Project Report : CS 7643
Improving 3D Point Cloud Classification

Mitchell Gray Shaheer Amjad Yonghao Li Ho Cheung Tsui
Georgia Tech
Atlanta, Georgia
{mgray88, htsui6, samjad7, yli3850}@gatech.edu

Abstract

We investigated methods for improving three-dimensional object classification from unordered point clouds with a focus on reducing overfitting while improving accuracy. To do this, we re-implemented two widely used architectures, PointNet [1] and PointPillars [2] from scratch using PyTorch [3]. While these baseline models achieve strong performance of around 89% accuracy [1], there is plenty of room to improve the accuracy and reduce prominent overfitting. We tried to improve these models by introducing improved data augmentation [4], dropout [5], and residual connections [6]. We evaluated the performance of the baseline and upgraded models on the ModelNet10 [7] and ModelNet40 [7] datasets. We observed that our upgraded models improve validation accuracy by up to 4% on both datasets. It is also important to note that overfitting is nearly completely fixed. Despite this, there is still more room for improvement for both accuracy and overfitting.

1. Introduction/Background/Motivation

We attempted to train two neural networks that could detect and classify different types of objects by using three dimensional points from a sensor as shown in Fig 1;

however, the neural networks would do this in two different approaches. The main problem we wanted to solve was learning how to better train the neural networks so that they generalize to make correct classifications more frequently than current state-of-the-art models.

There are a couple common approaches to solving this problem. One approach is to convert the 3D data points into a format that standard convolutional neural networks can handle. For example, some projects such as SPNet [8] convert the 3D images into slices (2D images) from different viewpoints and feed the resulting images into a 2D convolutional neural network. Another approach is to convert the

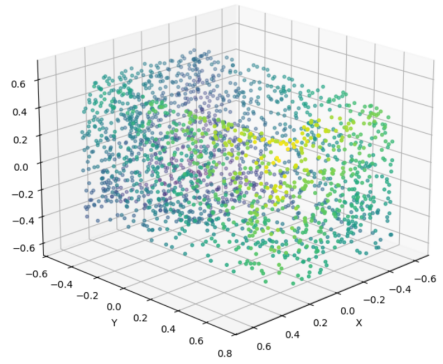

Figure 1. Sampled point cloud of a bathtub object.

3D object into voxel grids and use a 3D convolutional neural network such as ShapeNet [7]. More recently, there have been projects such as PointNet [1] and PointPillars [2] that operate directly on the sets of 3D points instead. The 2D images and voxel maps have multiple limitations such as losing the original data's invariance, removing 3D structure that isn't visible from the view angles, and throwing away detail that can be useful towards detecting the objects. The more modern approaches that work directly on the data have a few limitations such as computationally heavy algorithms, some information loss, and a tendency to overfit to the test sets. It is also important to note that this entire set of studies can be generalized to detecting structures in an unordered set of data points; this category of deep learning models is fairly understudied which leads modern approaches being far more experimental and less refined.

This project matters because these systems are at the core of many real-world technologies since this type of data comes from real-world sensors such as LiDaR e.g. for self-driving cars. Many self-driving cars from companies such as Waymo [9], and Zoox [10] utilize LiDaR. Having a model that is very accurate is necessary to keep drivers, pedestrians, and bikers safe. A model that incorrectly clas-

sifies objects or doesn't detect objects can lead to severe injuries or deaths. If we are able to improve the accuracy of the models used in these tasks, we may potentially contribute to saving lives worldwide as self-driving cars become more prevalent in society.

For this project we used the ModelNet10 [7] and ModelNet40 [7] datasets created at Princeton University. These are standard benchmark collections of three-dimensional object models. Both datasets consist of synthetic 3D CAD objects such as chairs, tables, sofas, etc. Each example is paired with a class label. ModelNet10 and ModelNet40 contain 10 and 40 object classes respectively. This data is curated for academic research on 3D shapes i.e. the data is not technically from the real-world; however, the performance gains on this dataset would likely transfer to real-world tasks since the same models are typically used in both cases.

2. Approach

We first built two baseline models from the ground up in PyTorch [3]. One model was the PointNet network [1], and the other was the PointPillars [2] network. For each dataset, we followed the standard train/test splits, sampled a fixed number of points per object, normalized them, and then trained both models using cross-entropy loss and monitored accuracy on the held-out validation set.

PointNet was implemented as close to the original paper as possible in both structure and base parameters, where descriptions are vague. References from the official code base [11] were used to fill in those gaps. Note that the official code base was updated after the original paper publish date, so there are slight differences even between the two official sources, and translating official pseudo code and code base in TensorFlow to PyTorch also lead to slight differences as follow.

Firstly, our dataset was obtained through the PyTorch data pipeline while the original method used raw 3D geometry to sample points with their own sampling method. Although functionally the same, there might be slight differences in the underlying sampled points. Secondly, the original code normalized the data during extraction implicitly while our implementation normalized this explicitly; this is functionally the same, but there might be slight differences in methodology. Thirdly, TensorFlow uses slightly different batch normalization and normalization behavior than PyTorch. Other differences include vague descriptions such as the number of epochs used, and specific positions of batch normalization layer placement. Our implementation inferred these from the official code base and descriptions. Lastly, the original code mentions non-deterministic predictions by using multiple random subsets of points sampled from an initial set during evaluation; this is not supported 1:1 in our data pipeline. Instead, we randomly resample from the geometry itself instead of the same set of points when non deterministic evaluation was used in our experiments.

The best parameters for baseline PointNet were found by experimenting with both ModelNet10 and ModelNet40 datasets using a linear sweep over key parameters. The search parameter set was based on the parameters listed in the paper where non changing parameter was frozen this default value. Parameters varied include model parameters as well as data pre-processing and evaluation like deterministic or non deterministic evaluation. In non-deterministic mode, we used 12 random resampled test points prediction with voting for the final evaluation prediction.

These experiments gave us a good idea of how the architecture behaved and allowed us to plot the models' performance. Since the existing models overfit fairly strongly, our main problem was trying to improve accuracy and decrease overfitting.

To do this, we modified both architectures with the specific goal of solving these problems. Architecturally, we attempted to introduce residual connections into the network. We've seen that residual connections can improve gradient flow, reduce overfitting, and improve performance [6]. On the data side, we added data augmentations to help the model see different versions of each object class. In other studies such as image processing, we've seen that data augmentation helps the model generalize better to data the model has never seen [4] especially when the datasets are not large. Lastly, we added dropout and tinkered with it to add regularization to our models. Dropout should help us reduce the model's overfitting [5] by training an ensemble of neural networks. After adding all of these new additions, we found good hyperparameter values, measured model performance, and created plots.

We expected this to be successful because each added model upgrade has a strong track record of reducing overfitting in neural networks. We've also seen all of these upgrades improve generalization, model complexity, and performance in other projects. PointNet and PointPillars mildly experimented with either dropout or data augmentation, but we wanted to take it further. Neither project even considered adding residual connections since doing such outside of recurrent neural networks was a bit of a newer idea at the time [6].

We expected two main problems when starting this project. One was getting the from-scratch implementations to work; especially since the reference code for PointNet was written using TensorFlow [11]. We also expected getting the models to generalize instead of overfitting would be difficult since the state-of-the-art models also struggle with this problem. We also knew that neither of our datasets were extremely large with ModelNet10 containing under 5,000 examples and ModelNet40 containing around 12,000 exam-

ples split across all classes. This meant that there would be less examples to learn from and the model would probably memorize some of the training set. This indicated to us that overfitting was more likely and the model would be more sensitive to hyperparameter tuning.

For PointPillars specifically, we had to adapt the architecture because the original network was designed only for 3D object detection[2], not classification. PointPillars normally outputs 2D Bird's Eye View (BEV) feature maps that are consumed by an anchor-based detection head. Since classification requires a single global prediction, we removed the detection head entirely and replaced it with a lightweight 2D CNN backbone followed by fully-connected layers to produce class logits. This modification allowed us to re-purpose the fast pillar encoder for shape classification while keeping the core PointPillars design intact.

3. Problem and Model Structure

Our problem had a very specific structure. Each object was represented as an unsorted set of 3D points sampled from its surface. The input is a set so order of points shouldn't matter. We are assigning a single label based on multiple points. The model also needed to be robust to simple transformations and small rotations. Our model architectures were designed to mirror this structure.

In PointNet, we used the same small network on every point independently, similar to a kernel. We also combined all per-point features with a symmetric aggregation operation, max pooling, which helps guarantee that the final global feature is insensitive to how the points are ordered and focuses instead on which geometric patterns are present in the set. The point data are input as a list of x,y,z coordinates, then transformed into different "feature spaces" by the TNet, and summarized using global max layers. To match the paper's stated benchmark, we set initial parameters as listed in the paper and then linearly searched for optimal parameters for our model.

In the PointPillars model, we grouped points into vertical pillars which reflects the fact that nearby points in space often belong to the same object part. We then process this image with convolutional layers to capture local spatial structure before making global decisions. The residual connections we added were layered on top of these, helping the networks learn deeper transformations without breaking the core properties of the model.

Unlike PointNet, PointPillars was never intended for classification tasks so its original detection head could not be used. We therefore converted the BEV feature map produced by the Pillar Feature Net (PFN) in addition with a scatter step into a global representation by applying a simplified 2D CNN backbone and global average pooling, followed by classification layers. This preserves the model's ability to learn local spatial structure from pillars while still producing a single label for the entire object.

4. Experiments and Results

We measured success by how well our models performed on data they had never seen before. We tracked cross-entropy loss and classification accuracy on the training and test splits. A model was considered more successful if it achieved lower test loss and higher test accuracy. Since we knew the models would overfit, we couldn't measure success solely on the training performance. We will now delve into the experiments and performance of our baseline models and upgrade models.

4.1. Baseline Models

4.1.1 PointNet

Experimentally, we first trained our PointNet and PointPillars models without any upgrades. We did a sweep over the hyper-parameters. We then ran the models and kept track of their loss and accuracy. After tuning the hyperparameters, we discovered the following settings performed the best:

For the PointNet baseline, search was performed in both ModelNet10 and ModelNet40 with deterministic and non-deterministic evaluations on test set. Note that in the original paper, the ModelNet10 benchmark results were not mentioned. So we focused more on ModelNet40.

The original set of parameters are listed in Table 1, which we used as our initial configuration.

Our results matched the original papers' claim of 89.2% accuracy for ModelNet40. In our experiments, we achieved a test accuracy of 92.84% on ModelNet10 and 89.26% on ModelNet40.

Parameters tested were number of epochs, batch size, learning rate, and dropout probability. 250 epochs was inferred from the original paper and our own observation since the paper did not specify how many epochs were used. By observing the training curve and result log, 250 was empirically determined to achieve converged results. Batch sizes of 16, 32, and 64 resulted in accuracy of 85.53%, 85.09%, and 84.52% respectively. Smaller batch size ensures that the overall gradient has greater variance per epoch, improving generalization because the model is less likely to get stuck in a local minima. Batch size of 64 under trains the model with fewer updates, and requires more epochs to achieve the same learning if at all possible due to minority class getting averaged out. Initial learning rates of 0.0001, 0.001, and 0.01 resulted in 83.87%, 85.09%, and 84.32% accuracy respectively. A learning rate too high or too low can impede learning by being way too slow, where each weight update is too tiny to traverse the loss landscape effectively. However, if the learning rate is too large, the training risks being too noisy as the model oscil-

lates in the loss landscape failing to land in smaller optima. A middle ground learning rate paired with the right decay gives the best results. For dropout, 0.0, 0.3, and 0.6 gave test accuracies of 85.69%, 85.08%, and 86.06% respectively. Surprisingly, a higher dropout rate than the paper's default resulted in better accuracy in our implementation. Dropout usually helps generalization by training "multiple" sub-neural nets, creating an "ensemble" which discourages over-reliance on certain features. Too high of a dropout can lead to representation power being too low. Here, the higher dropout rate did not negatively impact performance because PointNet's architecture—combining high-dimensional layers with a simple global max-pooling for context aggregation—maintained sufficient representational capacity.

The difference between our best parameters and the paper's can also be due to minor differences in the details of implementation. Notably, a high dropout rate being better here indicates that more regularization for better generalization seems to be a promising factor to improve the PointNet performance. This is corroborated by observing the learning curves in figs 6, 7, and 8, which show high variance between training and testing performance over epochs indicating overfitting. Therefore, our effort to focus on regularization methods to improve the model will likely lead to even better results!

4.1.2 PointPillars

For the PointPillars baseline, our best hyperparameter setting used a batch size of 32, a learning rate of 1×10^{-3}, weight decay of 1×10^{-4}, and 50 training epochs. These values were found to give the most stable optimization behavior. Architecturally, the baseline model consists of (1) a voxelization stage that partitions the point cloud into up to 1024 vertical pillars of size 0.08×0.08, (2) an 8-dimensional point-wise pillar feature representation followed by a max-pooled 64-dimensional Pillar Feature Net embedding, and (3) a compact three-block 2D CNN backbone with global average pooling for classification. This setup consistently produced test accuracies around 91% on ModelNet10 and 81% on ModelNet40, as seen in Fig 11 and 4 .

Across our hyperparameter search on ModelNet10, we observed that performance improved whenever we reduced the redundancy in the input and added moderate regularization. Reducing the number of points from 4096 to 1024 consistently increased the precision of validation (from 88.3% to 91.2%) by simplifying the distribution of the features of the pillar and reducing overfitting. Introducing a small amount of dropout in the backbone (0.1) provided an additional gain by stabilizing optimization, while excessive dropout (0.15) or higher weight decay (2×10^{-4}) hurt generalization. Adjusting geometric parameters such as pillar size or the number of points per pillar showed that too much sparsity (e.g., max 16 points per pillar) degrades local feature quality, whereas modest refinement yields only minor improvements. Overall, the best performance emerged when the model used fewer input points, light dropout, standard weight decay, and the original backbone width, suggesting that PointPillars benefits from controlled regularization rather than increased capacity.

In addition to the parameters tested, other architectural and voxelization settings were kept fixed throughout our experiments because they are theoretically well-motivated. We retained 1024 pillars and 32 points per pillar because the values ensure a balanced trade-off between spatial coverage and per pillar feature richness; reducing them would lead to information loss, while increasing them would inflate memory and computation without adding meaningful detail. The pillar feature dimensionality (8 to 64 after PFN) follows the original PointPillars formulation, where the input features (XYZ offsets and center-relative coordinates) are sufficient for capturing local geometry, and a 64-dimensional max-pooled embedding is empirically known to be expressive yet lightweight for downstream 2D CNNs. We also kept the backbone width (32, 64, 128 channels) and FC layer size (256) constant because classification requires less spatial reasoning than detection; a deeper or wider CNN would risk overfitting and provide diminishing returns on the relatively simple BEV representation produced from normalized ModelNet objects. Finally, the Adam optimizer and cross-entropy loss were not modified, as these are standard, stable choices for multi-class point-cloud classification tasks and performed reliably across all runs.

Although the BEV encoding provides strong geometric cues, the baseline PointPillars model still severely overfits due to the small dataset size and the deterministic structure of CAD objects, making it a good candidate for the upgrades introduced later.

4.2. Upgraded Models

We then ran the same benchmarks with the upgraded models. We got the following results after tuning the hyperparameters.

4.2.1 PointNet

Most of the PointNet upgrades focused on generalization and regularization to prevent the PointNet model from overfitting the training data and improving its performance on unseen examples. Adding additional dropout in the classification head, instead of the backbone, combated overfitting by randomly ignoring feature inputs where the final prediction is made, forcing the network to learn more robust features. Label smoothing regularized the loss function by replacing hard targets with soft targets, which prevents the

model from becoming overconfident and makes it more resilient to label noise. The use of fuller augmentation e.g. random jitter, random scaling (0.8 to 1.2) and random rotation ($\pm$ 15°) significantly increased the effective size and diversity of the training data through a wider range of transformations, making the model invariant to common variations in the point cloud. Finally, Adam weight decay acts as L2 regularization, penalizing large weights to encourage simpler models with smoother decision boundaries.

There were two upgrades that did not work: deepening the network and adding residual connections. PointNet's strength lies in efficiently learning local and global features primarily through the shared MLP layers and the max-pooling operation. Simply stacking more layers did not improve feature learning because the underlying issue is the ability to capture complex dependencies, especially since the final MLP classifier operates on a single global feature vector. PointNet is also relatively shallow. In this case, adding residual connections did not offer a significant advantage in a model as shallow as PointNet and mainly increased computational overhead. An interesting observation was that residual connections were still helpful in improving training stability but at the cost of accuracy (1 - 2 %); they were subsequently removed from the final model.

The final performance of the upgraded PointNet model was 93.6% accuracy on ModelNet10, Figs 9,10, and 89.7% on ModelNet40, Figs 2, 3.

Based on the accuracy and loss curves, the upgrades to the PointNet model were a success. The upgraded model demonstrates superior performance and generalization compared to the baseline. Specifically, 2 shows that the upgraded model achieved a higher final test accuracy (approaching 90% versus 88%) and a smaller gap between training and testing accuracy, suggesting reduced overfitting. 3 reinforces this conclusion by showing that the upgraded model stabilized with a significantly lower final test loss (around 0.5 versus 0.75). Finally, the separation between the train loss and test loss curves is much narrower in the upgraded model which is the key indicator that the model is generalizing better to unseen data from the ModelNet40 dataset.

The optimized hyper-parameters between baseline and upgraded models remained unchanged for the most part, except for the learning rate schedule (linear decay vs cosine annealing).

4.2.2 PointPillars

For the upgraded PointPillar model, we made multiple improvements. First, we implemented data augmentation that rotated the entire point cloud by $\pm 15°$. We also made minor translations to the point could. Next, we added dropout to the backbone. Lastly, we converted the PFN into

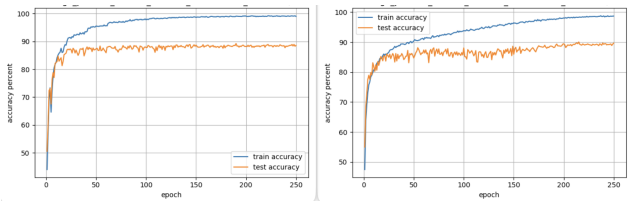

Figure 2. Baseline vs Upgraded PointNet accuracy (ModelNet40)

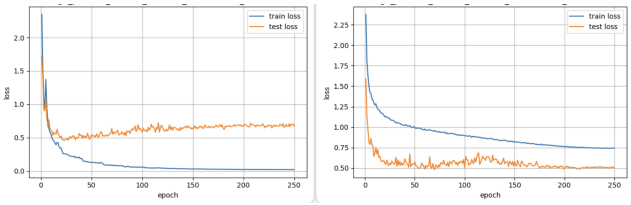

Figure 3. Baseline vs Upgraded PointNet loss (ModelNet40)

a module block and repeated it with a depth of 3. For each block, we added a residual connection.

To begin, we utilized the same hyper-parameters as the baseline PointPillars model, they provided good results with this new model as well. However, the baseline model didn't have dropout. Experimenting with dropout probability, 0.15 was the optimal value. As dropout approached 0, the training accuracy would improve but the model would strongly overfit. As dropout approached 0.3, the overfitting would be eliminated, but the model would begin underfitting the data. At the value of 0.15, the model nearly fitting the data and achieved optimal validation accuracy.

After running some experiments, we found that the upgraded PointPillars model was able to achieve a validation accuracy of 92.95% and 84.81% on ModelNet10 and ModelNet40 respectively. In addition, the model, as seen in Figs 4, 5, 11, and 12,

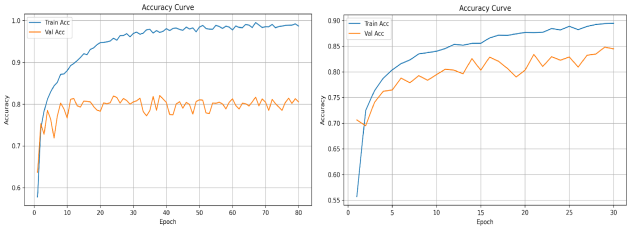

Figure 4. Baseline vs Upgraded PointPillars Accuracy (ModelNet40).

nearly fits the data with both accuracy and loss curves demonstrating very low variance. With these boosts in performance and such low variance, we can clearly see that these upgrades worked. Data augmentation and dropout reduced the overfitting as expected. The residual connections helped improve performance, but also helped stabilize the learning process. As seen in Figs 4, 5, 11, and 12, the curves from the upgraded model are significantly smoother and

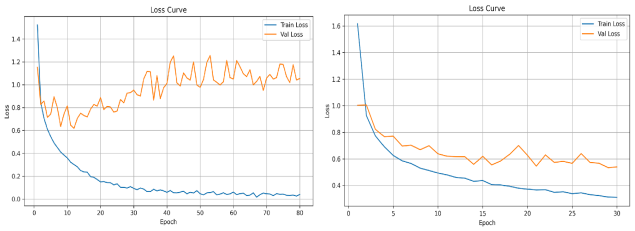

Figure 5. Baseline vs Upgraded PointPillars Loss (ModelNet40).

represent a more stable learning, likely because the residual connections improved the gradient flows in the model.

Although the overfitting could've been reduced a bit more and performance isn't as optimal as newer state of the art models, these results represent an overwhelming success.

4.3. Result Comparisons

Overall, we would say that we succeeded in the main goals we set for the project. We successfully implemented both state-of-the-art models from scratch and obtained good performance on our datasets. We were also able to modestly improve performance and massively decrease overfitting relative to the baselines by implementing our upgrades. However, there was still some overfitting that occurred, which means that we didn't necessarily fully solve the problem; we just provided a better solution than the baseline models. To see even better performance and overfitting reduction, it is clear that a new architecture is needed to make substantial improvements.

5. Learned Parameters

In both models there were learned parameters in all of the trainable layers that transform the data. The non-learned parts were everything that just reshapes, aggregates, or interprets those outputs. For both models, the weight matrices and biases of the fully connected layers and pointwise/convolutional layers were learned as well as the scale and shift parameters inside the batch normalization layers. The residual skip-path itself did not have any learnable parameters, only the layers within the residual block.

6. Optimizer

For all of our experiments we used the Adam optimizer [12] to train both models. This meant that parameters with smaller or more stable gradients can take relatively larger steps. This is useful with our project specifically due to all of the different layers and parameters which can experience very different gradient scales. We chose Adam because it is widely used in modern deep learning, is relatively insensitive to the exact choice of the learning rate compared to plain Stochastic Gradient Descent, and tends to provide faster and more stable convergence during training.

7. Loss Functions

The loss used in PointNet has two parts, firstly, a cross entropy loss for the classification net, where the predictions and the sample ground truth are compared. The second part is regularization loss; this is only found in the Transform Net for the features, limiting the freedom of the weights in "kernel" space. The regularization loss is multiplied by a scaling parameter.

8. Future Works

Our project focused on improving PointNet and PointPillars architectures on synthetic point-cloud classification benchmarks by adding upgrades. While we showed this reduced overfitting and improved validation performance relative to our baseline, there are several natural extensions that would make this work more robust, complete, and realistic. Going forward, one direction to improve is to move beyond clean CAD models and re-evaluate our work on real sensor data. Although we saw performance increases, whether or not this improves real-world models is not guaranteed. These datasets have the additional challenges of partial views, occlusions, and varying point densities. Our data augmentation had limited scaling and rotation whereas there can be huge variations in point density and object size across a kilometer-scale point cloud as seen in real-word environments. It is likely that the architectures we explored would not be able to recognize objects that are upside down or sideways for example. Another direction is a systematic exploration of regularization and architecture design for unordered point sets. In this project, we focused on classic methods to reduce overfitting, but there are other techniques such as stronger augmentations, alternative normalization techniques, and more label smoothing. It is also reasonable to run extensive hyper-parameter tuning such as a grid search. Together, all of these future extensions should hopefully help improve the performance of the model, generalize better for the real-world.

9. Conclusion

To conclude our investigation, we found that our upgrades noticeably increased performance and decrease overfitting of the model. We demonstrated that relatively simple and well-justified modifications can make point-cloud classifiers (or deep neural networks, in general) more robust and performant without changing the underlying task or architecture. Given the central role of 3D perception in safety-critical applications such as self-driving vehicles, we view this work as a step toward more reliable models and a foundation of future extensions on real LiDaR data, more advanced architectures, and a deeper focus on robustness.

References

[1] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," 2017. 1, 2

[2] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," 2019. 1, 2, 3

[3] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019. 1, 2

[4] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," 2017. 1, 2

[5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. 1, 2

[6] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," 2015. 1, 2

[7] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," 2015. 1, 2

[8] M. Yavartanoo, E. Y. Kim, and K. M. Lee, "Spnet: Deep 3d object classification and retrieval using stereographic projection," 2019. 1

[9] Waymo, "Waymo driver: Self-driving car technology for a reliable ride." https://waymo.com/waymo-driver/, 2025. 1

[10] Zoox, "Going beyond seeing to perceiving the world around us." https://zoox.com/journal/perception, 2023. 1

[11] C. R. Qi, "Pointnet: Deep learning on point sets for 3d classification and segmentation (github repository)." https://github.com/charlesq34/pointnet, 2017. 2, 9

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. 6

10. Work Division

Contributions of team members:

Student Name	Contributions
Mitchell (mgray88)	Team coordination. Project planning. PointPillars upgrades and experiments. Rubric question answering. Paper writing. Research background material.
Shaheer (samjad7)	PointNet upgrades and experiments. Paper writing and proofreading.
Yonghao (yli3850)	PointPillars initial implementation and experiments. Visuallization utility functions. Variation testing and research, parameter experiments. Paper writing.
Ho Cheung Tsui (htsui6)	PointNet initial implementation, variation testing and research, parameter experiments. Paper writing.

A. Our Code

Our code repository can be found on our Github: https://github.com/MitchellGray100/3D_Point_Cloud_Classification.
To implement the baseline models, we referenced the original PointNet and PointPillar whitepapers. We also referenced the PointNet implementation [11].

B. PointNet Hyperparameter Values

Table 1. Hyperparameter sets for PointNet Baselines

Hyperparameter	**Default Value**	**Found Best Value**
batch_size	32	32
num_epochs	250	250
learning_rate	0.001	0.001
learning_rate-decay_step	200000	200000
learning_rate-decay_factor	0.7	0.7
min_learning_rate	0	0
regularization-loss_weight	0.001	0.001
dropout_prob	0.3	0.3
adam_weight_decay	0.0	0.0
augment_training_data	True	False
num_points	1024	1024
batch_norm_init_decay	0.5	0.5
batch_norm_decay_rate	0.5	0.5
batch_norm_decay_step	200000	200000
batch_norm_decay_clip	0.99	0.99
ModelNet10-Test Deterministic Accuracy	91.1894	92.8414
ModelNet40-Test Deterministic Accuracy	85.0891	89.2625
ModelNet40-Test Non-Deterministic Accuracy	86.2642	89.1410

C. Extra Figures

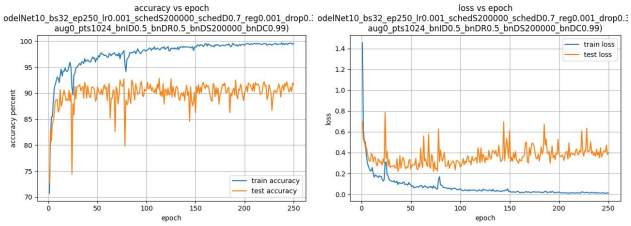

Figure 6. PointNet Baseline best Accuracy and Loss with ModelNet10, Deterministic Evaluation

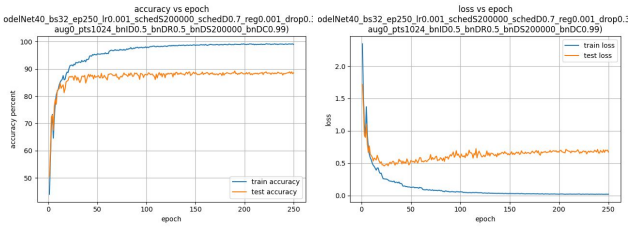

Figure 7. PointNet Baseline best Accuracy and Loss with ModelNet40, Deterministic Evaluation

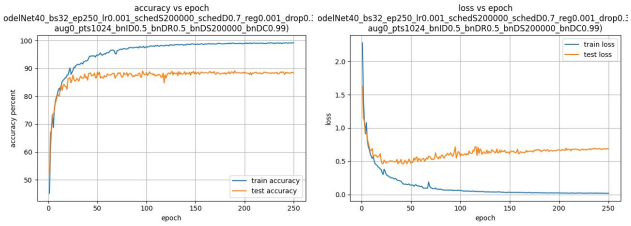

Figure 8. PointNet Baseline best Accuracy and Loss with ModelNet40, Non-Deterministic Evaluation

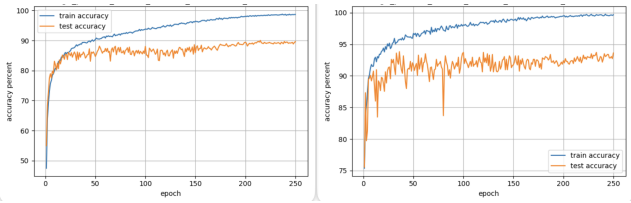

Figure 9. Baseline vs Upgraded PointNet accuracy (ModelNet10)

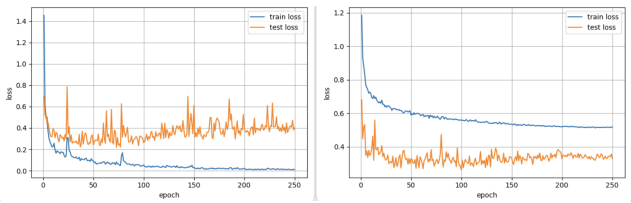

Figure 10. Baseline vs Upgraded PointNet loss (ModelNet10)

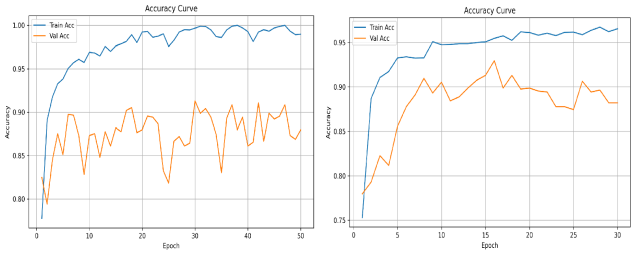

Figure 11. Baseline vs Upgraded PointPillars Accuracy (ModelNet10).

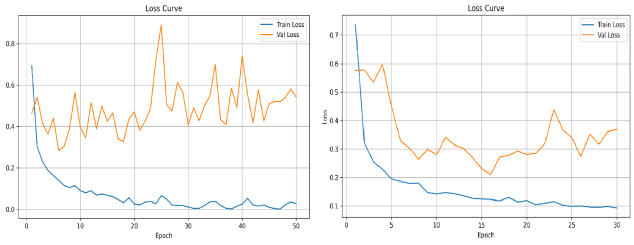

Figure 12. Baseline vs Upgraded PointPillars Loss (ModelNet10).