USING SIMULATION DATA FROM GAMING ENVIRONMENTS FOR TRAINING A DEEP LEARNING ALGORITHM ON 3D POINT CLOUDS

Steven Spiegel[1] and Jorge Chen[2]

[1]Oak Ridge Institute for Science and Education, 4692 Millennium Dr, Suite 101, Belcamp, Maryland 21017, USA, steve.s.spiegel@gmail.com
[2]Oak Ridge Institute for Science and Education, 4692 Millennium Dr, Suite 101, Belcamp, Maryland 21017, USA,jorgechen@geog.ucsb.edu

KEY WORDS: deep learning, point clouds, computer vision, gaming engines

ABSTRACT:

Deep neural networks (DNNs) and convolutional neural networks (CNNs) have demonstrated greater robustness and accuracy in classifying two-dimensional images and three-dimensional point clouds compared to more traditional machine learning approaches. However, their main drawback is the need for large quantities of semantically labeled training data sets, which are often out of reach for those with resource constraints. In this study, we evaluated the use of simulated 3D point clouds for training a CNN learning algorithm to segment and classify 3D point clouds of real-world urban environments. The simulation involved collecting light detection and ranging (LiDAR) data using a simulated 16 channel laser scanner within the the CARLA (Car Learning to Act) autonomous vehicle gaming environment. We used this labeled data to train the Kernel Point Convolution (KPConv) and KPConv Segmentation Network for Point Clouds (KP-FCNN), which we tested on real-world LiDAR data from the NPM3D benchmark data set. Our results showed that high accuracy can be achieved using data collected in a simulator.

1. INTRODUCTION

Low elevation and mobile generated point clouds are becoming increasingly easy to obtain. LiDAR scanners are no longer limited to high elevation aerial collects, nor are they restricted to cumbersome terrestrial scanners that need to be moved manually, as more companies move to using small unmanned aerial systems (sUAS) and laser scanners attached to vehicles. These newer platforms provide a more streamlined and affordable collection of point cloud data, which introduces new challenges to the point cloud community, particularly in the semantic segmentation of this data.

1.1 Traditional Methods

Traditional machine learning algorithms and non machine learning techniques have historically been used for classification and segmentation tasks. The pipeline of these algorithms is to compute pointwise geometric features, such as geometric features derived from a radial or k-nearest neighbors covariance matrix, and then use the features from the covariance matrix to run it through a traditional machine learning algorithm such as a support vector machine or a random forest classifier (Thomas, 2019). The features are based on using the resultant eigenvalues $(\lambda_1 > \lambda_2 > \lambda_3)$ and corresponding eigenvectors (e_1, e_2, e_3). These features include point omnivariance $\sqrt[3]{(\lambda_1\lambda_2\lambda_3)}$, linearity $(\frac{\lambda_1-\lambda_2}{\lambda_1})$, eigenentropy $(-\sum_{i=1}^{3}\lambda_i ln(\lambda_i))$, sum of eigenvalues $(\sum_{i=1}^{3}\lambda_i)$, and planarity $(\frac{\lambda_2-\lambda_3}{\lambda_1})$. These discriminate features allow for a traditional machine learning algorithm such as a random forest or support vector machine classifier to give accurate results (Weinmann et al., 2013). A study done by (Weinmann et al., 2013) used point features calculated from the covariance matrix as inputs for several classifiers on the Oakland 3D point dataset, obtaining an overall accuracy of 93.32 percent. The objects were labeled as wire, pole/trunk, facades, ground, and vegetation. The Oakland 3D dataset is a LiDAR point cloud of the CMU campus and comes split into a training, validation, and test points (Munoz et al., 2009).

One of the problems with using these more traditional techniques is they do not leverage the capability of deep learning algorithms, that is learning hierarchical structures of data, and are therefore limited in their ability automatically classify data. Although the accuracy seen in (Weinmann et al., 2013) is above 90 percent, it relies heavily on handcrafted features for accurate classification and using one dataset split into training and testing sections. A deep learning algorithm is able to learn these features on its own without the need for handcrafted features.

1.2 Deep learning in point clouds

Recently, more focus has been put on deep learning techniques due to their success in image classification. Neural networks, specifically convolutional neural networks, are able to learn abstract features of image data and give better results than using traditional machine learning on images. This success can also be found in 3D point datasets. Older methods use multiview representations for points, which consists of projecting 3D data onto multiple 2D planes and using 2D CNNs for segmentation (Guo et al., 2020). Other methods use a voxelization method where the point cloud is split into 3D occupancy grids and ran through a modified version of an image CNN (Chen et al., 2019). Pointnet and Pointnet++ were groundbreaking in applying deep learning to directly to points within point clouds. This eliminated the need to project points to 2D space and removed the need for computationally expensive voxel grids (Qi et al., 2017). Later pointwise algorithms improved on this concept and applied convolution neural networks to points as seen in (Thomas et al., 2019).

1.3 Simulated Data

Deep learning on point clouds has shown to be an effective method of point cloud segmentation, though their effectiveness

Approved for public release, 21-795

depends on the quantity and quality of training data. The newer generations of low elevation mobile scanning solutions address the quantity challenge but acquiring high quality pre-classified training data remains a non trivial endeavor. Point clouds typically consist of millions of points, which makes hand labelling points for semantic segmentation a difficult and time consuming task. There are benchmark datasets that come labeled for a variety of classification tasks, including semantic segmentation. Popular ones include the Semantic3D dataset (Hackel et al., 2017), Paris-rue-Madame dataset (Serna et al., 2014), and the TerraMobilita/iQumulus dataset (Vallet et al., 2015). The Semantic3D dataset is an outdoor point cloud collected using a terrestrial laser scanner containing over 4 billion hand annotated points (Hackel et al., 2017) while the TerraMobilita and Paris-rue-Madame datasets were collected using a mobile laser scanner (Vallet et al., 2015) (Serna et al., 2014). All of these datasets use traditional point segmentation techniques and hand annotation to label the points for training and testing. This requires a large amount of time and usually utilizes a team of people to accomplish. This is particularly true for semantic and panoptic segmentation tasks. Semantic and panoptic segmentation involve labeling every point in the scene, whether these are pixels in an image or points in a point cloud. Hence, collecting data in gaming engines and modeling suites help address the challenges presented in these segmentation tasks, and is one of the motivators of this study. In this paper, we explore using an autonomous vehicle simulator created in a gaming engine to collect training data and tested the results on a LiDAR dataset. We will show that training a deep learning algorithm on computer generated data can give superior results to that of a more traditional machine learning algorithm as described in section 1.1.

2. PREVIOUS WORK

Research into utilizing computer generated data for training is still relatively new. SqueezeSeg was one of the pioneers in this endeavor. They used a plugin for the video game *Grand Theft Auto V* (*GTA-V*) to attach a VLP-16 LiDAR scanner on top of a car in the video game. They used this scanner to collect point cloud data by driving the car around in game in order to supplement their training set (Wu et al., 2017). They then tested the trained algorithm on the benchmark KITTI dataset, a real world LiDAR collection utilizing a VLP system (Geiger et al., 2012). When only trained on the *GTA-V* data, the class level intersection over union (IoU) accuracy was 29.0 (Wu et al., 2017).

Another attempt at utilizing simulated data is from the creators of *SynthCity*. *Synth City* is a globally registered labeled point cloud collected in the modelling program Blender (Griffiths and Boehm, 2019). A Gaussian noise level of $\sigma = 0.5$ cm is applied on each axis to simulate real world sensor noise. The model simulates an urban environment with ground, natural, building, points (Griffiths and Boehm, 2019). Scanning simulation was done using the Blender plugin, BlenSor. BlenSor allows for the simulation of 3D sensors such as LiDAR and Kinect sensors (Gschwandtner et al., 2011).

Gaming engines, specifically Unreal Engine, offer a variety of solutions for vehicle and sensor simulations. Microsoft AirSim offers a simulation space to test out sensors attached to a sUAS. This allows for testing SLAM (simultaneous localization and mapping) based algorithms and learning based approaches within a simulator (Shah et al., 2017). CARLA (Car Learning to Act) is similar to AirSim, however it is primarily an autonomous vehicle simulator used to test autonomous driving algorithms and vehicles in a safe setting. It also provide a plethora of sensors to test in the simulator. These sensors include RGB cameras, radar sensors, depth cameras, LiDAR sensors, Global Navigation Satellite Systems (GNSS), and inertial measurement units (IMU). Many of the pipelines done in CARLA are for reinforced learning and sensor testing, as these are heavily used in autonomous driving (Dosovitskiy et al., 2017). The abundance of sensors allows for extensive testing and collection of LiDAR data, particualry labeled LiDAR data. Other researchers have conducted similar studies by combining LiDAR data from CARLA with the KITTI dataset for object detection (Dworak et al., 2019).

Our research differs from (Dworak et al., 2019) and (Wu et al., 2017) in that we only use simulated data collected in CARLA for the task of semantic segmentation. We train a deep learning algorithm on the simulated data and test it on real world point clouds. Our approach also differs from (Griffiths and Boehm, 2019) in how we apply noise to our simulated LiDAR data. *Synth City* applies a noise factor along each axis, but we apply noise to the range of the scanner. We also apply the noise per scan instead of applying the noise after the collection is done. We also use a algorithm that is well suited for semantic segmentation.

2.1 Kernel Point Convolution (KPConv)

In recent years, there have been many advances in 3D deep learning algorithms as mentioned in section 1.2. The algorithm we chose is Kernel Point Convolution, which applies a convolution operation directly to points (Thomas et al., 2019).

In image based CNNs, each pixel is directly multipilied to the weight matrix W_k followed by an activation function. Due to the irregular nature of point clouds, a point p is not likely to be aligned with the kernel points. Hence, KPConv introduces a kernel function that correlates distances between p and kernel points (Thomas, 2019). Let point cloud P with N points be such that $P \in \mathbb{R}^{Nx3}$ and point features $F \in \mathbb{R}^{NxD}$ We will let general point convolution be defined by the following equation (Thomas, 2019):

$$(F * g)(x) = \sum_{x_i \in \mathcal{N}_x} g(y_i) f_i \tag{1}$$

where $y_i = x_i - x$, $x_i \in P$, $f_i \in F$, and $\mathcal{N}_x = \{||x_i - x|| \leq r\}$ for a radius r. Kernel function g is defined as

$$g(x) = \sum_{k<K} h(y, \tilde{x}_k) W_k \tag{2}$$

where $\tilde{x}_k$ are kernel points and W_k are the associated weight matrices. h id a correlation function, where points closer to kernel points have a greater correlation factor than points further away. Generally it is written as

$$h(y_i, \tilde{x}_k) = max\left(0, 1 - \frac{||y_i - \tilde{x}_k||}{\sigma}\right) \tag{3}$$

Each feature vector is multiplied and summed over all K weight matrices. Further details of KPConv can be found in (Thomas et al., 2019).

3. METHODS

3.1 Collecting the Data in CARLA

CARLA is built using Unreal Engine 4, which is typically used for modelling, gaming, and simulation. The version of CARLA used in this experiment is CARLA 0.9.10. Data is collected using a semantic LiDAR scanner that is available in the CARLA sensor suite. Each return of a scan from the LiDAR sensor has an (x, y, z) coordinate as well as an object tag, object index, and cosine of the incidence angle. Note we are primarily interested in the object tag. It is also possible to adjust the number of channels, range, points per second, rotation frequency, field of view, and sensor tick of the scanner. This may be adjusted to better reflect different sensors. Our LiDAR sensor was set to 16 channels, had an upper and lower field of view of 15 degrees, and a maximum range of 100m. These settings are similar to a VLP-16 LiDAR scanner. The position of the LiDAR scanner was set to 1m forward of the car center and 2.8 m above center. Note also that CARLA uses a left handed coordinate system where x is forward, y is right, and z is up. Maps can either be created or downloaded using Unreal Engine. The map chosen is named "Town 3" in the CARLA map collection, which is a built-in map. Even though other maps are available, this one was chosen for its urban setting.

For real world mobile LiDAR scans, three sensors are involved: GNSS, IMU, and LiDAR scanner. The point cloud is generated using the following equation (Glennie, 2007):

$$p_G^l = p_{GPS}^l + R_b^l \cdot R_s^b \cdot r^s - R_b^l \cdot l^b \tag{4}$$

p_G^l are the coordinates in the local frame.
p_{GPS}^l are the GPS coordinates in the local frame.
R_b^l are the roll, pitch, and yaw rotations from the body frame to the local frame. This is given by the IMU
R_s^b is the rotation matrix from the scanner frame to the body frame
r^s are the point coordinates in the scanner frame
l^b is the lever arm offset from the scanner origin to the navigation origin with respect to the body frame

Errors in IMU attitude, boresight measurements, and GNSS measurements propagate throughout the system (Glennie, 2007). CARLA allows for testing all three of these systems, including a Gaussian error effect for both the IMU and GNSS (Dosovitskiy et al., 2017). For our purposes, we simplified the process by only considering error in the scanner's range measurements. The transformation and rotation matrix is taken from the LiDAR scanner in CARLA, and is the exact transform from the LiDAR scanner coordinate frame to the world (local) frame. Therefore, R_s^b is the identity matrix and l^b is $\overrightarrow{0}$. Hence, equation 4 reduces to

$$p_G^l = R_{rot} \cdot r^s \tag{5}$$

where R_{rot} is the $4x4$ affine transformation and rotation matrix from the scanner frame to the local frame of the form:

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{23} & R_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

The point r^s in the scanner frame is a $4x1$ column vector with coordinates $\begin{bmatrix} r_x & r_y & r_z & 1 \end{bmatrix}$ in the scanner coordinate reference frame. Entry R_{ij} is the result of 3 matrix multiplications of the rotation matrix about the z axis, y axis, and x axis given by $R_z \cdot R_y \cdot R_x$. This is often called the Tate-Bryant sequence of rotations, where for rotation angle α, β, γ about the x, y, z axis (respectively), we get:

$$R_z = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix} \tag{7}$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix}$$

So the first three rows and columns of $R_{rot} = R_z \cdot R_y \cdot R_x$ give the resultant $3x3$ rotation matrix. The translation column vector $\begin{bmatrix} t_x & t_y & t_z & 1 \end{bmatrix}$ give the x, y, z translation from the scanner coordinate frame to the local frame.

3.2 Adding Noise

Data collects such as Synth City apply noise to each axis, however this is not how noise is applied to laser scanners. Laser scanners such as the VLP-16 collect range measurements and then convert them to a Cartesian coordinate system. The semantic LiDAR scanner in CARLA collects points in (x, y, z) coordinate system. For our purposes, we assume that error in range values are normally distributed with a standard deviation ($\sigma_{scanner}$) of 2 cm. Points are collected in Cartesian coordinates and converted to spherical coordinates, where ρ is the range, ϕ is the elevation angle, and θ is the azimuth angle. Thus, for a given scan S with N points and $i \in [1, N]$, we have:

$$\begin{aligned} \rho_i &= \sqrt{x_i^2 + y_i^2 + z_i^2} \\ \phi_i &= \arccos\frac{z_i}{\rho_i} \\ \theta_i &= \arctan\frac{y_i}{x_i} \end{aligned} \tag{8}$$

Note that the arctan range is from $[-\pi, \pi]$ (In modern programming languages, this is the arctan2 function) to account for the different signs of x and y. So for a given range, ρ_i, a new range value is $\rho_i^{(n)} = \rho_i + n$, where n is a drawn sample from a normal distribution with $\mu_{scanner} = 0$ and $\sigma_{scanner} = 0.02$. So converting back to point $p_i^{(n)} = (x_i^{(n)}, y_i^{(n)}, z_i^{(n)})$, we get:

$$\begin{aligned} x_i^{(n)} &= \rho_i^{(n)} \sin\phi_i \cos\theta_i \\ y_i^{(n)} &= \rho_i^{(n)} \sin\phi_i \cos\theta_i \\ z_i^{(n)} &= \rho_i^{(n)} \cos\phi_i \end{aligned} \tag{9}$$

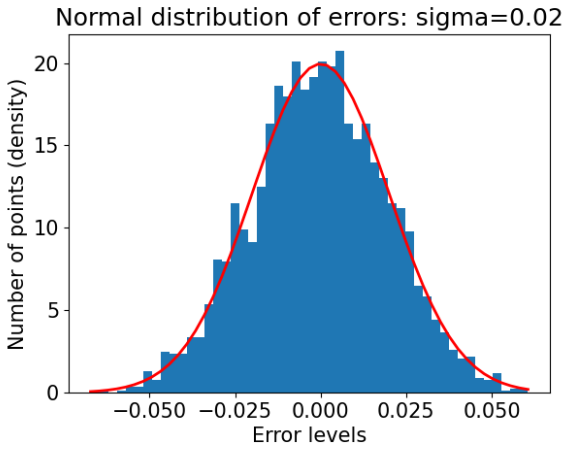

Figure 1. Error levels added to ρ

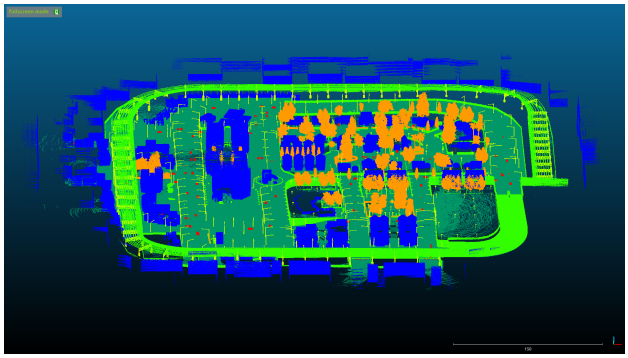

Figure 2. Training data from CARLA with added noise

This is done per scan. Note that each point, p_i has a label associated with it. Data is collected similarly to a mobile laser scan in an urban setting. The LiDAR scanner is attached to the car and driven using a heads up display (HUD). When the data is collected, it is combined into a single point cloud by applying equation 6 to the scan points, r^s after noise has been added. Note that r^s is a column vector of size $4xS$, where S is the number of points per scan. Hence,

$$r^s = \begin{bmatrix} t_x^0 & \dots & t_x^S \\ t_y^0 & \dots & t_y^S \\ t_z^0 & \dots & t_z^S \\ 1 & \dots & 1 \end{bmatrix} \tag{10}$$

So applying the rotation matrix per scan, R_{rot}^S we get

$$\begin{bmatrix} t_x^{final(0)} & \dots & t_x^{final(S)} \\ t_y^{final(0)} & \dots & t_y^{final(S)} \\ t_z^{final(0)} & \dots & t_z^{final(S)} \\ 1 & \dots & 1 \end{bmatrix} = R_{rot}^S \cdot \begin{bmatrix} t_x^0 & \dots & t_x^S \\ t_y^0 & \dots & t_y^S \\ t_z^0 & \dots & t_z^S \\ 1 & \dots & 1 \end{bmatrix} \tag{11}$$

This gives us the final (x, y, z) coordinates of the point cloud.

Following this the data density is reduced and made uniform through subsampling. The point cloud is first split into a voxel grid of size 1 cm. Then for a voxel grid V, and all points $p \in V$, the point that is closest to the center of the voxel is kept and all other points are removed. Additionally, the points are resampled as the point closest to the center of the voxel is moved to the voxel center (PDAL Contributors, 2018). They are then relabeled to buildings, ground, miscellaneous, pole-like, vegetation, and vehicles.

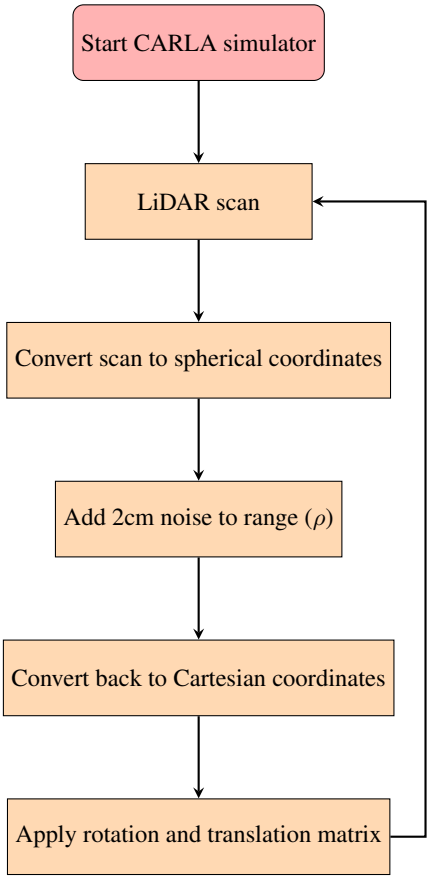

Figure 3. Flowchart of data collect

3.3 Test Data

Test data was selected from the NPM3D benchmark dataset, which we call the Lille dataset (Roynard et al., 2018). This real world LiDAR data was collected using a Novatek FlexPak 6 GNSS, an Ixsea PHINS IMU, and a Velodyne HDL-32E LiDAR scanner. Points come with (x, y, z) coordinates and RGB data. Data was collected in Paris and Lille, France, however the test data only uses 1150 m of the Lille data, totalling approximately 30 million points (Roynard et al., 2018). Original labels include buildings, poles, bollards, trash cans, barriers, pedestrians, cars, and natural, which were changed to fit our schema.

3.4 Running KPConv on training data

KPConv architecture comes in two varieties, classification (KP-CNN) and segmentation (KP-FCNN) (Thomas, 2019). Since this is a segmentation task, we chose the KP-FCNN, which consists of 5 encoder layers similar to ResNet blocks (He et al., 2016) and nearest upsampling for the decoder section. This gives pointwise feature classification, which is the goal for scene segmentation (Thomas, 2019). The KPConv algorithm was run using the Pytorch implementation of KPConv. Pytorch is a GPU accelerated deep learning python API allowing for the

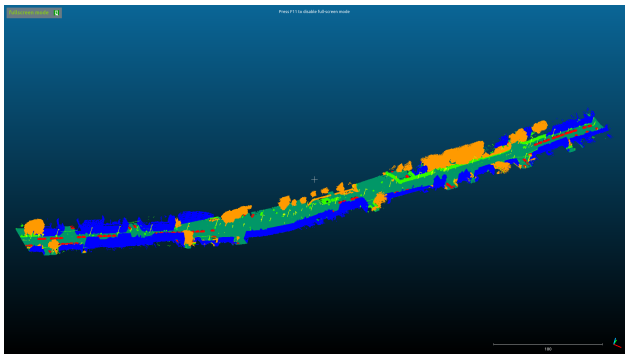

Figure 4. Lille, France test dataset

Figure 5. Heads up display of vehicle being driven in CARLA

training of deep neural and convolution neural networks (Paszke et al., 2019). The training data from CARLA was downsampled to 8 cm to save space. Training data was split into 9 sections (labeled 0 - 8). Section 3 was chosen for validation, as it had good representation from points of all classes. The algorithm was trained with a batch size of 4 and done for 500 epochs. The results of training were then tested on the Lille, France dataset.

3.5 Comparison with Traditional methods

We split the Lille dataset into training and testing data to compare traditional methods of segmentation with our method. We ran a random forest classifier on five handcrafted features similar to the ones used in (Weinmann et al., 2013). The five features used were eigenentropy, linearity, and sum of eigenvalues as shown in section 1.1. Local point density of k nearest neighbors, $D = \frac{k+1}{\frac{4}{3}\pi r^3_{knn}}$ and sum of the eigenvalues from the covariance matrix when the points are projected to 2D space were also included as done in (Weinmann et al., 2013).

4. RESULTS

4.1 Accuracy metrics

The result of testing can be seen in figure 7. Overall accuracy for the algorithm was 93.8 percent. Table 1 shows the confusion matrix of all values, rounded to 2 decimal places and table 2 shows the intersection over union (IoU) results. The best results came from ground and buildings, both having accuracy results above 90 percent and IoU scores above 0.80. The

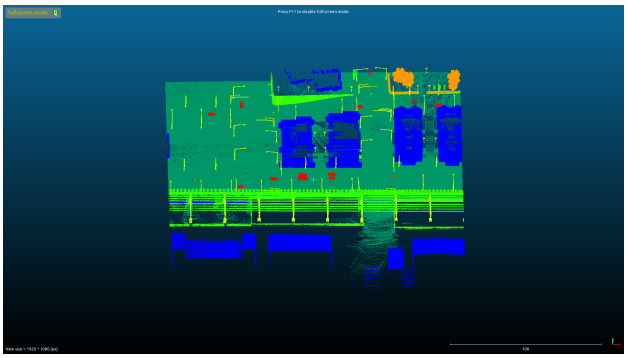

Figure 6. Validation set derived from CARLA

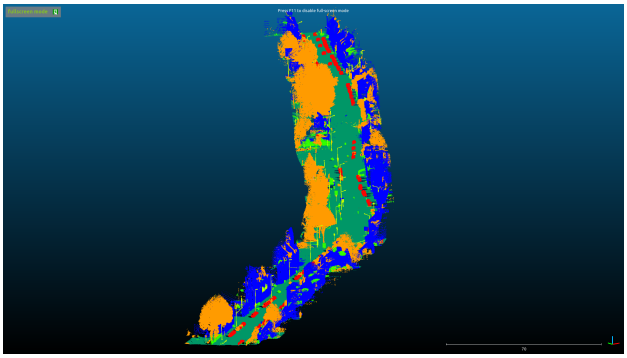

Figure 7. Visual results of KPConv

buildings	ground	miscellaneous	pole-like	vegetation	vehicles
0.95	0.00	0.02	0.00	0.02	0.00
0.01	0.99	0.00	0.00	0.00	0.00
0.39	0.01	0.46	0.01	0.13	0.00
0.21	0.00	0.09	0.56	0.13	0.00
0.01	0.00	0.02	0.00	0.97	0.00
0.00	0.00	0.03	0.00	0.17	0.80

Table 1. Confusion matrix of classes

buildings	ground	miscellaneous	pole-like	vegetation	vehicles	mIoU
0.86	0.98	0.38	0.44	0.73	0.79	0.70

Table 2. Intersection over Union (IoU) results on Lille, France

worst results were miscellaneous and pole-like objects. table 3 shows how a traditional machine learning classifiers compare to our method. The overall accuracy using random forest classifer was 0.78, with the KPConv learning algorithm outperforming in every category.

buildings	ground	miscellaneous	pole-like	vegetation	vehicles
0.78	0.04	0.02	0.00	0.15	0.02
0.06	0.89	0.00	0.00	0.03	0.01
0.43	0.06	0.11	0.00	0.34	0.05
0.30	0.01	0.02	0.41	0.25	0.01
0.08	0.02	0.01	0.00	0.87	0.02
0.22	0.16	0.03	0.00	0.31	0.28

Table 3. Confusion matrix using random forest classifier

4.2 Mislabeling: Miscellaneous

As table 1 shows, pole-like objects were mostly classified as vegetation or building objects. Miscellaneous objects were classified as buildings or vegetation. We can see that parts of miscellaneous objects, have a higher probability of being classified as buildings. We can see that in figure 8 that the staircase and fence had high probabilities of being building points when indeed they were miscellaneous. In particular, the staircase has a high probability of being buildings. Also, barrier

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-VIII-4-W2-2021-67-2021 | © Author(s) 2021. CC BY 4.0 License.

objects, such as barrier walls, have a similar geometry to building facades. Both object have similar geometric properties, yet they are labeled as two different object.

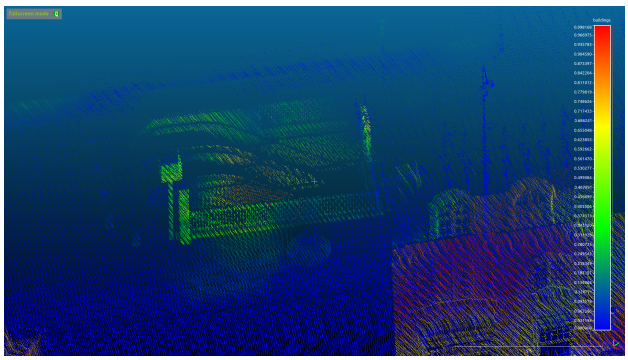

Figure 8. Probability of building points in a mislabeled section

4.3 Mislabeling: Pole-like

We can also see various mislabelling of pole-like objects, particularly in objects that significantly differ from poles in the training data. Light poles existed in both the training and testing set, hence light pole objects were often were correctly segmented in the test set. The variety of different pole-like objects in the test set made their classification more difficult, hence we see the mislabeling.

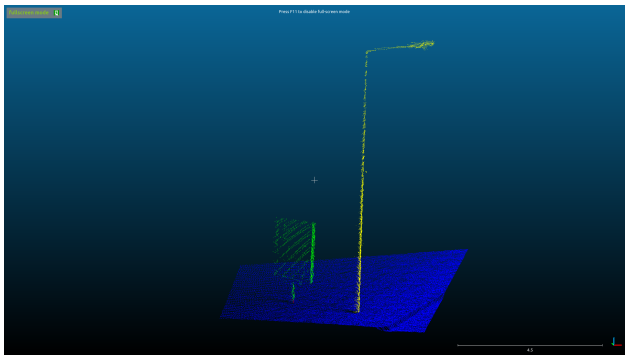

Figure 9. Correctly classified points for light pole and pole object mislabeled

5. DISCUSSION

Although KPConv trained on CARLA data produced satisfactory results for buildings, ground, vegetation, and vehicles, improvement can be made for the pole-like and miscellaneous objects. Note that many of the pole-like objects in the testing environments include street signs of various shape, bollards, and light poles. We can see that light poles and bollards were consistently classified, but street signs were often mislabeled. It is important to note that the training size consisted of approximately 52 million points which is relatively small compared to the test size, which consisted of approximately 30 million points.

The results given in section 4 show that using training data solely generated in a gaming environment can be effective in training deep learning algorithms and can give accurate results on testing real world data. Using these environments helps to solve the bottleneck issue explained in section 1. The customization and versatility allows for more simulation of real world environments, with the added benefit of fully labeled data to use for training. This saves time and resources, and makes this type data more accessible to researchers, as CARLA is open source (Dosovitskiy et al., 2017). This method also outperforms more traditional methods of segmentation as demonstrated in section 4.

5.1 Testing new networks

Future work will need to focus on testing other neural networks and creating more customized datasets. Other networks demonstrate good results on benchmark datasets and constructed more for semantic segmentation. RAndla-Net is a recent network designed specifically for semantic segmentation of large scale point clouds, using a point random sampling method and point feature aggregator (Hu et al., 2020). This allows for quick segmentation of large point cloud data.

5.2 Testing different sensors

Work will also continue in setting real world simulations for collection of training data. As stated in section 2 and section 3, there are other sensors attached to a mobile laser scanner system, namely the GNSS and IMU. For our experiment, we did not use these sensors and only used the transform data from the sensor. This gives the exact rotation and translation from the LiDAR scanner in the world frame. IMU and GNSS data can be used instead and we can simulate real world noise for both sensors, then the point cloud can be generated using equation 4.

5.3 Creating new and different models

Figure 10. Town 2, another urban environment in CARLA

To help the network better generalize a deep network, we need to add new data to the algorithm. While *Town 3* was good in how it represented an urban setting, the pole-like and miscellaneous objects saw higher rates of mislabelling. Adding more variance in the training data for these objects should allow the algorithm to generalize better, allowing for better test results.

6. CONCLUSION

Using simulation spaces for 3D deep learning shows promise due to the versatile nature of gaming engines. They have the ability to generate real world environments with fully labeled data, allowing for the training of convolution neural networks without the need for hand labeled training data. We see in this study that using only data and noise from the simulated LiDAR we get an overall mIoU accuracy of 0.70, showing that this can be done effectively. Further study needs to be done with other

algorithms as well as testing other sensors attached to a typical LiDAR suite, including IMU and GNSS sensors. As shown in equation 4, LiDAR point clouds are derived from using 3 sensors, and each one of these sensors comes with its own error distribution. Hence, further study will include how these sensors affect the overall accuracy of learning algorithms when trained on simulated data.

7. ACKNOWLEDGEMENTS

This research was supported in part by an appointment to the Postgraduate Research Participation Program at the U.S.National Geospatial-Intelligence Agency (NGA), administered by the Oak Ridge Institute for Science and Education through an interagency agreement between the U.S. Department of Energy and NGA.

REFERENCES

Chen, M., Feng, A., Prasad, P. B., McAlinden, R., Soibelman, L., Enloe, M., 2019. Fully Automated Photogrammetric Data Segmentation and Object Information Extraction Approach for Creating Simulation Terrain. *Los Angeles*, 12.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V., 2017. CARLA: An Open Urban Driving Simulator. *arXiv:1711.03938 [cs]*. http://arxiv.org/abs/1711.03938. arXiv: 1711.03938.

Dworak, D., Ciepiela, F., Derbisz, J., Izzat, I., Komorkiewicz, M., Wójcik, M., 2019. Performance of LiDAR object detection deep learning architectures based on artificially generated point cloud data from CARLA simulator. *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*.

Geiger, A., Lenz, P., Urtasun, R., 2012. Are we ready for autonomous driving? The KITTI vision benchmark suite. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 3354–3361. ISSN: 1063-6919.

Glennie, C., 2007. Rigorous 3D error analysis of kinematic scanning LIDAR systems. *Journal of Applied Geodesy*, 1(3). https://www.degruyter.com/document/doi/10.1515/jag.2007.017/html.

Griffiths, D., Boehm, J., 2019. SynthCity: A large scale synthetic point cloud. *arXiv:1907.04758 [cs]*. http://arxiv.org/abs/1907.04758. arXiv: 1907.04758.

Gschwandtner, M., Kwitt, R., Uhl, A., Pree, W., 2011. BlenSor: Blender Sensor Simulation Toolbox. G. Bebis, R. Boyle, B. Parvin, D. Koracin, S. Wang, K. Kyungnam, B. Benes, K. Moreland, C. Borst, S. DiVerdi, C. Yi-Jen, J. Ming (eds), *Advances in Visual Computing*, 6939, Springer Berlin Heidelberg, Berlin, Heidelberg, 199–208. Series Title: Lecture Notes in Computer Science.

Guo, Y., Wang, H., Hu, Q., Liu, H., Liu, L., Bennamoun, M., 2020. Deep Learning for 3D Point Clouds: A Survey. *arXiv:1912.12033 [cs, eess]*. http://arxiv.org/abs/1912.12033. arXiv: 1912.12033.

Hackel, T., Savinov, N., Ladicky, L., Wegner, J. D., Schindler, K., Pollefeys, M., 2017. SEMANTIC3D.NET: A NEW LARGE-SCALE POINT CLOUD CLASSIFICATION BENCHMARK. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-1/W1, 91–98. http://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-1-W1/91/2017/.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Hu, Q., Yang, B., Xie, L., Rosa, S., Guo, Y., Wang, Z., Trigoni, N., Markham, A., 2020. RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Munoz, D., Bagnell, J. A., Vandapel, N., Hebert, M., 2009. Contextual Classification with Functional Max-Margin Markov Networks. 975–982.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, R. Garnett (eds), *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 8024–8035.

PDAL Contributors, 2018. PDAL Point Data Abstraction Library.

Qi, C. R., Yi, L., Su, H., Guibas, L. J., 2017. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *arXiv:1706.02413 [cs]*. http://arxiv.org/abs/1706.02413. arXiv: 1706.02413.

Roynard, X., Deschaud, J.-E., Goulette, F., 2018. Paris-Lille-3D: a large and high-quality ground truth urban point cloud dataset for automatic segmentation and classification. *arXiv:1712.00032 [cs, stat]*. http://arxiv.org/abs/1712.00032. arXiv: 1712.00032.

Serna, A., Marcotegui, B., Goulette, F., Deschaud, J.-E., 2014. Paris-rue-Madame Database - A 3D Mobile Laser Scanner Dataset for Benchmarking Urban Detection, Segmentation and Classification Methods. M. D. Marsico, A. Tabbone, A. L. N. Fred (eds), *ICPRAM*, SciTePress, 819–824.

Shah, S., Dey, D., Lovett, C., Kapoor, A., 2017. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. *arXiv:1705.05065 [cs]*. http://arxiv.org/abs/1705.05065. arXiv: 1705.05065.

Thomas, H., 2019. Learning new representations for 3D point cloud semantic segmentation. PhD thesis, Université Paris sciences et lettres.

Thomas, H., Qi, C. R., Deschaud, J.-E., Marcotegui, B., Goulette, F., Guibas, L. J., 2019. KPConv: Flexible and Deformable Convolution for Point Clouds. *Proceedings of the IEEE International Conference on Computer Vision*.

Vallet, B., Brédif, M., Serna, A., Marcotegui, B., Paparoditis, N., 2015. TerraMobilita/iQmulus urban point cloud analysis benchmark. *Computers & Graphics*, 49, 126–133. https://linkinghub.elsevier.com/retrieve/pii/S009784931500028X.

Weinmann, M., Jutzi, B., Mallet, C., 2013. Feature relevance assessment for the semantic interpretation of 3D point cloud data. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-5/W2, 313–318. https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/II-5-W2/313/2013/.

Wu, B., Wan, A., Yue, X., Keutzer, K., 2017. Squeeze-Seg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud. *arXiv:1710.07368 [cs]*. http://arxiv.org/abs/1710.07368. arXiv: 1710.07368.