6-9-2022

# An Empirical Study On Sampling Approaches For 3D Image Classification Using Deep Learning

Nicholas Michelette
*Rowan University*

# AN EMPIRICAL STUDY ON SAMPLING APPROACHES FOR 3D IMAGE CLASSIFICATION USING DEEP LEARNING

by

Nicholas Michelette

A Thesis

Submitted to the
Department of Computer Science
College of Science and Mathematics
In partial fulfillment of the requirement
For the degree of
Master of Science in Computer Science
at
Rowan University
May 17, 2022

Thesis Chair: Shen-Shyang Ho, Ph.D., Associate Professor, Department of Computer Science

Committee Members:
Bo Sun, Ph.D., Associate Professor, Department of Computer Science
Ganesh R. Baliga, Ph.D., Professor, Department of Computer Science

## Dedications

This work is dedicated to my parents and grandparents, who have supported me throughout my entire life and education. I would not be here without their continued assistance. My father's hard work in both academia and in the real world has also inspired me throughout my studies.

**Acknowledgments**

I would like to thank my supervisor/advisor Dr. Shen-Shyang Ho for his guidance for the entirety of this thesis. He helped me pinpoint exactly what I wanted this thesis to be as well as giving me direction. Additionally, his experience has helped tremendously to ensure that this thesis is the best it could have been, from knowing how to pace the work needed to technical writing. I would also like to thank Dr. Sun and Dr. Baliga for providing some great insight for this thesis. Without everybody, this thesis would not have been possible.

## Abstract

Nicholas Michelette
AN EMPIRICAL STUDY ON SAMPLING APPROACHES FOR 3D IMAGE CLASSIFICATION USING DEEP LEARNING
2021-2022
Shen-Shyang Ho, Ph.D.
Master of Science in Computer Science

A 3D classification method requires more training data than a 2D image classification method to achieve good performance. These training data usually come in the form of multiple 2D images (e.g., slices in a CT scan) or point clouds (e.g., 3D CAD modeling) for volumetric object representation. The amount of data required to complete this higher dimension problem comes with the cost of requiring more processing time and space. This problem can be mitigated with data size reduction (i.e., sampling).

In this thesis, we empirically study and compare the classification performance and deep learning training time of PointNet utilizing uniform random sampling and farthest point sampling, and SampleNet which utilizes a reduction approach based on weighted average of nearest neighbor points, and Multi-view Convolution Neural Network (MVCNN). Contrary to recent research which claimed that SampleNet performs outright better than simple form of sampling approaches used by PointNet, our experimental results show that SampleNet may not significantly reduce processing time and yet it achieves a poorer classification performance. Additionally, resolution reduction for the views in MVCNN achieves poor accuracy when compared to view reduction. Moreover, our experimental result shows that simple sampling approaches used by PointNet as well as using simple view reduction when using a multi-view classifier can maintain accuracy while decreasing processing time for the 3D classification task.

## Table of Contents

Abstract ..... v

List of Figures ..... viii

List of Tables ..... ix

Chapter 1: Introduction ..... 1

1.1 Problem Statement and Investigation ..... 2

1.2 Thesis Outline ..... 3

Chapter 2: Literature Review ..... 4

Chapter 3: Compared Methodologies ..... 8

3.1 VoxNet ..... 8

3.2 PointNet ..... 10

3.3 Multi-View Convolutional Neural Network (MVCNN) ..... 15

Chapter 4: Experimental Setup and Design ..... 17

4.1 Dataset Description ..... 17

4.1.1 PointNet and SampleNet ..... 17

4.1.2 MVCNN ..... 17

4.2 Evaluation Measures ..... 17

4.3 Hardware and Code Used ..... 18

4.3.1 PointNet and SampleNet Code ..... 18

4.3.2 MVCNN Code ..... 18

4.4 Investigated Issues ..... 19

4.4.1 PointNet and SampleNet Classification Performance Comparison ..... 19

4.4.2 MVCNN ..... 19

## Table of Contents (Continued)

Chapter 5: Experimental Results and Analysis.....21

5.1 Sampling with PointNet Versus SampleNet.....21

5.2 Different Types of Sampling With MVCNN.....37

Chapter 6: Conclusions and Future Work.....52

References.....54

## List of Figures

Figure Page

Figure 1. Furthest Point Sampling (FPS) vs Uniform Random Sampling [5] ..... 14

Figure 2. PointNet With 1024 Randomly Sampled Points (PointNet 1024) vs SampleNet With Different Sampling Size ..... 22

Figure 3. Comparing Classification Performance for PointNet 1024 vs PointNet With Smaller Sample Sizes ..... 24

Figure 4. Comparing Classification Performance for PointNet vs SampleNet With Fixed Sample Size ..... 26

Figure 5. Comparing Classification Performance for PointNet Random vs Furthest Point Sampling With Fixed Sample Size ..... 28

Figure 6. Comparing Classification Performance for PointNet Random vs PointNet FP vs SampleNet Sampling Accuracy ..... 31

Figure 7. Comparing Training Time for PointNet and SampleNet Sampling ..... 32

Figure 8. Comparing Classification Performance for PointNet and SampleNet 1024, 256, 128, 64, 32 ..... 33

Figure 9. SampleNet With Different Number of Points in Initial Point Cloud and PointNet Using 2048 Points. ..... 35

Figure 10. MVCNN Validation Accuracy Over Time ..... 43

Figure 11. MVCNN Validation Accuracy - Maximum vs Minimum Evaluation Accuracy ..... 45

Figure 12. MVCNN Sampling Method Loss ..... 47

Figure 13. MVCNN Average Time per Trial vs Number of Views ..... 48

Figure 14. MVCNN Validation Accuracy Using Various Sampling Methods (From Table *4*) ..... 49

## List of Tables

Table Page

Table 1. Comparing Classification Performance and Training Time for PointNet Random vs PointNet FP vs SampleNet Sampling Accuracy .....30

Table 2. MVCNN Test Results – Batch Size 4.....37

Table 3. MVCNN Test Results – Batch Size 1.....40

Table 4. MVCNN Test Results – Batch Size 16 With 64 Epochs.....41

# Chapter 1

## Introduction

Object classification is one of many computer vision problems, whether it is done through single 2D images or using more three-dimensional (3D) data. A 2D image classification model usually takes a long time to train but a 3D model requires even more data in general to be processed and therefore takes longer to complete training, even if there is less instances or objects to process. 3D data generally comes in the form of meshes, point clouds, or multiple images [2]. Some point clouds contain thousands of points and multiple images contain many more pixels than 2D images. For some classification methods, the point cloud is converted to a grid of binary voxels. With more data, the training of neural networks becomes more computationally expensive.

3D image classification can be used in a variety of different situations. Robotics require classification of objects before doing their task [26] and medical imaging can use 3D classification to identify diseases like the coronavirus [28]. Some specific applications include self-driving cars [25], industrial production, domestic assistance, and healthcare services [26]. Industrial production includes welding, assembly, and shipping of products [26] as well as industrial part classification [23] and industrial bin picking [24]. Domestic assistance applications include household chores, entertainment, personal assistance like a chef assistant, and home defense [26]. Healthcare service applications include surgery, caring for patients, being a receptionist, and being a nurse assistant,

One method of reducing the training time as well as inference time is called sampling, which reduces the amount of input data to a subset of the original input data or

a representation of the original input data. This can be done in many different ways [2, 3, 4, 6, 16, 30, 31, 32, 34, 35] for each classification method, but the objective of sampling is to maintain the accuracy of the classifier while reducing the amount of data the classifier needs for training. A simple example of sampling is randomly choosing points from a point cloud to use as input into the classifier instead of using the whole point cloud.

There are many different ways to sample. For point clouds, the most common methods of sampling include random point sampling as well as furthest point sampling [6], both of which are simple functions. There are more complex sampling methods, like those found in S-Net [6] and SampleNet [4], which attempt to learn which points are the best points to sample and are overall more complex than furthest point sampling or random sampling.

Sampling generally is not done with multi-view classifiers, but the images are taken at a certain resolution or resized to that resolution [3]. However, there are a few simple sampling methods possible like reducing the resolution further or selecting a subset of images from the multiple images taken for an object. Any sampling method that could apply to multi-view classifiers could also apply to the voxel based classifiers, like reducing the resolution of the voxel grid [1].

### 1.1 Problem Statement and Investigation

The main problem with 3D classification is that there is a lot of data needed to train a classifier and training the model is computationally expensive. Some possible solutions include sampling as well as view reduction. This thesis investigates the effect of

sampling or multi-view reduction approaches on existing 3D deep learning approaches, in particular PointNet [2], SampleNet [4], and Multiview Convolution Neural Network (MVCNN) [3] for 3D image classification.

### 1.2 Thesis Outline

In Chapter 2, we will describe recent work on using deep learning for classifying objects using 2D images to 3D representations and provide an overview of sampling methods used in object classification. In Chapter 3, we will describe how each baseline 3D classification approach accomplishes the task of 3D classification. In Chapter 4, we will describe the experimental setup and design for testing the selected sampling methods for PointNet, SampleNet, and MVCNN. This chapter includes which dataset was used and how each method was tested. In Chapter 5, we will present the results from the experiment along with an analysis of those results. In Chapter 6, we present the conclusion along with some potential future work.

# Chapter 2

## Literature Review

To start, most image classifiers or object classifiers use some form of machine learning, and most classifiers use a neural network. There are image processing techniques that do not involve machine learning that helps with image classification with one of the first being found in “Machine Perception of Three-Dimensional Solids” [10]. The first 2D image classifier, used to classify Japanese characters, was called Neocognitron and created by Kunihiko Fukushima [11]. This was effectively the first convolutional neural network and was one of the first instances of deep learning. After Neocognitron came LeNet, created by Yaan LeCun [12] in the 1990s and based off of Neocognitron. This project took the convolutional neural network from Neocognitron and applied gradient backpropagation to it, effectively allowing the convolutional neural network to learn more efficiently. This project was used to classify characters as well, giving rise to the popular MNIST dataset [13].

After LeNet, some more datasets were compiled and challenges to classify the datasets were made, notably the Pascal Visual Object Classes [14] and ImageNet [15] datasets, the latter of which contains over 1 million images with 1000 classes. In the 2012 ImageNet competition, AlexNet [16] achieved an error rate much lower than any previous classification projects for ImageNet. AlexNet used a convolutional neural network similar to the neural network found in LeNet, but with many more layers and parameters. Some reasons for the success of AlexNet include using the ReLU activation function, dropout regularization, and the much deeper neural network that could finally

be used because technology finally had the processing power to do so along with the ability to use a GPU for processing. 2D image classifiers continue to evolve with one example being DenseNet [17].

The next big step for learning on 3D objects is ShapeNet [18], which used 3D CAD models to create a volumetric occupancy grid to use as input into a three-dimensional convolutional neural network (3D CNN). Along with the paper, Wu *et al.* released the largest 3D CAD model dataset at the time called ModelNet [19]. This allowed for many other methods to be developed. One of the methods improved upon what ShapeNet had already done and improved the accuracy of using a voxel grid along with a 3D CNN. Since VoxNet [1] was an improvement over ShapeNet, most projects attempting to improve on voxel grid classifiers use VoxNet as a baseline. One example of this being OctNet [20].

Another foundational method of classifying the models from ModelNet uses point cloud data directly and is called PointNet [2]. This method takes a fixed number of input points and uses their coordinates as input into a neural network, which then classifies the object. One attempted improvement that targeted the accuracy of point cloud classifiers is DensePoint [21], which applies the concept of DenseNet [17] to the point cloud instead of an image. Some implementations attempt to reduce the training and interpretation time of PointNet, like S-Net [6] or SampleNet [4]. They attempt to do this by sampling the original point cloud using another neural network.

The last foundational method of classifying the models from ModelNet goes back to using images instead of converting or using the point cloud directly. Multi-view

convolutional neural networks (MVCNN) [3] use multiple images of a single object to help classify it. The MVCNN still uses the ModelNet dataset, opting to render pictures at regular intervals around each object to use as input. Each object in the dataset has a specified number of images associated with it, referred to as views. The MVCNN essentially uses the image classifier found in AlexNet [16] on each view and does further processing using the output of the previous step. According to the ModelNet website, an implementation of MVCNN called RotationNet [22] currently holds the record for the highest accuracy classifier on the ModelNet dataset. RotationNet attempts to align a subset of views correctly on the object and take the classification of each view after the correct alignment is achieved. In a way, it is not using all of the views of a single object to classify it.

Sampling with point clouds starts with the simplest method: choosing random points from the point cloud [2]. More complex sampling methods exist like Furthest Point Sampling [2, 30], clustering [31, 32], iterative simplification [31], and particle simulation [31]. Some sampling methods are more specific to image classification, like the sampling found in S-Net [6], SampleNet [4], and "Task-Aware Sampling Layer for Point-Wise Analysis" [32]. PointNet uses either random sampling or furthest point sampling.

Sampling with 2D images is typically not done with classifiers other than rescaling images and pooling functions within the classification model [16], which stays true for multi-view classifiers [3]. There are many image sampling methods to choose from which include uniform, random, nonuniform, and measurement-adaptive sampling algorithms [34]. Furthest point sampling can also be used with images [30]. Additionally, some data compression algorithms can be used to reduce the size of images [35]. With

multi-view classifiers, it is important to select the necessary number of views needed to classify datasets [3].

# Chapter 3

## Compared Methodologies

In this chapter, we describe in detail how each basic 3D classifier functions. In Section 3.1, we describe how voxel-based classifiers classify objects based on their volumetric occupancy grid representations. In Section 3.2, we describe how point cloud based classifiers classify an object using their point cloud representations. In section 3.3, we describe how multi-view or multi-image based classifiers classify an object based on their multi-view representations.

### 3.1 VoxNet

VoxNet [1] is one of the first successful 3D classification programs. There are three different types of datasets that VoxNet uses: CAD models, LiDAR point clouds, and RGBD point clouds. The points sampled from the CAD models, or the raw LiDAR point cloud data is then converted into a volumetric occupancy grid by mapping the point coordinates (x, y, z) to voxel coordinates (i, j, k), depending on the origin, orientation, and resolution of the voxel grid. Some constraints in this process are that the origin is assumed to be given as input, the z-axis is aligned with gravity, and the resolution is consistent for all objects. In VoxNet, the occupancy grid is 32 x 32 x 32 voxels, with the object fitting within a 24 x 24 x 24 voxel sub volume. Each voxel with LiDAR data has a fixed size, which is usually $(0.1\text{m})^3$.

Working with a volumetric occupancy grid allows for easy and simple manipulation of the data to be used in training. There are a number of reasons to not use the points directly in PointNet [2] and VoxNet found that the volumetric occupancy grid

was more consistent than previous methods. CAD models can be voxelized easily, but there are also a number of different types of occupancy grids that are used with LiDAR data: the binary occupancy grid, the density grid, and the hit grid. The binary occupancy grid is where each voxel is given a probability that it is occupied depending on if there are points in that part of the grid and the state of the voxel before it. The density grid is where the object is expected to have a uniform density, so the density of points within each voxel with respect to the point density of voxels is given as input into the neural network. Lastly, the hit grid is where each voxel has a number of points within it, which is used as input for the neural network. The differences in performance between these three is minimal, but it is a difference nonetheless and some of the grid types are faster to process than others.

To classify the volumetric occupancy grid, VoxNet uses a 3D convolutional neural network, or 3D CNN. 3D CNNs can make use of spatial features and learn local spatial features, which are important for classification. Also, increasing the number of layers may allow the network to recognize more complex features. Additionally, once trained, inference is feed-forward and fast, especially on modern hardware. The neural network consists of the input, 3D convolutional layers, 3D pooling layers, and fully connected layers.

The structure of the VoxNet CNN starts with the input of the occupancy grid of size I x J x K, followed by two convolutional layers, a pooling layer, and two fully connected layers. The specific number for the CNN they found optimal are a 32 x 32 x 32 grid, one convolutional layer with 32 filters, kernel size of 5 x 5 x 5, and stride of 2, another convolutional layer with 32 filters, kernel size of 3 x 3 x 3, and stride of 2,

followed by a pooling layer of size 2 x 2 x 2, with a fully connected layer going to 128 nodes, which then fully connect to another set of K nodes, where K is the number of classes. This structure has 921,736 parameters.

There is also another major problem with this approach: it is not rotation-independent. Since the object is only aligned to gravity, the object could not be facing "forward." To combat this, during preprocessing, there must be n copies of the object generated, each rotated 360o/n intervals around the z axis, with n being 12 or 18 in most cases. This means that the object is rotated by $30^o$ or $20^o$ from the previous rotation. The CNN is then trained on all rotations of every object, while at evaluation time, every copy of each rotation of an object is pooled together to classify the object. This works relatively well, but still occasionally fails.

Training of the CNN uses stochastic gradient descent with momentum, dropout regularization, and adds randomly perturbed copies of each instance of objects, which consists of mirroring and shifting between -2 and 2 voxels.

The methods of sampling on MVCNN [3] described later could also apply to VoxNet, as both use different "views" of the object to help classify it more accurately as well as have a defined resolution.

### 3.2 PointNet

PointNet [2], instead of transforming the data into voxels or an occupancy grid, uses point cloud data directly and each point in the cloud can be represented by the coordinates (x, y, z). There are three properties of point sets that are important: they are unordered, the points interact with each other, and invariance under transformation.

Interaction among points means that neighboring points form meaningful subsets, so a classifier model must capture local structures and interactions between local structures. Invariance under transformation just means that if all points undergo a transformation of some sort, whether it be translation or rotation, no data should be lost, and the classification should not change. Most importantly, point sets are unordered, which gives up to the factorial of the number of points, or N!, possible permutations on how to choose points, which can get very large as the number of points is usually in the thousands and the factorial operation results in extremely large numbers. This was the most important problem to solve when creating the classifier for PointNet.

Most deep learning focuses on regular inputs and not unordered sets of information like a point cloud. VoxNet [1] gets around this by mapping the points to an occupancy grid, which is an ordered input. However, using voxel-based methods have their limitations and some important data might be lost, so PointNet tries to use just the raw points as input. Some options include sorting the points or using an RNN, but PointNet found that symmetric functions give the best results. Operations like addition and multiplication are symmetric because the order of the numbers do not matter in the operation. PointNet accomplishes this with a combination of multi-layer perceptrons and a max pooling function.

The PointNet paper summarizes the overall function of the network well by stating “effectively the network learns a set of optimization functions/criteria that select interesting or informative points of the point cloud and encode the reason for their selection. The final fully connected layers of the network aggregate these learnt optimal values into the global descriptor for the entire shape” [2]. One effect of this network is

that it finds a sparse set of points that can effectively be used to construct a skeleton of the object.

A simple explanation of the PointNet neural network structure is that each point gets its own multi-layer perceptron or MLP. Each MLP has the input of each coordinate in the point, so the x value, y value, and z value of the current point. After a few fully connected layers, a max pool function is used to extract features out of the multiple MLPs. The result from the max pool function is then used in another MLP to finally classify the object. The symmetric function used is the single max pool function. Within the basic structure of PointNet, there are also input and feature transformations to help the network a bit. They found that these transformations improve the performance by about 1.9%. PointNet without these transformations has about 800,000 parameters with 148 million floating point operations (FLOPs) per sample and 3.5 million parameters with 440 million FLOPS per sample while using the transformations. According to Theorem 1 in [2], in the worst case scenario, the network can voxelize a point cloud, but ultimately finds a better way to probe the 3D space.

During training, noise is added to some of the points and the overall object is slightly rotated about the up-axis. Inference is also quick because of the use of simple fully connected layers and a max pool function. Inputs are also normalized to a unit sphere as some objects may have larger coordinates than others despite being a similar shape. PointNet is also more robust than earlier methods as deleting points or forcing the "important" points to be outliers do not affect the results much until extreme levels, like 80% of the data missing, are achieved.

As stated previously, the more accurate version of PointNet has around 3.5 million parameters with 440 million FLOPS per sample. This was much more space and time efficient than the previous 3D classifiers that were slightly more accurate like MVCNN [3], which has 60 million parameters and 62 billion FLOPs according to the PointNet paper. Point-based classifiers also scale with points linearly or O(N), while 2D convolutional networks squarely or $O(N^2)$, with image resolution in image-based methods, and cubically or $O(N^3)$ in volumetric convolution.

Some more applications of PointNet include retrieving models based on the selected "important" points from another point cloud, selecting points that correlate in two separate point clouds, and extracting the general shape of objects that have no category [2].

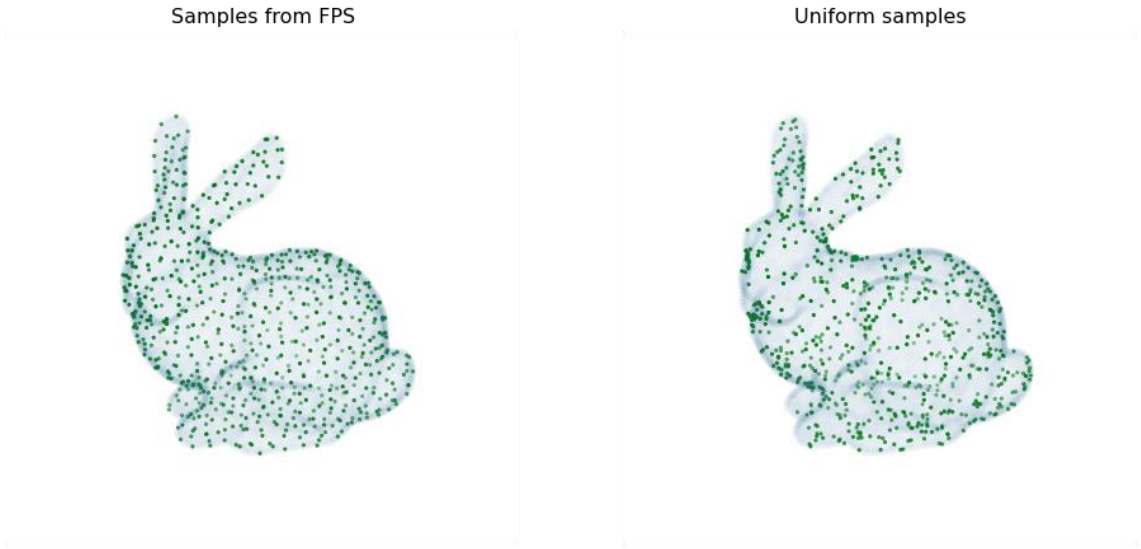
Since point clouds could have a large number of points as well as having an inconsistent number of points between many objects, sampling in some form is required. The dataset used for performance evaluation in [2] already has 2048 points sampled from the surface of a 3D CAD mesh, but 2048 points is too large to be performant. Therefore, PointNet uses further sampling on these point clouds by uniform random sampling or Furthest Point sampling.

Furthest Point Sampling (FPS) [30] is a simple sampling algorithm that consists of only a few steps. There are two sets: one set for all of the sampled points and one set for all of the remaining points that are not yet in the sampled points set. To start, a single point is randomly selected and put into the sampled set. Then, for each point in remaining, find the nearest neighbor in sampled, and save the distance to the point it

corresponds to in the remaining set. Then, select the point in the remaining set where the nearest neighbor distance is the largest and move that point from the remaining set to the sampled set. Then, repeat these steps until the sampled set has the specified number of points in it. Figure 1 shows how FPS gives a better representation of an object [5].

**Figure 1**

*Furthest Point Sampling (FPS) vs Uniform Random Sampling [5]*




SampleNet [4] proposes that classical sampling approaches like FPS do not consider the processing of the sampled point cloud. According to the work of Dovrat *et al.* [6], a neural network can be used to produce a set of simplified points which is optimized for point cloud learning, which then uses a post-processing step to match the simplified points to their nearest neighbors in the original point cloud. SampleNet continues this work by introducing a differentiable version of nearest neighbor selection

during training, which they call soft projection. Soft projection replaces the simplified points with a weighted average of the points nearest neighbors. This effectively approximates the nearest neighbor selection portion of the algorithm proposed by Dovrat *et al.* It is important to note that normal nearest neighbor sampling is a non-differentiable function with regard to neural networks, and therefore, cannot be learned on, but the approximation of the nearest neighbor is differentiable, and therefore, learnable, while also being able to use the original point cloud directly.

### 3.3 Multi-View Convolutional Neural Network (MVCNN)

Instead of classifying an object using a point cloud or volumetric occupancy grid as input, MVCNN [3] uses multiple 2D images as input. MVCNN uses multiple CNNs to classify objects. To use MVCNN, multiple images of the same object at different angles are required. The distance of each image origin to the object must be similar as well as the at regular intervals around the object for maximum performance.

To create the rendered images of the objects, shapes are scaled to fit into a viewing volume and cameras are set up to capture the images within the virtual scene. The cameras are placed in specific spots around the object depending on the assumed situation the object is in. For example, one of the cameras placing methods involves placing 12 cameras in a circle around the center of an object at 30 degrees intervals while being elevated 30 degrees from the ground plane. This is used assuming the objects are consistently aligned upright in the z-axis, which is the case for most 3D model datasets. Some datasets may not do this, so other methods for placing the camera may be needed.

MVCNN uses image descriptors based on Fisher vectors and CNN activation features to start the classification process. The first CNN in the network is shared between all 12 different views and when classifying, is used on all 12 views where the results are aggregated together in a “view pool,” which is similar to a max pool function. A separate neural network is used after this view pooling to finally classify the object. The parameters across all 12 views in the first neural network are shared.

# Chapter 4

## Experimental Setup and Design

In this chapter, we describe in detail which dataset, evaluation metrics used, hardware, and code were used to conduct the set of experiments.

### 4.1 Dataset Description

All experiments in this paper use the ModelNet [19] dataset in the form that they need it in. ModelNet is a dataset containing 12,311 CAD-generated meshes across 40 different classes, 9,843 meshes set aside for training and 2,468 for testing [27].

#### *4.1.1 PointNet and SampleNet*

PointNet and SampleNet use a point cloud representation of ModelNet. Additionally, the experiments in this thesis use the entire dataset.

#### *4.1.2 MVCNN*

MVCNN uses multiple images to represent every mesh in the ModelNet dataset. Additionally, the experiments in this paper use a subset of the dataset found on the MVCNN web page, consisting of 10 classes with 100 objects per class, 80 of which are for training and 20 for testing.

### 4.2 Evaluation Measures

Overall, the metrics for all tests in this paper are accuracy and training time. All tests in this paper use either validation accuracy after each epoch as a metric for comparison or evaluation accuracy. The MVCNN tests use average, maximum,

minimum, and median accuracies as evaluation accuracies as each method was tested multiple times. Additionally, the MVCNN tests use average training time for each trial.

### 4.3 Hardware and Code Used

All tests were done using a RTX 2080 super GPU, Ryzen 3700x CPU, and 32 gigabytes of RAM. The tests were conducted on the Ubuntu operating system with the assistance of Docker [28].

#### *4.3.1 PointNet and SampleNet Code*

For the experiments done in this paper, the code from the SampleNet GitHub page [7] were used along with the Docker image provided by that GitHub page. Implementations of PointNet and SampleNet were included in the code, so both PointNet and SampleNet were tested using the code found on this GitHub page. By default, PointNet uses random sampling, so Furthest Point sampling was implemented manually based on the Chainer [8] implementation of the code. This code uses python 3.6.9 along with Tensorflow 1.13.2. The sampling for PointNet random sampling happened at the beginning of each epoch while the Furthest Point sampling happened as a preprocessing step before training.

#### *4.3.2 MVCNN Code*

For the experiments done in this paper, the implementation of MVCNN from MVCNN with CRF-RNN [9] was used. The CRF-RNN portion was ignored. A subset of the image data found at the MVCNN website [3] was used. Some custom python scripts were made to organize the data in a way that worked with this MVCNN implementation.

By default, this implementation chooses the first views found, so if there are 12 total views and 6 views were to be selected, the first 6 would be selected each epoch. A modification was made so that this could be 6 random views selected per epoch or 6 evenly spaced views instead of the first 6. The resolution reduction was done at the beginning of the neural network with a max pool function, which was also not included in the original code.

## 4.4 Investigated Issues

### *4.4.1 PointNet and SampleNet Classification Performance Comparison*

We investigate the effect of varying the methods of sampling and the number of points sampled on evaluation accuracy and training time.

The suite of experiments that are performed are as follows:

1. Use PointNet random sampling and vary between using 32, 64, 128, 256, 512, and 1024 points.
2. Use PointNet Furthest Point sampling and vary between using 32, 64, 128, and 256 points.
3. Use SampleNet sampling and vary between using 32, 64, 128, 256, and 512 points.

### *4.4.2 MVCNN*

We investigate the effect of varying the methods of sampling, the number of views sampled to, batch size, and number of epochs of training on average, maximum, minimum, and median accuracies as well as average training time.

The suite of experiments that are performed are as follows:

1. Fix Batch size to 4 and number of epochs to 16 and vary sampling method between first views, random views, selected views, resolution reduction with first views, resolution reduction with random views, and resolution reduction with selected views, additionally vary number of views used between 3, 4, 6, 8, and 12 views.
2. Fix Batch size to 1 and number of epochs to 16 and vary sampling method between first views, random views, selected views, resolution reduction with first views, resolution reduction with random views, and resolution reduction with selected views, additionally vary number of views used between 3, 4, 6, 8, and 12 views.
3. Fix Batch size to 16 and number of epochs to 64 and vary sampling method between first views, random views, selected views, resolution reduction with first views, resolution reduction with random views, and resolution reduction with selected views, additionally vary number of views used between 3, 4, 6, 8, and 12 views.

# Chapter 5

## Experimental Results and Analysis

In this chapter, we present the results from the experiment outlined in chapter 4 along with an analysis of the experimental results.

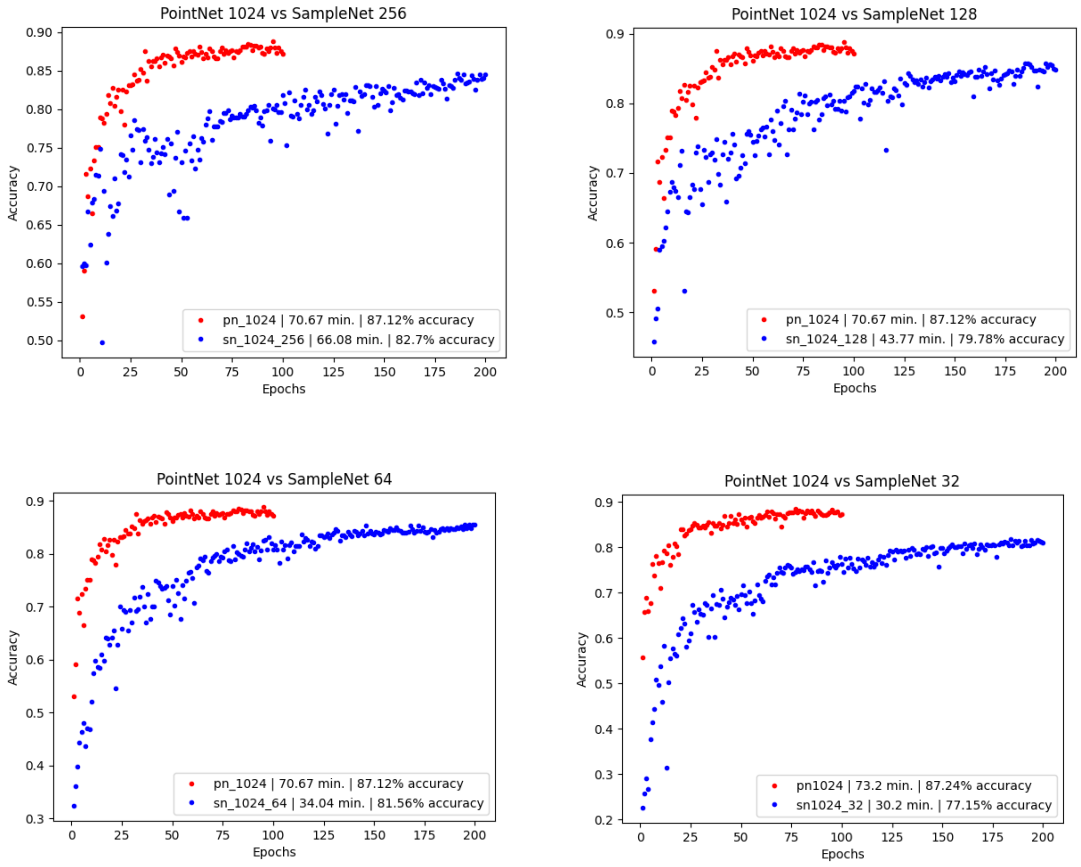
### 5.1 Sampling with PointNet Versus SampleNet

The set of graphs in Figure 2 show the validation accuracy after each epoch of PointNet (red) with 1024 points sampled using random sampling versus SampleNet (blue) with various amounts of points sampled along with the testing accuracy in the legend at the bottom right of each graph. PointNet 1024 was run twice, taking 70.7 minutes and 73.2 minutes to complete 100 epochs respectively while having accuracies of 87.1% and 87.2% respectively. The graph at the bottom right is the only graph that uses the second run of PointNet. Each run of SampleNet ran for 200 epochs.

SampleNet 256 (top left) took 66.1 minutes to complete with an 82.7% accuracy, SampleNet 128 (top right) took 43.8 minutes to complete with a 79.8% accuracy, SampleNet 64 (bottom left) took 34 minutes to complete with an 81.6% accuracy, and SampleNet 32 (bottom right) took 30.2 minutes to complete with an accuracy of 77.2%.

**Figure 2**

*PointNet With 1024 Randomly Sampled Points (PointNet 1024) vs SampleNet With Different Sampling Size*




*Note.* Compares Classification Performance for PointNet with 1024 randomly sampled points (PointNet 1024) vs SampleNet with different sampling size x = 32, 64, 128, 256 (SampleNet x)

SampleNet does not maintain accuracy while reducing the amount of work needed. In Figure 2, the default PointNet implementation with 1024 points is considerably more accurate than any SampleNet run. The closest accuracy that SampleNet has to the default PointNet implementation is 82.7% accuracy with 256 points sampled, which is a whole 4.42% less accurate than PointNet's 87.12% accuracy using

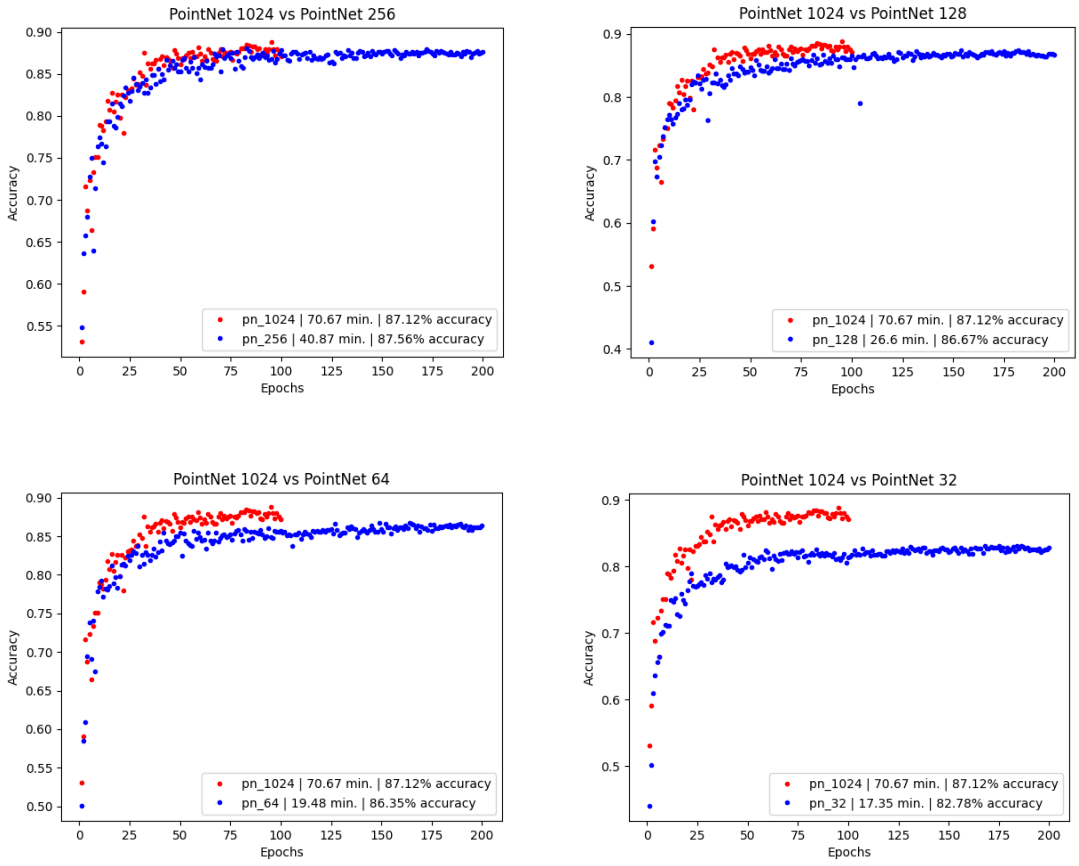
1024 points. This significant decrease in accuracy comes saving roughly 4 minutes of training time, which is only 7% faster than PointNet with 1024 points.

The set of graphs in Figure 3 show the validation accuracy after each epoch of PointNet with 1024 points sampled (red) versus PointNet with various amounts of points sampled (blue), both using random sampling, along with the testing accuracy in the legend at the bottom right of each graph. PointNet 1024 took 70.7 minutes to complete 100 epochs while having an accuracy of 87.1%. Every run of PointNet 256, 128, 64, and 32 with ran for 200 epochs.

PointNet 256 (top left) took 40.87 minutes to complete with an 87.6% accuracy, PointNet 128 (top right) took 26.6 minutes to complete with an 86.7% accuracy, PointNet 64 (bottom left) took 19.48 minutes to complete with an 86.4% accuracy, and PointNet 32 (bottom right) took 17.4 minutes to complete with an accuracy of 82.8%.

**Figure 3**

*Comparing Classification Performance for PointNet 1024 vs PointNet With Smaller Sample Sizes*

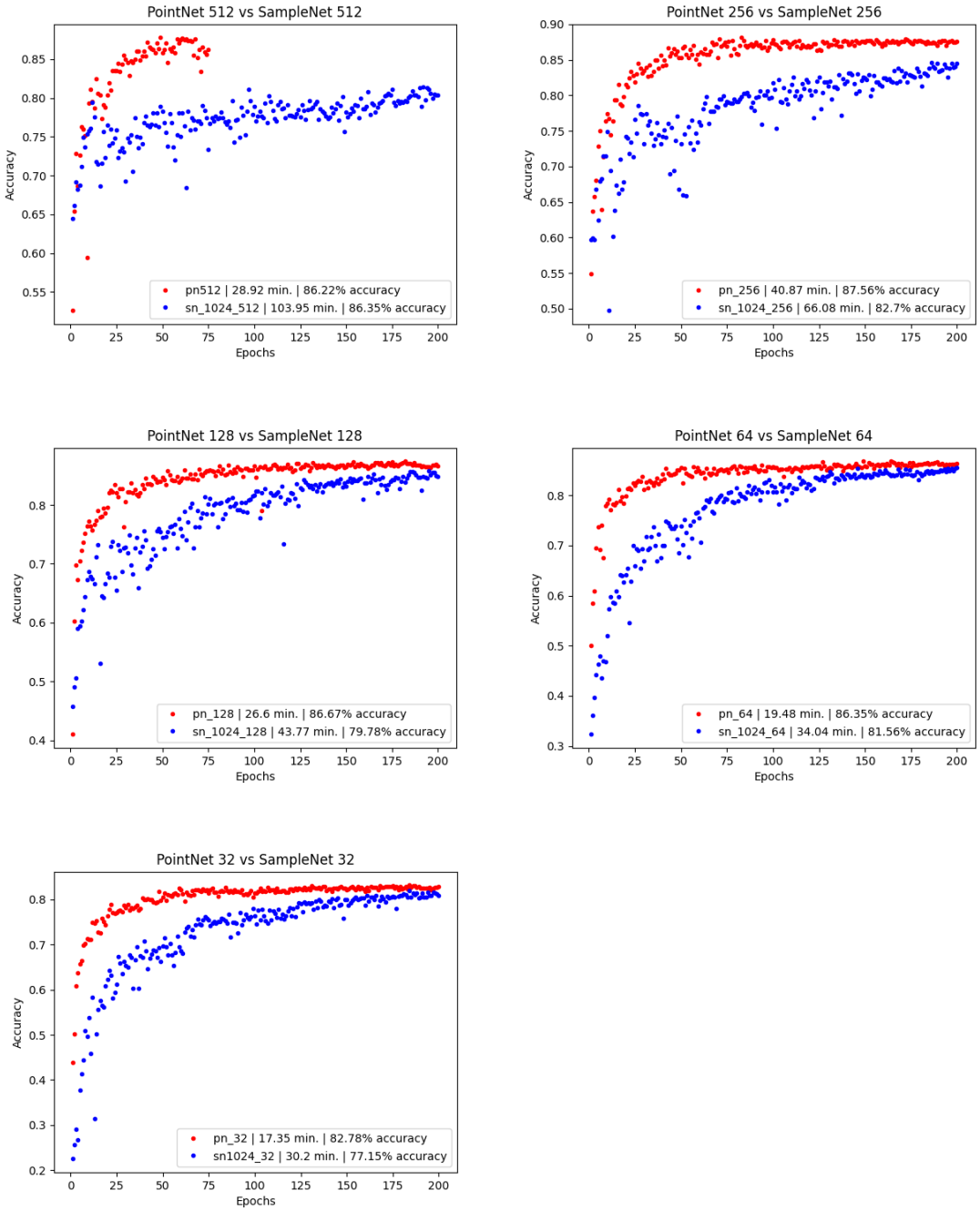



The default PointNet sampling method, which is uniform random sampling, performs much better than SampleNet. In Figure 3, the difference between 1024 points and smaller sample sizes is noticeable, but not significant until the number of points sampled is in the double digits. The evaluation accuracy of using 256 points sampled is comparable to the accuracy of using 1024 points. There is a 0.44% accuracy difference between 1024 points and 256 points sampled in favor of 256 points sampled. Sampling to 256 points improved speed by 72.9%, which is a considerable speed up for no accuracy

lost. This is reflected in the top left graph of Figure 3, where the validation accuracy of using 256 points sampled is slightly lower than using 1024 points but catches up once it is allowed to run for more epochs. The gap between using 1024 points and sampling to 128 or 64 points is apparent, although still small. The speed up is larger, with sampling to 128 points being 165.7% faster and sampling to 64 points being 262.8% faster than using 1024 points. Sampling to 32 points is significantly less accurate than using 64 or more points while only being 12% faster than using 64 points.

**Figure 4**

*Comparing Classification Performance for PointNet vs SampleNet With Fixed Sample Size*

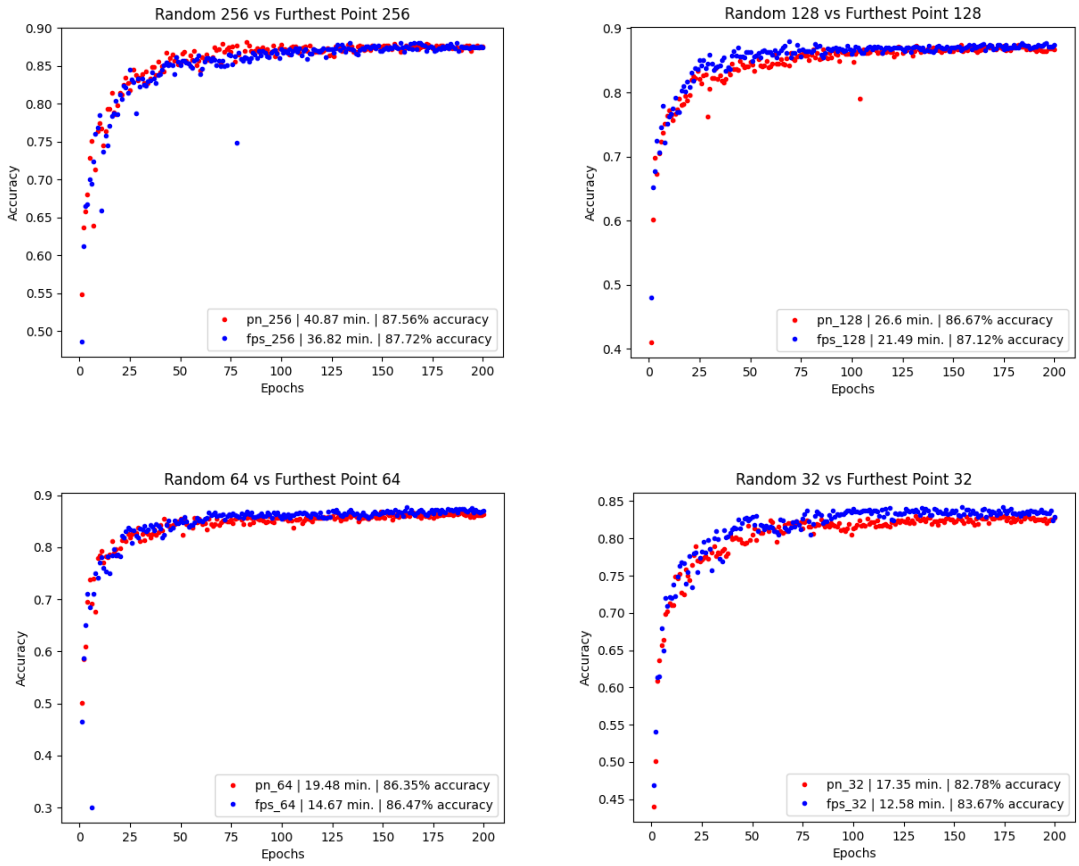


The set of graphs in Figure 4 show the validation accuracy after each epoch of PointNet (red) with various amounts of points sampled using random sampling versus SampleNet (blue) with various amounts of points sampled along with the testing accuracy in the legend at the bottom right of each graph. The number of points sampled is the same for both methods in each graph. Each method was run for 200 epochs with the exception of PointNet 512, which ran for 75 epochs.

Comparing PointNet random sampling with SampleNet with the same number of points sampled shows how random sampling is better. Throughout the entirety of Figure 4, PointNet sampling consistently outperforms SampleNet. The gap between the validation accuracies after each epoch is significant between sampling methods as well as the evaluation accuracies and time spent training, all in favor of random sampling. SampleNet starts to catch up in accuracy towards the later epochs, but still takes around 70% more time to complete than random sampling.

**Figure 5**

*Comparing Classification Performance for PointNet Random vs Furthest Point Sampling With Fixed Sample Size*




The set of graphs in Figure 5 show the validation accuracy after epoch of PointNet with random sampling (red) with various amounts of points sampled versus PointNet with furthest point sampling (blue) with various amounts of points sampled along with the testing accuracy in the legend at the bottom right of each graph. The number of points sampled is the same for both methods in each graph. Each method ran for 200 epochs.

Furthest Point 256 (top left) took 36.82 minutes to complete with an 87.7% accuracy, Furthest Point 128 (top right) took 21.5 minutes to complete with an 87.12% accuracy, Furthest Point 64 (bottom left) took 14.67 minutes to complete with an 86.5% accuracy, and Furthest Point 32 (bottom right) took 12.6 minutes to complete with an accuracy of 83.7%.

PointNet using random sampling is not too different from PointNet using furthest point sampling. The accuracy of each, shown in Figure 5, is within 1% of each other for each respective sampling size. However, FPS is slightly more accurate in all cases, and this is most apparent when sampling to 32 points. The validation accuracies after each epoch are relatively close, with FPS being slightly more accurate and learning faster than random sampling, which is reflected in the evaluation accuracy of FPS also being higher. It is important to note that furthest point technically did take less time, but that is because the furthest point sampling algorithm took place before training, while random sampling happened before each epoch. The furthest point algorithm does start with a random point, so it could possibly benefit from sampling before each epoch, but also requiring an algorithm that can run on the GPU. The CPU implementation takes too long to complete before each epoch, especially for an algorithm that will result in a similar structure on across different runs on the same set of points.

**Table 1**

*Comparing Classification Performance and Training Time for PointNet Random vs PointNet FP vs SampleNet Sampling Accuracy*

Accuracy

| | 1024 | 512 | 256 | 128 | 64 | 32 |
|---|---|---|---|---|---|---|
| Random | 87.12 | 86.22 | 87.56 | 86.67 | 86.35 | 82.78 |
| Furthest Point | - | - | 87.72 | 87.12 | 86.47 | 83.67 |
| SampleNet | - | 86.35 | 82.7 | 79.78 | 81.56 | 77.15 |

Time

| | 1024 | 512 | 256 | 128 | 64 | 32 |
|---|---|---|---|---|---|---|
| Random | 70.67 | 28.92 | 40.87 | 26.6 | 19.48 | 17.35 |
| Furthest Point | - | - | 26.82 | 21.49 | 14.67 | 12.58 |
| SampleNet | - | 103.95 | 66.08 | 43.77 | 34.04 | 30.2 |

*Note.* The column names are the number of points sampled to and the row names are the sampling methods used. The accuracies are percentages, and the time is in minutes.

Table 1 shows the accuracy and time values for each method and number of points sampled to pair.

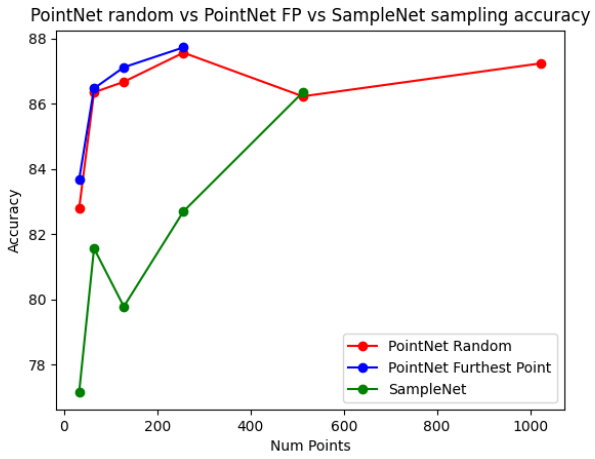
With 512 points, PointNet random is 0.13% less accurate than SampleNet. However, SampleNet also ran for many more epochs. Also, in the top left graph of Figure 4, SampleNet's validation accuracy is much lower than the test accuracy. With 256 points, PointNet random is 4.86% more accurate than SampleNet and PointNet took 25.2 less minutes than SampleNet to complete. With 128 points, PointNet random is 6.89% more accurate than SampleNet and PointNet took 17.2 less minutes than SampleNet to

complete. With 64 points, PointNet random is 4.79% more accurate than SampleNet and PointNet took 14.6 less minutes than SampleNet to complete. With 32 points, PointNet is 5.63% more accurate than SampleNet and PointNet took 12.9 less minutes than SampleNet to complete.

With 256 points, Furthest Point is 0.12% more accurate than random and Furthest Point took 4 less minutes than random to complete. With 128 points, Furthest Point is 0.45% more accurate than random and Furthest Point took 5.1 less minutes than random to complete. With 64 points, Furthest Point is 0.12% more accurate than random and Furthest Point took 4.8 less minutes than random to complete. With 32 points, Furthest Point is 0.89% more accurate than random and Furthest Point took 4.77 less minutes than random to complete.

**Figure 6**

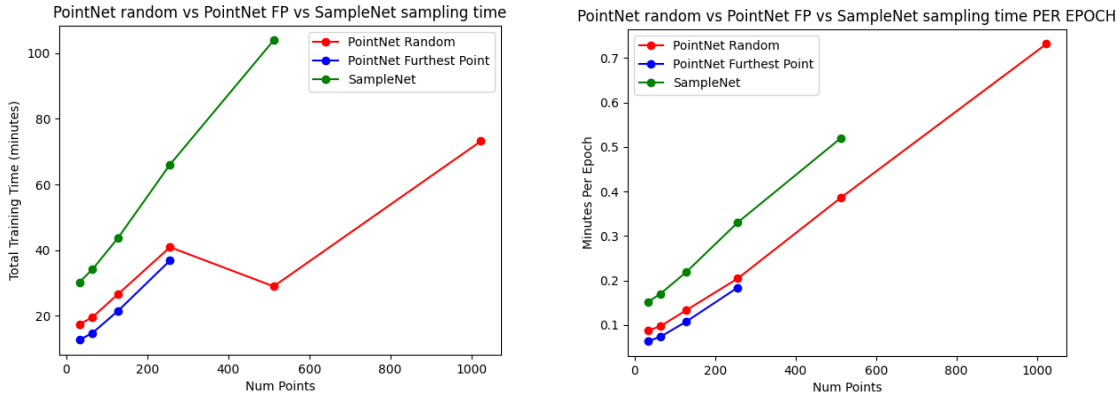
*Comparing Classification Performance for PointNet Random vs PointNet FP vs SampleNet Sampling Accuracy*



The graph in Figure 6 shows the accuracy of using different numbers of points when sampling. Num points is the number of points used when sampling, ranging from 32 to 1024 points sampled. The specific accuracies of each are found in Table 1.

**Figure 7**

*Comparing Training Time for PointNet and SampleNet Sampling*



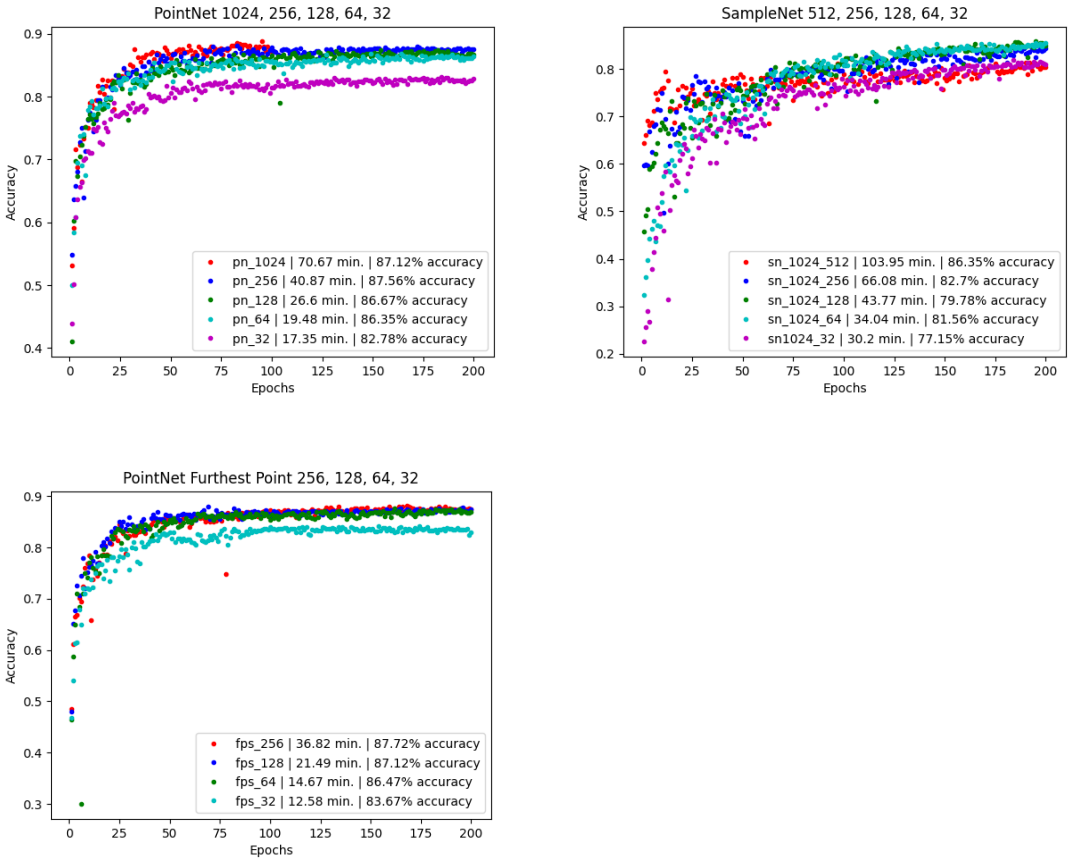

The set of graphs in Figure 7 show the amount of time training took with different numbers of points when sampling with PointNet Random (red), PointNet Furthest Point(blue), and SampleNet (green). Num points is the number of points used when sampling, ranging from 32 to 1024 points sampled. The left graph shows the total amount of time taken for that method to run and the right graph shows the amount of time per epoch per method.

Looking at the training time per epoch graph in Figure 7, it is confirmed that PointNet takes a linear amount more time with more points used, which also stands true for SampleNet. The slopes of each method are linear and almost identical to each other,

with SampleNet starting with a higher time per epoch. PointNet took less time per epoch with 128 points than SampleNet with 32 points. Additionally, PointNet with 256 points took less time per epoch than SampleNet with 128 points.

**Figure 8**

*Comparing Classification Performance for PointNet and SampleNet 1024, 256, 128, 64, 32*




The set of graphs in Figure 8 show the validation accuracy after each epoch of each sampling method, comparing various amounts of points sampled.

The top left graph shows the accuracy of PointNet using Random sampling and different amounts of points sampled to. Red is 1024 points, blue is 256 points, green is 128 points, cyan is 64 points, and purple is 32 points. The final evaluation accuracies and how long training took are on the graph legend.

The top right graph shows the accuracy of SampleNet using different amounts of points sampled to. Red is 512 points, blue is 256 points, green is 128 points, cyan is 64 points, and purple is 32 points. The final evaluation accuracies and how long training took are on the graph legend.

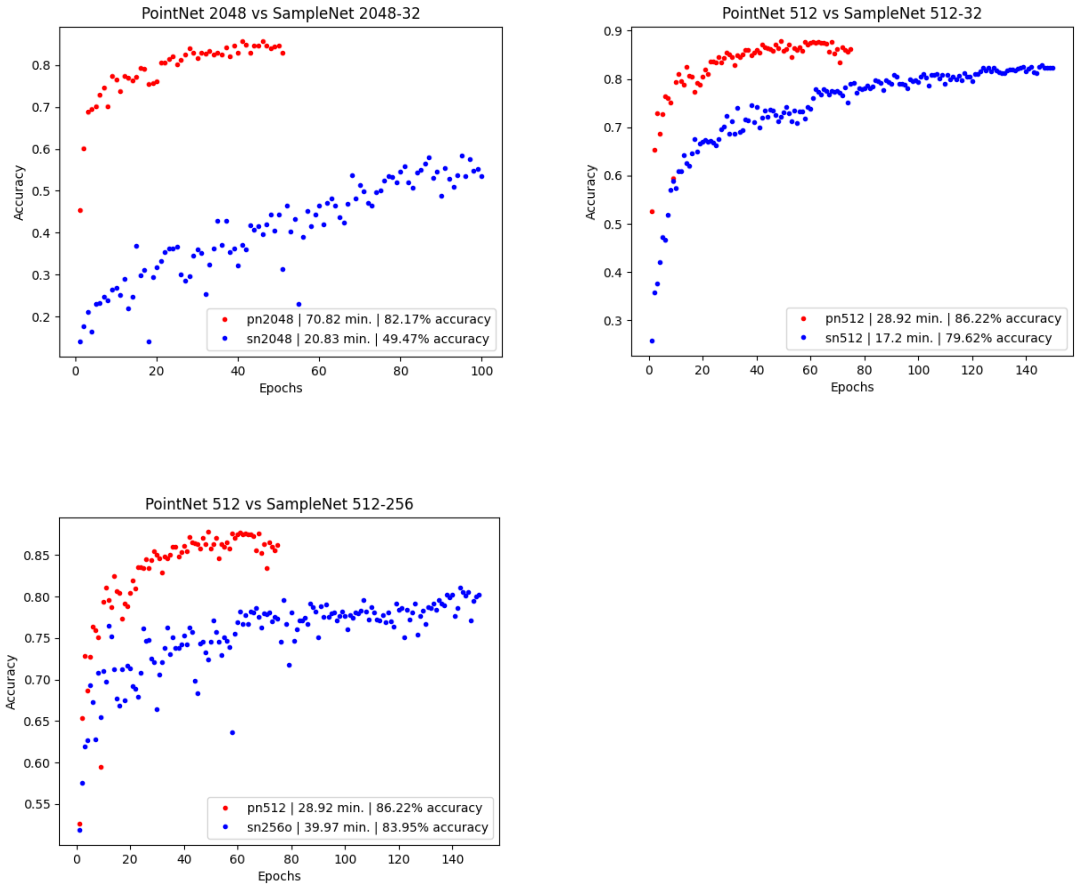
The bottom left graph shows the accuracy of PointNet using Furthest Point sampling and different amounts of points sampled to. Red is 256 points, blue is 128 points, green is 64 points, and cyan is 32 points. The final evaluation accuracies and how long training took are on the graph legend.

Comparing the methods with themselves at different number of points sampled also shows some interesting results. In Figure 8, the two PointNet sampling methods show steep increases in accuracy at the beginning of training followed by a plateau of validation accuracy, improving at a slow rate. This stays true for any number of points sampled. The validation accuracy after each epoch between the number of points sampled for random sampling seems to be more spread than with furthest point sampling. SampleNet also improves the validation accuracy after epoch quickly, but this sharp increase stops sooner than in PointNet, leading into a somewhat linear increase in accuracy after that. This is technically an improvement over PointNet as SampleNet seems like it will plateau at a later epoch than PointNet. However, SampleNet is still

much less accurate than PointNet and would take significantly longer to train to have a similar accuracy to PointNet. These observations stay true for any number of points sampled with SampleNet.

**Figure 9**

*SampleNet With Different Number of Points in Initial Point Cloud and PointNet Using 2048 Points*




The set of graphs in Figure 9 are some auxiliary graphs that showcase some other situations. Ordinarily, SampleNet first uses Furthest Point Sampling (FPS), the type of

sampling PointNet uses, before doing further sampling. These graphs show the effect of changing the number of points sampled from FPS compared to the PointNet counterpart.

In PointNet 2048 vs SampleNet 2048-32 (top left), PointNet (red) is sampled to 2048 points while SampleNet first samples to 2048 points using random sampling, then to 32 points using the SampleNet in the second part of sampling. PointNet took 70.8 minutes to complete 50 epochs with an accuracy of 82.2% while SampleNet took 20.83 minutes to complete 100 epochs with an accuracy of 49.47% accuracy.

In PointNet 512 vs SampleNet 512-32 (top right) and PointNet 512 vs SampleNet 512-256 (bottom left), PointNet (red) took 28.92 minutes to complete 75 epochs with an accuracy of 86.2%. SampleNet 512-32 took 17.2 minutes to complete 150 epochs with an accuracy of 79.6%. SampleNet 512-256 took 40 minutes to complete 150 epochs with an accuracy of 84%.

Overall, the best type of sampling to use with PointNet is furthest point sampling. It maintains accuracy until there are about 64 points sampled. Random sampling also maintains accuracy in the same way but is less accurate to start. Sampling to 256 points seems to be a sweet spot where accuracy is above sampling using 1024 points and training time is significantly reduced. Sampling to 64 points will still keep the accuracy within 1% of using 1024 points, along with any number of points above 64. SampleNet does not maintain accuracy while taking significantly longer to train than either PointNet method.

## 5.2 Different Types of Sampling With MVCNN

### Table 2

*MVCNN Test Results – Batch Size 4*

AVERAGES

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 90.1 | - | - | 89.8 | - | - |
| 8 | 90.3 | 91.3 | 89.9 | 90.1 | 89.2 | 87.4 |
| 6 | 93.0 | 91.9 | 91.1 | 86.6 | 91.5 | 88.2 |
| 4 | 91.1 | 92.9 | 90.8 | 92.6 | 89.6 | 88.0 |
| 3 | 91.5 | 90.2 | 89.4 | 89.0 | 89.1 | 87.3 |

MAX

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 94.0 | - | - | 94.0 | - | - |
| 8 | 93.0 | 95.0 | 92.0 | 93.0 | 92.0 | 91.0 |
| 6 | 95.0 | 94.0 | 94.0 | 91.0 | 95.0 | 90.0 |
| 4 | 95.0 | 95.0 | 93.0 | 95.0 | 92.0 | 90.0 |
| 3 | 93.0 | 96.0 | 92.0 | 92.0 | 94.0 | 92.0 |

MIN

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 86.0 | - | - | 87.0 | - | - |
| 8 | 88.0 | 87.0 | 87.0 | 88.0 | 87.0 | 84.0 |
| 6 | 92.0 | 88.0 | 88.0 | 84.0 | 88.0 | 81.0 |
| 4 | 86.0 | 88.0 | 87.0 | 89.0 | 87.0 | 84.0 |
| 3 | 89.0 | 75.0 | 86.0 | 87.0 | 83.0 | 82.0 |

RANGE

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 8.0 | - | - | 7.0 | - | - |
| 8 | 5.0 | 8.0 | 5.0 | 5.0 | 5.0 | 7.0 |
| 6 | 3.0 | 6.0 | 6.0 | 7.0 | 7.0 | 9.0 |
| 4 | 9.0 | 7.0 | 6.0 | 6.0 | 5.0 | 6.0 |
| 3 | 4.0 | 21.0 | 6.0 | 5.0 | 11.0 | 10.0 |

MEDIAN

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 90.0 | - | - | 89.0 | - | - |
| 8 | 90.0 | 91.0 | 90.5 | 90.0 | 89.5 | 87.0 |
| 6 | 93.0 | 92.0 | 91.5 | 86.5 | 91.5 | 89.0 |
| 4 | 91.5 | 94.0 | 91.0 | 93.0 | 90.0 | 89.0 |
| 3 | 92.0 | 91.0 | 89.5 | 89.0 | 89.5 | 87.5 |

TIME

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 264.6 | - | - | 141.73 | - | - |
| 8 | 203.65 | 203.74 | 204.07 | 105.55 | 105.77 | 105.62 |
| 6 | 175.47 | 175.63 | 175.65 | 91.52 | 90.3 | 90.37 |
| 4 | 148.23 | 148.38 | 148.2 | 74.7 | 74.71 | 74.68 |
| 3 | 134.74 | 134.75 | 134.75 | 66.8 | 66.7 | 66.9 |

Table 2 consists of 6 separate sub-tables, due to each method being tested 10 separate times, training for 16 epochs per trial. There are 6 sampling methods listed at the top – firstx-rot, random-rot, selected-rot, reduction-firstx, reduction-random, and

reduction-selected. Firstx-rot selects the first few views, random-rot selects a few random views, and selected-rot selects evenly spaced views (for example, 6 “selected” views would be every other view if there were 12 views total for an object). Reduction-firstx selects the first few views along with reducing the resolution of the images to half the original value for each axis, reduction-random selects a few random views along with reducing the resolution of the images, and reduction-selected selects evenly spaced views along with reducing the resolution of the images.

There are also 5 different number of views used on the left. Since there were 10 trials for each view-method combination, the average, maximum, minimum, and median accuracies get their own tables. The range table is just the maximum minus the minimum accuracies. The time table displays the average time in seconds that one training run takes for that view-method combination.

Each MVCNN sampling method found in Table 2 were tested 10 times each, which is why the sub-tables have maximum, minimum, and average accuracies. The top left entry in each table (firstx-rot 12) does not use any sampling method. Most of the view sampling methods maintain an accuracy around or higher than using no sampling at all. In particular, using the first 6 views yields the best overall results, having the second highest maximum accuracy of 95%, the highest minimum accuracy of 92%, and the highest average accuracy. The highest accuracy of 96% was achieved using 3 random views, but the lowest accuracy of 75% was also achieved using 3 random views. This makes using 3 random views one of the more volatile than other methods of sampling, but it still maintains the average accuracy from using no sampling. In all non-reduction methods, using 6 views seems to be the most accurate, while selecting every few views

tends to have the worst accuracy out of the non-reduction methods. The reduction methods all perform worse than the non-reduction counterparts with the exception of a couple methods.

The amount of time saved from reducing the amount of data used is not equal to the amount of data sampled. For example, looking at the Time section of Table 2, reducing the number of views to 6 or ½ the original amount reduces the training time to 2/3. This becomes more apparent when getting closer to ¼ of the original number of views, where the time spent on training is more than half of the original training time. This also applies to the resolution reduction as well, where using 2x2 resolution reduction results in the training time still taking over half of the no sampling training time. The combination of resolution reduction and view reduction results in much lower training times, but still do not maintain accuracy from using no sampling.

Since the resolution reduction implemented uses a 2x2 max pooling function, the amount of input information reduces to roughly a quarter of the original size, going from 224 pixels high and 224 pixels wide pixels to 112 pixels high and 112 pixels wide, reducing the number of pixels from 50176 to 12544. If the number of views is kept the same, then this method reduces the amount of information by the same amount as reducing the number of views to ¼ of the total number of views. In this case, resolution reduction has the equivalent amount of information as reducing the number of views to 3. Resolution reduction overall performs worse than view reduction with 3 views with the average and median accuracies of resolution reduction being lower than of using 3 views. The average time of one trial for view reduction is also lower than that of resolution reduction, making view reduction to 3 views strictly superior to resolution reduction.

**Table 3**

*MVCNN Test Results – Batch Size 1*

AVERAGES

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 91.1 | - | - | 87.7 | - | - |
| 8 | 90.56 | 90.9 | 90.3 | 89.5 | 90.6 | 88.78 |
| 6 | 91.3 | 92.11 | 89.0 | 88.8 | 92.0 | 87.6 |
| 4 | 90.9 | 89.6 | 89.4 | 92.1 | 89.0 | 87.4 |
| 3 | 90.88 | 90.33 | 88.44 | 89.3 | 87.3 | 84.9 |

MAX

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 94.0 | - | - | 94.0 | - | - |
| 8 | 93.0 | 94.0 | 93.0 | 92.0 | 94.0 | 91.0 |
| 6 | 95.0 | 95.0 | 94.0 | 92.0 | 96.0 | 93.0 |
| 4 | 94.0 | 91.0 | 93.0 | 95.0 | 92.0 | 90.0 |
| 3 | 94.0 | 93.0 | 92.0 | 93.0 | 91.0 | 89.0 |

MIN

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 86.0 | - | - | 75.0 | - | - |
| 8 | 87.0 | 88.0 | 86.0 | 81.0 | 86.0 | 86.0 |
| 6 | 88.0 | 89.0 | 85.0 | 86.0 | 89.0 | 84.0 |
| 4 | 86.0 | 87.0 | 87.0 | 89.0 | 84.0 | 85.0 |
| 3 | 85.0 | 86.0 | 84.0 | 85.0 | 84.0 | 82.0 |

RANGE

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 8.0 | - | - | 19.0 | - | - |
| 8 | 6.0 | 6.0 | 7.0 | 11.0 | 8.0 | 5.0 |
| 6 | 7.0 | 6.0 | 9.0 | 6.0 | 7.0 | 9.0 |
| 4 | 8.0 | 4.0 | 6.0 | 6.0 | 8.0 | 5.0 |
| 3 | 9.0 | 7.0 | 8.0 | 8.0 | 7.0 | 7.0 |

MEDIAN

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 91.0 | - | - | 89.5 | - | - |
| 8 | 91.0 | 90.5 | 90.0 | 90.5 | 91.0 | 89.0 |
| 6 | 91.5 | 92.0 | 88.0 | 89.0 | 92.0 | 87.5 |
| 4 | 91.5 | 90.0 | 89.0 | 92.0 | 89.0 | 87.5 |
| 3 | 91.5 | 91.0 | 88.0 | 89.5 | 87.5 | 84.5 |

TIME

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 721.74 | - | - | 420.18 | - | - |
| 8 | 594.39 | 594.37 | 594.7 | 329.12 | 329.22 | 329.2 |
| 6 | 537.1 | 537.12 | 537.1 | 289.83 | 290.17 | 290.3 |
| 4 | 479.79 | 479.73 | 479.86 | 247.96 | 247.88 | 247.99 |
| 3 | 455.69 | 457.89 | 456.31 | 230.52 | 230.97 | 230.37 |

Table 3 is similar to Table 2 and uses the same set of tables as but uses a batch size of 1 instead of a batch size of 4. Some full testing runs never actually learned and got stuck at 10% accuracy, so those entries were omitted. This only happened with a batch size of 1.

Table 3 also demonstrates many of the conclusions found from analyzing Table 2. Selecting the first few views maintains evaluation accuracy down to 3 views, while random view selection also maintains evaluation accuracy down to 6 views. Most other methods do not maintain accuracy. Selecting the first 6 views also performs the best overall. Interestingly, resolution reduction with no view reduction (reduction-firstx 12) takes less time per trial than reducing the number of views to 3 views, which was the opposite in Table 2. However, using view reduction down to 3 views maintains accuracy better than simple resolution reduction.

**Table 4**

MVCNN Test Results – Batch Size 16 with 64 Epochs

AVERAGES

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 92.8 | - | - | 90.0 | - | - |
| 8 | 93.6 | 91.8 | 93.4 | 89.2 | 88.8 | 91.2 |
| 6 | 91.25 | 93.4 | 93.4 | 92.0 | 92.6 | 89.4 |
| 4 | 95.2 | 94.0 | 93.4 | 94.2 | 87.8 | 90.8 |
| 3 | 94.4 | 91.0 | 90.4 | 91.6 | 90.6 | 90.8 |

MAX

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 94.0 | - | - | 91.0 | - | - |
| 8 | 95.0 | 94.0 | 96.0 | 91.0 | 91.0 | 93.0 |
| 6 | 95.0 | 95.0 | 96.0 | 94.0 | 95.0 | 92.0 |
| 4 | 97.0 | 96.0 | 94.0 | 95.0 | 89.0 | 92.0 |
| 3 | 97.0 | 95.0 | 92.0 | 93.0 | 93.0 | 93.0 |

MIN

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 91.0 | - | - | 89.0 | - | - |
| 8 | 92.0 | 87.0 | 89.0 | 88.0 | 86.0 | 89.0 |
| 6 | 83.0 | 92.0 | 89.0 | 88.0 | 91.0 | 86.0 |
| 4 | 93.0 | 92.0 | 93.0 | 93.0 | 85.0 | 88.0 |
| 3 | 90.0 | 88.0 | 88.0 | 89.0 | 87.0 | 90.0 |

RANGE

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 3.0 | - | - | 2.0 | - | - |
| 8 | 3.0 | 7.0 | 7.0 | 3.0 | 5.0 | 4.0 |
| 6 | 12.0 | 3.0 | 7.0 | 6.0 | 4.0 | 6.0 |
| 4 | 4.0 | 4.0 | 1.0 | 2.0 | 4.0 | 4.0 |
| 3 | 7.0 | 7.0 | 4.0 | 4.0 | 6.0 | 3.0 |

MEDIAN

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 94.0 | - | - | 90.0 | - | - |
| 8 | 94.0 | 93.0 | 94.0 | 89.0 | 89.0 | 91.0 |
| 6 | 93.5 | 93.0 | 95.0 | 93.0 | 92.0 | 89.0 |
| 4 | 95.0 | 93.0 | 93.0 | 94.0 | 88.0 | 91.0 |
| 3 | 95.0 | 91.0 | 91.0 | 92.0 | 92.0 | 90.0 |

TIME

| # of Views | firstx-rot | random-rot | selected-rot | reduction-firstx | reduction-random | reduction-selected |
|---|---|---|---|---|---|---|
| 12 | 557.58 | - | - | 366.15 | - | - |
| 8 | 386.87 | 390.5 | 392.68 | 257.39 | 256.72 | 258.44 |
| 6 | 314.92 | 313.88 | 316.33 | 205.44 | 204.62 | 207.91 |
| 4 | 236.81 | 236.69 | 239.51 | 151.29 | 151.61 | 154.21 |
| 3 | 198.42 | 198.89 | 201.33 | 124.28 | 123.72 | 126.5 |

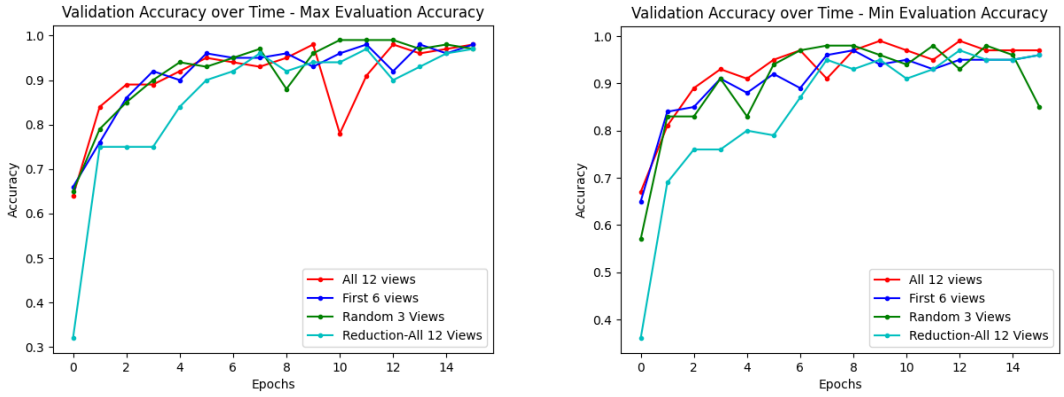
Table 4 is similar to Table 2 and uses the same set of tables but uses a batch size of 16 instead of 4 along with training for 64 epochs instead of 16 epochs. There were also only 5 training trials per method instead of 10 training trials per method.

Running each method for more epochs with a larger batch size gave some interesting results. First, the maximum evaluation accuracy was 97% instead of 96%. Also, the average accuracy of each sampling method increased. Additionally, using the first 6 rotations was not the most accurate and had one trial that had 83% evaluation accuracy, which was the lowest in this test run. This indicates that the trial using 3 random views in Table 2 that got an accuracy of 75% was due to luck to some extent. Most of the view reduction methods maintained accuracy at or above the method using no sampling, with the exception of a few methods, namely being 8 random views, 3 random views, and 3 selected views. Surprisingly, selected views mostly maintained accuracy in this test run. Additionally, almost all of the resolution reduction methods did not maintain accuracy. Comparing the average time per trial between choosing the 3 views against resolution reduction shows that resolution reduction takes almost double the amount of time as choosing the 3 views. This indicates that batch size potentially has

an effect on the time that each method takes to complete, as using a batch size of 1 showed that resolution reduction was faster while using a batch size of 4 and 16. A simple solution to this problem would be to simply select the views (if not using random view selection) or do the resolution reduction before training, as resolution reduction on the same image will always produce the same result. Likewise, view reduction that is not random will always select the same views to train on, so they do not need to be selected at the beginning of epoch. Instead, these can be done as a preprocessing step before training. Nonetheless, every method of reducing the views to 3 without resolution reduction are more accurate than resolution reduction.

**Figure 10**

*MVCNN Validation Accuracy Over Time*




The set of graphs in Figure 10 show the validation accuracy after each epoch of a few different methods of sampling. The left graph shows the accuracy after each epoch for the run with the maximum evaluation accuracy out of that sampling method's training

trials. The right graph is the same but for the minimum evaluation accuracy of the sampling method's training runs. All final performance results can be found in Table 2.

In both graphs, the baseline of all 12 views used (red) is firstx-rot with 12 in Table 2, which does not use any sampling method. The first 6 views sampling method (blue) is firstx-rot with 6 views on the table, 3 random views (green) is random-rot with 3 views, and all 12 views with resolution reduction (cyan) is reduction-firstx with 12 views.

In the left graph showing the maximum evaluation accuracies, the final evaluation accuracy of using all 12 views is 94%, using the first 6 views is gives an accuracy of 95%, using 3 random views give an accuracy of 96%, and using resolution reduction with no view reduction gave an accuracy of 94%.

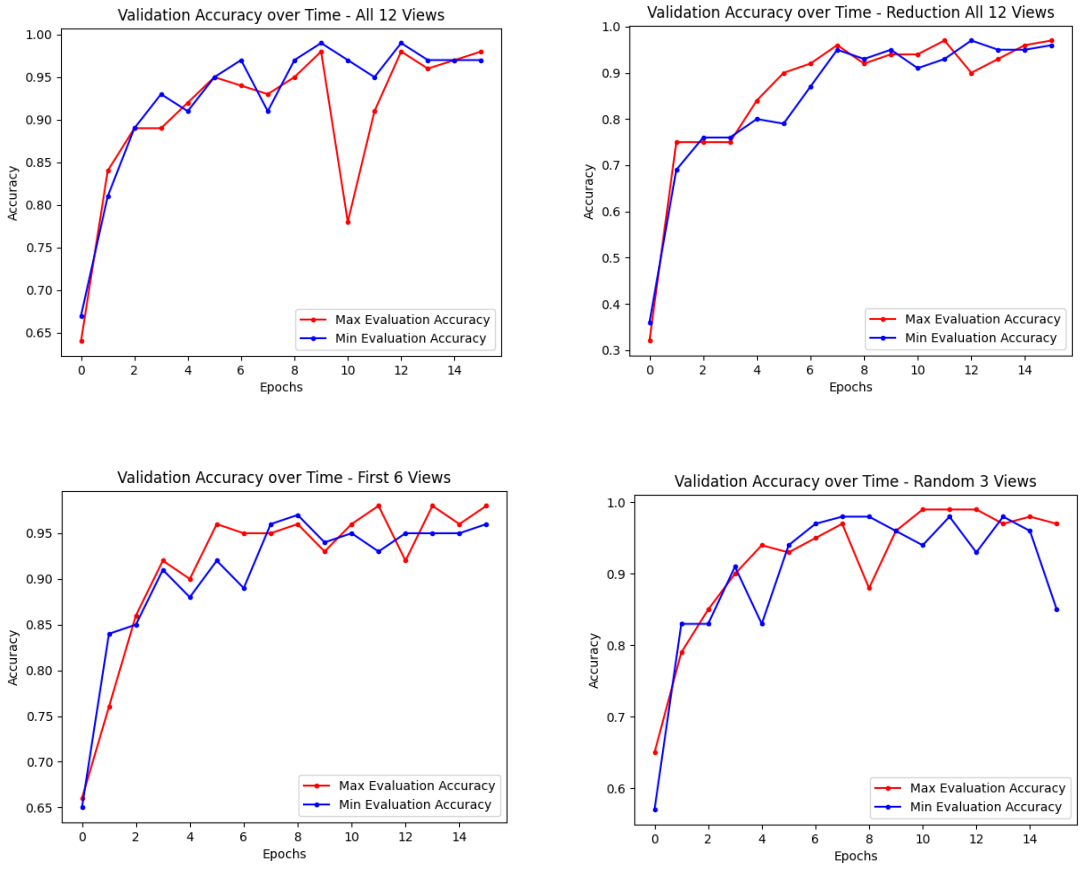
In the right graph showing the minimum evaluation accuracies, the final evaluation accuracy of using all 12 views is 86%, using the first 6 views is gives an accuracy of 93%, using 3 random views give an accuracy of 75%, and using resolution reduction with no view reduction gave an accuracy of 87%.

Looking at Figure 10, the first 6 views method has the most stable validation accuracy throughout each epoch in both the maximum and minimum evaluation accuracy trials. The resolution reduction method starts off at a much worse accuracy as well as taking longer to learn toward the beginning of the trials, but it catches up by around 8 epochs into the trials. The two notable decreases in validation accuracy occur between epoch 9 and 10 of the maximum evaluation accuracy for using all 12 views, which it then recovers from to attain a high evaluation accuracy. The second decrease happens for the minimum evaluation accuracy trial for using 3 random views, which occurs at the end of

the trial, producing the model with the lowest accuracy of all the trials. Additionally, using 3 random views gives the highest validation accuracy of any of the selected trials in Figure 10.

**Figure 11**

*MVCNN Validation Accuracy - Maximum vs Minimum Evaluation Accuracy*




The set of graphs in Figure 11 take the max evaluation accuracy training runs (red) of each sampling method and compares it with the minimum evaluation training runs (blue) of each respective sampling method, using the validation accuracy after each

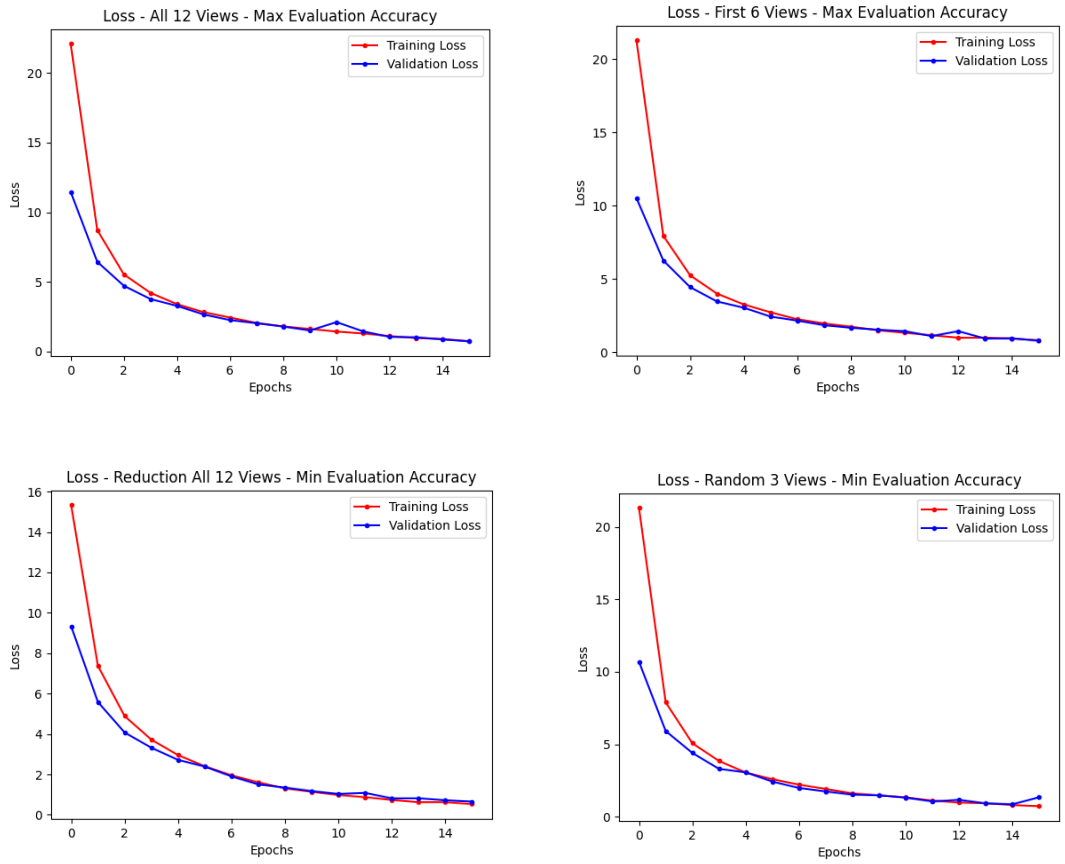
epoch. The top left graph is the baseline of all 12 views being used, or no sampling. The top right graph is the resolution reduction with all 12 views sampling method, The bottom left graph is the first 6 views sampling method, and the bottom right is the random 3 views sampling method.

In the top left graph (all 12 views/no sampling), the maximum evaluation accuracy is 94% while the minimum evaluation accuracy is 86%. In the top right graph (resolution reduction using no view reduction), the maximum evaluation accuracy is 94% while the minimum evaluation accuracy is 87%. In the bottom left graph (first 6 views), the maximum evaluation accuracy is 95% while the minimum evaluation accuracy is 93%. In the bottom right graph (random 3 views), the maximum evaluation accuracy is 96% while the minimum evaluation accuracy is 75%.

Comparing each method's maximum and minimum evaluation accuracy trials with each other in Figure 11 shows the impact of random chance. Overall, the minimum accuracy trials had more major decreases in accuracy between 2 epochs than the maximum accuracy trials. This also applies to the beginning of the trials as well, where learning is the quickest. Additionally, the minimum accuracy trial of using 3 random views has a large decrease in accuracy at 4 epochs into the trial, which caused the rest of the epochs to be less stable than other trials. Apart from the one massive decrease in accuracy using all 12 views, using 3 random views have the most frequent large decreases in validation accuracy throughout any trial.

**Figure 12**

*MVCNN Sampling Method Loss*

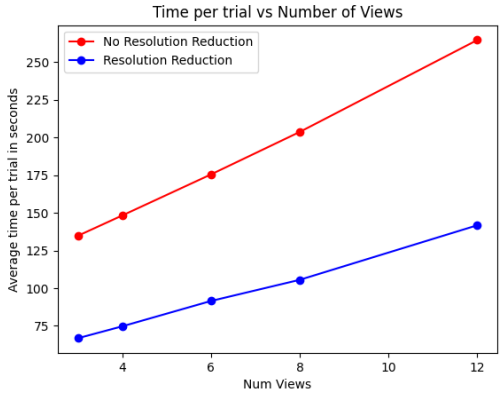



The set of graphs in Figure 12 show the training loss versus the validation loss of a few different training runs across a few sampling methods. The top two graphs use the maximum evaluation accuracy runs, while the bottom two graphs use the minimum evaluation accuracy runs for each respective sampling method. The top left graph is the baseline of all 12 views being used, or no sampling. The top right graph is the first 6 views sampling method, the bottom left graph is the resolution reduction with all 12 views sampling method, and the bottom right is the random 3 views sampling method.

The loss of each trial decreases at almost every epoch. Looking at Figure 12 shows that the validation loss after each epoch decreases almost every step. Some of the larger decreases in accuracy, like the final epoch of the minimum evaluation accuracy trial for 3 random views, are reflected in the respective loss graph, but the loss generally decreases after each epoch, regardless of the change in accuracy. For example, epoch 10 of the minimum evaluation accuracy trial for 3 random views sees a decrease in accuracy, but no rise in loss. Additionally, the loss graphs indicate that overfitting is not happening in these trials. Figure 13 indicates that reducing the number of views reduces the amount of time per trial linearly, which is to be expected.

**Figure 13**

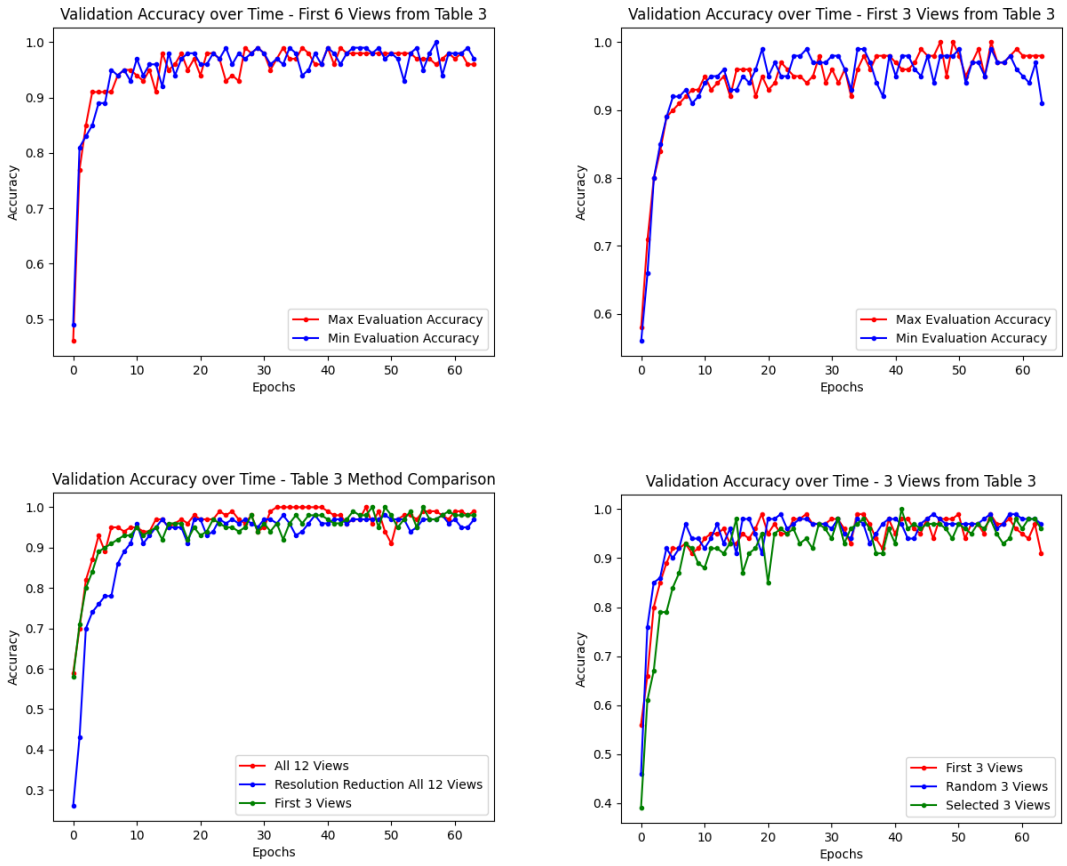
*MVCNN Average Time per Trial vs Number of Views*




The graph in Figure 13 shows the average time per trial of using no resolution reduction (red) and using resolution reduction (blue) with various numbers of views.

**Figure 14**

*MVCNN Validation Accuracy Using Various Sampling Methods (From Table 4)*




The set of graphs in Figure 14 show the validation accuracy after each epoch using various sampling methods from Table 4.

The top left graph compares the validation accuracy after each epoch of the maximum evaluation accuracy trial (red, 95% accuracy) and minimum accuracy trial (blue, 83% accuracy) using the first 6 views method.

The top right graph compares the validation accuracy after each epoch of the maximum evaluation accuracy trial (red, 97% accuracy) and minimum accuracy trial (blue, 90% accuracy) using the first 3 views method.

The bottom left graph shows the validation accuracy after each epoch using different sampling methods: no sampling (red, 94% accuracy), resolution reduction (blue, 91% accuracy), and first 3 views (green, 97% accuracy). This graph uses the maximum evaluation accuracy trial for each method.

The bottom right graph shows the validation accuracy after each epoch using view reduction down to 3 views using each method of doing so: first 3 views (red, 90% accuracy), random 3 views (blue, 88% accuracy), and selected 3 views (green, 88% accuracy). This graph uses the minimum evaluation accuracy trial for each method.

Training for more epochs and having a larger batch size seems to lead to more stable results, as shown in Figure 14. Both the minimum evaluation accuracy trial of using the first 6 views as well as using resolution reduction have a final epoch validation accuracy that is significantly higher than the final evaluation accuracy of the respective trial. For the first 3 views method, the validation accuracy decreased after the final epoch and was close to the evaluation accuracy of that trial. Looking at the top right and both bottom graphs of Figure 14, reducing the number of views to 3 views makes the validation between each epoch more unstable, having the validation accuracy decrease significantly then increase significantly repeatedly, than using more views.

Overall, sampling using view reduction will maintain accuracy while also reducing training time. Resolution reduction without view reduction using a max pooling

function does not maintain accuracy. Combining both resolution reduction and view reduction also does not maintain accuracy. However, combining resolution reduction and view reduction does significantly reduce training time, even when compared to the fastest view reduction only method.

# Chapter 6

## Conclusions and Future Work

We have presented empirical evidence that sampling can be used to reduce training time while maintaining evaluation accuracy. On a point cloud, sampling using random sampling or furthest point sampling will reduce training time significantly while still maintaining classification accuracy, with furthest point sampling performing slightly better than random sampling. The sampling found in SampleNet drastically decreases accuracy while increasing training time. On multiple images, sampling by view reduction maintains accuracy on a multi-view classifier while reducing processing time as well, while resolution reduction does not accomplish this task.

In the future, the most obvious test to conduct is applying the sampling methods used on MVCNN to VoxNet to see if VoxNet can also maintain accuracy while undergoing "view" reduction or resolution reduction. In theory, VoxNet is similar to MVCNN where VoxNet has "rotations" instead of "views" as well as having a specified resolution, so the observations from MVCNN should apply to VoxNet. However, sampling on VoxNet may not give the same results as sampling on MVCNN in practice.

Testing RotationNet [22] in the future would be good as it is claims to be the most accurate multi-view classifier and does not use every view. The RotationNet paper does not include training and inference time, so that would also need to be tested. The RotationNet method of correctly aligning a subset of views could also be applied to VoxNet to hopefully improve accuracy.

As for point cloud classifiers like PointNet, the idea of sampling using a differentiable function seems enticing as then it could be learned. However, SampleNet simply did not perform well. Perhaps the SampleNet learning method was too complex and a simpler but still differentiable sampling method is needed. Furthest point sampling maintains accuracy until around 64 points are sampled, so the neural network sampling the point cloud needs to be quicker than SampleNet or maintain accuracy past 64 points sampled while still taking less time than furthest point sampling with a higher number of points sampled. There could also be an undiscovered sampling method that performs better than furthest point sampling without learning.

## References


[1] Maturana, D., & Scherer, S. (2015). Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 922-928). IEEE.

[2] Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 652-660).

[3] Su, H., Maji, S., Kalogerakis, E., & Learned-Miller, E. (2015). Multi-view convolutional neural networks for 3d shape recognition. In Proceedings of the IEEE international conference on computer vision (pp. 945-953).

[4] Lang, I., Manor, A., & Avidan, S. (2020). Samplenet: Differentiable point cloud sampling. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 7578-7588).

[5] (2021). Furthest Point Sampling. Retrieved from https://minibatchai.com/ai/2021/08/07/FPS.html

[6] Dovrat, O., Lang, I., & Avidan, S. (2019). Learning to sample. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 2760-2769).

[7] Lang, I., Manor, A., & Avidan, S. (2020). Samplenet: Differentiable point cloud sampling. Retrieved from https://github.com/itailang/SampleNet

[8] (2018). Chainer-PointNet. Retrieved from https://github.com/corochann/chainer-pointnet/blob/master/chainer_pointnet/utils/sampling.py

[9] Lee, Ko., Yu, Y., Ha, B., & Lee, Kw. (2021). MVCNN with CRF-RNN for BIM. Retrieved from https://github.com/kwanhoonlee/crfrnn-mvcnn

[10] Roberts, L. G. (1963). Machine perception of three-dimensional solids (Doctoral dissertation, Massachusetts Institute of Technology).

[11] Fukushima, K., & Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In Competition and cooperation in neural nets (pp. 267-285). Springer, Berlin, Heidelberg.

[12] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[13] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). The MNIST Database. Retrieved from http://yann.lecun.com/exdb/mnist/

[14] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2010). The pascal visual object classes (voc) challenge. International journal of computer vision, 88(2), 303-338.

[15] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. International journal of computer vision, 115(3), 211-252.

[16] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25.

[17] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708).

[18] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., & Xiao, J. (2015). 3d shapenets: A deep representation for volumetric shapes. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1912-1920).

[19] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., & Xiao, J. (2015). ModelNet. Retrieved from https://modelnet.cs.princeton.edu/

[20] Riegler, G., Osman Ulusoy, A., & Geiger, A. (2017). Octnet: Learning deep 3d representations at high resolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3577-3586).

[21] Liu, Y., Fan, B., Meng, G., Lu, J., Xiang, S., & Pan, C. (2019). Densepoint: Learning densely contextual representation for efficient point cloud processing. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 5239-5248).

[22] Kanezaki, A., Matsushita, Y., & Nishida, Y. (2018). Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 5010-5019).

[23] Merino, I., Azpiazu, J., Remazeilles, A., & Sierra, B. (2021). 3D Convolutional Neural Networks Initialized from Pretrained 2D Convolutional Neural Networks for Classification of Industrial Parts. Sensors, 21(4), 1078.

[24] Azpiri, A. I., Ortega, E. L., & Cobo, A. A. (2021). Affordance-based grasping point detection using graph convolutional networks for industrial bin-picking applications. Sensors, 21(3), 816.

[25] Qian, R., Lai, X., & Li, X. (2021). 3D object detection for autonomous driving: a survey. arXiv preprint arXiv:2106.10823.

[26] Manzoor, S., Joo, S. H., Kim, E. J., Bae, S. H., In, G. G., Pyo, J. W., & Kuc, T. Y. (2021). 3D Recognition Based on Sensor Modalities for Robotic Systems: A Survey. Sensors, 21(21), 7120.

[27] Qiu, S., Anwar, S., & Barnes, N. (2021). Geometric back-projection network for point cloud classification. IEEE Transactions on Multimedia.

[28] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. Linux journal, 2014(239), 2.

[29] Yi, L., Pei, X., & Guo, Y. (2021). 3D CNN classification model for accurate diagnosis of coronavirus disease 2019 using computed tomography images. Journal of Medical Imaging, 8(S1), 017502.

[30] Eldar, Y., Lindenbaum, M., Porat, M., & Zeevi, Y. Y. (1997). The farthest point strategy for progressive image sampling. IEEE Transactions on Image Processing, 6(9), 1305-1315.

[31] Pauly, M., Gross, M., & Kobbelt, L. P. (2002). Efficient simplification of point-sampled surfaces. In IEEE Visualization, 2002. VIS 2002. (pp. 163-170). IEEE.

[32] Lin, Y., Chen, L., Huang, H., Ma, C., Han, X., & Cui, S. (2022). Task-Aware Sampling Layer for Point-Wise Analysis. IEEE Transactions on Visualization and Computer Graphics.

[33] Alexa, M., Rusinkiewicz, S., Nehab, D., & Shilane, P. (2004). Stratified point sampling of 3D models. In Proc. Eurographics Symp. on Point-Based Graphics (pp. 49-56).

[34] Taimori, A., & Marvasti, F. (2017). Measurement-Adaptive Sparse Image Sampling and Recovery. arXiv preprint arXiv:1706.03129.

[35] Dhawan, S. (2011). A review of image compression and comparison of its algorithms. International Journal of electronics & Communication technology, 2(1), 22-26.