

Remote Sensing Studios

Semester 2: 2023

Studio 1: 3D Semantic Segmentation of Urban Areas

Dr. Fatemeh Alidoost

Submitted by:

Bryan Ramirez Franco (1005724)

Lazuardy F.P. Sulaeman (1005745)

Muhtasimul Islam Rushdi (1005729)

Nishat Tasnim Priyanka (1005720)

3D Semantic Segmentation of Urban Areas

Nishat Tasnim Priyanka, Muhtasimul Islam Rushdi, Lazuardy F.P. Sulaeman, Bryan Ramirez Franco.

Hochschule für Technik Stuttgart

ABSTRACT

This project aims to implement semantic segmentation of 3D point clouds using the PointNet++ network. The objective is to establish a balanced hyperparameter configuration for optimum segmentation. Experiments with various training parameter combinations have been done to optimize the network's performance. The dataset consists of airborne lidar data and corresponding semantic labels, covering areas in Jacksonville, Florida, and Omaha, Nebraska, USA. This project's input is 120 ASCII text files, including x,y,z, and separated label files, with 110 for training and 10 for testing the network. The hyperparameters investigated here are NumberOfPoints, BlockSize, Epoch, and mini-batch size. The number of training data is also changed to see the effect on the accuracy. Test data 4 (JAX_328) and test data 10 (OMA_290) are the data to assess the accuracy. The highest accuracy achieved for test data 4 and 10 is 0.92 and 0.87, respectively. The best configuration is 50.000 NumberofPoints, 150 blocksize, 35 epochs, and 60 training data. The accuracy is mainly affected by the number of points and the number of train data.

Keywords: Semantic segmentation, 3D point clouds, PointNet++, Training hyperparameters, Semantic labels, Spatial data analysis.

1. INTRODUCTION

This project focuses on developing a practical approach for implementing semantic segmentation of 3D point clouds using the PointNet++ network. The project's main objective is to validate the network's performance through trials applying various training parameters. By adjusting these parameters during training, we aim to optimize the network's performance and establish a balanced configuration that ensures meaningful segmentation accuracy, which will be considered assessment criteria.

The dataset used in this project consists of airborne lidar data and corresponding semantic labels. It covers an area of approximately 100 square kilometers across Jacksonville, Florida, and Omaha, Nebraska, USA. The dataset is divided into 120 tiles. The airborne lidar data provide ground-truth geometry with a spacing of around 80 cm. The point clouds are stored in ASCII text files, representing each point's coordinates (x, y, z), intensity, and return number. Semantic labels are provided as separate ASCII text files, categorizing points into the ground, trees, buildings, water, and bridges. The dataset includes 110 files for training and ten files for testing.

The PointNet++ network is employed to accomplish this task. The network is available in MATLAB. By utilizing the strengths of the PointNet++ network and carefully analyzing the dataset, an effective approach for semantic segmentation of 3D point clouds is aimed to be developed, paving the way for improved understanding and analysis of complex spatial data.

2. THEORETICAL BACKGROUND

PointNet++ is a deep learning architecture designed for analyzing point cloud data, commonly used in applications such as 3D object recognition, segmentation, and scene understanding. It enables the accurate semantic segmentation of unorganized lidar point clouds, associating each point with a class label. Here the key concepts of PointNet++ and its hierarchical feature learning approach are based mainly on the article titled "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space" by Qi et al. (2017).

Input Processing

The initial phase of PointNet++ involves processing a set of points in a metric space, typically a 3D point cloud dataset. These points undergo several steps, starting with local feature extraction using the PointNet layer. The PointNet layer extracts feature independently from each point, employing a shared multilayer perceptron (MLP) network consisting of fully connected layers (Matlab, 2023). In each fully connected layer, the neurons compute a weighted sum of the inputs from the previous layer, incorporating a bias term. The weighted sum is then passed through a rectified linear unit (ReLU) activation function, which applies a non-linear transformation to each neuron's output. The ReLU activation function operates independently on each point's output.

Following the ReLU activation, the local features are aggregated using max pooling. Max pooling selects the maximum value within a neighborhood for each feature dimension, resulting in a global feature vector. This global feature vector serves as the input for the subsequent stages of the PointNet++ network (Qi et al., 2017; Matlab, 2023).

Hierarchical Feature Learning

PointNet++ incorporates hierarchical feature learning to capture local features at different contextual scales. This is achieved through recursive processing on a hierarchical structure known as nested partitioning. The hierarchical structure consists of multiple levels of set abstraction, progressively abstracting larger local regions. This hierarchical organization allows the network to capture features with increasing contextual scales, providing a richer input data representation (Matlab, 2023).

PointNet++ is designed to be invariant to input point permutations, enabling it to recognize patterns regardless of the order of the points. The network learns local structures induced by the metric space, facilitating the recognition of fine-grained patterns within the input point cloud. Skip connections are crucial in preserving fine-grained details during the hierarchical feature learning process. These connections propagate information from lower levels to higher levels, allowing the combination of local features from different scales in the set abstraction layers. The concatenated features are then passed through a fully connected layer to produce the final output features (Matlab, 2023).

Set Learning Layers

PointNet++ incorporates set learning layers to handle varying densities in point sets. These layers combine features from multiple scales, addressing density challenges. They are permutation invariant, enabling the network to handle sets of different sizes while maintaining performance. PointNet++ uses a multi-scale grouping strategy to capture local structures and contextual information. The network extracts relevant information for tasks like semantic segmentation by propagating features from lower to higher levels, resulting in accurate predictions. (Qi et al., 2017; Matlab, 2023)

Multilayer Perceptron (MLP)

Within PointNet++, MLP networks extract local features from the input point set. These networks contain fully connected layers with ReLU activation functions. The number and size of these layers can be configured based on the task and dataset requirements (Qi et al., 2017; Matlab, 2023).

Max Pooling

Max pooling is employed in PointNet++ to aggregate local features into a global feature vector. It selects the maximum value within non-overlapping regions, effectively summarizing salient features of the input point set. Max pooling is applied within each set abstraction level, enabling the aggregation of features from different scales (Qi et al., 2017; Matlab, 2023).

Downsampling of Feature Maps

The downsampling of feature maps in PointNet++ is achieved through operations such as max pooling. This process reduces the spatial resolution of the feature maps while capturing larger-scale patterns. Downsampling enhances the network's computational efficiency and memory usage, facilitating the analysis of large-scale point cloud data (Matlab, 2023).

By reducing the spatial resolution, downsampling allows the network to focus on higher-level features and abstract representations. It helps capture and understand larger-scale patterns and structures within the point cloud. Downsampling also improves computational efficiency and reduces memory requirements by reducing the amount of data that needs to be processed (Matlab, 2023).

Skip Connections

Skip connections are essential in recognizing complex patterns within PointNet++. They propagate information from lower to higher levels of the network and preserve fine-grained details. In PointNet++, skip connections are implemented by combining local features from different scales in set abstraction layers. The concatenated features are then passed through a fully connected layer to produce the final output features.

Hyperparameter description

The hyperparameters investigated for this task are BlockSize, Number of Points, Epoch, MiniBatch size, and L2Regularization.

1. **BlockSize and Number of Points:** Block size is the size of the blocks used for sampling points in each layer of PointNet++. While the number of points that are sampled from the input point cloud. More number of points leads to a more detailed representation but increases computational complexity. The constraints considered for deciding the number of these parameters are Object Size, Density of Point Cloud, Computational Resources, and Overfitting.
2. **Max Epochs:** specify the maximum number of iterations during training. Increasing the number of Epochs potentially improves accuracy. However, too many Epochs can lead to overfitting and inefficient running time.
3. **Mini-Batch Size:** The dataset is divided into smaller subsets called mini-batches during training. A typical range for this parameter is between 4 and 128. Increasing the MiniBatch Size can speed up training due to parallel processing but may require more memory. A larger MiniBatch Size can lead to more stable gradients but might compromise generalization if the dataset is small.
4. **L2 Regularization:** this regularization technique adds a penalty term to the loss function, helping to prevent overfitting. It controls the magnitude of the regularization term and is typically set between 0.001 and 0.01. Higher values will increase the penalty for large weights, discouraging overfitting but potentially reducing accuracy if regularization is too strong.

3. DATA AND SOFTWARE

The main objective of this task is object detection for an urban scene using 3D point cloud segmentation techniques. The necessary data for this task consisted of airborne lidar data and corresponding semantic labels. The dataset encompassed approximately 100 square kilometers, comprising 120 tiles that covered Jacksonville, Florida, and Omaha, Nebraska in the United States.

Airborne LiDAR data are used to provide ground-truth geometry. The aggregate nominal pulse spacing (ANPS) is approximately 80 cm. Point clouds are provided in ASCII text files with format {x, y, z, intensity, return number}. **Semantic labels** are provided as ASCII text files. Semantic classes in the contest include buildings, bridges, high vegetation, ground, and water.

Folder structure:

- Test-Track4: 10 ASCII text files (comma separated): {x, y, z, intensity, return number}
- Test-Track4-Truth: 10 ASCII text files: {label}
- Train-Track4: 110 ASCII text files (comma separated): {x, y, z, intensity, return number}
- Train-Track4-Truth: 110 ASCII text files: {label}

Table 1: Label Classes

Class Index	Class Description
2	Ground
5	High Vegetation
6	Building
9	Water
17	Bridge Deck

The numeric computing environment MATLAB (R2022b), along with Computer Vision Toolbox, Deep Learning Toolbox, Lidar Toolbox, and Parallel Computing Toolbox for GPU support, was used for this studio project. For the GPU support, the Citrix digital workspace platform was used to access and operate desktops in the cloud remotely. It was ensured that the remote desktop had a recent graphics driver.

4. METHODOLOGY

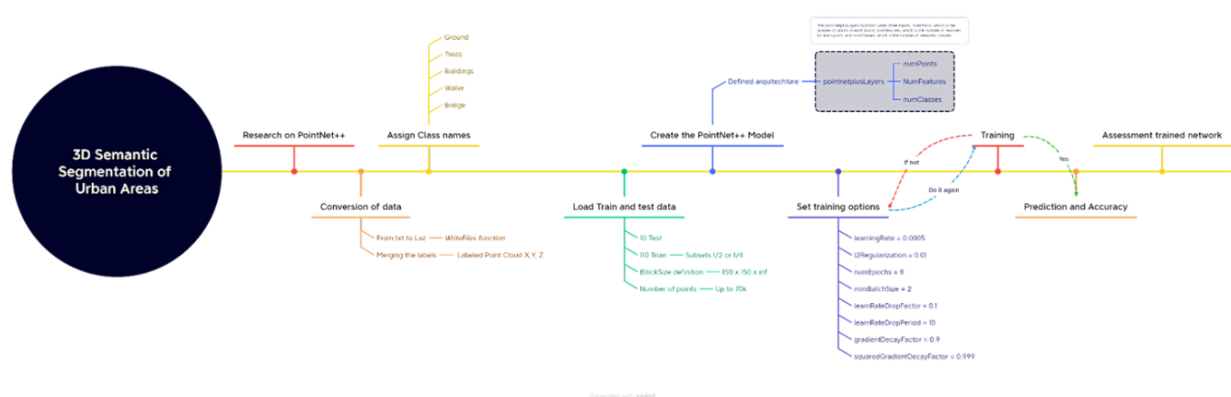


Figure 1 The outlined methodology includes various stages: data pre-processing, dataset loading into a datastore, designing a convolutional neural network, configuring training options, training the model using PointNet++, and evaluating the performance of the trained detector.

Conversion of Data and Assign the Class Name

The definition of data classes is given first. Then the pre-processing of the training data (ASCII text file) is required to specify the text (.txt) file conversion to .las (or LAS) file format and the path to the training data. The points with a label of 0 (indicating unclassified points) from the pointCloud were removed and label variables using the 'select' function. This ensures that only points with valid semantic labels are retained for further processing and visualization. To ensure proper display, the class labels in the label variable are remapped to a range of 1 to 5. This adjustment allows for correct visualization and interpretation of the different classes present in the point cloud

Finally, the modified ptCloud and its associated attributes, including the updated classification labels, are written to a LAS file using the writePointCloud function. This enables storing the processed point cloud data for future use or analysis.

Load Train and Test Data

The point cloud data was divided into small, non-overlapping grids for efficient memory processing. The grid dimensions were defined (e.g., [150, 150]), and a fixed number of points per grid was fixed (e.g., 50000) to enable faster training. During the training process, the outputs of 'maxLabel' and 'maxWeight' play a crucial role in achieving a balanced loss function across all classes. These values are utilized to ensure that the loss function is evenly balanced, accounting for the varying importance of different classes in the training procedure.

Create a network, Set Training Options, and Training the Model

For the exercise, the PointNet++ model was used. The model was trained using a GPU. The PointNet++ architecture takes the Number of points in each input point cloud, the Number of coordinate channels in each point (typically 3 for x, y, and z), and the number of classes in the dataset as input layers. Then it creates a deep learning network with multiple layers, including convolutional, max-pooling, fully connected, and output layers.

The 'pixelClassificationLayer' is a custom layer used for semantic segmentation problems. It applies a softmax function to the input tensor and calculates the cross-entropy loss between the predicted and ground truth labels.

The 'Adam' optimization algorithm is employed to train the network. The hyperparameters are specified using the trainingOptions function. The hyperparameters, including learningRate, l2Regularization, numEpochs, and miniBatchSize, were fine-tuned by systematically exploring different combinations of these options to optimize the training process to enhance the overall performance and convergence of the learning algorithm. A custom training loop was used to train the model.

Prediction and Accuracy

The full test point cloud data was read to assess the accuracy of the model's predictions. The point cloud is divided into non-overlapping grids, and the model predicts the labels for the points within each grid. Finally, the output predictions are obtained, enabling evaluation of the model's accuracy and classification performance.

Assessment Trained Network

The model calculates a confusion matrix by comparing the predicted dense labels with the target dense labels. It then proceeds to evaluate a range of metrics to assess the performance of the trained network for semantic segmentation. These metrics include dataset metrics (such as Global Accuracy, mean accuracy, mean IoU, and weighted IoU), class metrics (including Accuracy and IoU for each class), and image metrics. By utilizing the confusion matrix, these metrics provide valuable insights into the overall effectiveness of the trained network in performing semantic segmentation at both the dataset level and the level of each specific class.

5. EXPERIMENT

a. Investigation

The network validation process involved using two out of the ten available test datasets. We conducted multiple experiments by adjusting various training options, such as the train data, epochs, block size, number of points, minibatch size, and learning rate. We conducted 16 trial runs and shared the best corresponding results of the two test datasets: JAX_328 and OMA_290.

Here, we have shown an example of the final epoch result of Trial 16. The model training progress is shown for different epochs and iterations. The accuracy, loss, and learning rate are recorded at each iteration, indicating the model's performance and progress over time.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Mini-batch Loss	Base Learning Rate
35	10900	30:58:21	96.78%	0.2433	5.0000e-07
35	10950	31:06:52	58.45%	1.2241	5.0000e-07
35	11000	31:15:22	96.65%	0.3293	5.0000e-07
35	11050	31:23:51	93.11%	0.3516	5.0000e-07
35	11100	31:32:26	94.71%	0.3007	5.0000e-07
35	11150	31:40:57	49.06%	1.2249	5.0000e-07
35	11200	31:49:30	92.63%	0.6936	5.0000e-07

Figure 2 Final Epoch Result of Trial 16

Here we have showcased the experiments conducted on the test data JAX_328 and OMA_290. We have highlighted the best results of each class. Specifically, for JAX_328, Trial 15 demonstrated the highest Global accuracy and WeightedIoU values. Similarly, for OMA_290, Trial 16 exhibited the maximum Global accuracy and WeightedIoU values. Throughout the trials, we deliberately modified the parameters to effectively observe the impact of each parameter and highlight their influence.

Table 2 Experiments with JAX_328_PC3 Test Data

	Test Data	JAX_328_PC3					
	Trial number	Trial 8	Trial 10	Trial 11	Trial 13	Trial 15	Trial 16
	Train Data	1 to 40	1 to 40	1 to 40	41 to 90	41 to 100	41 to 100
Hyper Parameters	Mini Batch	10	8	8	6	6	4
	Epoch	15	15	15	15	15	35
	Block Size	100	150	150	150	150	150
	NumPoint	30000	40000	50000	50000	50000	50000
	Learning Rate	0.0005	0.0005	0.0005	0.0005	0.0005	0.0005
Accuracy Assessment	Run Time	954 min	533 min	613 min	572 min	788 min	1909 min
	Ground	0.86994	0.78677	0.91274	0.96124	0.99725	0.99007
	Trees	0.94596	0.92395	0.93428	0.98845	0.99205	0.99235
	Building	0.59699	0.51466	0.5692	0.28963	0.20044	0.21586
	water	0.041339	0.31197	0.035898	0.006877	0.005479	0.013226
	bridge	0	0	0	0	0	0
	Global	0.85824	0.81954	0.86986	0.90634	0.9205	0.91838
	WeightedIoU	0.79824	0.76518	0.80698	0.83892	0.85102	0.8485
	Mean Accuracy	0.49084	0.50747	0.49042	0.44924	0.43904	0.4423

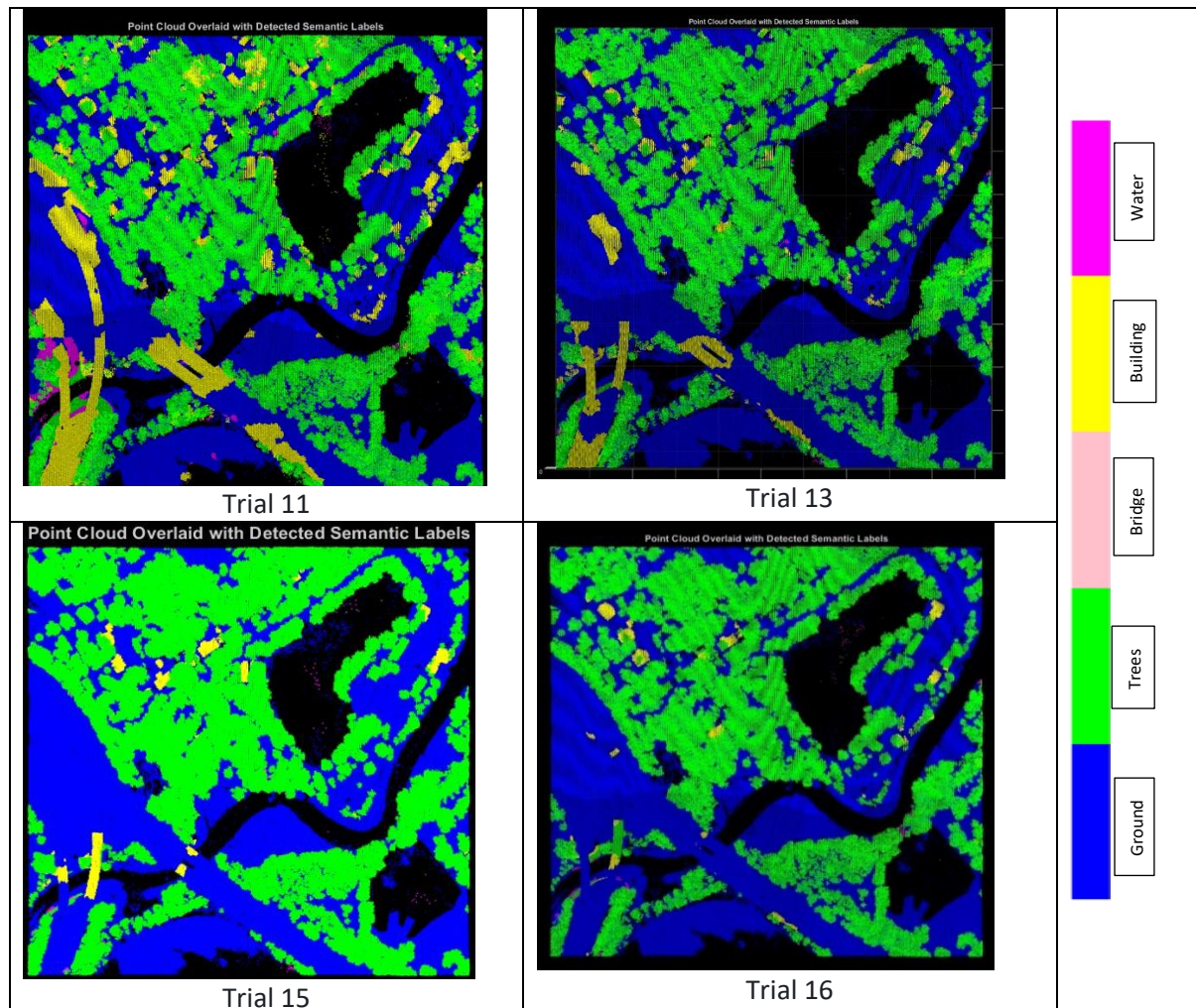


Table 3 Experiments with OMA_290_PC3 Test Data

	Test Data	OMA_290_PC3				
	Trial number	Trial 8	Trial 11	Trial 13	Trial 15	Trial 16
	Train Data	1 to 40	1 to 40	41 to 90	41 to 100	41 to 100
Hyper Parameters	Mini Batch	10	8	6	6	4
	Epoch	15	15	15	15	35
	Block Size	100	150	150	150	150
	NumPoint	30000	50000	50000	50000	50000
	Learning Rate	0.0005	0.0005	0.0005	0.0005	0.0005
Accuracy Assessment	Run Time	954 min	613 min	572 min	788 min	1909 min
	Ground	0.80057	0.50238	0.99733	0.99995	0.9778
	Trees	0.33946	0.023362	0.38595	0.27944	0.33801
	Building	0.62777	0.77279	0.6539	0.68256	0.7533
	water	NaN	NaN	NaN	NaN	NaN
	bridge	NaN	NaN	NaN	NaN	NaN
	Global	0.71878	0.58584	0.84609	0.85441	0.87003
	WeightedIoU	0.65456	0.42992	0.74265	0.7481	0.77483
	Mean Accuracy	NaN	NaN	NaN	NaN	NaN

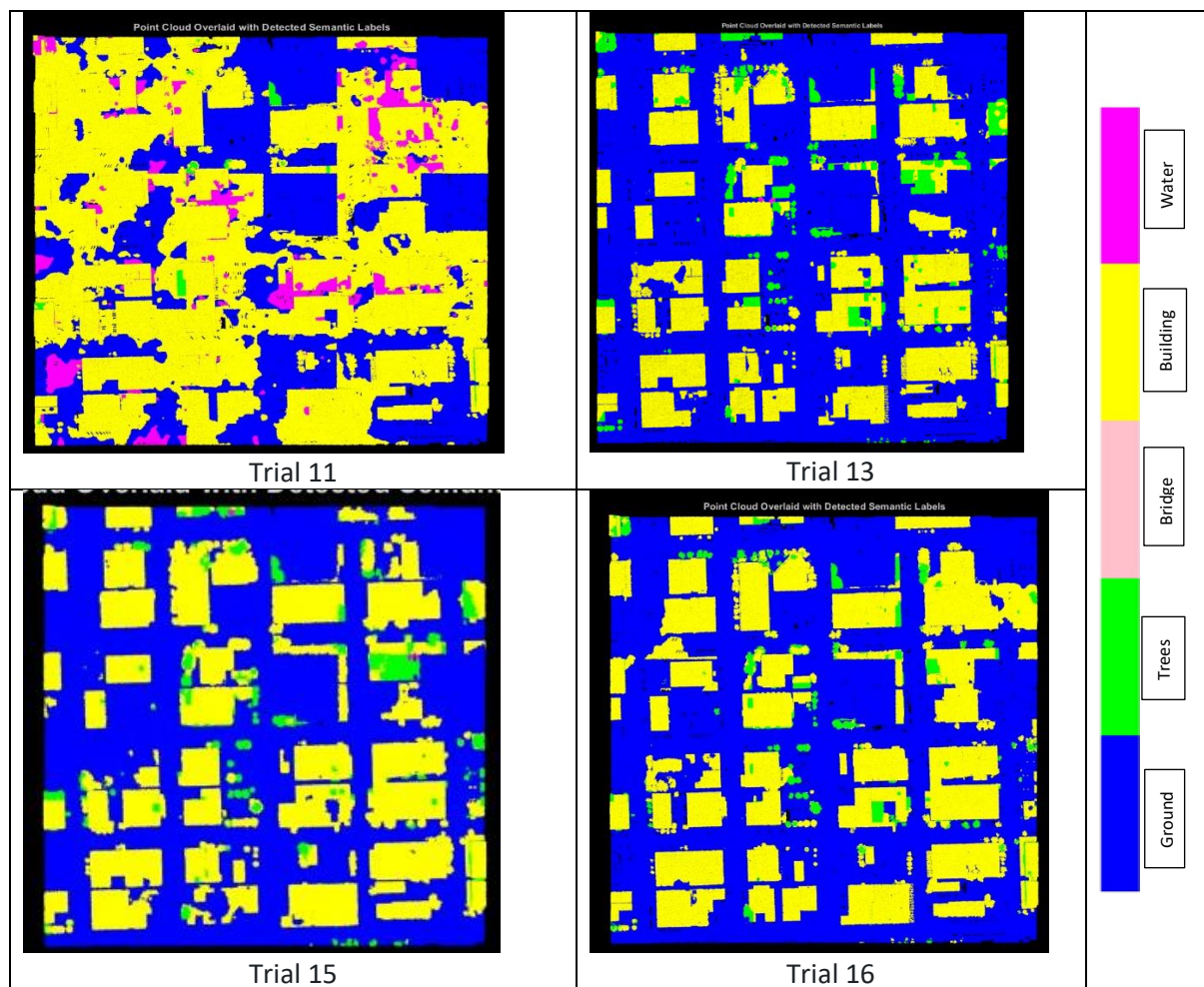


Figure 4 OMA_290 Result of Different Trials

b. RESULT

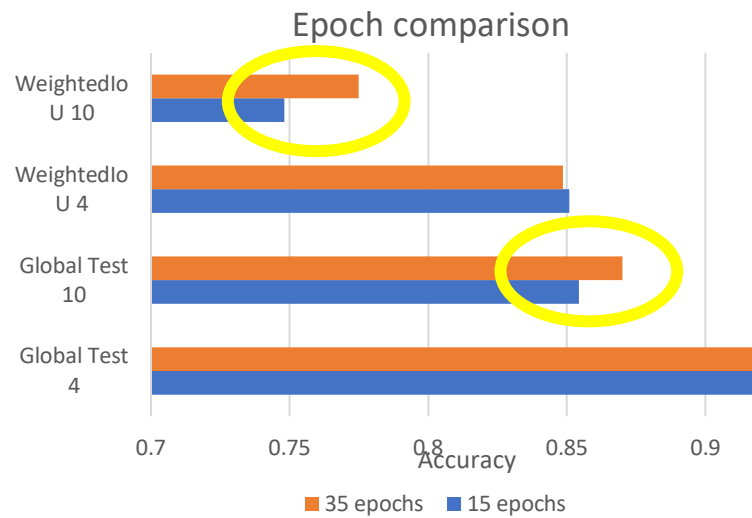


Figure 5 Epoch comparison in test data 10 and 4 with 15 and 35 epoch

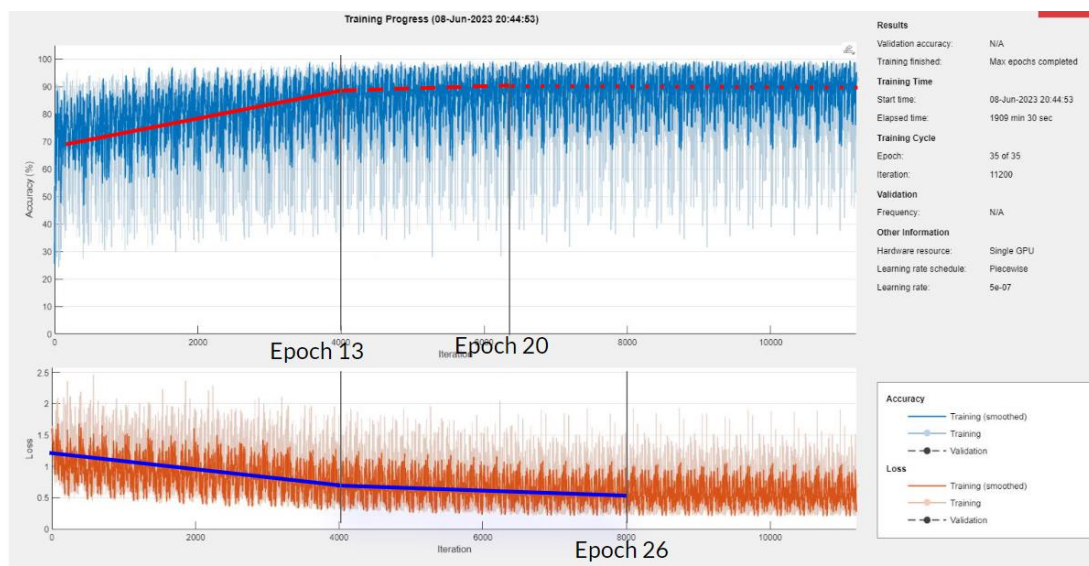


Figure 6 Training graph for epoch 35

The comparison of the results reveals interesting trends. The Global test accuracy for Test Tile 4 is 0.9205 at 15 epochs and slightly reduced at 35 epochs (0.91838), but the difference in accuracy is insignificant. Similarly, the WeightedIoU also shows slight improvement at 15 epochs. On the other hand, the Global test accuracy for Test Tile 10 is 0.85441 at 15 epochs and increases to 0.87003 at 35 epochs, indicating a more substantial improvement at 35 epochs. The WeightedIoU follows a similar trend. The analysis of Test Tile 10 reveals a clear difference in accuracy improvement with increasing numbers of epochs.

Further analysis indicates that the accuracy continues to improve until epoch 20 and then plateaus, while the loss function consistently decreases until epoch 26. This suggests that the optimal epoch for this project is 26, striking a balance between accuracy and loss reduction. Considering this, Test Tile 10 provides more reliable results than Test Tile 4.

However, it is important to consider the runtime length as well. The training process for 35 epochs takes significantly longer, with a runtime of 1909 minutes, which is more than double the runtime of 15 epochs (788 minutes).

In summary, although there is a slight improvement in accuracy with more training epochs, considering the time constraint, it is more efficient to use 15 epochs for this project.



Figure 7 Train data comparison for 40,50, and 60 train data in test 4 and test 10.

For Test Tile 4, the global test accuracy gradually increases from 40 Training tiles (0.86986) to 50 Training tiles (0.90634) and reaches the highest value at 60 Training tiles (0.9205). Similarly, for Test Tile 10, the global test accuracy shows a similar trend, increasing from 40 Training tiles (0.58584) to 50 Training tiles (0.84609) and reaching its highest value at 50 Training tiles (0.85441). Weighted IoU exhibits a similar pattern.

In conclusion, a general improvement in both the global test accuracy and the weighted IoU for both the 4 and 10 test tiles can be seen as the train data increases from 40 tiles to 60 tiles. This suggests that the model's performance becomes more accurate and effective in segmenting urban areas as the train data becomes more comprehensive and representative.

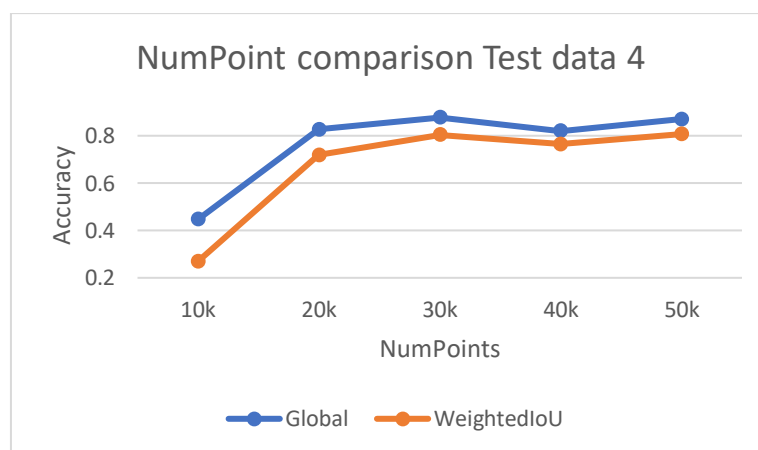


Figure 8 NumPoint comparison in test data 4 with using 10,000 until 50,000

Initially, the global test accuracy starts at a lower value of 0.44652 for a number of points of 10,000. As the number of points increases, the global test accuracy improves significantly, reaching its highest value of 0.87678 at 30,000 Numpoints. However, at 40,000 Numpoints, there is a slight drop in global test

accuracy to 0.81954, followed by a subsequent increase to 0.86986 at 50,000 points, which is similar to 30,000 Numpoints.

Like the global test accuracy, the weighted IoU exhibits an increasing trend as the NumPoint increases. When the number of points exceeds 50,000, it surpasses the computational capacity of the GPU.

In summary, the results indicate that as the number of points increases, the model's performance in terms of both global test accuracy and weighted IoU generally improves. The highest performance in terms of global test accuracy and weighted IoU is achieved with 30k points, followed by 50k points, although the difference between the two is minimal.

However, using 50k points provides a better visual representation of the blocks, with points evenly distributed for clearer object visualization. Thus, 50k points are chosen for this project to achieve slightly better overall performance and a more visually comprehensive representation of the objects.

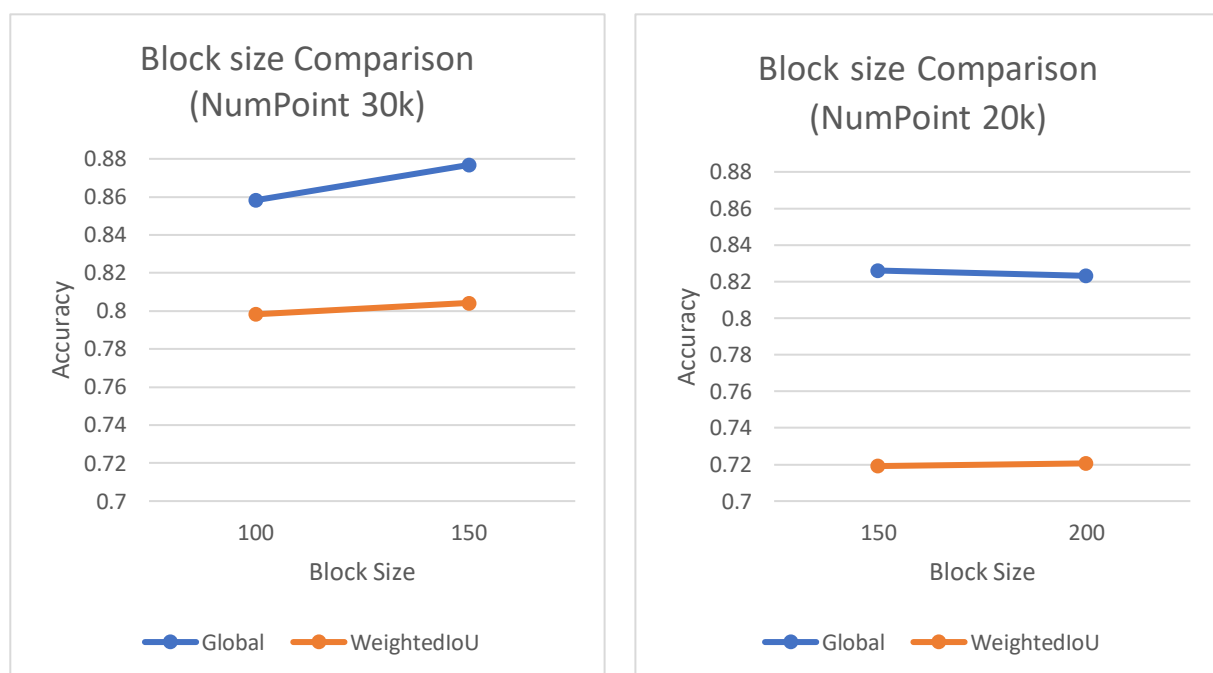


Figure 9 Blocksize comparison for test data 4 in 100 and 150 block size in NumPoint 30k (left) and 150 and 200 blocksize in NumPoint 20k

A block size of 150 achieves higher global test accuracy (0.87678) compared to a block size of 100 (0.85824) with 30k points. Similarly, block size 150 has slightly higher global test accuracy (0.82607) than block size 200 (0.82317) with 20k points. These differences suggest that using a block size of 150 provides better results in terms of both global test accuracy and weighted IoU. Overall, a block size of 150 is more effective in capturing and segmenting objects within urban areas for the project.

6. CONCLUSION

In conclusion, the analysis of the results reveals several key findings for the project of 3D semantic segmentation of urban areas using the PointNet++ network, which are described below:

For training epochs, there is a trade-off between runtime and accuracy. Training for 35 epochs slightly improves accuracy but significantly increases runtime. Considering the diminishing returns and time constraints, it's more practical to use 15 epochs.

Furthermore, comparing test data sets shows that the model's performance improves as the number of training tiles increases. Both global test accuracy and weighted IoU consistently improve as the train data becomes more comprehensive and representative. However, further analysis and validation are required to assess the model's performance regarding data diversity and potential biases.

In summary, increasing the number of points improves the model's performance, with the best results observed at 30,000 and 50,000 points. Using 50,000 points provides a visually comprehensive representation of objects. Further analysis can explore additional factors like data quality and model architecture to understand performance with different data sizes comprehensively.

Regarding the block size, the analysis suggests that using a block size of 150 generally yields better results in terms of both global test accuracy and weighted IoU.

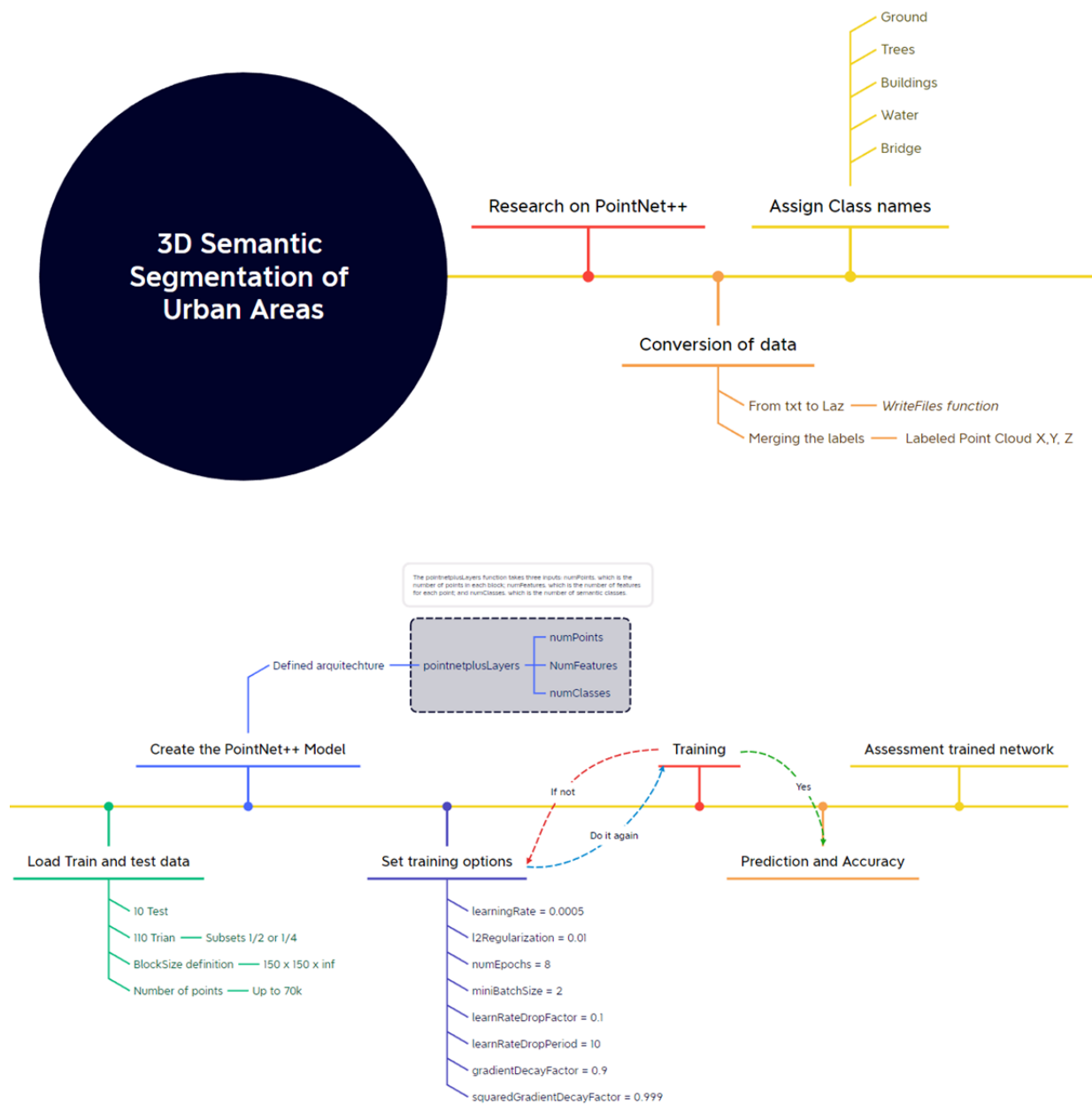
Considering the trade-offs between accuracy, runtime length, and visualization, the recommended approach for this project would be to train the model for 15 epochs, use 50,000 points, and have a block size of 150. These choices balance achieving satisfactory performance, minimizing training time, and ensuring a visually comprehensive representation of the objects within urban areas.

In this study, two key parameters, NumPoints and the number of train data, significantly influenced accuracy. The study's runtime was primarily affected by epoch, minibatch, and block size. The highest accuracy was achieved with a block size of 150, 50k NumPoints, and 60 train data sets, but epochs varied between 15 epochs (test data 4) and 35 epochs (test data 10).

REFERENCES

1. MatLab. (2023). Getting Started with PointNet++—MATLAB & Simulink—MathWorks Deutschland [Educational]. MathWorks Help Center. <https://de.mathworks.com/help/lidar/ug/get-started-pointnetplus.html> [Access Date: 20th March, 2023]
2. Qi, Charles R., Li Yi, Hao Su, and Leonidas J. Guibas. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space." ArXiv:1706.02413 [Cs], June 7, 2017. <https://arxiv.org/abs/1706.02413>.
3. Varney, Nina, Vijayan K. Asari, and Quinn Graehling. "DALES: A Large-Scale Aerial LiDAR dataset for Semantic Segmentation." ArXiv:2004.11985 [Cs, Stat], April 14, 2020. <https://arxiv.org/abs/2004.11985>.
4. MatLab. (2023). Aerial Lidar Semantic Segmentation Using PointNet++ Deep Learning — MathWorks Deutschland [Educational]. MathWorks Help Center. <https://de.mathworks.com/help/lidar/ug/aerial-lidar-segmentation-using-pointnet-network.html> [Access Date: 4th April, 2023]

Appendix 1 WORKFLOW

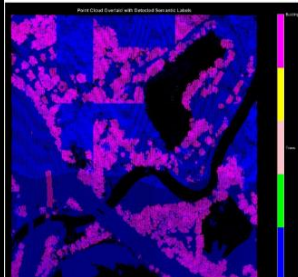
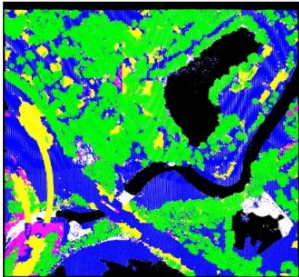
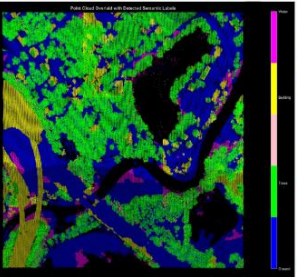
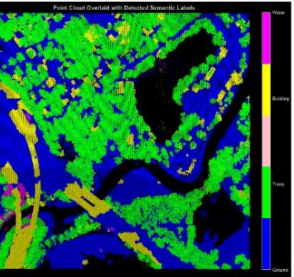


Appendix 2 Experiments Tables and Figures

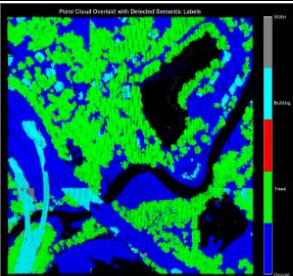
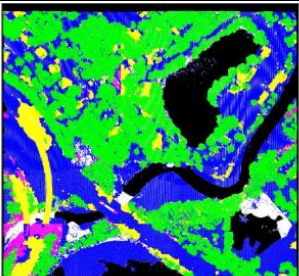
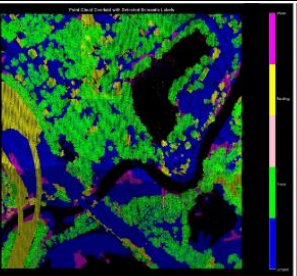
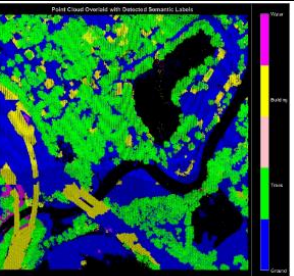
	Trial number	Trial 1	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8	Trial 10	Trial 11	Trial 13	Trial 15	Trial 16
Hyperparameters	Train Data	1 to 40	1 to 40	1 to 40	1 to 40	1 to 40	1 to 40	1 to 40	1 to 40	41 to 90	41 to 100	41 to 100
	Test Data	4 (JAX_328)										
	Mini Batch	6	2	4	10	6	10	8	8	6	6	4
	Epoch	4	15	15	15	15	15	15	15	15	15	35
	Block Size	250	150	150	200	150	100	150	150	150	150	150
	NumPoint	40000	10000	20000	20000	30000	30000	40000	50000	50000	50000	50000
	Learning Rate	0.0025	0.0005	0.0005	0.0005	0.0005	0.0005	0.0005	0.0005	0.0005	0.0005	0.0005
Accuracy Assessment	Run Time	-	435 min	390 min	212 min	431 min	954 min	533 min	613 min	572 min	788 min	1909 min
	Ground	0.75874	0.99978	0.95749	0.91696	0.93374	0.86994	0.78677	0.91274	0.96124	0.99725	0.99007
	Trees	0.70891	0.000182	0.81134	0.84167	0.9373	0.94596	0.92395	0.93428	0.98845	0.99205	0.99235
	Building	0.75321	0.42729	0.51374	0.46177	0.48465	0.59699	0.51466	0.5692	0.28963	0.20044	0.21586
	water	Nan	NaN	0.0017	0.042662	0.018402	0.041339	0.31197	0.035898	0.006877	0.005479	0.013226
	bridge	0	NaN	0	0	0	0	0	0	0	0	0
	Global	0.74902	0.44652	0.82607	0.82317	0.87678	0.85824	0.81954	0.86986	0.90634	0.9205	0.91838
	WeightedIoU	0.62937	0.26932	0.71916	0.72063	0.80424	0.79824	0.76518	0.80698	0.83892	0.85102	0.8485
	Mean Accuracy	NaN	NaN	0.45685	0.45261	0.47482	0.49084	0.50747	0.49042	0.44924	0.43904	0.4423

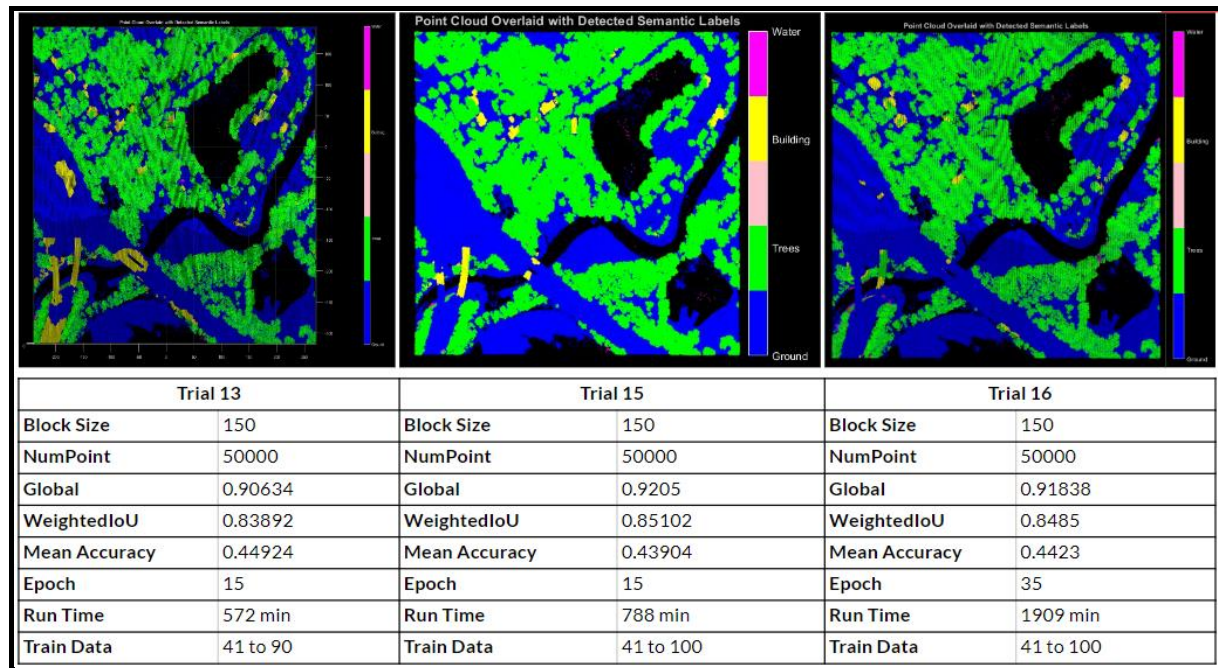
	Trial number	Trial 8	Trial 11	Trial 13	Trial 15	Trial 16
Hyperparameters	Train Data	1 to 40	1 to 40	41 to 90	41 to 100	41 to 100
	Test Data	10 (OMA_290)				
	Mini Batch	10	8	6	6	4
	Epoch	15	15	15	15	35
	Block Size	100	150	150	150	150
	NumPoint	30000	50000	50000	50000	50000
	Learning Rate	0.0005	0.0005	0.0005	0.0005	0.0005
Accuracy Assessment	Run Time	954 min	613 min	572 min	788 min	1909 min
	Ground	0.80057	0.50238	0.99733	0.99995	0.9778
	Trees	0.33946	0.023362	0.38595	0.27944	0.33801
	Building	0.62777	0.77279	0.6539	0.68256	0.7533
	water	NaN	NaN	NaN	NaN	NaN
	bridge	NaN	NaN	NaN	NaN	NaN
	Global	0.71878	0.58584	0.84609	0.85441	0.87003
	WeightedIoU	0.65456	0.42992	0.74265	0.7481	0.77483
	Mean Accuracy	NaN	NaN	NaN	NaN	NaN

Test Data 4 (JAX_328)

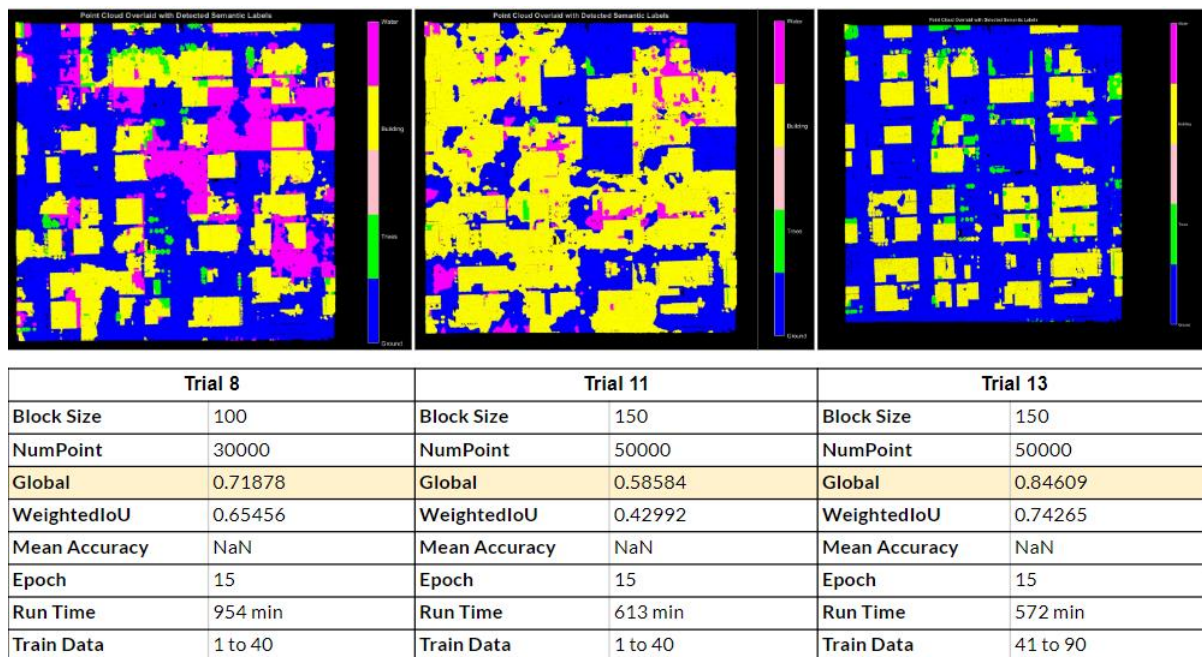





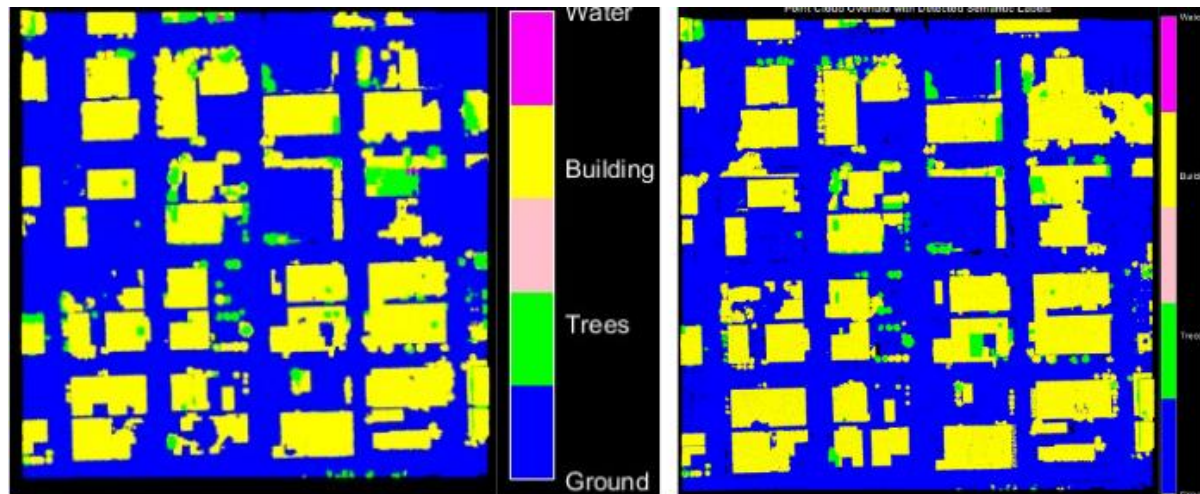
Trial 4		Trail 8		Trial 10		Trial 11	
Block Size	150	Block Size	100	Block Size	150	Block Size	150
NumPoint	10000	NumPoint	30000	NumPoint	40000	NumPoint	50000
Global	0.44652	Global	0.85824	Global	0.81954	Global	0.86986
WeightedIoU	0.26932	WeightedIoU	0.79824	WeightedIoU	0.76518	WeightedIoU	0.80698
Mean Accuracy	Nan	Mean Accuracy	0.49084	Mean Accuracy	0.50747	Mean Accuracy	0.49042
Epoch	15	Epoch	15	Epoch	15	Epoch	15
Run Time	435 min	Run Time	954 min	Run Time	533 min	Run Time	613 min
Train Data	1 to 40	Train Data	1 to 40	Train Data	1 to 40	Train Data	1 to 40

							
Trial 7		Trail 8		Trial 10		Trial 11	
Block Size	150	Block Size	100	Block Size	150	Block Size	150
NumPoint	30000	NumPoint	30000	NumPoint	40000	NumPoint	50000
Global	0.87678	Global	0.85824	Global	0.81954	Global	0.86986
WeightedIoU	0.80424	WeightedIoU	0.79824	WeightedIoU	0.76518	WeightedIoU	0.80698
Mean Accuracy	0.47482	Mean Accuracy	0.49084	Mean Accuracy	0.50747	Mean Accuracy	0.49042
Epoch	15	Epoch	15	Epoch	15	Epoch	15
Run Time	431 min	Run Time	954 min	Run Time	533 min	Run Time	613 min
Train Data	1 to 40	Train Data	1 to 40	Train Data	1 to 40	Train Data	1 to 40



Test Data 10 (OMA_290)





Trial 15		Trial 16	
Block Size	150	Block Size	150
NumPoint	50000	NumPoint	50000
Global	0.85441	Global	0.87003
WeightedIoU	0.7481	WeightedIoU	0.77483
Mean Accuracy	NaN	Mean Accuracy	NaN
Epoch	15	Epoch	35
Run Time	788 min	Run Time	1909 min
Train Data	41 to 100	Train Data	41 to 100