Bayesian Optimisation of PointNet

Laxman Singh[1], Niranchana Periasamy[1], Siew Yaw Hoong[1], Tan Chee Wei[1]

Affiliations:
[1] M. Tech students, Institute of Systems Science, National University of Singapore, Singapore

ABSTRACT

Point clouds represent unstructured set of data points which are often generated by 3D scanners intended to measure 3D positions in space. PointNet is a simple but efficient algorithm that shows a novel way to process unordered 3D point cloud directly without pre-processing. PointNet is advantageous due to the invariance to input permutations by adopting a symmetry function that aggregates the information from each point in the point cloud. Our research focuses on PointNet and other related frameworks which utilizes point cloud to extract 3D model by point classification. In this study, we perform a comparative analysis between PointNet, PointNet++ and 3D Modified Fisher Vector(3DmFV) to investigate the likelihood of optimising PointNet using Bayesian Optimisation and automate hyperparameters selection to understand the effect of batch sizes on training time and accuracy.

Keywords: Point Cloud; PointNet; Neural Networks; Bayesian Optimisation;

1. INTRODUCTION

Point cloud is widely scanned using scanners, LiDAR sensors and RGBD cameras to represent the external surface of an object. Yet, to establish deep learning on point clouds comes with its own challenges related to the learning process. Point clouds are unstructured data distributed in space and invariant to input permutations.

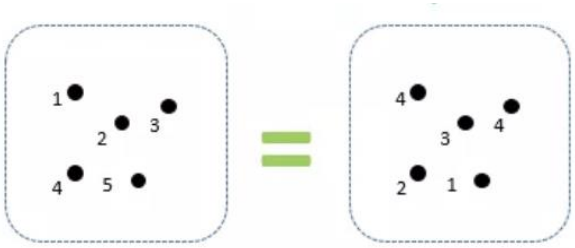


Fig 1: Point Cloud Geometry

#	x	y	z
1	x_1	y_1	z_1
2	x_2	y_2	z_2
3	x_3	y_3	z_3
4	x_4	y_4	z_4
5	x_5	y_5	z_5

≠

#	x	y	z
1	x_1	y_1	z_1
2	x_2	y_2	z_2
3	x_3	y_3	z_3
4	x_4	y_4	z_4
5	x_5	y_5	z_5

Fig 2: Permutation invariance - unordered points in underlying matrices

In this paper, we explore methods to improve the accuracy of object classification of point clouds using PointNet. Pointnet is a unified architecture that is capable of processing point clouds and outputting classification for the entire input or per point segment/part labels for each point of the input [1]. Point cloud is often represented using three coordinates (x,y,z) and/or with additional dimensions. PointNet uses single symmetric function, max pool approach as it's learning key. With this, the network learns a set of optimization functions to hand pick informative points of the point cloud and also encode the reason for the selection. The final fully connected layers aggregates these value to be able to perform shape classification on a global descriptor level.

Next, we study PointNet++, which is a hierarchical version of PointNet. As PointNet does not make use of the local structure by design, the new architecture uses three layers at every level to build a hierarchical grouping of points to abstract local regions into larger

ones. These three layers are Sample, Group and PointNet. The main idea behind this approach is to divide the point sets into overlapping local regions using a distance metric based on the local space. These local features can later be grouped and processed using Pointnet to learn the global features [2]. This method is able to improve on the result of the previous PointNet model.

Our paper also includes the study of 3DmFV (3D Modified Fisher Vector) which also uses point cloud as input. This model uses uses a point cloud hybrid representation which describes the point by their deviation from the Gaussian Mixture model (GMM) and with the grid like structures allows for the usage of ConvNets for the object classification. [3] This approach which differs from standard deep network principle uses non-learned features to represent the data however this model achieved the state of the art accuracy as compared to the other models that use point cloud as input.

Based on the study of the three models, we weigh upon the available optimization approaches which are designing a new architecture, a new loss function or a new training strategy, and given the limited time and resources available, we diverted the focus towards improving the training strategy for the Pointnet model by hyperparameters optimization using Bayesian optimization.

This optimization approach builds a probability model of the objective function that maps input values to a probability of a loss: p (loss | input values). This probability model, also called the surrogate or response surface, is computationally lighter to optimize than the actual objective function. Bayesian methods select the next values to evaluate by applying a criteria, Expected Improvement, to the surrogate. Each time the algorithm proposes a new set of candidate hyperparameters, it evaluates them with the actual objective function and records the result in a pair (score, hyperparameters). These records form the history which can be use to guide the algorithm to form a probability model of the objective function that improves with each iteration. Tree structured Parzen Estimator (TPE)'s best model had been shown to outperformed the best model found using random and manual search [5].

We applied this approach on the narrow set of batch size hyperparameters (2, 4, 8, 16, 32 and 64) [6] and study the results in the rest of this paper.

2. LITERATURE REVIEW

Point cloud gained prominence since 2017 when point cloud scanning was widely used by scanners for converting multiple point cloud to a 3D model. However, there are very few papers that apply deep learning on the point cloud directly.

Deep Learning on Point Cloud Point clouds are XYZ coordinates distributed in space and thus there is no structured grid. Also, it is invariant to permutations which means that the order of a point can be represented in two different matrices geometrically. PointNet was introduced as a novel architecture to apply deep learning on point cloud directly and also respects its invariance to input permutations. This also served as a unified framework for point classification and segmentation applications. It then formed as a basis for all future studies that involves applying neural network on point cloud. PointNet++ is enhanced version of PointNet architecture essentially introducing an hierarchical structure to point clouds. To capture local structures and fine-grained patterns at high definition(limitations of PointNet), PointNet is recursively applied on nested partitioned of input point set. By exploiting metric space distances, our network is able to learn local features with increasing contextual scales.

3D Modified Fisher Vector(3DmFV-Net) was later proposed to address severe memory size tradeoffs of PointNet. 3DmFV-Net architecture introduces hybrid method that combines discrete grid structure with continuous generalized Fisher vectors which is feature aggregation. Using the grid enables us to design a new CNN architecture for real-time point cloud classification. Kd-Network emphasize on using Kd-tree to create an order in the point cloud. The structure problem was solved by applying weights for each node in the tree. As our research focuses on applying deep learning on point cloud directly, we pursue our study basing PointNet and PointNet architecture.

Optimizing Model Performance Revisiting small batch training for deep Neural Network paper uses Mini-batch Stochastic Gradient Descent (SGD) optimization to encourage training data in small batches. It also claims that best performance is observed for mini-batch sizes between 2 and 32. This aids to bump up the overall model accuracy. Although this increases the training time, data parallelism can be induced to minimise the

impact. Algorithms for Hyper-Parameter Optimization focuses on automating parameter optimization to overcome human intervention in manually optimizing parameters to tweak the model performance.

3. DATA

In this study, we used the same data set that the Pointnet authors, namely the ModelNet40 database from Princeton University. The dataset consists of 12,311 CAD models from 40 man-made object categories, split into 5 training files with a total of 9843 items, and 2 testing files with 2468 items.

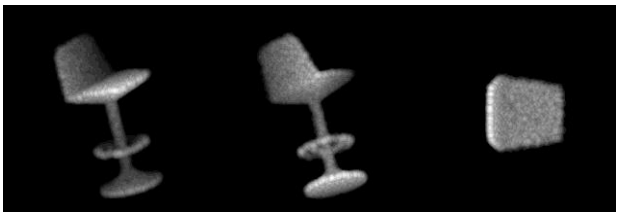


Fig 3: An example of the ModelNet40 3D CAD file used in Pointnet object classification

4. METHODOLOGY

There are 3 parts to our study:

4.1 Pointnet, Pointnet++ and 3DmFV verification

We ran the codes for the original Pointnet, Pointnet++ and 3DmFV to verify the results and confirm our system setup is correct. The original Pointnet study applied the algorithm on 3 types of applications – 3D object classification, object part segmentation, and semantic scene segmentation. Since the computational resources required for the applications are quite intensive, we focused our study on 3D object classification.

The training is done by running the training Python file, which creates a model that can be used to evaluate the test files. The evaluation generate an event log that can be visualised with Tensorboard, as well as a dump folder with all the misclassified object pictures.

4.2 Hyperparameter tuning with Bayesian optimisation

After comparing the 3 methods, we narrow down our study on hyperparameters optimisation of Pointnet in 3D object classification. We modified the Pointnet codes to work with the Hyperopt package in Python. The function searches a pre-defined space to return the optimum hyperparameters that minimise an objective function. Our target here is to maximise the classification accuracy, but since Hyperopt works with minimising a function, our objective function was set to negative accuracy.

Hyperparameters available for tuning and their default values in Pointnet are:

- Batch size - 32
- Number of points - 1024
- Max epoch - 250
- Learning rate - 0.001
- Momentum - 0.9
- Optimizer - Adam
- Decay step - 200,000
- Decay rate - 0.7

To limit the search space and processing time, we focused our tuning to batch size selection. Batch sizes selected are 2, 4, 8, 16, 32, and 64. Likewise, Hyperopt evaluation cycle was set to 50 with the epoch set to 12 (vis the original 250). While this may not provide enough epochs for the accuracy to converge, we believe that it's a fair trade-off that would still allow us to make a reasonable comparison between batch sizes.

4.3 Effect of batch size on classification accuracy

With the results from the Hyperopt, we conducted full 250-epoch runs of Pointnet classification for each batch size to verify the effect of batch sizes on training time, loss and accuracy. As training time is dependant on the computing setup, all processing from this part is done with a laptop with nVidia GeForce 1060 GPU.

Computational platform

The original Pointnet was written with Tensorflow in Python 2.7 for Ubuntu operating system. We adapted the files to Windows OS and ran them on Tensorflow and Python 3.6. Processing was done primarily with Graphics Processing Unit (GPU) to take advantage of its faster parallel computing capabilities. As the computational requirements for running the files are quite intensive, the work was distributed between the following cloud and local resources:

1. Google Cloud Platform with Tesla P100 GPU
2. Google Colaboratory with Tesla K80 GPU
3. Laptop with nVidia GeForce 1060 GPU

With our setup, each training cycle of 250 epochs of the 9843 training items takes about 12 hours. Evaluation cycles are considerably faster at a few minutes each.

However, the long training time limited the number of cycles that we can perform.

5. RESULTS

5.1 Pointnet, Pointnet++ and 3DmFV verification

The verification results between the 3 methods are shown in Table 1. From the table we can see that our results are very similar to the published results. The minor variations can be attributed to the inherent machine learning process.

	Training			Evaluation			Published
	Mean Loss	Accuracy	Average Class Accuracy	Mean Loss	Accuracy	Average Class Accuracy	Accuracy
Pointnet	0.552	0.885	0.859	0.546	0.883	0.856	0.892
Pointnet++	0.451	0.895	0.872	0.453	0.895	0.873	0.907
3DmFV	0.497	0.896	0.868	-	-	-	0.914

Table 1: Comparing evaluation results for classification between Pointnet, Pointnet++ and 3DmFV. Evaluation results for 3DmFV are not available

5.2 Hyperparameter selection with Bayesian optimisation

The results of the hyperparameter tuning of the batch size are shown in Fig 1. The bar chart shows the accuracy of the 3D object classification in Pointnet when tested with different batch sizes. The highest accuracy is achieved with the batch size of 32.

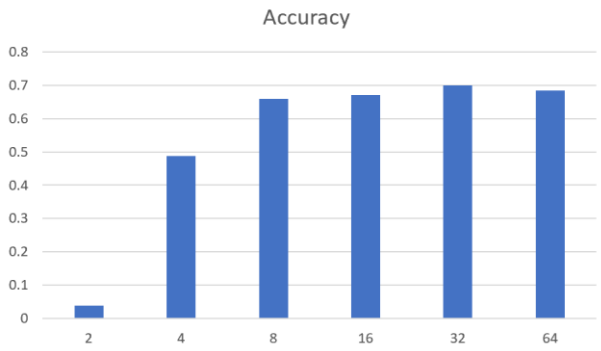


Fig 1: Comparing the effect of batch sizes on accuracy using Bayesian hyperparameter optimisation

5.3 Effect of batch size on classification accuracy

In our last step, we ran through the Pointnet object classification with different batch sizes to verify the results of the Hyperopt. Table 2 shows the summary of the training time, mean loss, accuracy and average class accuracy in evaluation.

Batch	Training	Evaluation

Size	Time (min)	Mean Loss	Accuracy	Average Class Accuracy
2	825	357.940	0.073	0.045
4	686	17988.35	0.706	0.663
8	649	0.505	0.876	0.848
16	609	0.497	0.888	0.858
32	703	0.546	0.883	0.856
64	665	0.668	0.878	0.844

Table 2: Comparing evaluation results for 3D object classification between different batch sizes in Pointnet

Figure 2 shows the effect of batch size on evaluation accuracy. As predicted by Hyperopt, lower batch sizes such as 2 and 4 returns poor accuracy, while batch sizes from 8 onwards offer a more stable accuracy of about 88%. This accuracy is higher than the 70% from Hyperopt, possibly because of the full 250 epochs trained here, compared to the 50 epochs in Hyperopt. That also explains why the accuracies for batch sizes 8 and 16 here are higher than that in Hyperopt.

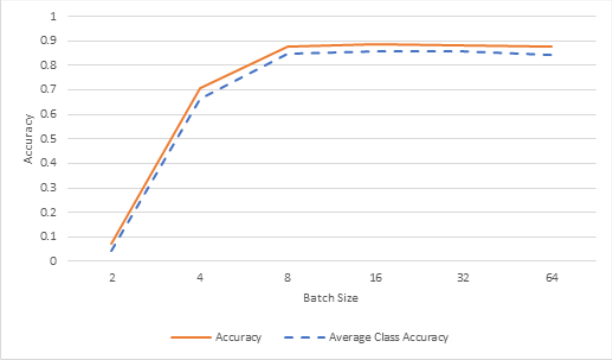


Fig 2: The effect of batch sizes on accuracy using evaluation results for 3D object classification in Pointnet

Given the long training time of Pointnet, we also compared the time needed to train the model using different batch sizes. Fig 6 shows that batch size 16 offers the shortest training time. However, more iterations of the process is needed in order to confirm that this is indeed so.

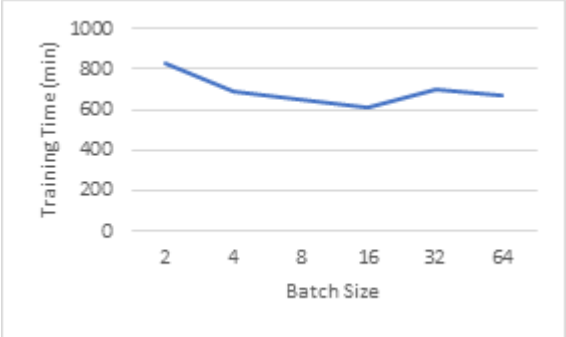
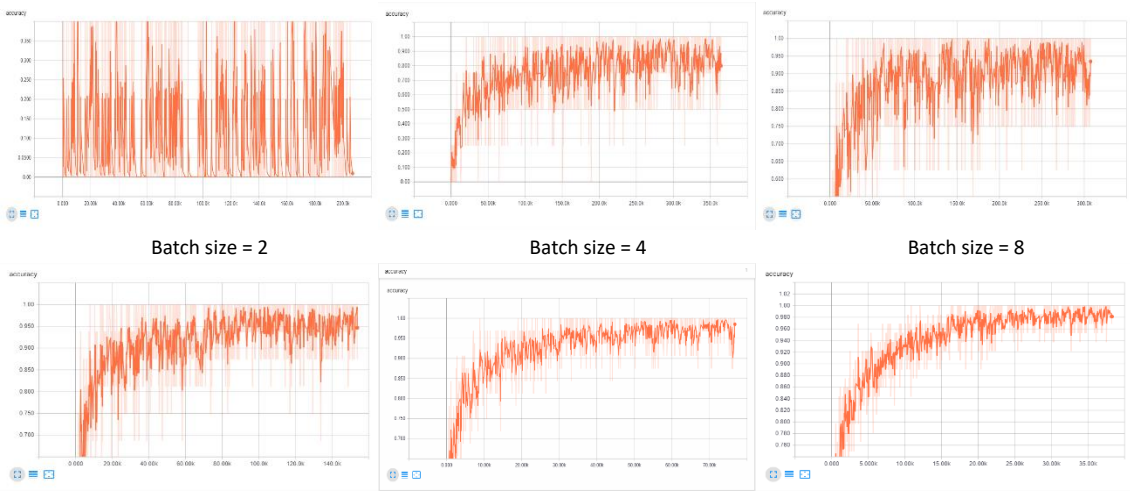


Fig 6. The effect of batch sizes on training time using evaluation results for 3D object classification in Pointnet

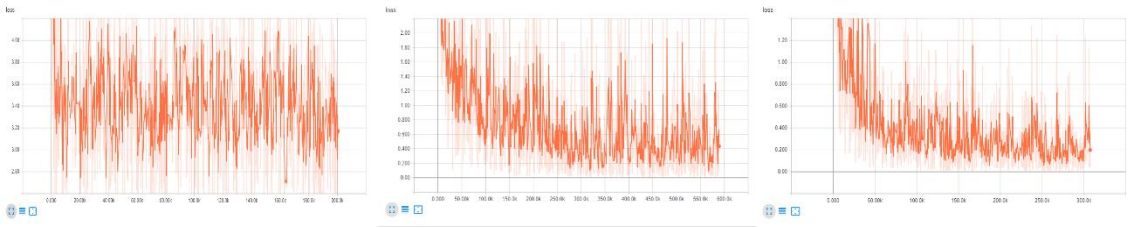
Fig 3 to 5 show that as the batch size increases, the accuracy and loss converge sooner, and from batch size 32 onwards the process is significantly less volatile than that of the lower batch sizes.



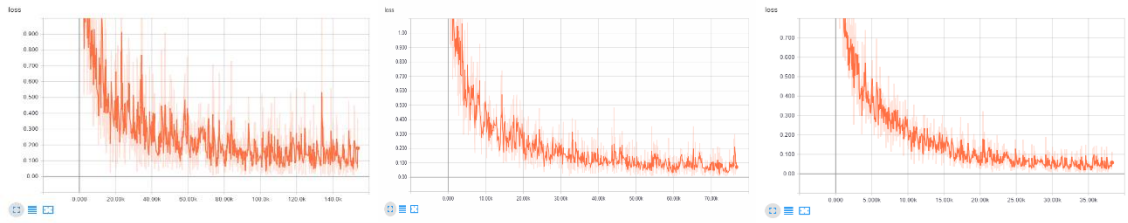
Batch size = 2 | Batch size = 4 | Batch size = 8

Batch size = 16 | Batch size = 32 | Batch size = 64

Fig 3: The effect of batch sizes on accuracy using evaluation results for 3D object classification in Pointnet

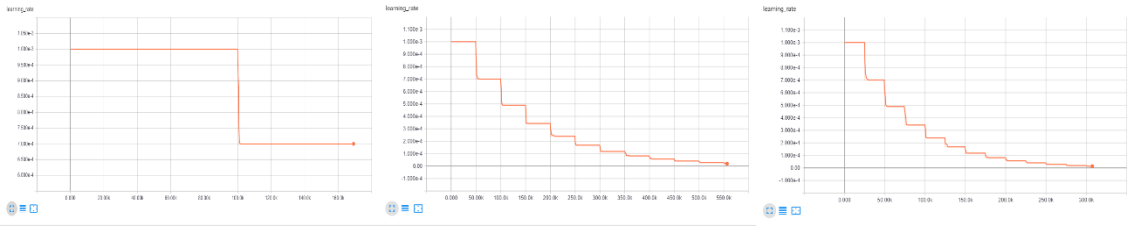


Batch size = 2 | Batch size = 4 | Batch size = 8

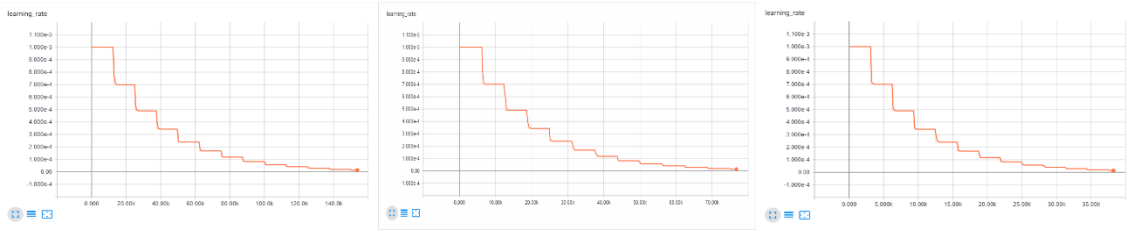


Batch size = 16 Batch size = 32 Batch size = 64

Fig 4: The effect of batch sizes on loss using evaluation results for 3D object classification in Pointnet



Batch size = 2 Batch size = 4 Batch size = 8



Batch size = 16 Batch size = 32 Batch size = 64

Fig 5: The effect of batch sizes on learning rate using evaluation results for 3D object classification in Pointnet

DISCUSSION

The proposed approach shows the best result can be achieved by using the batch size of 32 with the next best options to be either 16 or 64. Comparing this against our actual training result are achieved between batch size of 16 and 32 (the default given by the model) shown in Figure 2. The epoch runs needed to get a close approximation to the best result is much smaller as compared to doing a grid search which would have required a full 250 epoch run per batch size. This shows that the bayesian optimization can be even more effective when the hyperparameters range is even wider.

However comparing our best model's result against the published result, our best accuracy 0.858 is still below 0.892 (Figure 1). This highlights the difficulties in reproducing results due to variance caused by hardware setup or additional variables. Thus this also introduced an additional complexity in comparing results between better models against better tuned models. Smaller batch size (2, 4 and 8) shows higher volatility especially for the batch size 2 which after 250 epochs of training only manage to achieve the average class accuracy of 0.045. The same plot for batch size 16, 32 and 64 appears to be smoother in contrast. In terms of the training time, there's no significant difference that we observe across the different batch size.

For this paper, we have also experienced running the training using a few setups in order to complete the training. Google Colab provided additional challenge in terms of result storage and session live time. Due to the long training time needed, most Colab session does not complete the full 250 epochs run, and this motivated us to try different hardware setup such as running the training using local laptop with Nvidia GPU and subsequently moving to Google Cloud Platform which are better suited for this task as compared to Google Colab. The usage of cloud platform can also be leveraged to create reproducible results by allowing the Virtual Machine instance to be easily cloned and reproduced and the results running can be executed anywhere without restriction.

CONCLUSIONS

The best result that we have achieved in our training uses batch size 16 and the result for the mean loss, accuracy and average class accuracy were the best amongst the other batch sizes. This shows that the hyperparameters optimization is a viable option when it comes to to working with deep learning models which requires extensive training time. However as shown in

the earlier graph, due to the volatility of the effect of the batch size has on its performance and the fact that our simulation returned result that is sub par as compared to the published result, there should be rooms for other optimization in terms of hardware or other hyperparameters which could be better suited for our setup. Given the restriction of the time needed to complete each training run, the hyperparameters option were limited to batch size, in future works, additional hyperparameters can be considered as well as more powerful hardware setup to reduce the training time.

REFERENCES

1. C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

2. C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *Conference on Neural Information Processing Systems*, 2017.

3. Ben-Shabat, Y., Lindenbaum, M., Fischer, A. 3D Point Cloud Classification and Segmentation using 3D Modified Fisher Vector representation for Convolutional Neural Networks. *arXiv preprint arXiv:1711.08241*, 2017.

4. R. Klokov and V. Lempitsky. Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2017.

5. Masters, D., and Luschi, C. Revisiting Small Batch Training for Deep Neural Networks. *arXiv:180407612 [preprint]*, 2018.

6. James S. Bergstra, Remi Bardenet, Yoshua Bengio, and B ´ al´ azs K ´ egl. Algorithms for Hyper- Parameter Optimization. In *Advances in Neural Information Processing Systems*, 2011.