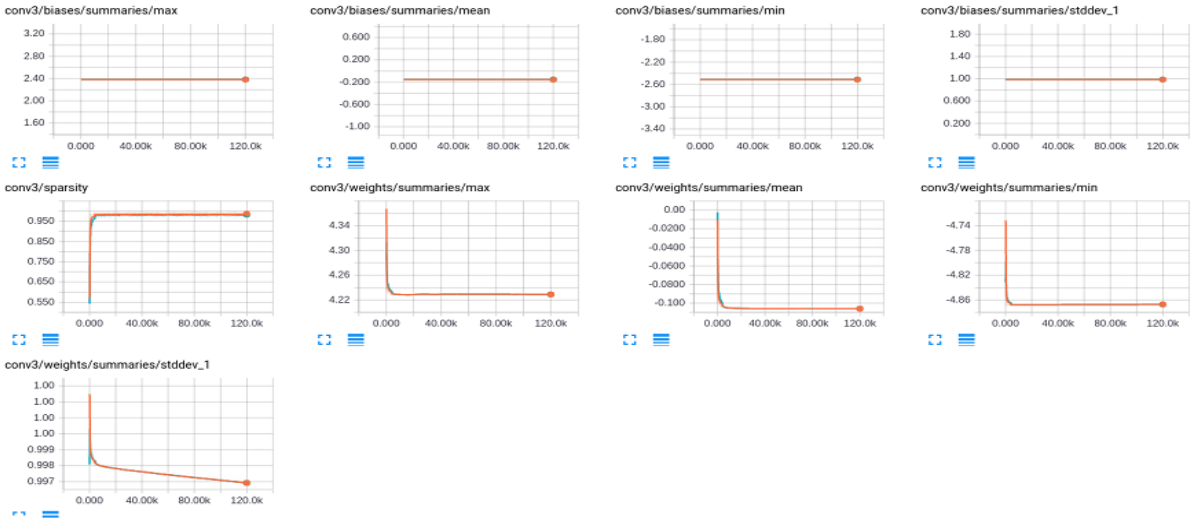


Figure 38. Conv3 Layer Summaries of ModelNet10 Top Layer Model

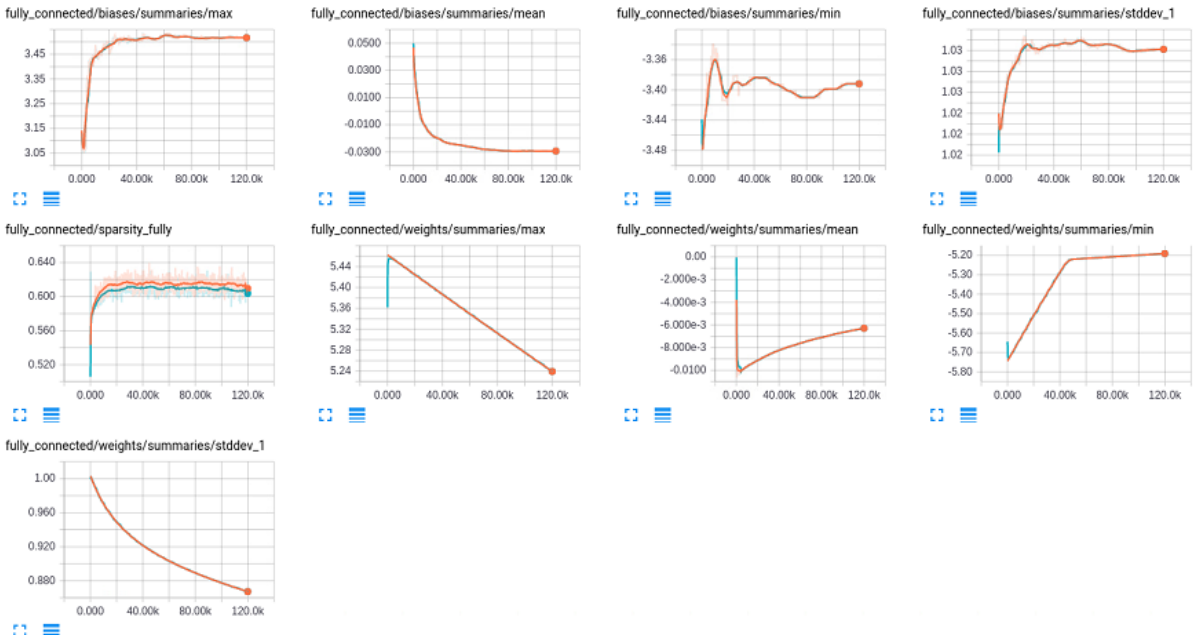


Figure 39. Fully Connected Layer Summaries of ModelNet10 Top Layer Model

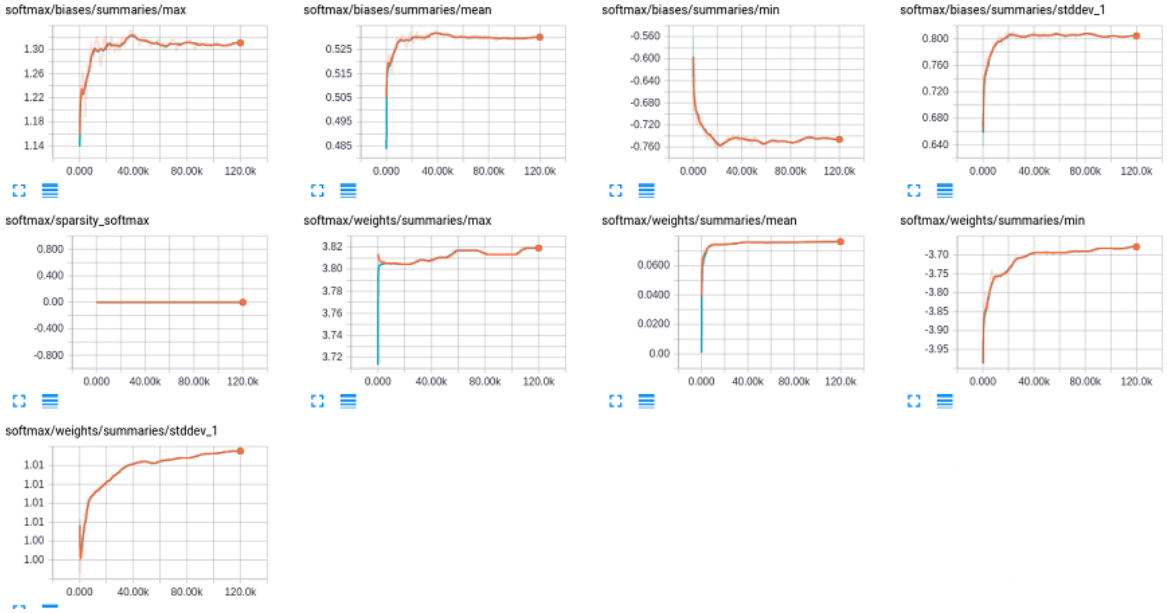


Figure 40. Softmax Layer Summaries of ModelNet10 Top Layer Model

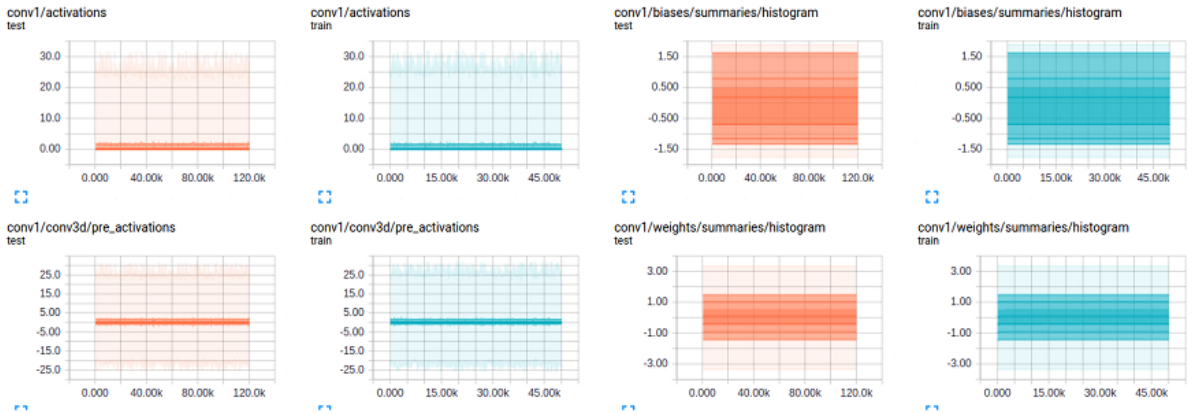


Figure 41. Conv1 Layer Activations of ModelNet10 Top Layer Model

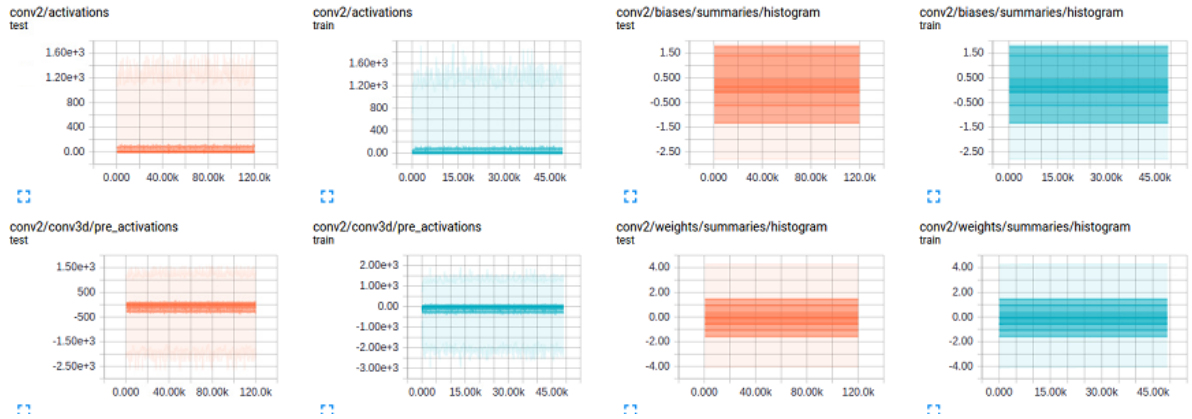


Figure 42. Conv2 Layer Activations of ModelNet10 Top Layer Model

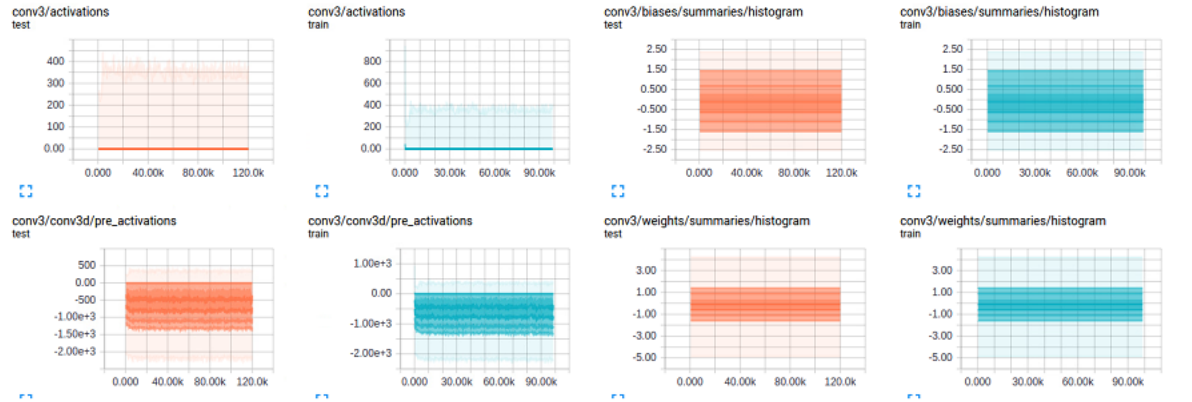


Figure 43. Conv3 Layer Activations of ModelNet10 Top Layer Model

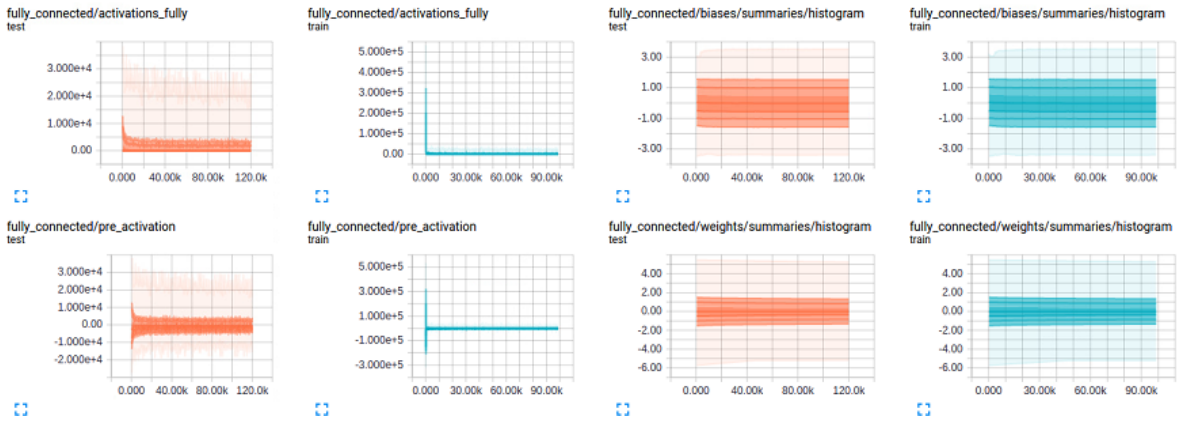


Figure 44. Fully Connected Layer Activations of ModelNet10 Top Layer Model

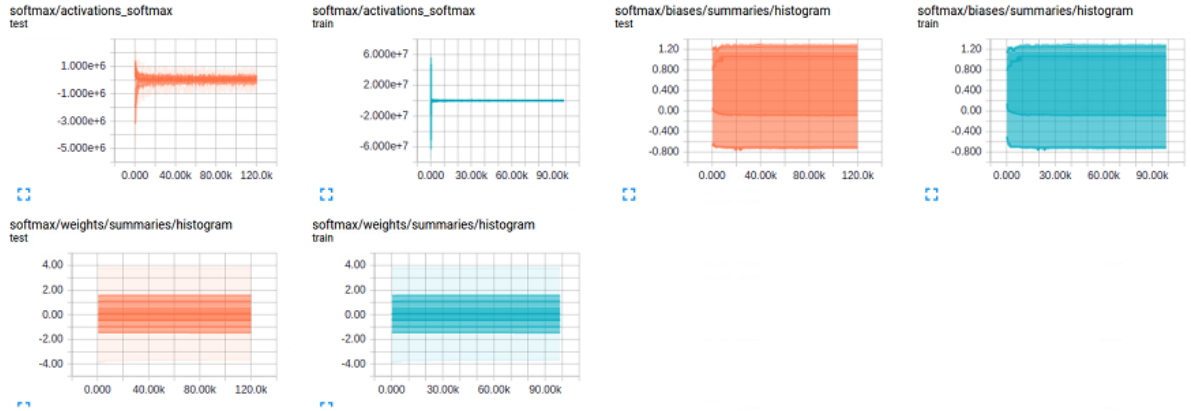


Figure 45. Softmax Layer Activations of ModelNet10 Top Layer Model

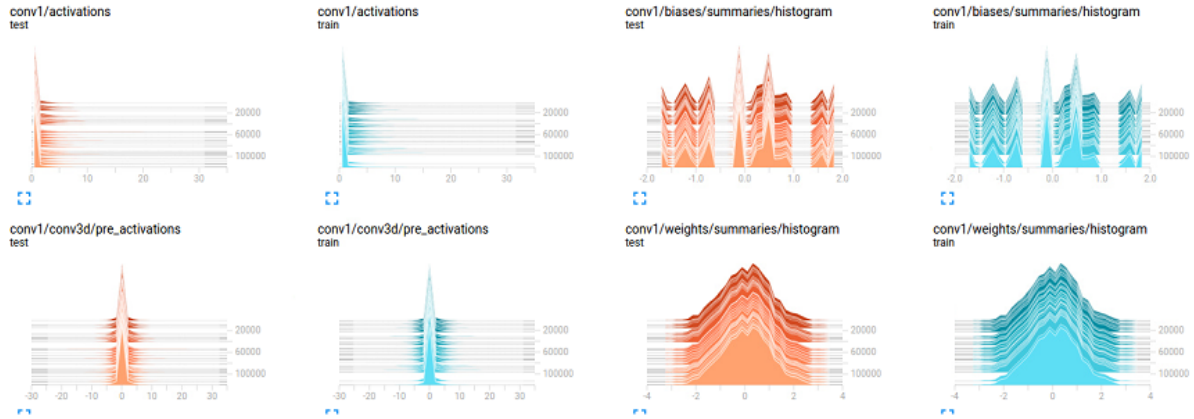


Figure 46. Conv1 Layer Histograms of ModelNet10 Top Layer Model

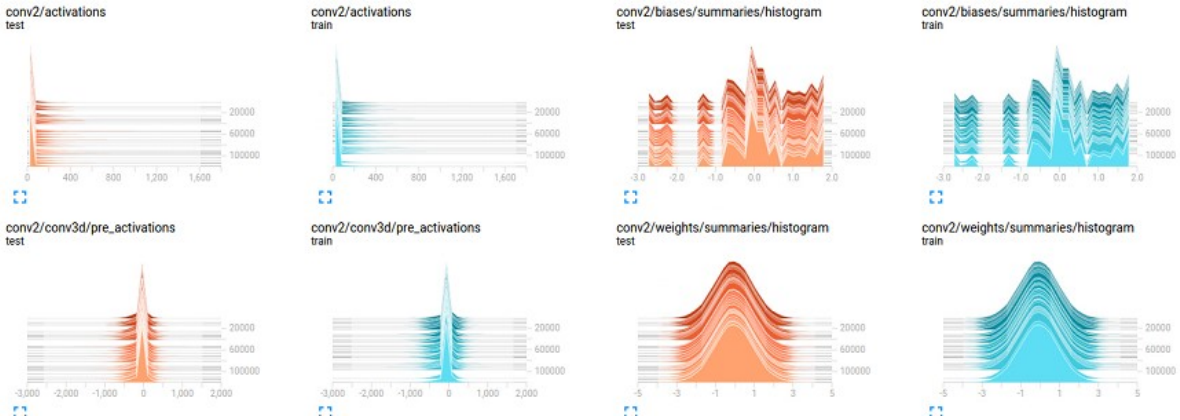


Figure 47. Conv2 Layer Histograms of ModelNet10 Top Layer Model

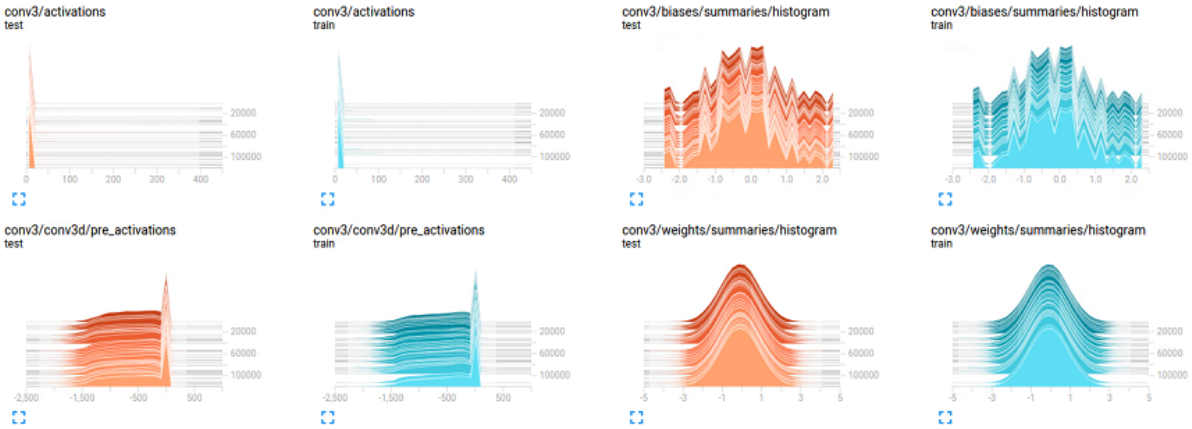


Figure 48. Conv3 Layer Histograms of ModelNet10 Top Layer Model

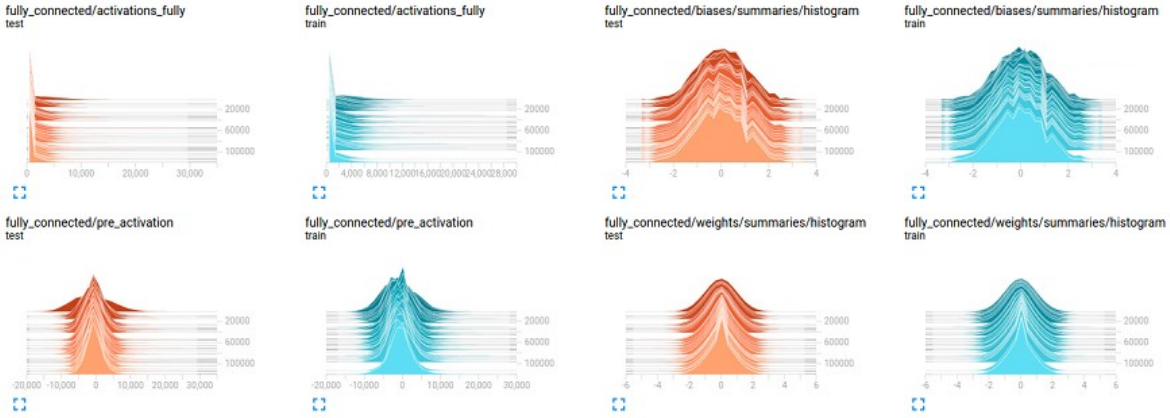


Figure 49. Fully Connected Layer Histograms of ModelNet10 Top Layer Model

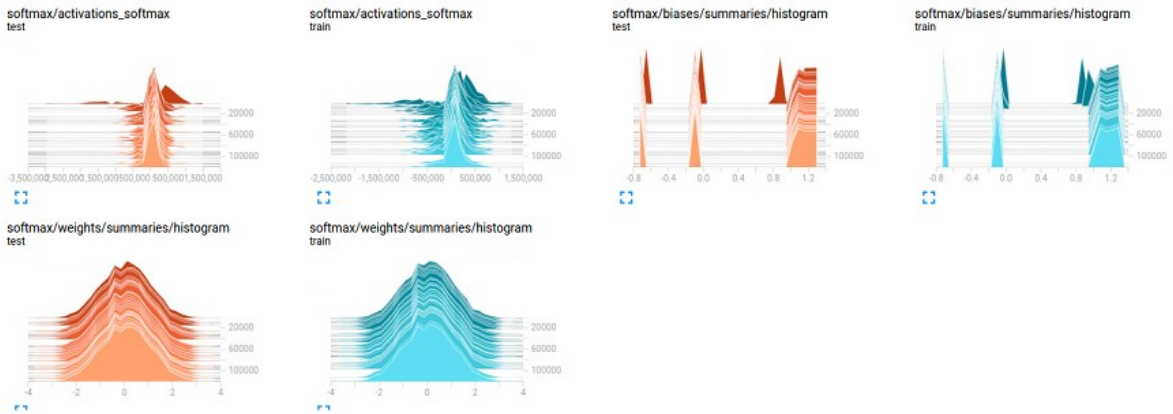


Figure 50. Softmax Layer Histograms of ModelNet10 Top Layer Model

After the redistribution of classes and moving the confused classes, the accuracy for each lower level subproblems and the higher level general model increase above the performance of state of art algorithms. The accuracy for all first level five classes are shown in Figure 51.

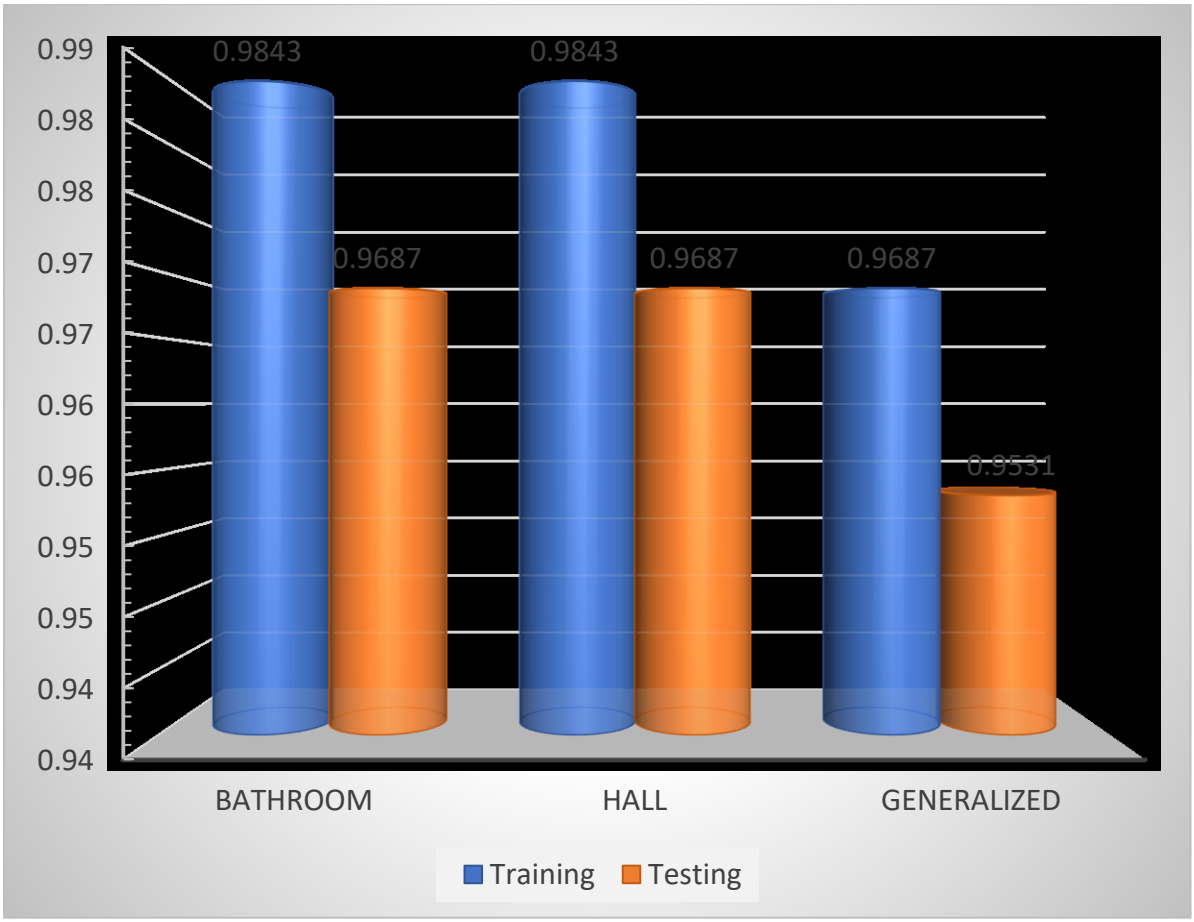


Figure 51. Accuracy for Subproblems of ModelNet10

Based on the final accuracy of the upper level and two subproblems at the lower level, Figure 52 shows the mathematically calculated accuracy for two lower level subproblems bathroom and hall and also for three separate classes bed, nightstand and desk at the upper level which is really good when compared with initial distribution accuracy in figure 30.

Subproblem	Training	Testing
Bathroom	95.34%	92.32%
Hall	95.34%	92.32%
Bed	96.87%	95.31%
Desk	96.87%	95.31%
Night Stand	96.87%	95.31%

Figure 52. Mathematical Accuracy for Subproblems and Individual Classes after Optimization

Prediction for ModelNet10 with k=1:

To evaluate the prediction accuracy for ModelNet10, we have tried to predict the labels for 5 objects per class. The procedure is basically top-down approach. The procedure includes giving the 3D input to the upper layer model and predict the labels at the upper layer. If predicted label at the upper layer is the label for class then that is the final label for class and if it is the label for another subproblem, load model for subproblem and predict the label. Follow this top-down procedure until final label for the class is predicted. Figure 53 shows the final confusion matrix for ModelNet10 50 objects. The accuracy in this case 86% for 50 objects.

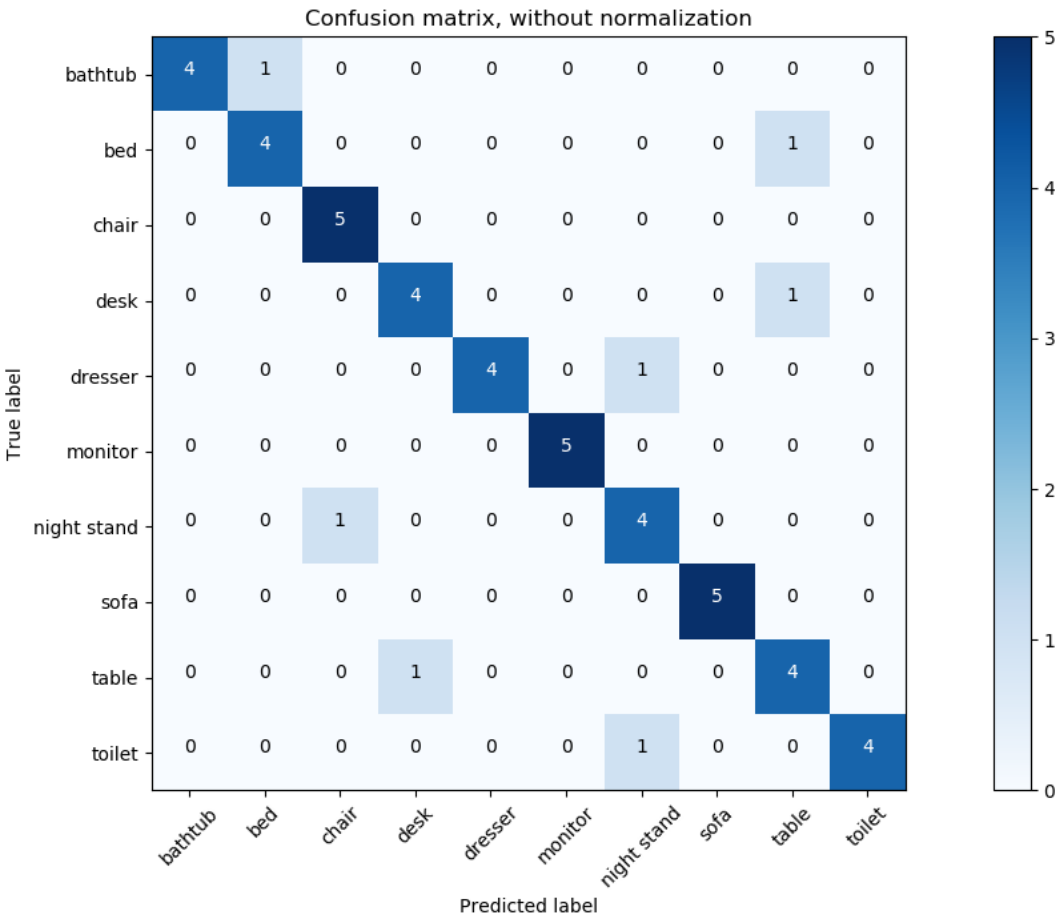


Figure 53. Confusion Matrix for ModelNet10 50 objects for k=1

Figure 54 shows the normalized confusion matrix which gives idea about the accuracy per classes.

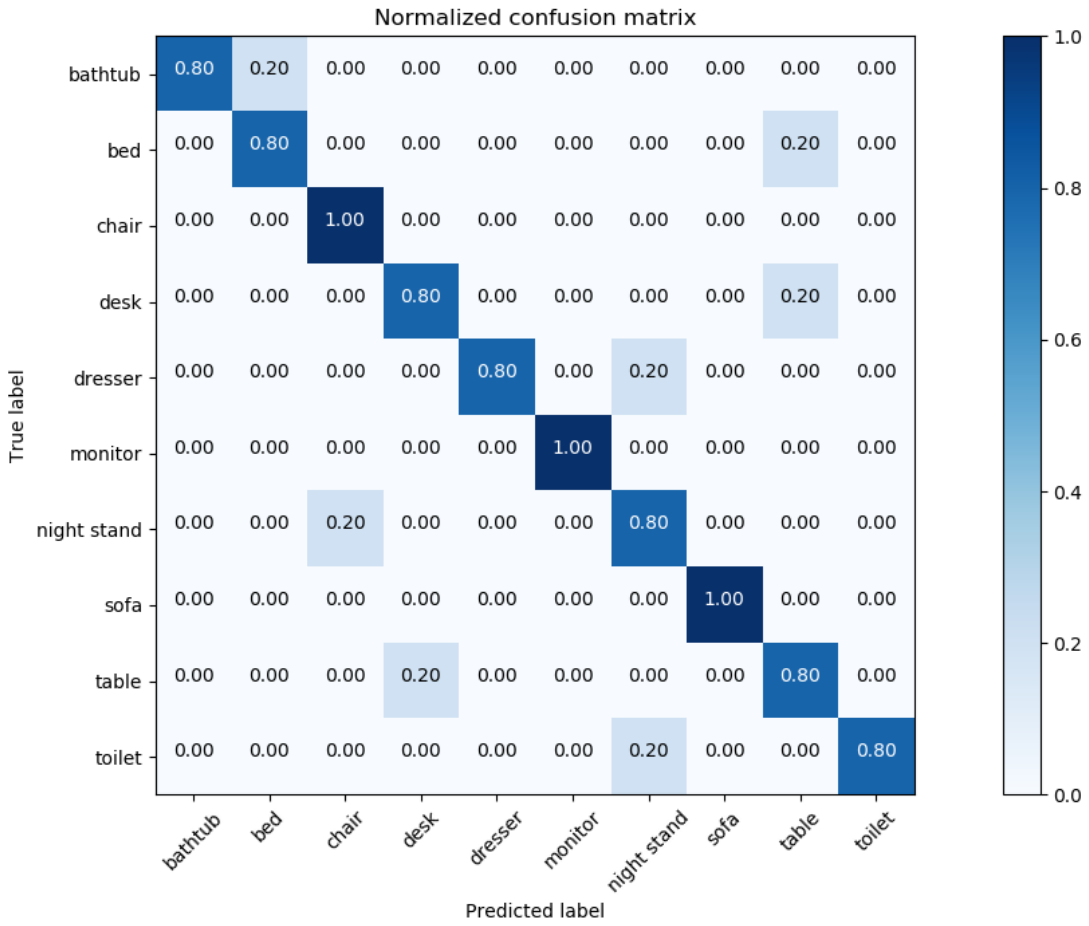


Figure 54. Normalized Confusion Matrix for ModelNet10 50 objects with k=1

From the confusion matrix, we can see that accuracy are affected by some of the classes and these classes are confused with each other at even lower level. The similarity between their shapes and spatial features justify this confusion between them. Figure 55 shows the confused classes which give the idea of their shapes and features.

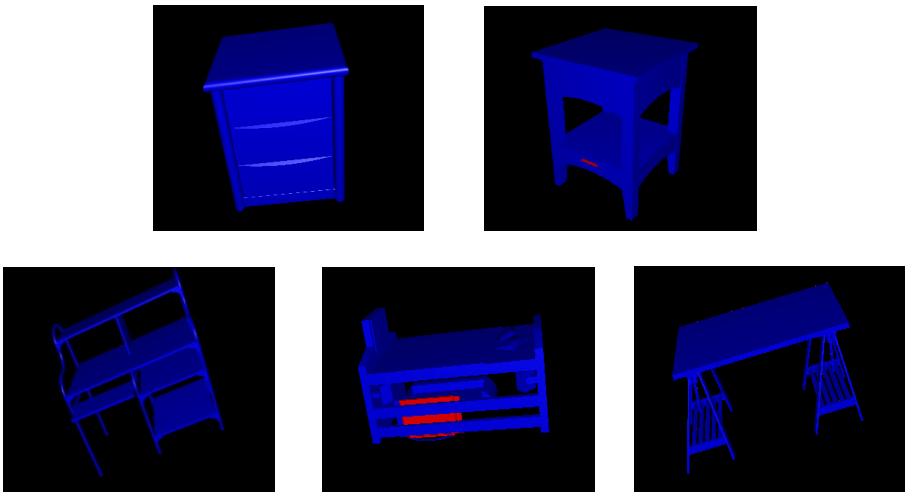


Figure 55. Confused Classes (Dresser & Night Stand, Bed – Desk and Table)

From Figure 56, we can see that desk and nightstand has almost similar shapes and that is the reason why they are confused. It is even possible that human may confuse about these classes, so it makes sense to predict the range of labels instead of a single label. We also try to check performance by predicting the two labels at the higher level.

Prediction for ModelNet10 with k=2:

The fuzzy parameter k =2 means the predicting 2 labels at first level. The prediction strategy is same as the k=1 but instead of taking label with the highest probability, takes 2 labels with first and second highest priority then follow the same top-down approach for prediction. Figure 56 shows the confusion matrix for ModelNet10 50 objects with k=2. The accuracy, in this case, is 94% for 50 objects.

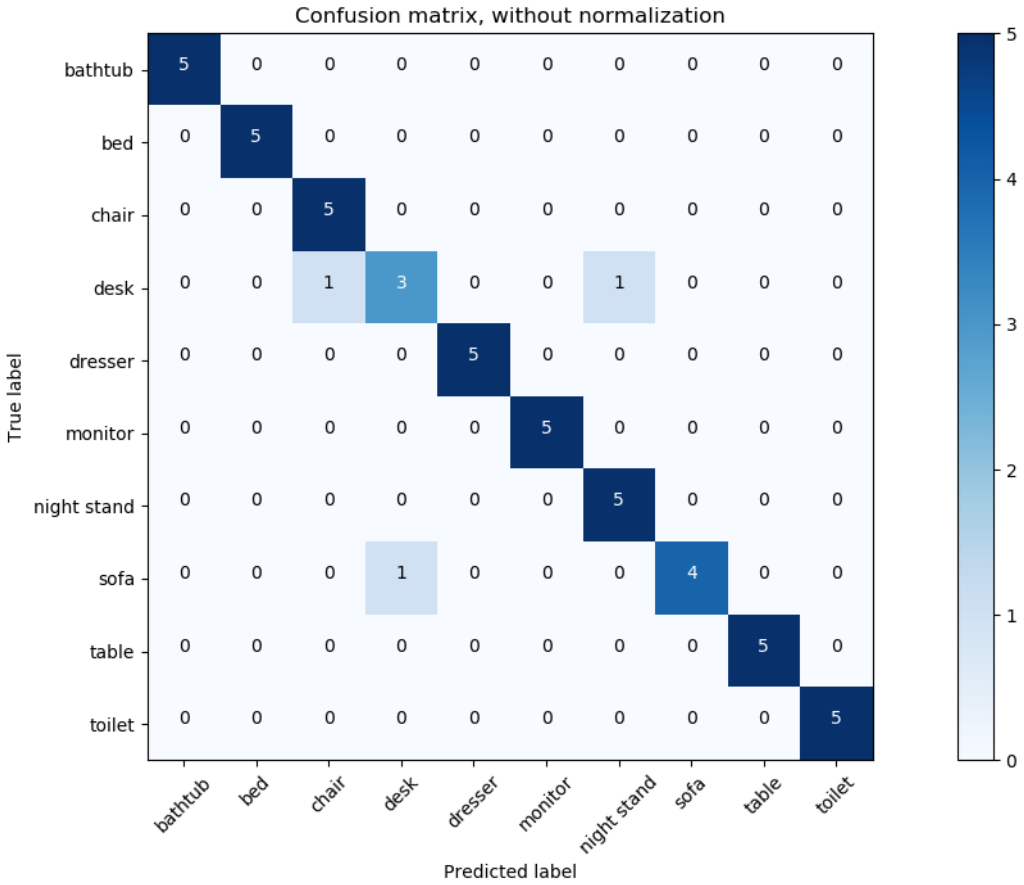


Figure 56. Confusion Matrix for ModelNet10 50 objects for k=2

Figure 57 shows the normalized confusion matrix which gives idea about the accuracy per classes.

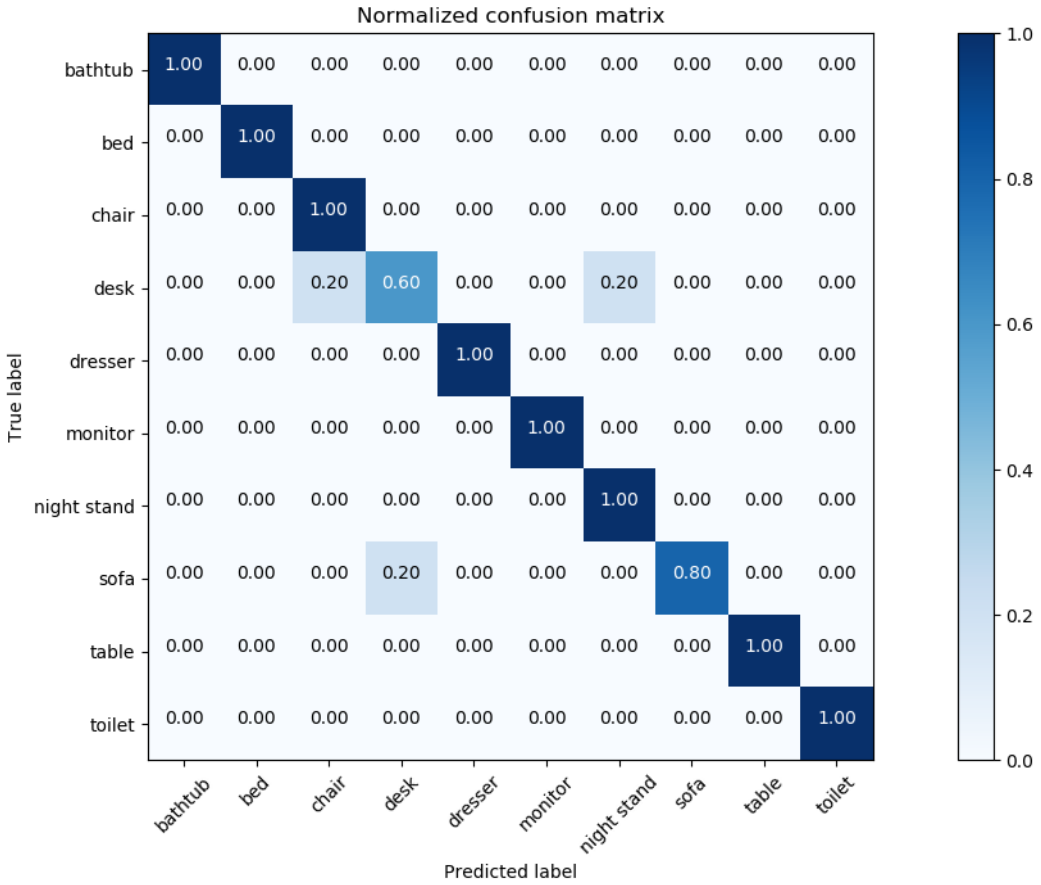


Figure 57. Normalized Confusion Matrix for ModelNet10 50 objects k=2

From the confusion matrix, we can say that by predicting 2 labels at higher level eliminate few similarities between the shapes and spatial features, thus predict the correct labels for some confused classes. Figure 58 shows the prediction time for k=1 and k=2 for the cases of predicting label at first level and predicting label at the second level. The time is in the milliseconds.

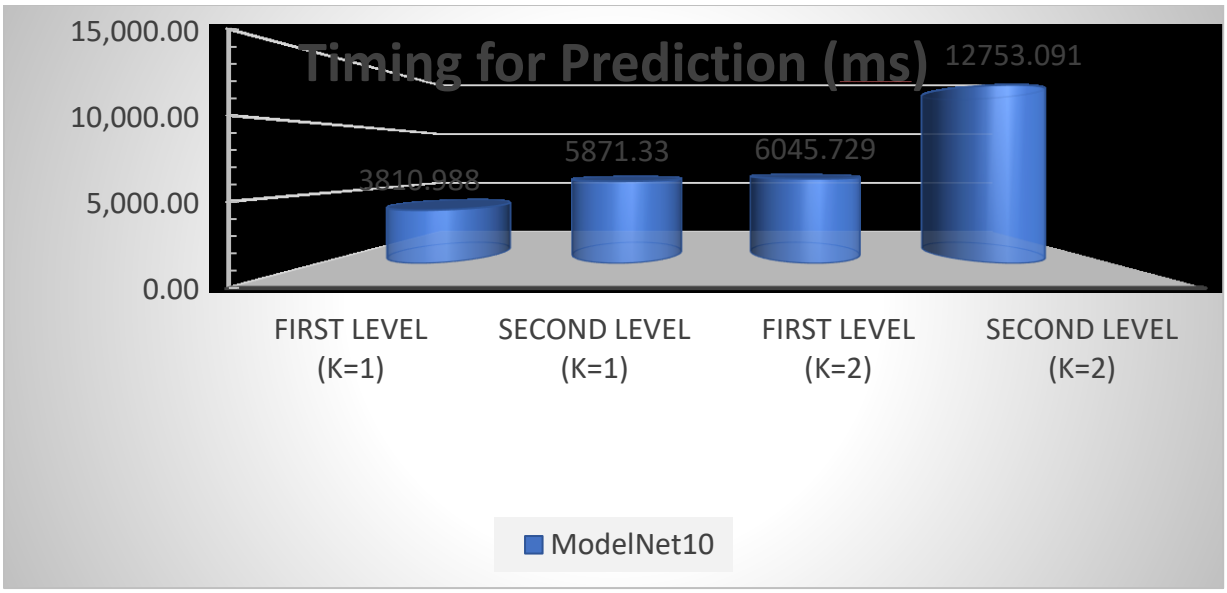


Figure 58. Prediction Time for ModelNet10 with k=1 and k=2

From the Figure 58, we can say that as we increase the layers and value of k, prediction time also increased. For k=1, if labels are predicted at first layer time is 3.8s while if it is predicted at second layer time is 5.871s. For k=2, if labels are predicted at first layer time is 6.045s while if it is predicted at second layer time is 12.753s.

4.6.2 ModelNet40 Performance evaluation

In this section, we are evaluating our approach using the ModelNet40 dataset. Figure 59 shows the statistics about the ModelNet40 classes in terms of number of object per class.

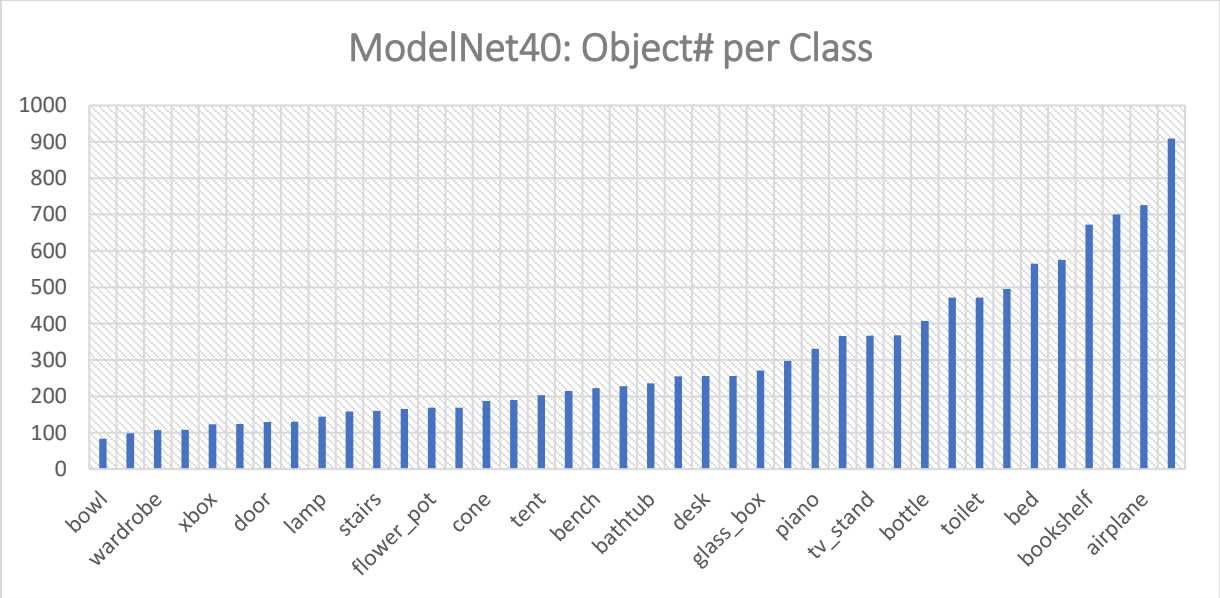


Figure 59. Number of Object per Class

In this case, the classes are divided into subproblems using ImageNet image hierarchy. Initially, classes are divided into the seven subproblems using ImageNet image hierarchy. The name of subproblems is bathroom, bedroom, electronics, external, hall, kitchen, and others. Figure 60 shows the initial class distribution per subproblem.

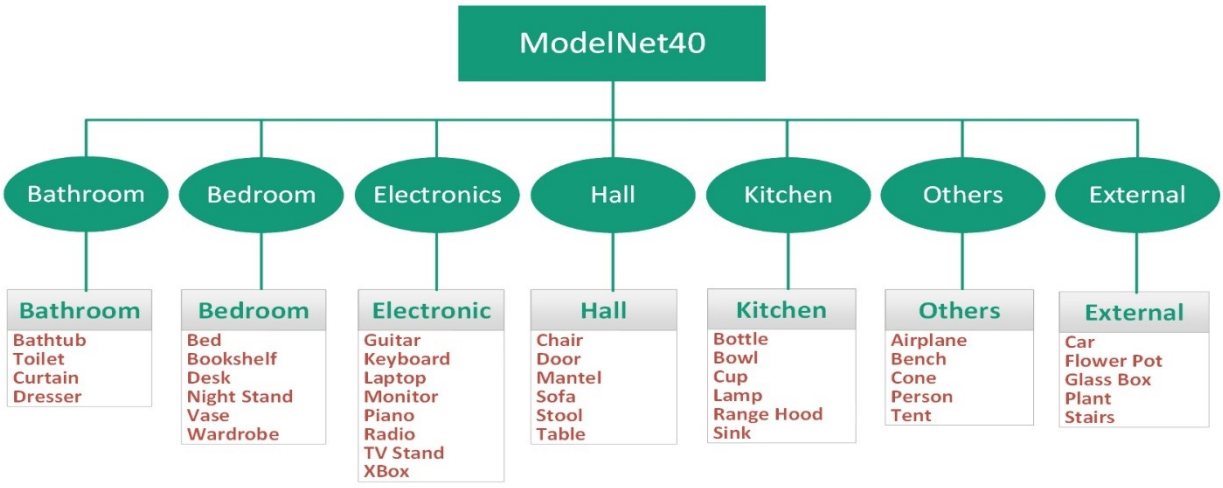


Figure 60. Initial Class Distribution per Subproblem in ModelNet40

As the ModelNet10 initial distribution, this distribution was also not giving the expected performance so there was a need for optimization. Figure 61 shows the class hierarchy after the redistribution and optimization.

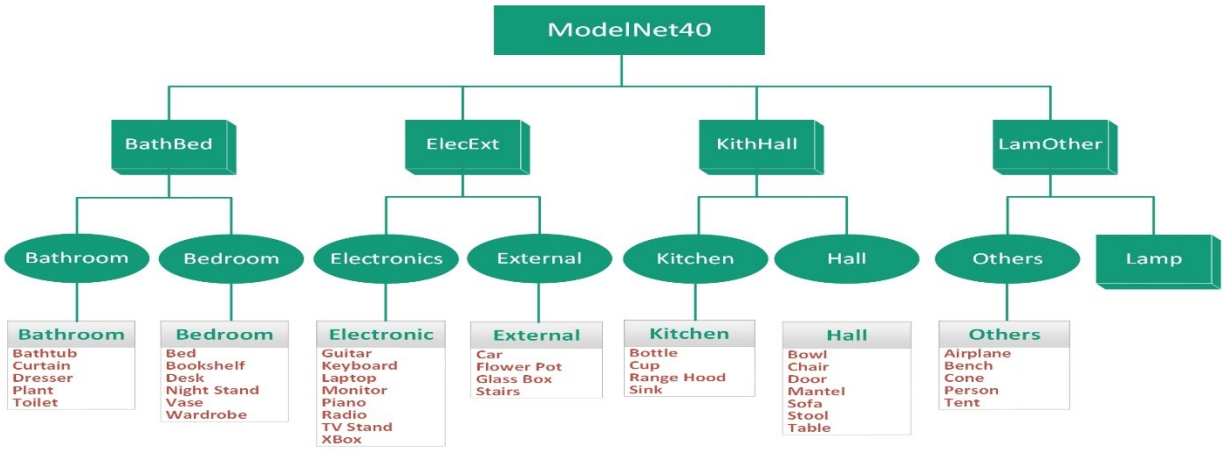


Figure 61. Class hierarchy after redistribution and optimization for ModelNet40

In the process of optimization, we have added one more layer of subproblems which classify the two subproblems at third level. The lamp class is also moved to upper level as it was confusing with all the classes in lower level subproblems. The name of lower level subproblems are same as the initial distribution but some classes are redistributed. At second level, 4 subproblems are added with name bathbed, elecext, kithhall, and lampother. These names represent the subproblems it classifies at the third level such as bathbed subproblems classify between bathroom and bedroom and similarly for others.

CNN Model for ModelNet40 Bathroom subproblems:

The bathroom subproblem is last layer subproblem which classifies the bathtub, curtain, dresser, plant and toilet classes. Figure 62 and Figure 63 shows the accuracy and losses throughout the training and testing of subproblem.

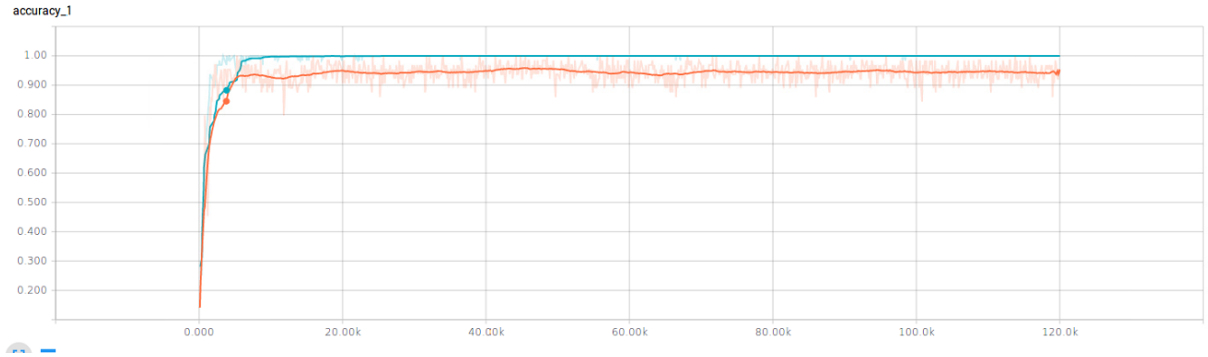


Figure 62. Accuracy of ModelNet40 Bathroom Subproblem for Conquer Learning

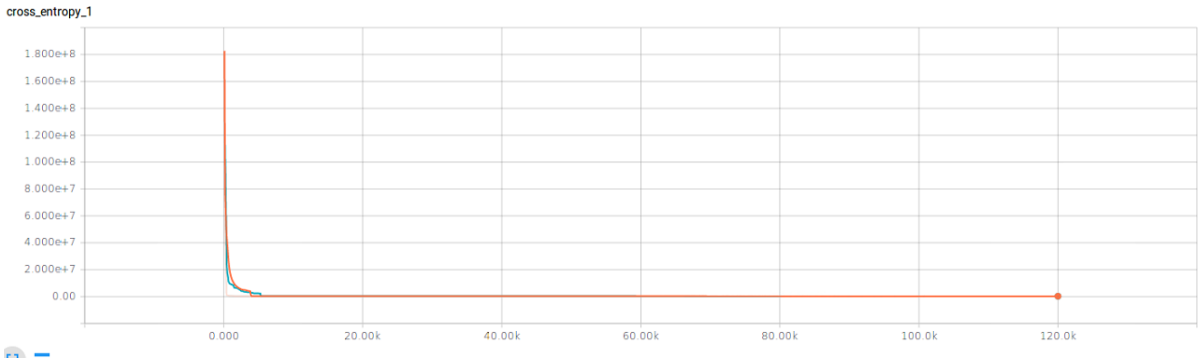


Figure 63. Antropy of ModelNet40 Bathroom Subproblem for Conquer Learning

CNN Model for ModelNet40 Bedroom Subproblem:

The bedroom subproblem is last layer subproblem which classifies the bed, bookshelf, nightstand, desk, vase and wardrobe classes. Figure 64 and Figure 65 shows the accuracy and losses throughout the training and testing of subproblem.

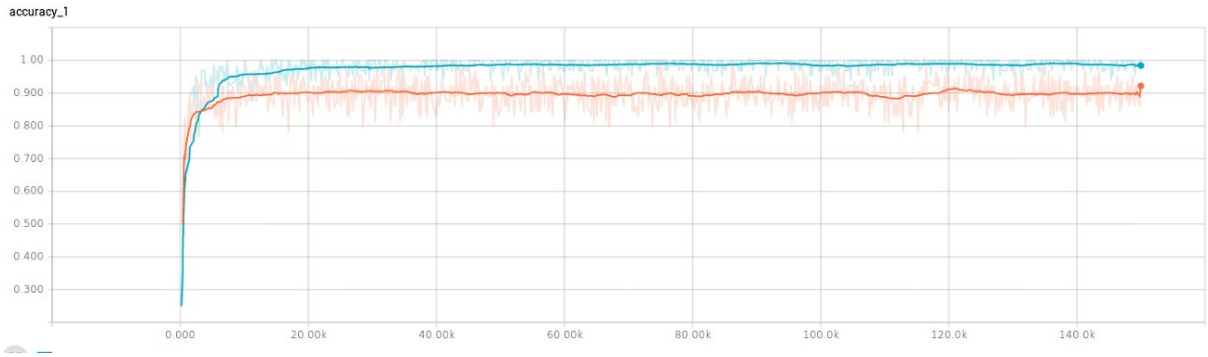


Figure 64. Accuracy of ModelNet40 Bedroom Subproblem for Conquer Learning

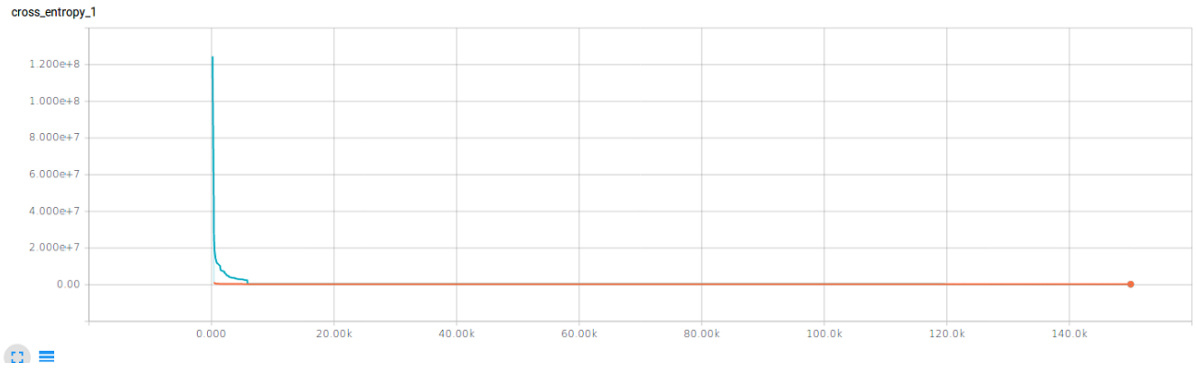


Figure 65. Antropy of ModelNet40 Bedroom Subproblem for Conquer Learning

CNN Model for ModelNet40 Electronics Subproblem:

The electronics subproblem is last layer subproblem which classifies the guitar, keyboard, laptop, monitor, piano, radio, tv stand and Xbox classes. Figure 66 and Figure 67 shows the accuracy and losses throughout the training and testing of subproblem.

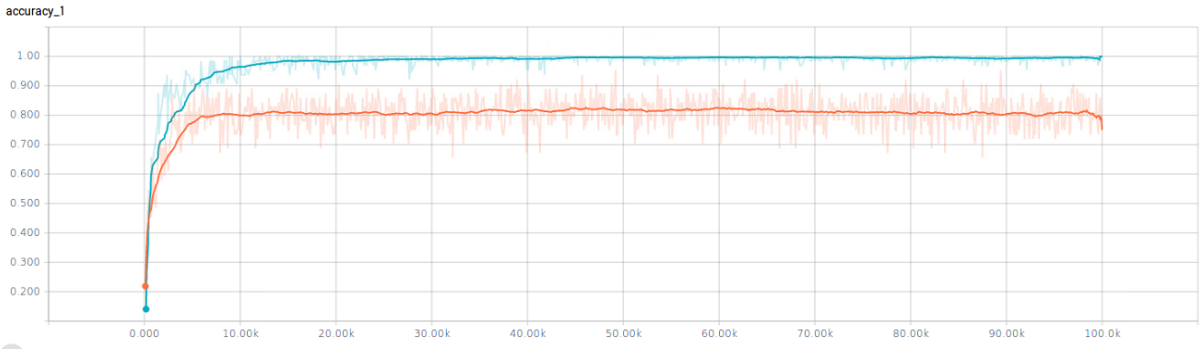


Figure 66. Accuracy of ModelNet40 Electronics Subproblem for Conquer Learning

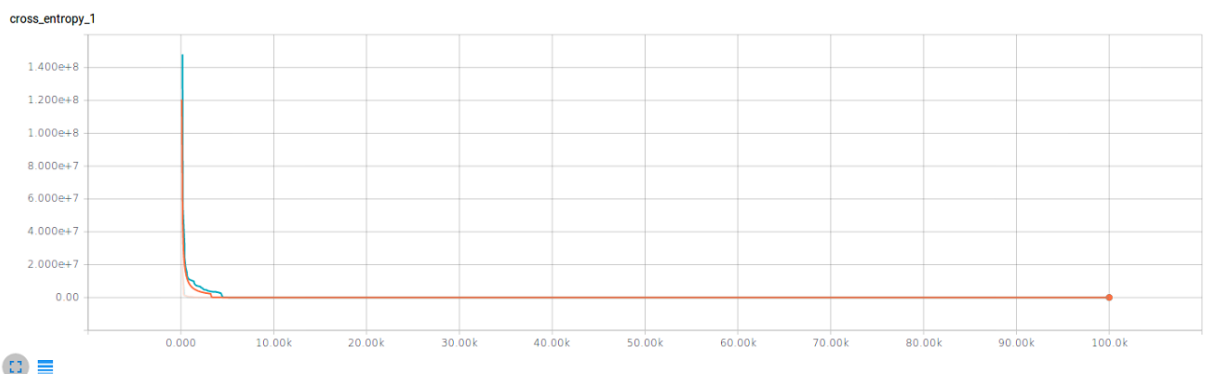


Figure 67. Antropy of ModelNet40 Electronics Subproblem for Conquer Learning

CNN Model for ModelNet40 External Subproblem:

The external subproblem is last layer subproblem which classifies the car, flower pot, glass box and stairs classes. Figure 68 and Figure 69 shows the accuracy and losses throughout the training and testing of subproblem.

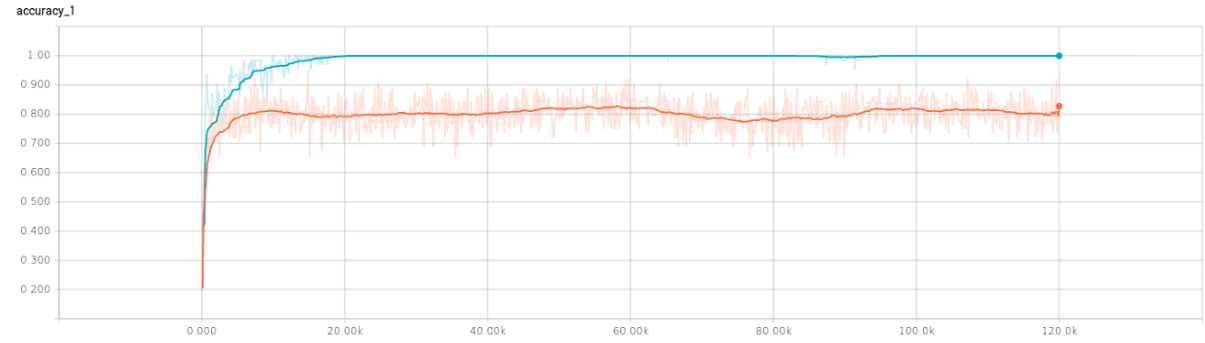


Figure 68. Accuracy of ModelNet40 External Subproblem for Conquer Learning

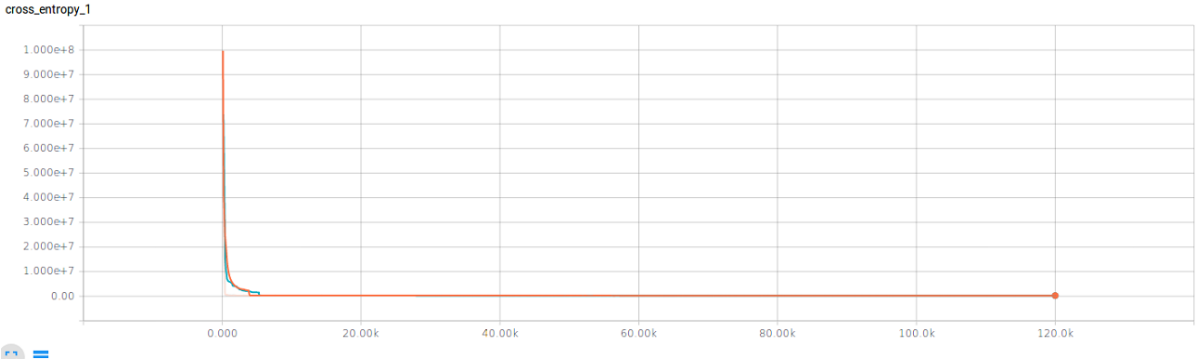


Figure 69. Antropy of ModelNet40 External Subproblem for Conquer Learning

CNN Model for ModelNet40 Kitchen Subproblem:

The kitchen subproblem is last layer subproblem which classifies the bottle, cup, range hood and sink classes. Figure 70 and Figure 71 shows the accuracy and losses throughout the training and testing of subproblem.

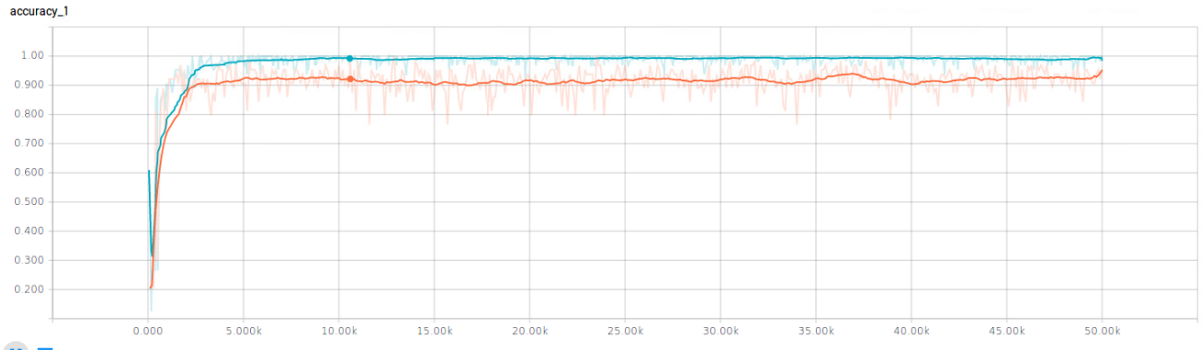


Figure 70. Accuracy of ModelNet40 Kitchen Subproblem for Conquer Learning

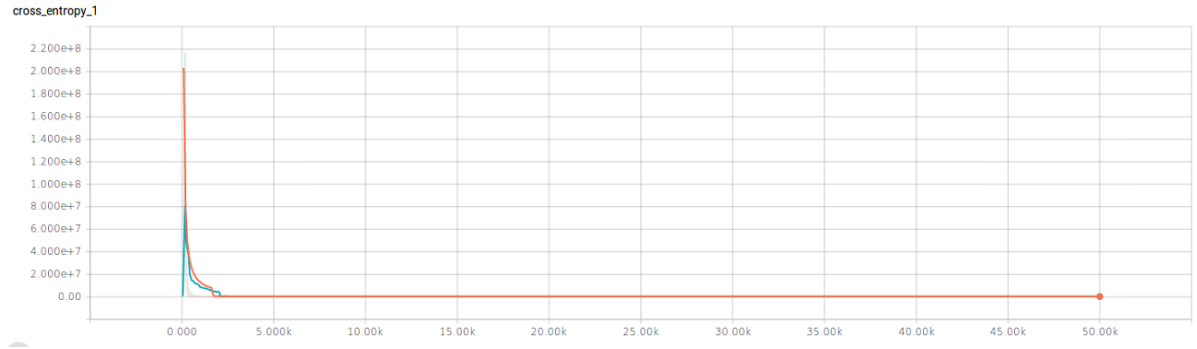


Figure 71. Antropy of ModelNet40 Kitchen Subproblem for Conquer Learning

CNN Model for ModelNet40 Hall Subproblem:

The hall subproblem is last layer subproblem which classify the bowl, chair, mantel, door, sofa, stool and table classes. Figure 72 and Figure 73 shows the accuracy and losses throughout the training and testing of subproblem.

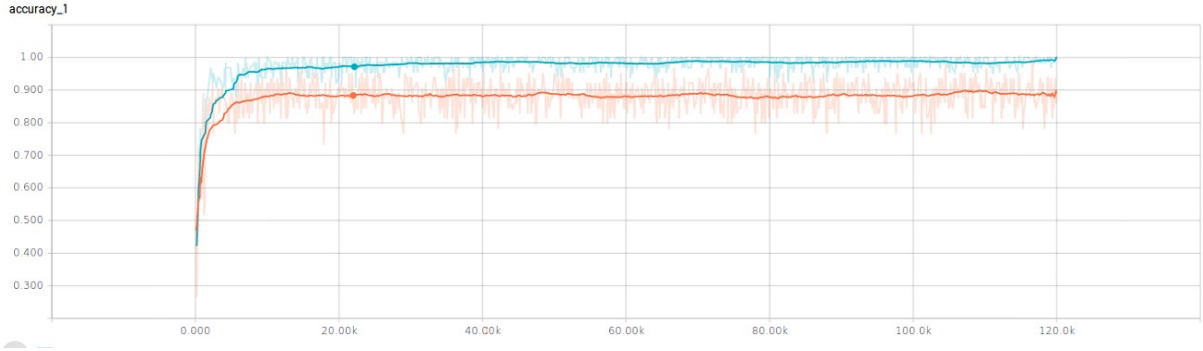


Figure 72. Accuracy of ModelNet40 Hall Subproblem for Conquer Learning

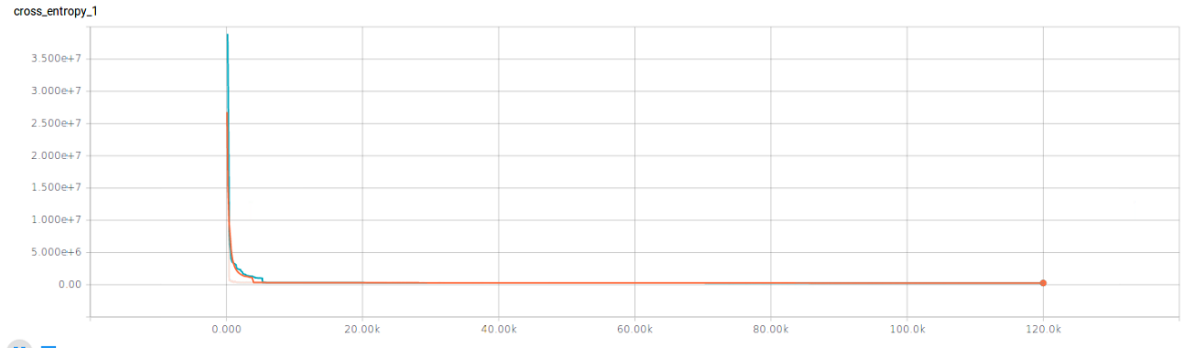


Figure 73. Antropy of ModelNet40 Hall Subproblem for Conquer Learning

CNN Model for ModelNet40 Others Subproblem:

The others subproblem is last layer subproblem which classifies the airplane, bench, cone, person, and tent classes. Figure 74 and Figure 75 shows the accuracy and losses throughout the training and testing of subproblem.

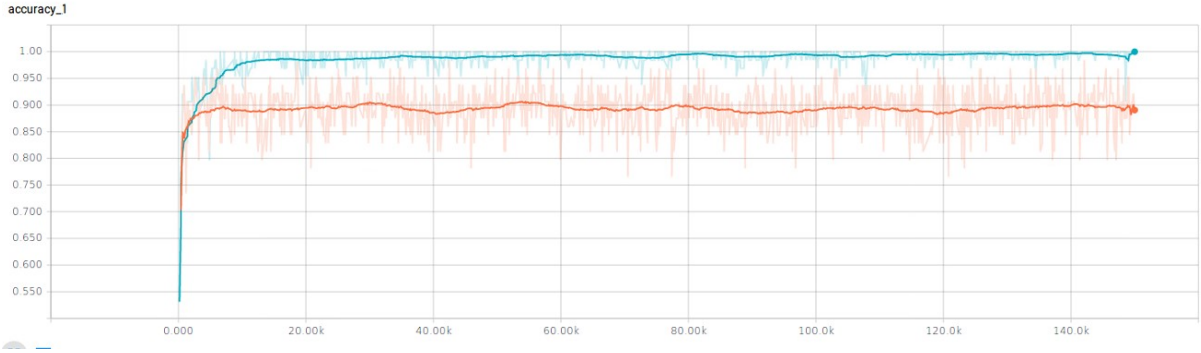


Figure 74. Accuracy of ModelNet40 Others Subproblem for Conquer Learning

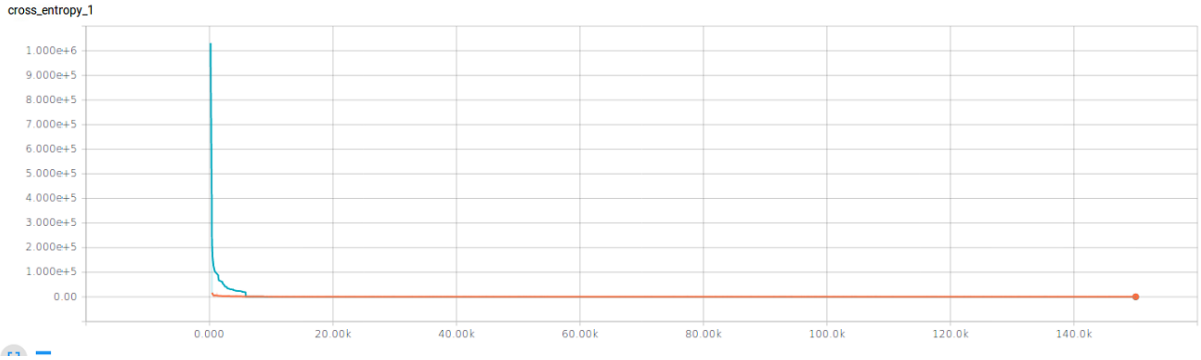


Figure 75. Antropy of ModelNet40 Others Subproblem for Conquer Learning

CNN Model for ModelNet40 Bathbed Subproblem:

The bathbed subproblem is second layer subproblem which classifies the bathroom and bedroom subproblems. Figure 76 and Figure 77 shows the accuracy and losses throughout the training and testing of subproblem.

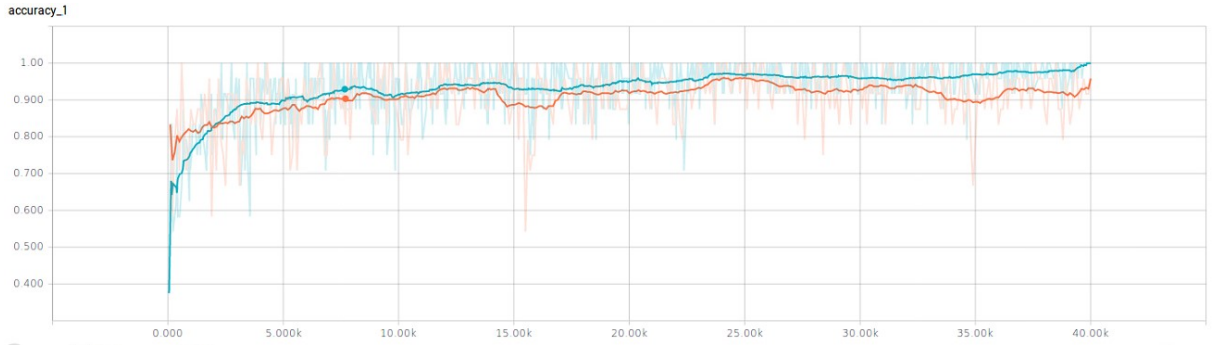


Figure 76. Accuracy of ModelNet40 Bathbed Subproblem for Conquer Learning

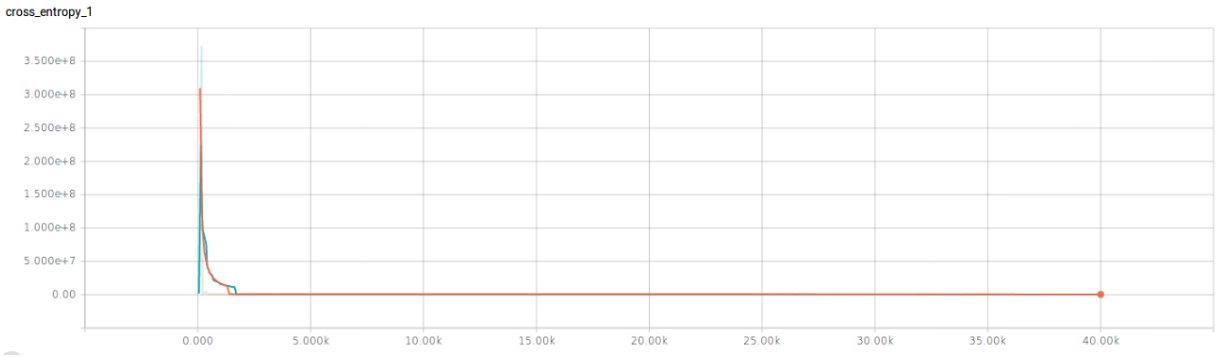


Figure 77. Antropy of ModelNet40 bathbed Subproblem for Conquer Learning

CNN Model for ModelNet40 ElecExt Subproblem:

The elecext subproblem is second layer subproblem which classifies the electronics and external subproblems. Figure 78 and Figure 79 shows the accuracy and losses throughout the training and testing of subproblem.

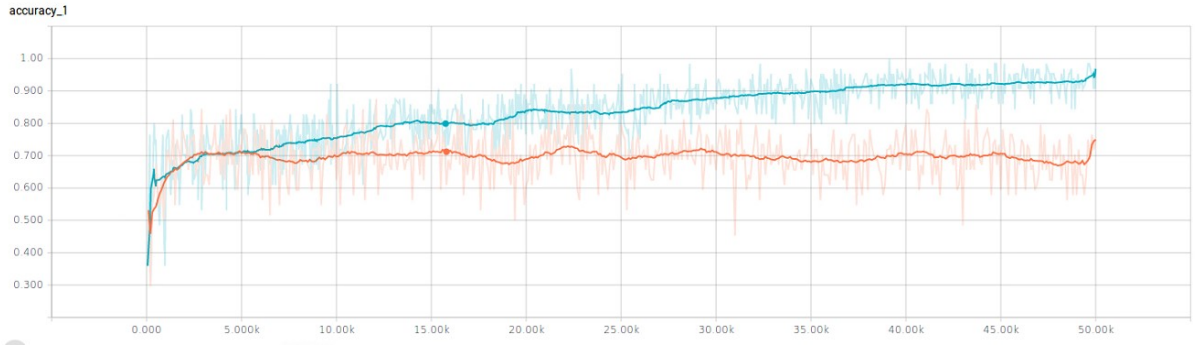


Figure 78. Accuracy of ModelNet40 ElecExt Subproblem for Conquer Learning

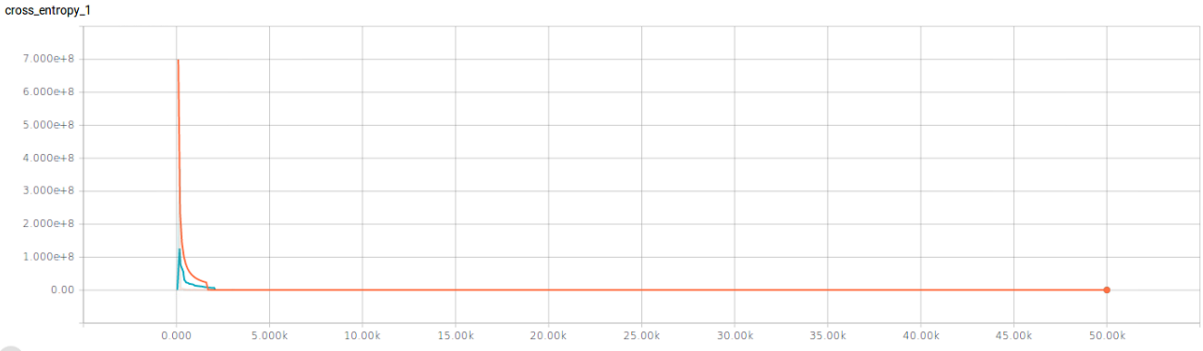


Figure 79. Antropy of ModelNet40 ElecExt Subproblem for Conquer Learning

CNN Model for ModelNet40 HallKith Subproblem:

The hallkith subproblem is second layer subproblem which classifies hall and kitchen subproblems. Figure 80 and Figure 81 shows the accuracy and losses throughout the training and testing of subproblem.

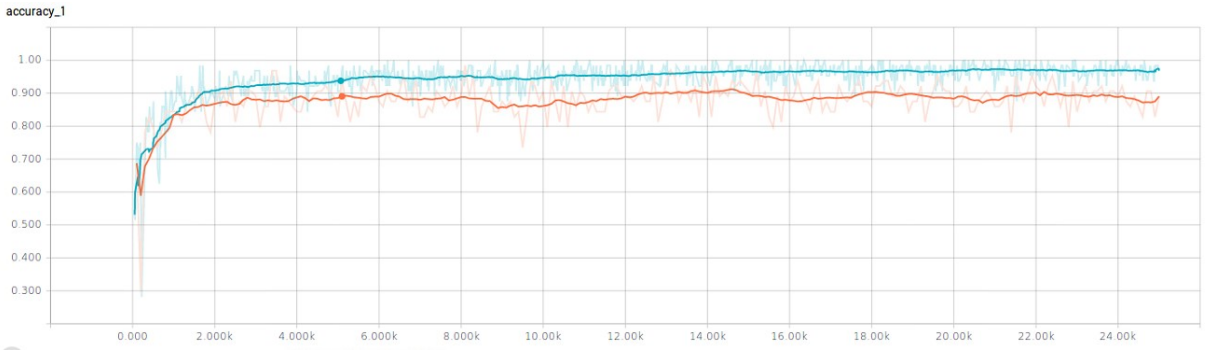


Figure 80. Accuracy of ModelNet40 HallKith Subproblem for Conquer Learning

Figure 81. Antropy of ModelNet40 HallKith Subproblem for Conquer Learning

CNN Model for ModelNet40 LampOthers Subproblem:

The LampOthers subproblem is second layer subproblem which classifies the others subproblem and the lamp class. Figure 82 and Figure 83 shows the accuracy and losses throughout the training and testing of subproblem.

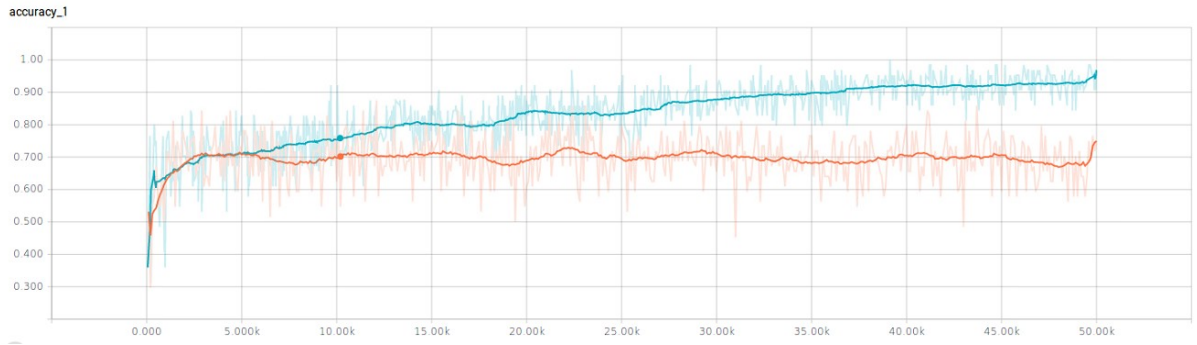


Figure 82. Accuracy of ModelNet40 LampOther Subproblem for Conquer Learning

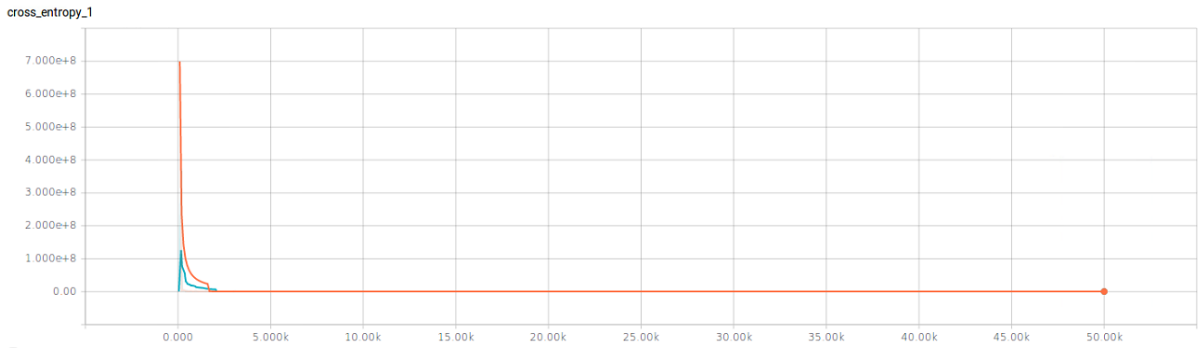


Figure 83. Antropy of ModelNet40 LampOther Subproblem for Conquer Learning

CNN Model for ModelNet40 Top Layer Subproblem:

The top layer subproblem classifies the bathbed, elecext, hallkith and lampothers subproblems which are the second layer subproblems. There are the operations for the different layers, gradient update, accuracy calculation. Figure 84 to Figure 101 below shows the graph for accuracy, cross-entropy, summaries, activations, and histograms for all the layers in the model. The weights and parameters are getting updates in each iterations of mini-batches which can be verified by looking at pre-activations and activations which get stabilized with consistency of accuracy and cross-entropy.

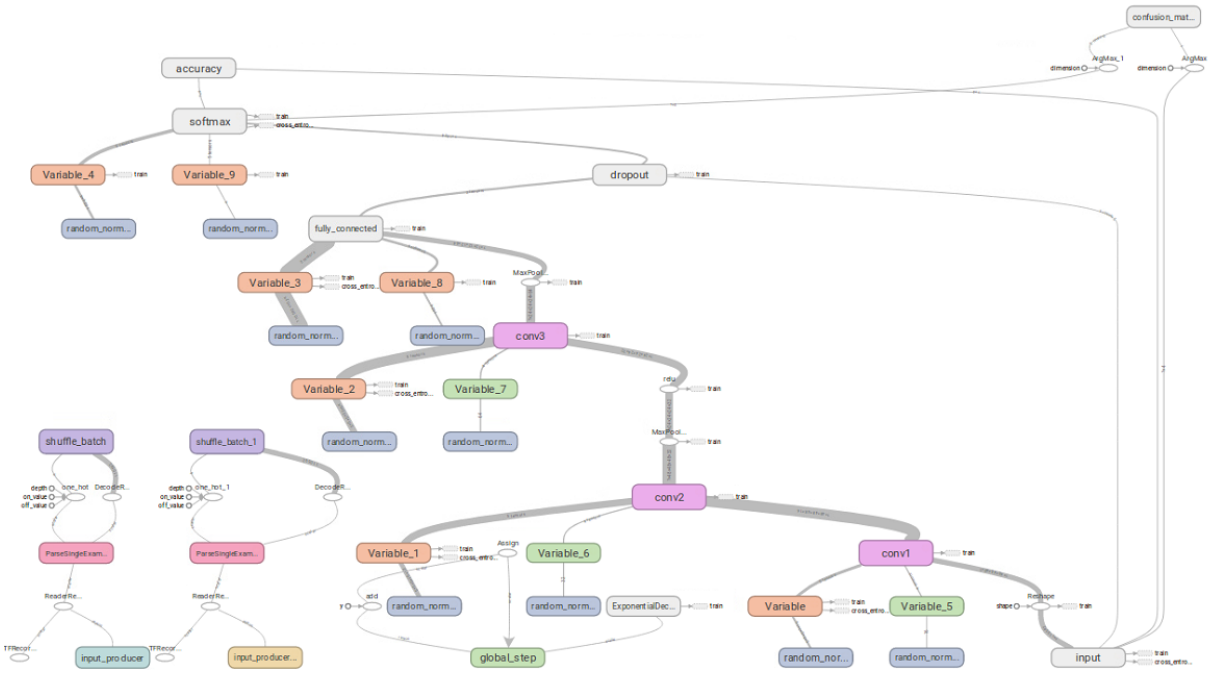


Figure 84.Computation Graph of ModelNet40 Top Layer Subproblem for Conquer Learning

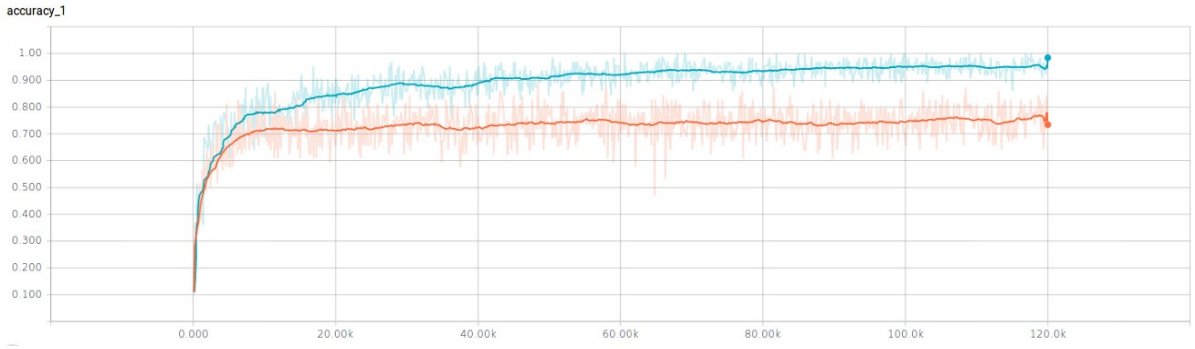


Figure 85. Accuracy of ModelNet40 Top Layer Subproblem for Conquer Learning

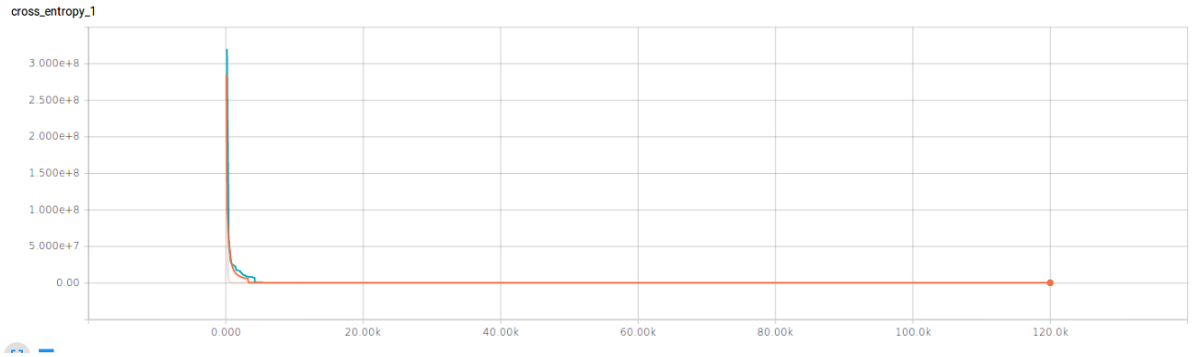


Figure 86. Antropy of ModelNet40 Top Layer Subproblem for Conquer Learning

Figure 87. Conv1 Summaries of ModelNet40 Top Layer Subproblem for Conquer Learning

Figure 88. Conv1 Activations of ModelNet40 Top Layer Subproblem for Conquer Learning

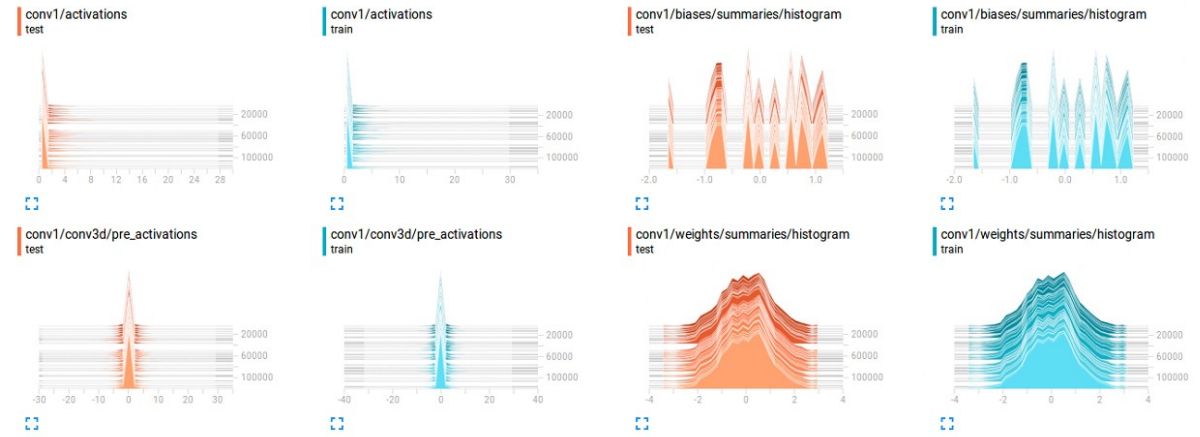


Figure 89. Conv1 Histograms of ModelNet40 Top Layer Subproblem for Conquer Learning

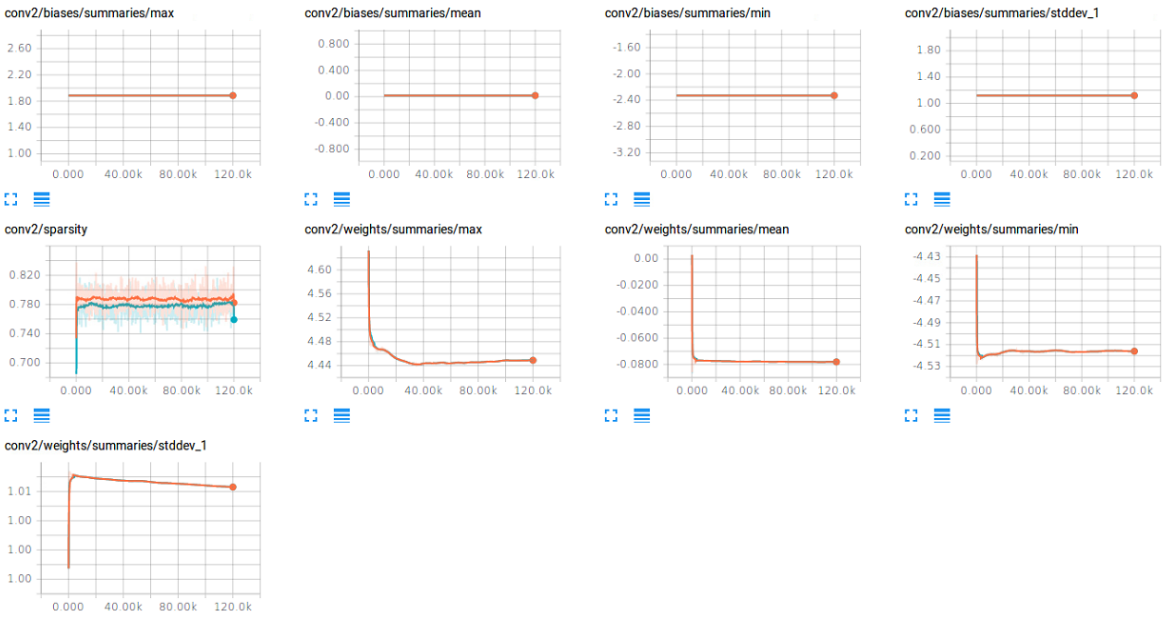


Figure 90. Conv2 Summaries of ModelNet40 Top Layer Subproblem for Conquer Learning

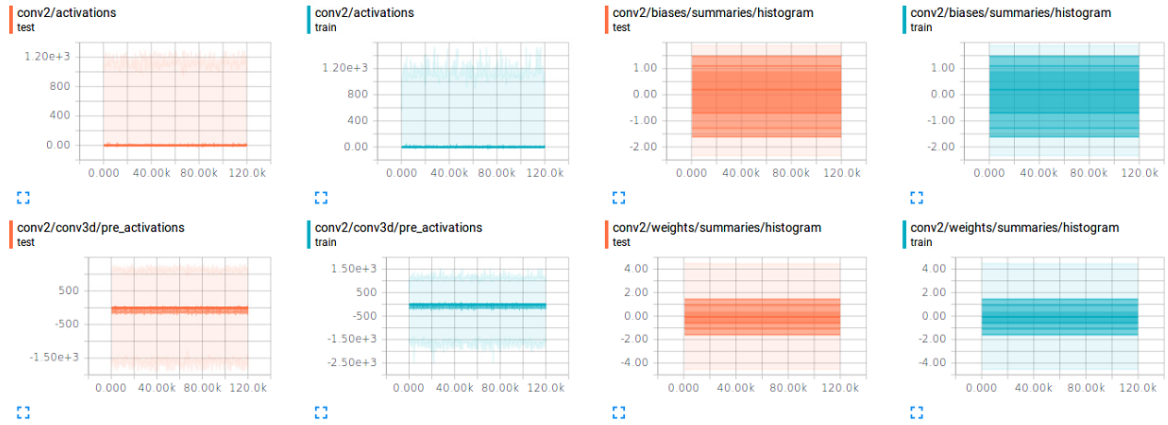


Figure 91. Conv2 Activations of ModelNet40 Top Layer Subproblem for Conquer Learning

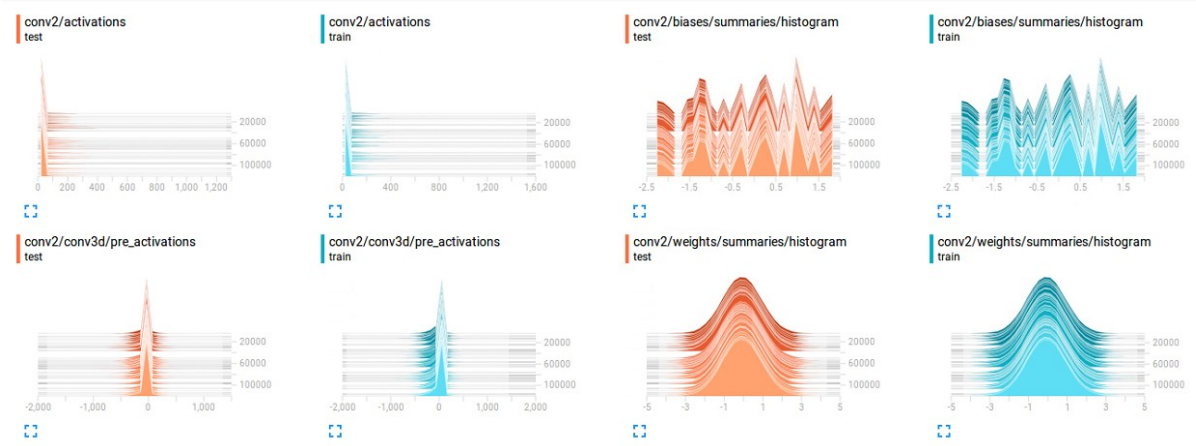


Figure 92. Conv2 Histograms of ModelNet40 Top Layer Subproblem for Conquer Learning

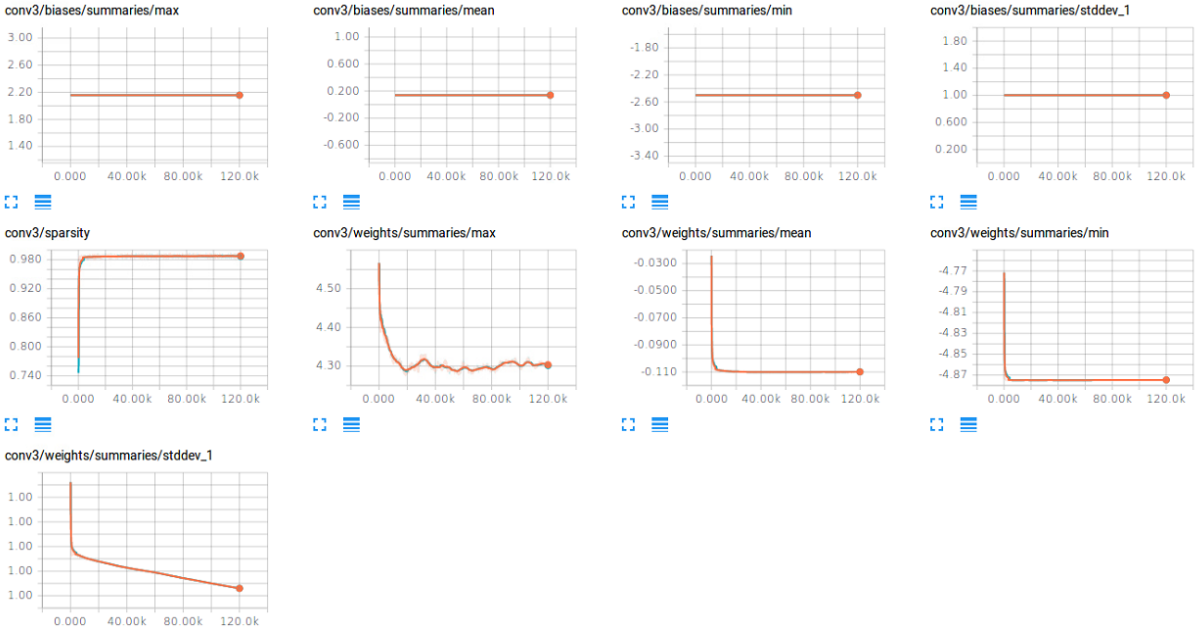


Figure 93. Conv3 Summaries of ModelNet40 Top Layer Subproblem for Conquer Learning

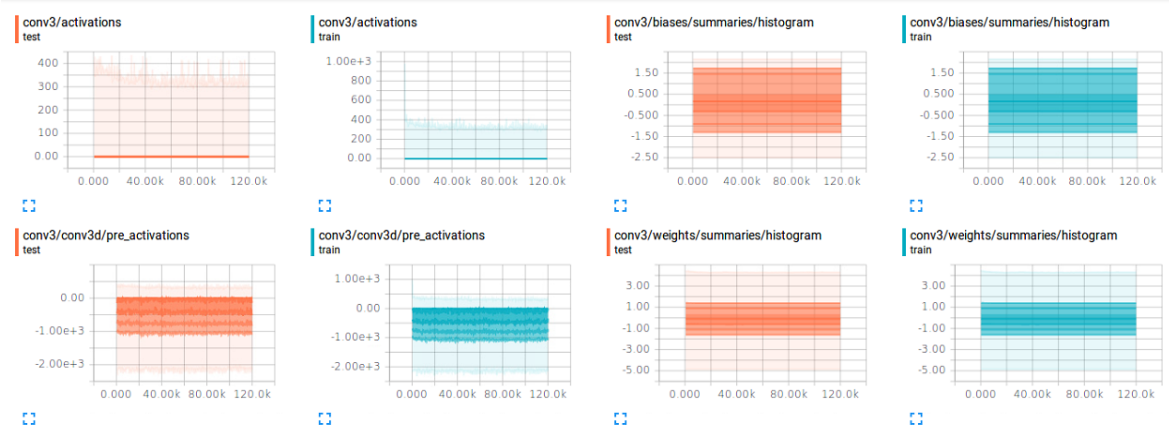


Figure 94. Conv3 Activations of ModelNet40 Top Layer Subproblem for Conquer Learning

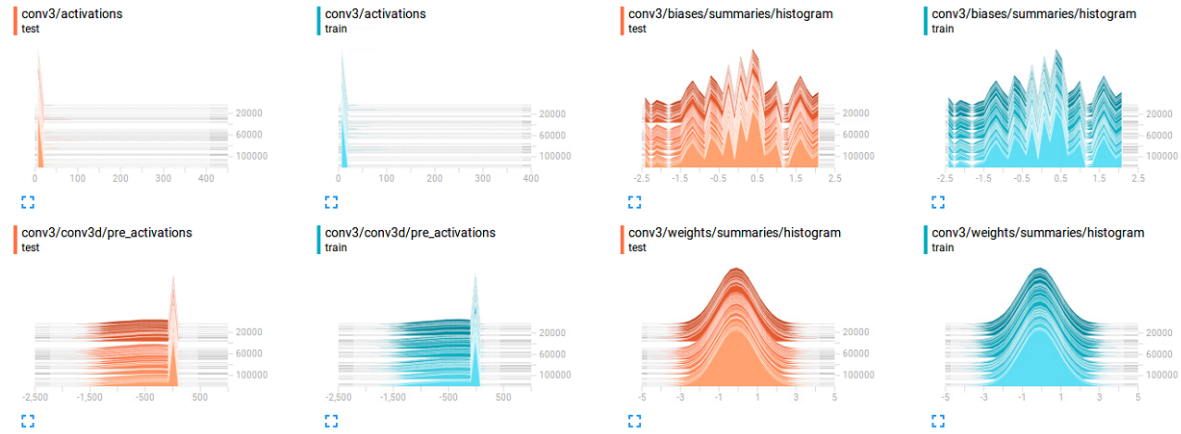


Figure 95. Conv3 Histograms of ModelNet40 Top Layer Subproblem for Conquer Learning

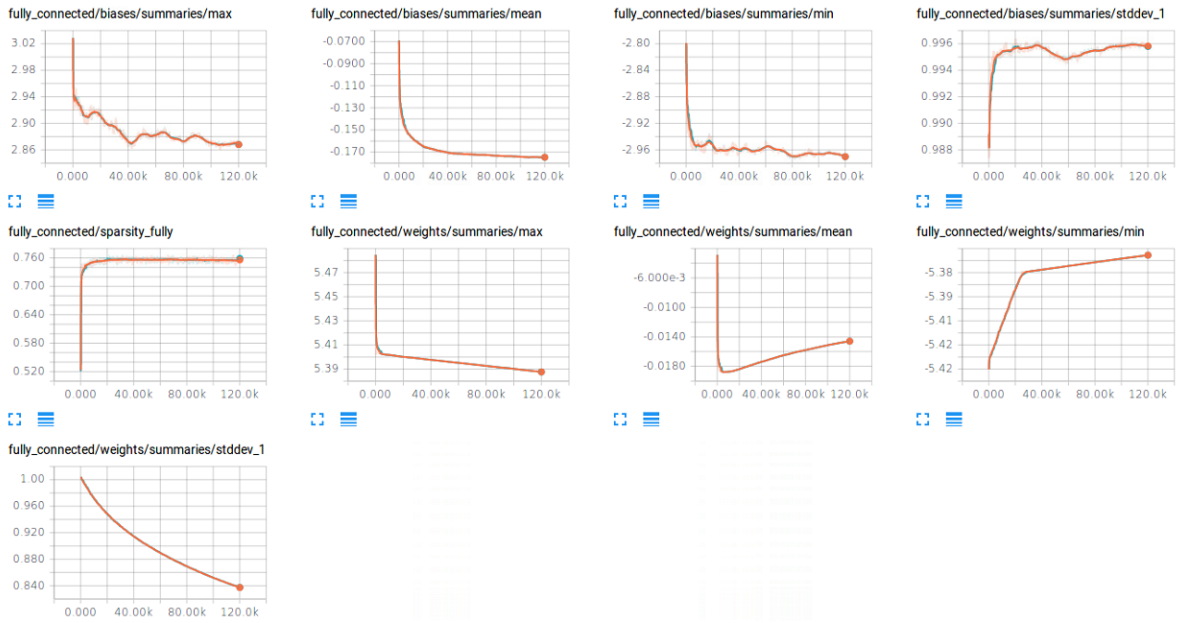


Figure 96. Fully Connected Summaries of ModelNet40 Top Layer Subproblem for Conquer Learning

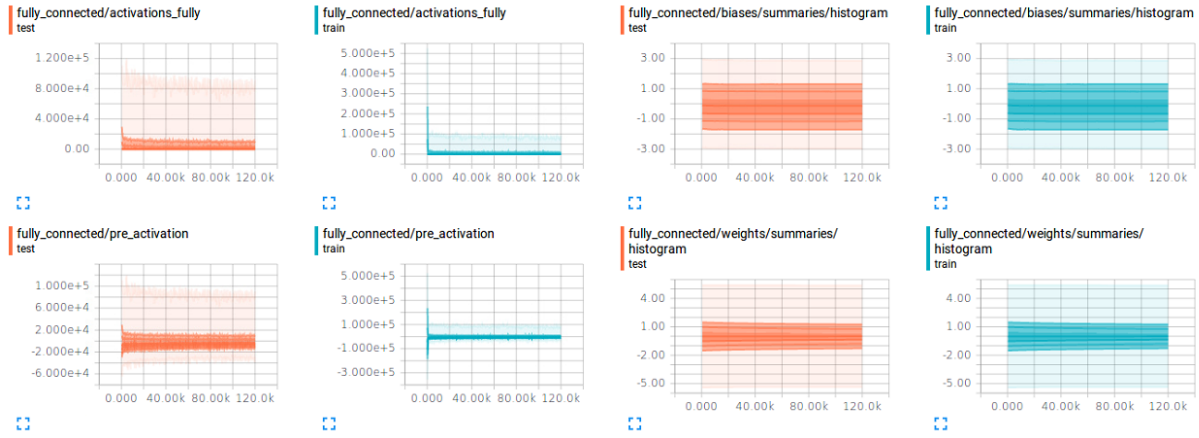


Figure 97. Fully Connected Activations of ModelNet40 Top Layer Subproblem for Conquer Learning

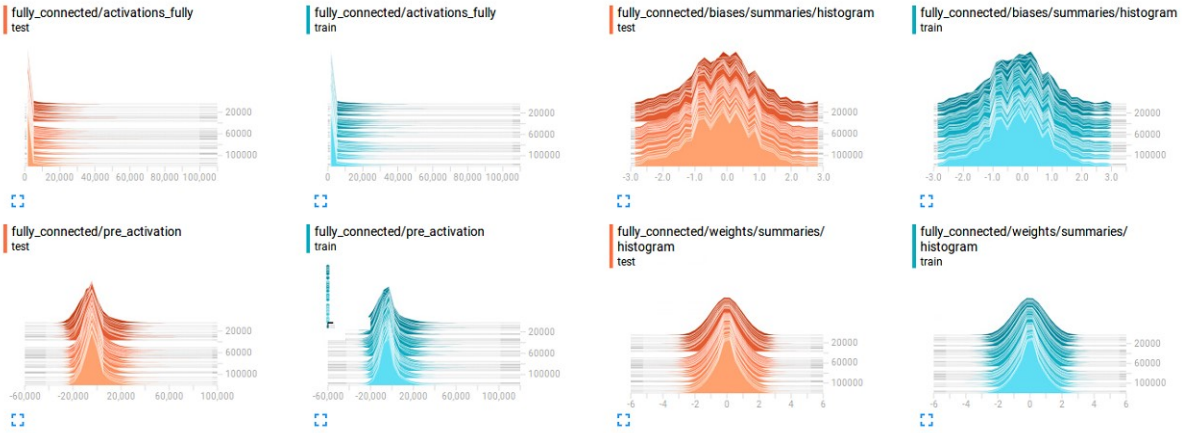


Figure 98. Fully Connected Histograms of ModelNet40 Top Layer Subproblem for Conquer Learning

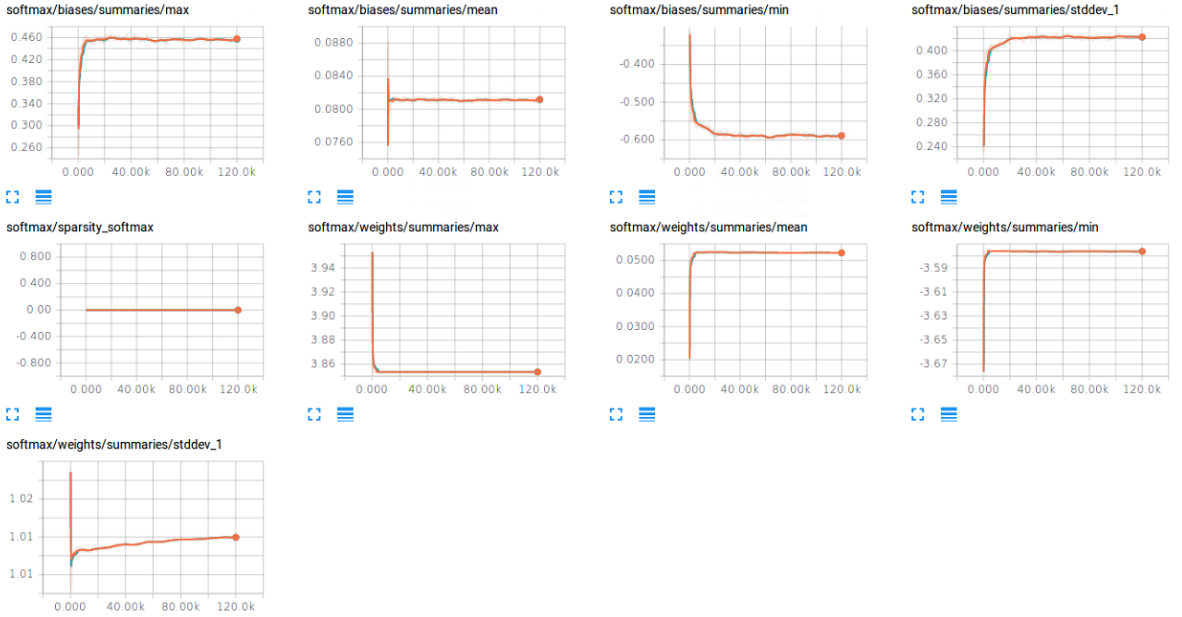


Figure 99. Softmax Summaries of ModelNet40 Top Layer Subproblem for Conquer Learning

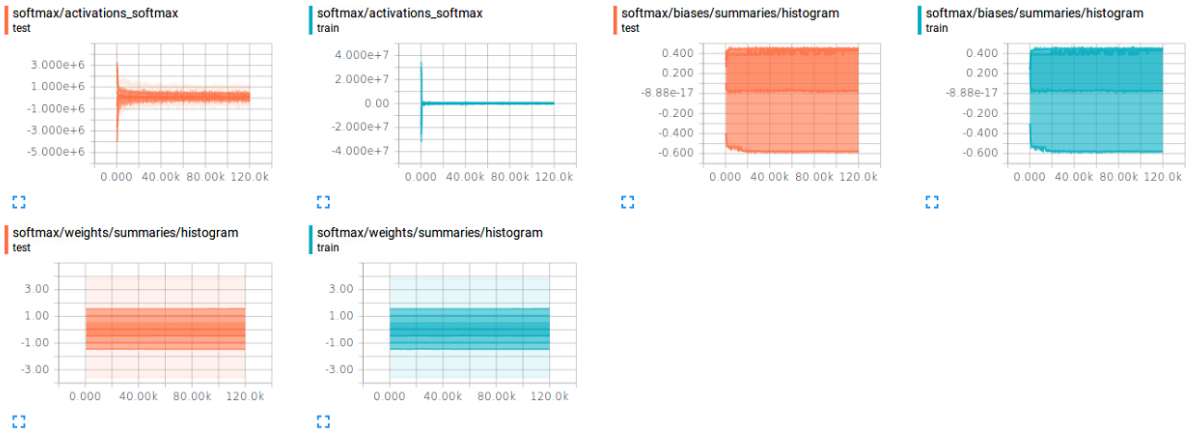


Figure 100. Softmax Activations of ModelNet40 Top Layer Subproblem for Conquer Learning

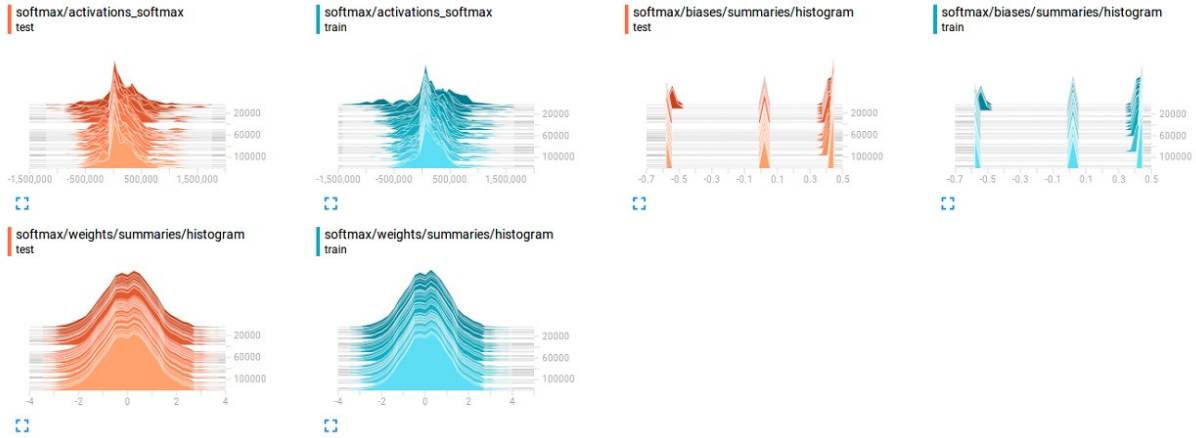


Figure 101. Softmax Histograms of ModelNet40 Top Layer Subproblem for Conquer Learning

After optimization and training, the accuracy for lower layer subproblems is shown in Figure 102.

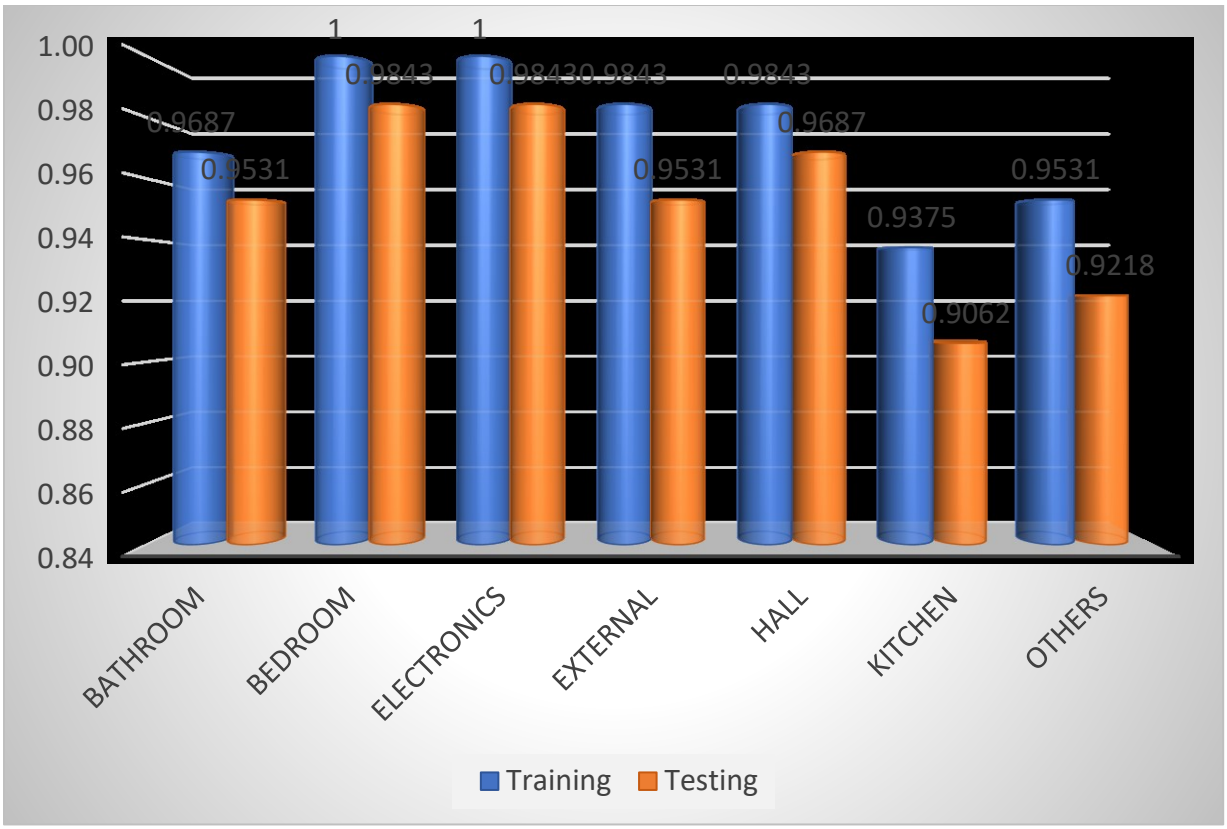


Figure 102. Accuracy of Lower Layer Subproblems (ModelNet40)

Prediction for ModelNet40:

The prediction strategy for ModelNet40 is same as the ModelNet40 but the only difference is we need to follow three level hierarchy instead of two level. Using the same top-down approach for prediction, we tried to predict labels for 299 objects over 40 classes. With the fuzzy parameter k = 1 (predicting one label at top layer) out of 299 objects 246 objects label are predicted correctly while with the fuzzy parameter k = 2 (predicting two labels at top layer) out of 299 objects 260 objects were predicted correctly. Figure 103 shows these statistics with accuracy for k = 1 and k = 2.

	Total Object	Correctly Classified	Accuracy
K=1	299	246	0.8227
K=2	299	260	0.8695

Figure 103. Accuracy for k =1 and k = 2 (ModelNet40)

Figure 104 shows the per class accuracy of ModelNet40 for k = 1.

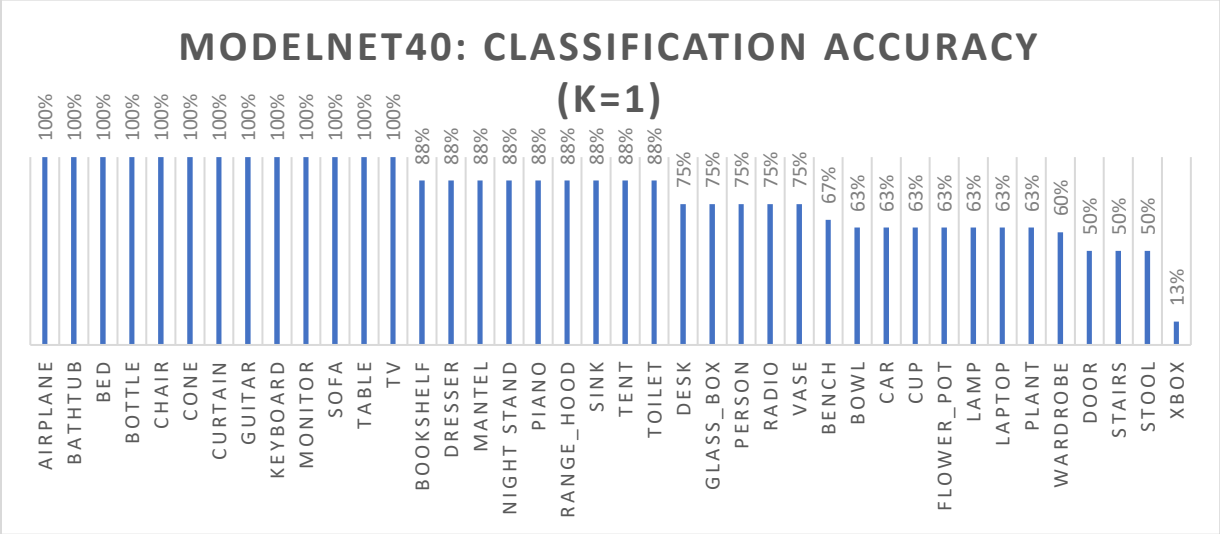


Figure 104. Per class Accuracy for k =1 (ModelNet40)

Figure 105 shows the per class accuracy of ModelNet40 for k = 2.

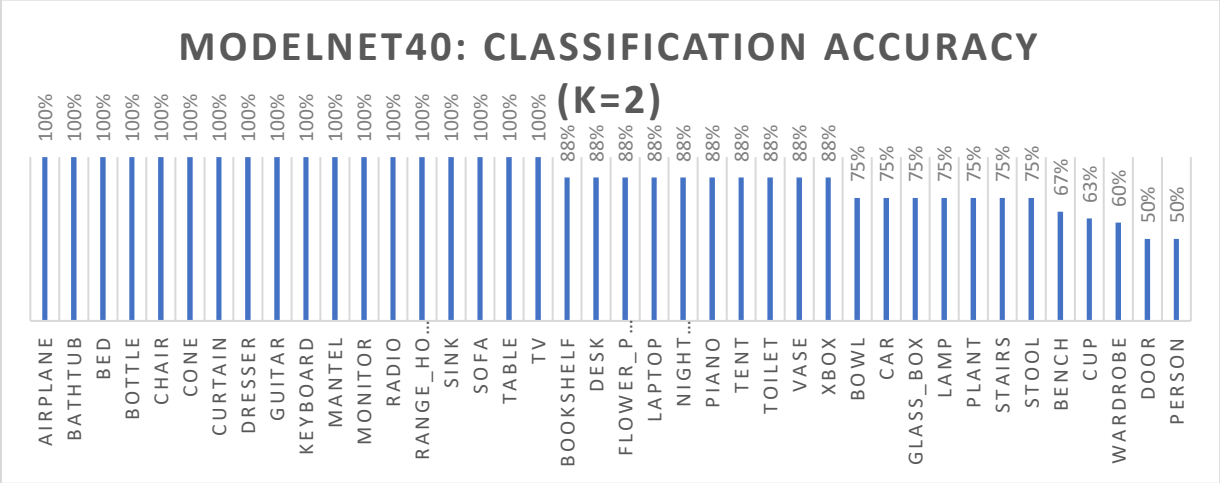


Figure 105. Per Class Accuracy for k = 2 (ModelNet40)

The accuracy of ModelNet40 is highly affected by some classes because of two reasons. One is the similarity between their 3D features which was also the case for ModelNet10 prediction. While the other reason is that the classes which have less than 200 objects such as the wardrobe. Because of the less number of objects, the model was not training properly for these classes and thus it was not able to predict them correctly. Figure 106 shows the classes which have less than 200 objects. If

we remove these classes from the evaluation, then the accuracy for k = 1 is 88.94% while for k = 2 is 94.21%. It is necessary to have a large number of objects to train the model optimally.

Class	# of Objects	Class	# of Objects
Bowl	84	Laptop	169
Cone	187	Person	108
Cup	99	Plant	130
Curtain	158	Radio	124
Door	129	Stairs	160
flower_pot	169	Stool	190
Keyboard	165	wardrobe	107
Lamp	144	Xbox	123

Figure 106. Classes with less than 200 objects

Figure 107 shows the prediction time in milliseconds for k = 1 and k = 2 with respect to different layers.

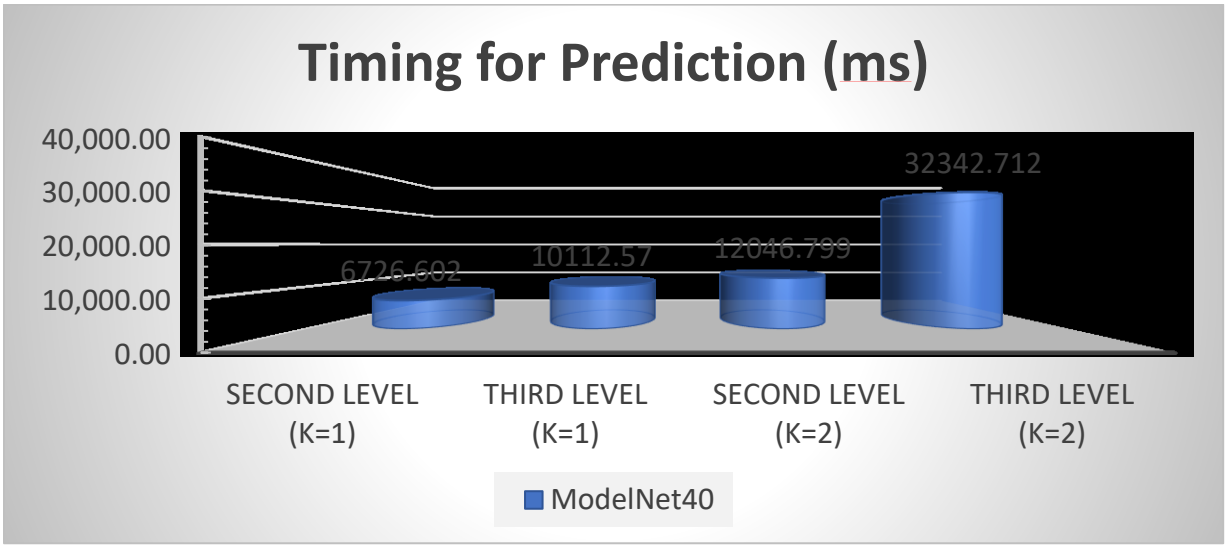


Figure 107. Prediction Time for ModelNet40

The prediction time increased as the number of layers and value of k increased. In our class hierarchy, first class appears at the second level which is lamp while all other 39 class are at last level. For k = 1, if label predicted at second level the time will be 6.726s while if it is predicted at the third level then the time will be 10.112s. For k = 2, if label predicted at second then the prediction time will be 12.046s while if it is predicted at the third level then prediction time will be 32.2s. It is very important to choose the correct value of levels and k.

4.6.3 Performance Comparison

Performance Comparison for ModelNet10

Figure 108 shows the accuracy comparison between ShapeNet, VoxNet[7], PointNet and H3DNET.

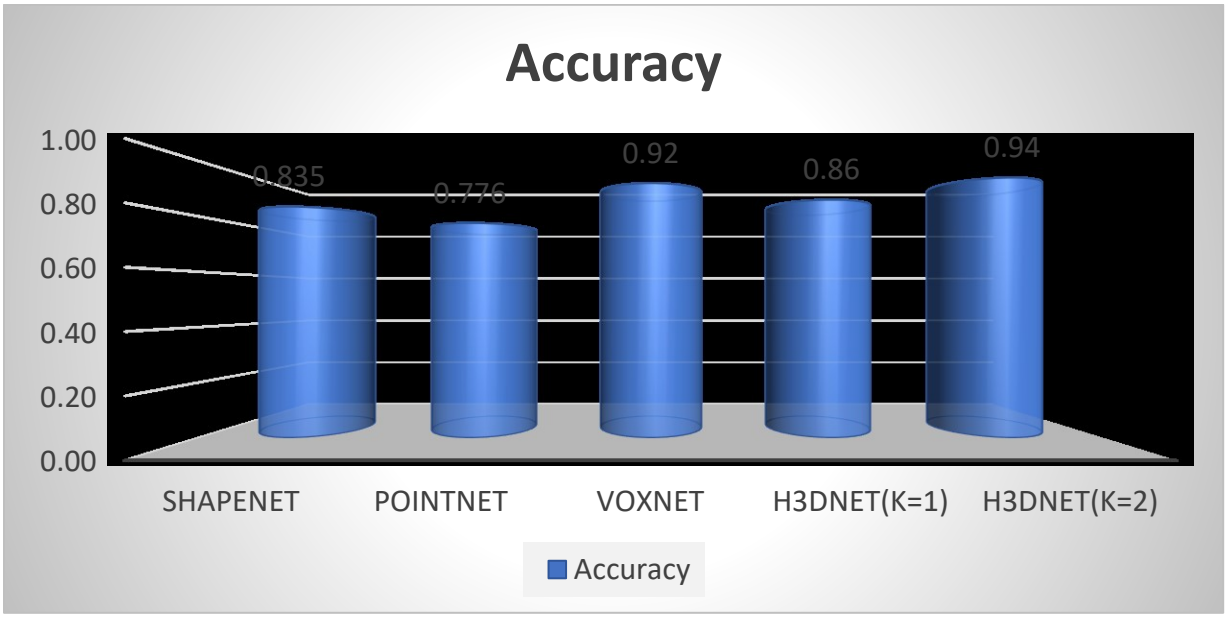


Figure 108. Performance Comparison for ModelNet10

With H3DNET, the state of the art accuracy is achieved for k = 2 while for k = 1 it is the bit less than the current state of the art accuracy. In comparison with Pointnet which is not based on voxelization, the performance for H3DNET is very good.

Performance Comparison for ModelNet40:

Figure 109 shows the accuracy comparison between the ShapeNet, VoxNet[7] and H3DNET.

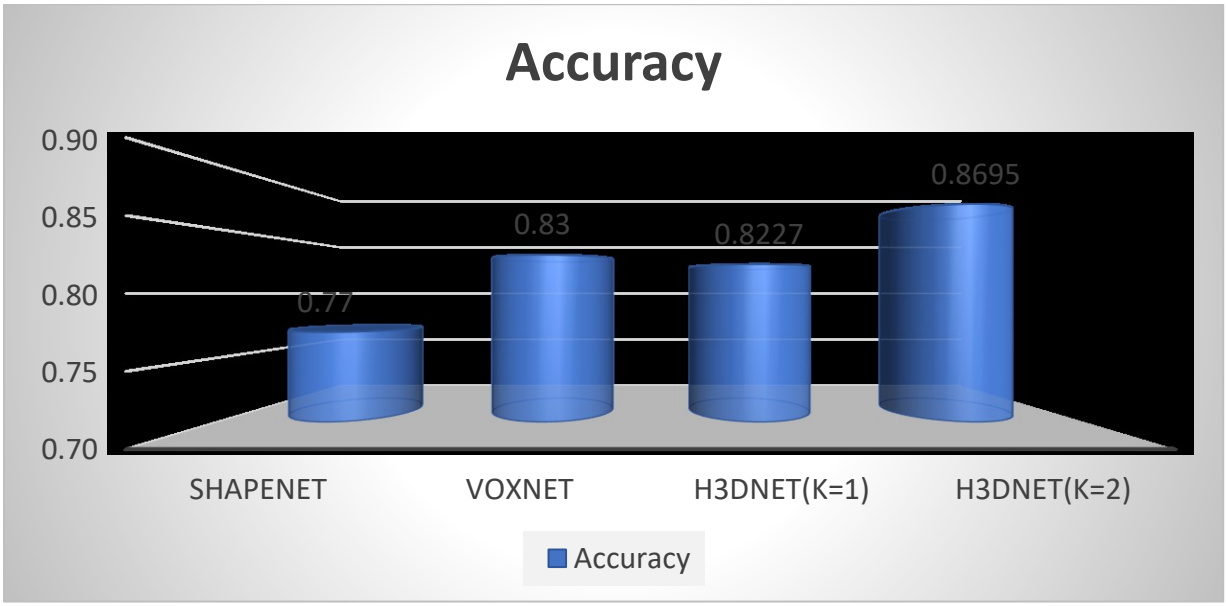


Figure 109. Performance Comparison for ModelNet40

For the ModelNet40, the accuracy for k=2 is better than the state of the art algorithm's accuracy while for k = 1 it is comparable to the it. The performance is very good than SHAPENET which has the accuracy of 77%.

Dataset Size Comparison:

Because of the voxelization and converting objects into binary records, there is a significant reduction in the size of the dataset. For ModelNet10, it is reduced from 4.8 GB to 314 MB while for s ModelNet40, it is reduced from 9.8 GB to 1.4 GB. Figure 110 shows the statistics about the size.

	Original Size	After Preprocessing Size
ModelNet10	4.8 GB	0.3 GB
ModelNet40	9.8 GB	1.5 GB

Figure 110. Dataset Size Comparison before and after preprocessing

Training Time Comparison:

Our model takes around 11s for 100 iterations and preprocessing of ModelNet40 requires nearly 1 hours. If we train the network for 100K iteration, it will collectively take 5 to 7 hours which was 6 to 12 hours in the case of VoxNet[7] but it cannot be compared until tested on the same environments.

4.7 Summary

The results from the test cases clearly show that the approach presented here showcases the distributed learning with a desired accuracy and training time. The approach also gives the optimal performance even with the larger dataset by dividing the classes into smaller subproblems. There are so many possibilities to improve the approach to make it more robust and autonomous.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

The presented framework H3DNET can be used to prepare the distributed machine learning model for 3D object classification or any other machine learning problem. The distribution and optimization technique presented can be scale well with larger datasets to divide them into the smaller problem and get the optimal overall accuracy with less complex networks. The divide learning and conquer learning is not dependent on any of the parameters of another model so it possible to train them in the separate machine and then use them to predict the class. It is also possible to add new subproblems at higher layer or add the new class to lower layer model which does not require to retrain the other models, so this approach can also achieve distributed and incremental learning. Here we have used the same models for all the learning, but it is not necessary. We can use a different model for different subproblems.

Even during inference stage, it is possible to implement the distributed computation to reduce the resource usage of the machine. As discussed above, in the system where new data comes very often, this approach can be the better choice as adding new data does not require to rebuild the network but the partial update of the model. This together with distributed learning and prediction, makes it efficient compare to another model which performs 3D object classification. This approach can also be applied to any similar problems, such as image classification, gesture recognition or any classification problems which is not same for other models.

5.2 Limitations

The overall performance of approach presented in this thesis dependent on the higher layer model accuracy. Another big time-consuming task is optimization of the distribution of class as there is no straightforward rule to select the optimum grouping of classes. The top layer learning takes too

much time to optimize as there might be the classes which look same and distributed among different subproblems. To solve this, it requires redesigning the top layer model with more complex structure with a large number of parameters. In this hierarchical approach, the number of levels is also important as there is a trade-off between the number of levels and performance of the network.

5.3 Future Work

The thesis presented here can be more reliable and accurate if we implement some of the below extension to the approach.

- The presented architecture is evaluated on object classification, which can be applied to any other supervised problems
- For 3D object classification, density voxel grid can also be used instead of binary grid to provide the color information of 3D object.
- The Presented architecture can be improved by using a more appropriate method in selecting the proper conquer learning model group size and group set.
- A dynamic machine learning model can be the extension to this framework where the distribution of classes can be done automatically by looking at the generated confusion matrix and this continues until desired performance is not achieved
- The proposed model can also be useful for multitasking such as object recognition and gesture recognition which can provide some insights about the transfer learning.

REFERENCES

[1] Rémi Cadène, Thome Nicolas, and Cord Matthieu. 2016. Master's Thesis: Deep Learning for Visual Recognition. In arXiv preprint arXiv:1610.05567. Retrieved from https://arxiv.org/abs/1610.05567

[2] Mayanka Chandrasekhar. 2017. Multi-class Discrimination Distribution Model. (to be Published)

[3] Jeffrey Dean et al. 2012. Large scale distributed deep networks. NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (2012), 1223–1231.

[4] Stephen Jones. 2016. How the GPU Is Revolutionizing Machine Learning | NVIDIA Blog. (October 2016). Retrieved April 3, 2017 from https://blogs.nvidia.com/blog/2015/12/10/machine-learning-gpu-facebook/

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Advances in neural information processing systems. 1097–1105

[6] S. W. Wang and A. E. Kaufman 1993, "Volume sampled voxelization of geometric primitives" *, 1993 IEEE Conference on Visualization (1993)*, pp. 78-84. DOI: https://doi.org/10.1109/VISUAL.1993.398854

[7] Zhirong Wu *et al*. 2015, "3D ShapeNets: A deep representation for volumetric shapes," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)*, pp. 1912-1920. DOI: https://doi.org/10.1109/CVPR.2015.7298801

[8] D. Maturana and S. Scherer 2015, "VoxNet[7]: A 3D Convolutional Neural Network for real-time object recognition," *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2015)*, pp. 922-928. DOI: https://doi.org/10.1109/IROS.2015.7353481

[9] Garcia-Garcia, F. Gomez-Donoso†, J. Garcia-Rodriguez, S. Orts-Escolano, M. Cazorla, J. Azorin-Lopez 2016, "PointNet: A 3D Convolutional Neural Network for Real-Time Object Class Recognition," 2016 arXiv preprint arXiv:1612.00593. DOI: https:// arxiv.org/abs /1612.00593

[10] Bernard Marr. 2016. The Top 10 AI and Machine Learning Use Cases Everyone Should Know About. (September 2016). Retrieved April 3, 2017 from https://www.forbes.com/sites/bernardmarr/2016/09/30/what-are-the-top-10-use-cases-for-machine-learning-and-ai/#7e532cb194c9

[11] Tom M. Mitchell. 1997. Machine learning, New York: McGraw-Hill Book Company

[12] Karen Simonyan, and Zisserman Andrew. 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556. DOI: https:// arxiv.org/abs /1409.1556

[13] Christian Szegedy et al. 2015. Going deeper with convolutions. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015). DOI:http://dx.doi.org/10.1109/cvpr.2015.7298594

[14] Anon. Convolutional Neural Networks | TensorFlow. Retrieved May 28, 2017 from https://www.tensorflow.org/tutorials/deep_cnn.

[15] Anon. TensorFlow. Retrieved April 3, 2017 from https://www.tensorflow.org/

[16] Anon. ModelNet from http://modelnet.cs.princeton.edu/

[17] Anon. Online Courses from Top Universities. Join for Free. Retrieved April 3, 2017 from https://www.coursera.org/learn/machine-learning

[18] Anon. Supervised Learning. Retrieved April 3, 2017 from https://www.mathworks.com/discovery/supervised-learning.html

[19] Alex Hern. 2016. Google says machine learning is the future. So I tried it myself. (June 2016). Retrieved April 3, 2017 from https://www.theguardian.com/technology/2016/jun/28/google-says-machine-learning-is-the-future-so-i-tried-it-myself

[20] Anon. Welcome to Deep Learning. Retrieved April 3, 2017 from http://deeplearning.net/

[21] Olga Russakovsky et al. 2015. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) 115, 3 (2015), 211–252. DOI:http://dx.doi.org/10.1007/s11263-015-0816-y

VITA

Marmikkumar Patel completed his Bachelor's degree in Computer Engineering from Gujarat Technological University in Ahmedabad, India and he started his Masters in Computer Science at the University of Missouri-Kansas City (UMKC) in January 2016, with an emphasis on Data Sciences and he graduated in December 2017. While he was studying in UMKC, he has worked as Graduate Teaching Assistant for Advanced Software Engineer and Graduate Research Assistant for School of Computing and Engineering. He was also Big Data Hadoop Developer Intern in Oalva Inc. from June 2017 to December 2017. Upon completion of his requirements for the Master's Program, he plans to work as a Data Scientist/Machine Learning Engineer.