Unstructured Point Cloud Normal Estimation Using Deep Learning

Chrysostomos Chadjiminas

May 16, 2020

1 Problem Statement

Many point cloud processing applications have a better performance when the normals for the point cloud are correctly estimated. Normal estimation is the process of predicting the normal for each point in the cloud. This process is easy on flat surfaces but can be harder on noisy data and surfaces with sharp features and point density variations. Data captured with LiDAR, photogrammetry or other point cloud capturing techniques are usually noisy, present a lot of sharp features and have sample density variations.

Boulch and Marlet present a new method for normal estimation that tries to address all these problems on surfaces with sharp edges in [BM16]. This paper is summarised in the next section.

2 Paper Summary

The method includes a randomised Hough Transform that is created for each point and a Convolutional Neural Network that learns the function that maps the Hough Accumulator to its normal. Generally, the process can break down to several steps. First there is the Hough Transformation that transforms the data to Hough Space and a voting scheme to create a hough accumulator, then there's the training data generation process to create the training samples for the network to learn, and finally, there's the Convolutional Neural Network architecture along with the training algorithm.

Hough Transform

The **Hough transform** was traditionally used for shape extraction by creating a (multidimensional) histogram where each bin represents a shape hypothesis. The histogram is also called a **Hough accumulator** because it accumulates the votes in each bin. The transformation can be used for any feature descriptors other than shape. This transformation is achieved by a voting scheme and the feature extraction is achieved by getting the feature descriptor from the most voted bin in the hough accumulator. Note that a feature descriptor or a range of them can be mapped to each bin and there can be a process to recover it from the selected bin.

In this paper, the Hough transform is used to predict the normal for each point. A previous technique is mentioned from an earlier work done in [BM12b] where a robust

randomised Hough Transform is proposed to estimate the normals. To predict then normal using Hough Transform there are three main steps, first the **hypothesis generation** where triplets are randomly selected for possible normal directions, then **voting in the Hough Space** where each possible normal votes in the 2D accumulator and, finally, the **election of a normal** from the accumulator. In [BM12b] the voting happens in 2D spherical accumulator parametrised by two spherical coordinates θ and ϕ. The normal election happens by selecting the max bin and averaging the normals that have voted for the specific bin.

In this paper, the hypothesis generation step is the same but the voting is simpler and the normal election happens with a CNN. In summary a 2D Hough accumulator is created for each point. The normal hypotheses for the accumulator are created by randomly selecting triplets of points from the neighbourhood of the point which define a plane and therefor a possible normal for it. The voting scheme is defined by the following equation:

$$(x, y) = (\frac{n_x + 1}{2} * m, \frac{n_y + 1}{2} * m)$$

where x, y are the vote's coordinates in the accumulator, (n_x, n_y, n_z) is the normal of the random triplet, and m is the number of rows and columns of the hough accumulator. The result can be retrieved from the max bin of the accumulator by solving for n_x, n_y

$$(n_x, n_y) = (\frac{2 * x}{m} - 1, \frac{2 * y}{m} - 1)$$

Because the normal has magnitude 1, the n_z component can be retrieved as:

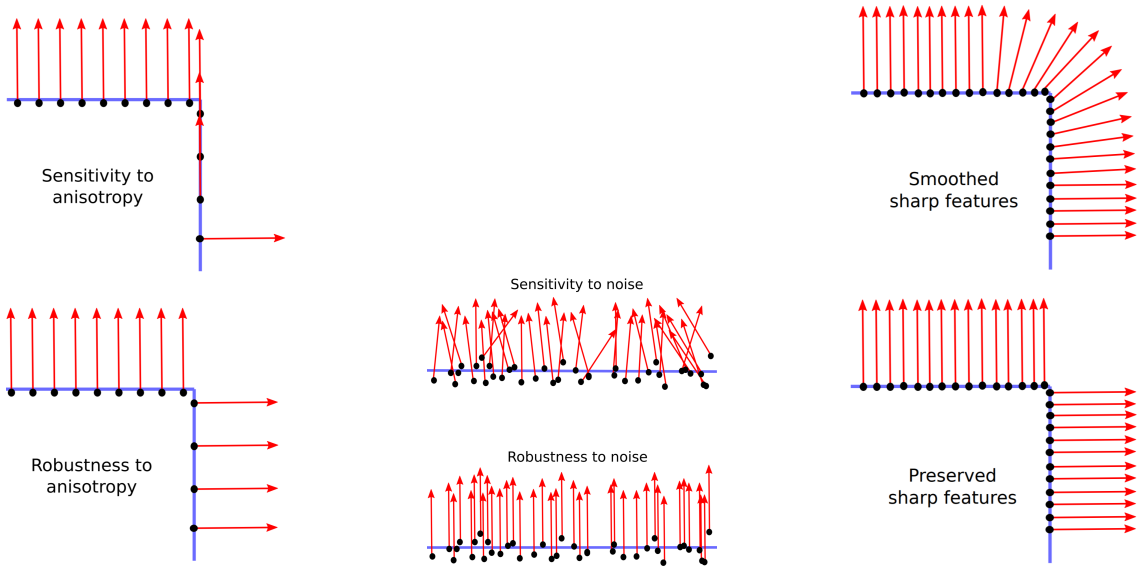
$$n_z = \sqrt{1 - (n_x^2 + n_y^2)}$$

Unfortunately, this method is not robust to noise, to outliers and can not perform well with sharp edges1 as it tends to smoothen out normals in sharp features, makes wrong decisions in noisy sections and is biased towards densely packed neighbourhoods.

This method is not robust because the votes choose a vector that is the average of the normals from the triplets. Therefor, it is suggested that a CNN should learn the function that maps the points accumulator to it's normal. This method consists of several steps. First the accumulator should be created as it was already described. In addition, an accumulator normalisation method is described to decrease the pattern variations to help learning. It includes two principal component analysis transformations, a 3D transformation and one that maps the normal to 2 dimensions. The result is an accumulator that is not guaranteed to have a consisted normalisation but not something the non linearity and non locality of the CNN can not handle. The next step includes training data generation.

Training Data Generation

Corners of 5000 points are generated with random angles and from the corners 1000 points are sampled. The accumulators for those points are created. The sample points are near 3 side corners and edges to help the network make the right decisions for similar points in the test data. The ability of the network to generalise will also make the network to decide for other cases as well. To provide robustness to noise, noise is also added to the training dataset.



(a) Density Anisotropy (b) Noise (c) Sharp features

Figure 1: Images from slides in [BM12a]. Challenges for normal estimation.1a When some areas are more densely sampled than other areas, the votes are unequally distributed leading to wrong normal estimation. This is prevalent with lidar data as well. 1bWhen data is noisy a sensitive to noise estimator can fail even for flat surfaces. 1cIt's usual for estimators to smoothen sharp features by assigning an average normal to normals in the edges.

Convolutional Neural Network (CNN) for normal estimation

The network architecture that is used is based on LeNet5[LeC+89]. It is comprised from several convolution layers, ReLU activations and a Dense Network at the end for classification. The network can be seen in Fig 2

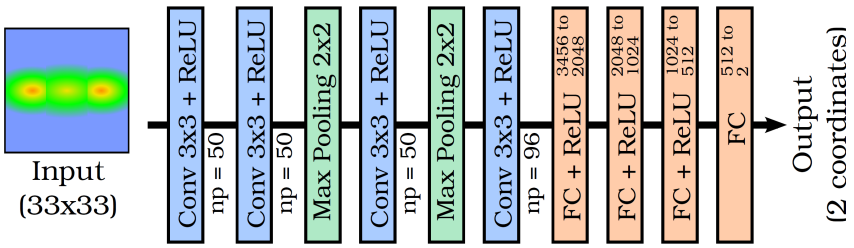


Figure 2: Figure from [BM16]. Convolutional neural network architecture. The predicted normal is after the PCA transformations and the original normal can be recovered by applying the inverse of the transformations.

The network is trained using mean square metric (**MSE**):

$$\{\hat{\mathbf{n}}_p\} = \underset{\{\hat{\mathbf{n}}_p\}}{\operatorname{argmin}} \sum_{p \in P} (\mathbf{n}_p - \mathbf{n}_p^*)^2 \tag{1}$$

For the training process the normal n is known for each point p. As previously mentioned, a Hough accumulator for each point since for each point a neighborhood of fixed size is selected. The result is a grayscale image with the most dominant votes being close to

white. For the training data there can be mostly 2 or 3 dominant votes for edges and corners respectively. When there's one dominant vote that means that the training sample is a plane.

A method to combat density variation is presented along with a method to deal with the scale that the scene is observed. The multi scale approach of accumulator will be discussed more thoroughly in the implementation section.

3 Paper Implementation

To implement this paper, we chose to use Python 3. We used common machine learning and data science packages. Here is a full list:

- numpy (linear algebra)
- scipy (geometric rotations)
- sklearn (KDTree)
- panda (tracking model data)
- pytorch (deep learning)
- plotly (3D visualisations)
- matplotlib (2d visualisations)
- trimesh (loading obj files)
- cython (optimising the performance of critical paths)

3.1 Hough Accumulator

The hough accumulator is simply a 2 dimensional square histogram. Each bin in the histogram represents the number of votes for that specific normal. To generate the accumulator for a point $\mathbf{p}$ a fixed size neighbourhood $\mathbf{K}$ around the points is given. The neighbourhood is created using a K-D tree that is build for the point cloud we want to estimate the normals for. For the K-D tree implementation, Scikit-learn's[Ped+11] KDTree was used because it's fast and gives accurate results.

Voting

To select the normals an amount $\mathbf{T}$ of uniformly random triplets of points are sampled. This amount $\mathbf{T}$ was fixed to be 1000 but is configurable. Each triplet defines a plane and therefor a potential normal for $\mathbf{p}$. To find the normal the cross product of two vectors is taken, this can produce two results which are along the same line but different sign. We chose to assume that the normals in the data are oriented in the general direction of the positive z axis. This is the case with aerial LiDAR data. The expected orientation can be configured per point. The normal (n_x, n_y, n_z) at this state could be used to vote in an accumulator $\mathbf{M_{m \times m}}$ with the following2:

$$(x, y) = (\frac{n_x + 1}{2} * m, \frac{n_y + 1}{2} * m) \tag{2}$$

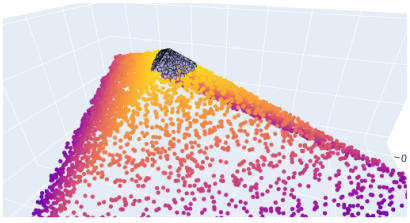
The normal can be simply retrieved back with 3 and 4:

$$(n_x, n_y) = (\frac{2 * x}{m} - 1, \frac{2 * y}{m} - 1) \tag{3}$$

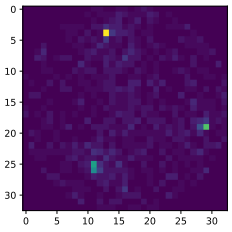
$$n_z = \pm\sqrt{1 - (n_x^2 + n_y^2)} \tag{4}$$

. Notice that n_z can have two possible values but we can orient the normal with the assumption we made about the orientation of the normals.

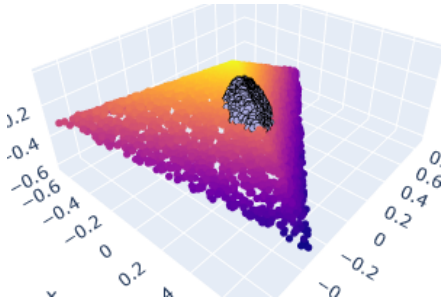
To decrease pattern variation and to ease learning two transformations are used. Firstly, as is suggested in the paper, a 3D PCA is performed on the mean-centred neighbourhood and the z axis is oriented on the axis with the least variation. After the data are projected to the accumulator plane, a second 2D PCA is used on the neighbourhood points in the accumulator plane to align the most variance with the x axis. The accumulator is then normalised between 0 and 1. Results for a corner and an edge can be seen in Fig 3



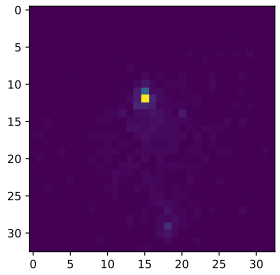
(a) Neighbourhood corner



(b) Accumulator of corner 3a



(c) Neighbourhood edge



(d) Accumulator of edge 3c

Figure 3: Neighbourhoods and their corresponding accumulators. Bright spots represent highly voted bins. 3a 3b A neighbourhood at the tip of a pyramid where there are three dominant normals, this is visible in the accumulator. 3c 3d A neighbourhood of an edge and the corresponding accumulator. There are two dominant normals but one one is farm more dominant than the other because of number of points in one side are more than the other.

Multiscale Accumulators

As was suggested in the paper, multiscale accumulators were implemented. This is an extension to the the method that was described above. The multiscale accumulators are similar to regular accumulators but they have multiple channels instead of one. Each channel can be thought of as a single channel accumulator created for the same point

but for different neighbourhood sizes. Three and five scale multiscale accumulators are implemented. For the three scale the channels are for $[\mathbf{K}/2, \mathbf{K}, 2 * \mathbf{K}]$ neighbourhood sizes and for the five scale accumulator the channels are $[\mathbf{K}/4, \mathbf{K}/2, \mathbf{K}, 2 * \mathbf{K}, 4 * \mathbf{K}]$ neighbourhood sizes. The PCA transforms are only created for the neighbourhood with size $\mathbf{K}$ and are applied for all the rest of the channels. A 5 scale multiscale accumulator can be seen in Fig 4

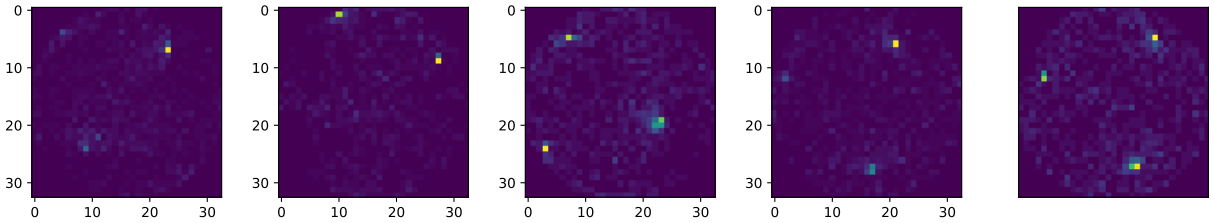


Figure 4: Channels 1 through 5 for a 5 scale accumulator created for the same point as in the corner in 3. K=500, the corresponding neighbourhood sizes are [K/4, K/2, K, 2K, 4K]. It can be observed that different channels have different votes.

3.2 Synthetic Data Generation

We deploy a similar strategy to what is described in the paper and generate synthetic corner examples of various angles and rotations.

Our implementations generates 3 axes (unit vectors) at a specified angle. Since our Hough estimator assumes distinct orientation, we make sure all synthetic data that is generated has normals oriented towards positive z. We can then create 3 triangles by combining the zero vector and all combinations of pairs form the axes. These triangle will be the 3 sides of our corner. Next, we fill each side with points distributed randomly using barycentric coordinates. Once we have the point cloud we can apply gaussian noise to each point. Lastly, all corners are rotated randomly5 on all axes to provide further variation and robustness to the CNN.

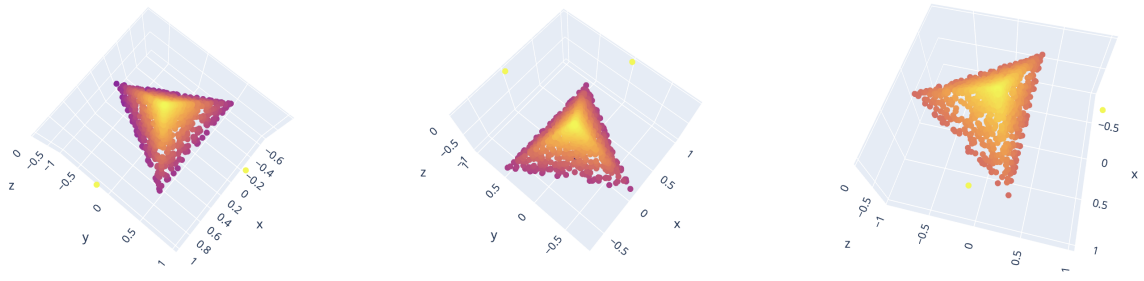


Figure 5: Left: 80deg; Middle: 90deg; Right: 120deg

3.3 CNN Architecture

Overview

We implement our deep convolutional neural network using PyTorch[Pas+19]. As proposed in the paper, we use a stack of 4 convolutional layers with a Max Pool after the

2nd and the last. This is followed by a stack of 4 fully connected layers. Every layer uses a ReLU activation function. We train the network with a root mean square criterion. In the convolution layers additional Batch Norm units are included for speed and stability improvements while the fully connected layers feature Dropout functions before each activation function to reduce overfitting and improve generalisation.

$$\{\hat{\mathbf{n}}_p\} = \underset{\{\mathbf{n}_p\}}{\operatorname{argmin}} \sum_{p \in \mathcal{P}} (\mathbf{n}_p - \mathbf{n}_p^*)^2$$

We found the Adam optimiser can sometimes give better results for this architecture but in some experiments we trained with Stochastic Gradient Descent (SGD). To facilitate faster learning we employ PyTorch's learning rate scheduler. The learning rate is reduced automatically while training when the if no improvement to the loss occurs in a given number of epochs (referred to as the "patience" of the scheduler). To find the optimal loss we use the Early stopping technique by tracking and saving our loss and counting the number of epochs in which it hasn't improved. Once a threshold is passed we stop learning and save the model with the best loss.

Using the model

The previous section goes into detail how we create accumulator using Hough transforms. The resulting gray images are fed to the network as the training data, while the x and y components of the normal vector is passed as the label in the supervised learning process. Since the network is trained on projected normals after applying the two PCA rotations, we need to do the same for the label. For each training example we derive a PCA on the 3D coordinate system and 2D PCA on the projected space. We apply those rotations to the ground-truth normal and save the x and y pair as the label.

This whole process is repeated when using the trained CNN model to estimate a normal. We need to derive and apply two PCAs when passing a Hough-projected point cloud neighbourhood to the model. We need to store these rotation matrices as their inverses will be used to unrotate the resulting coordinate that was generated. Once that is done, we have a pair of coordinate back in the original 3D coordinate system. We only have the x, y of the CNN but since we can assume this is a normalised vector we can easily derive the z.

$$z = \pm\sqrt{1 - (x^2 + y^2)}$$

The only remaining step is to reorient the normal towards an assumed direction.

3.4 Extentions

In this section we will highlight the aspects of our implementation that were not proposed in the original paper but we introduced as additional extensions.

3.5 Synthetic corner with edge bias

A common problem during our training process was that the CNN would not learn to predict correct normals on some points near the edges. One strategy we devised was to bias the synthetic data and generate less points on the flat areas. This way we could

generate more edge examples when randomly picking neighbourhoods. Examples are shown in Fig 6

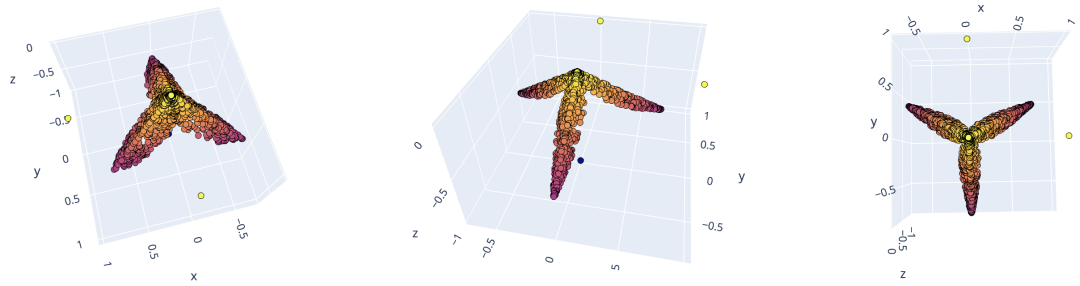


Figure 6: Bias for edges in synthetic data

3.6 Corner balancer

An alternative approach to biasing the synthetic data is to control the process of picking random neighbourhoods. When a random point in the point cloud is chosen at random and a neighbhourhood generated, we can analyse the ground-truth normals of all points in the neighbourhood. We can then group the normals by their general direction which we assume is one per corner side (with certain sensitivity measures accounted for if noise is applied). We can then take the number of unique directions and decide if the given neighbourhood is a flat plane, a two-sided edge, or three-sided corner. By counting how much any of these cases were generated, we can extend the data generation process until a desired ratio is achieved. Although this will penalise the performance of the data generation process, we can have more fine-grained control over the type of examples the CNN will train on.

3.7 Training extensions

As covered in the CNN section, we introduced the inbuilt PyTorch learning rate scheduler into the training process

4 Evaluation

To evaluate the performance of the network it was decided to compare it against the max bin method.

The max binning method selects the normals with the biggest vote from the accumulator using the equations in 3 and 4. The predicted normal is then transformed from the Hough space back to the points cloud original coordinate system by applying the inverse of the two PCA transforms.

The evaluation metric is Root Mean Square which, for a point cloud P with $|P|$ points p, with predicted normals $\{\hat{\mathbf{n}}_p\}$ and actual normals $\{\mathbf{n}^*_p\}$ is defined by:

$$RMS = \sqrt{\frac{1}{|P|}\sum_{p\in P}\widehat{(\mathbf{n}^*_p\hat{\mathbf{n}}_p)}} \tag{5}$$

The point cardinality P was replaced with the number of good points $|GP|$. Good points are the points where the angle between the predicted normal and the ground truth is less than a threshold. In code this threshold is called a *pgp_threshold* and is configurable. The threshold f is set to be 8 degrees for the experiments. With that in mind 5 becomes:

$$RMS = \sqrt{\frac{1}{|GP|} \sum_{p \in P} \widehat{(\mathbf{n}_p^* \hat{\mathbf{n}}_p)}} \tag{6}$$

Experiments were executed on a bumpy terrain point cloud which has a lot of sharp features and on a regular pyramid point cloud. The RMS error was measured for fixed point clouds but with increasing amount of noise applied to them. The good point ratio is also measured which is defined as:

$$\text{GPR} = \frac{|GP|}{|P|} \tag{7}$$

The neighbourhood size **K** was set to 100. The scales for the accumulator, for training and classifying with the CNN are 3. The CNN was trained with 25000 training samples and 1000 validation samples. The plots for validation and training loss can be see in Fig 7

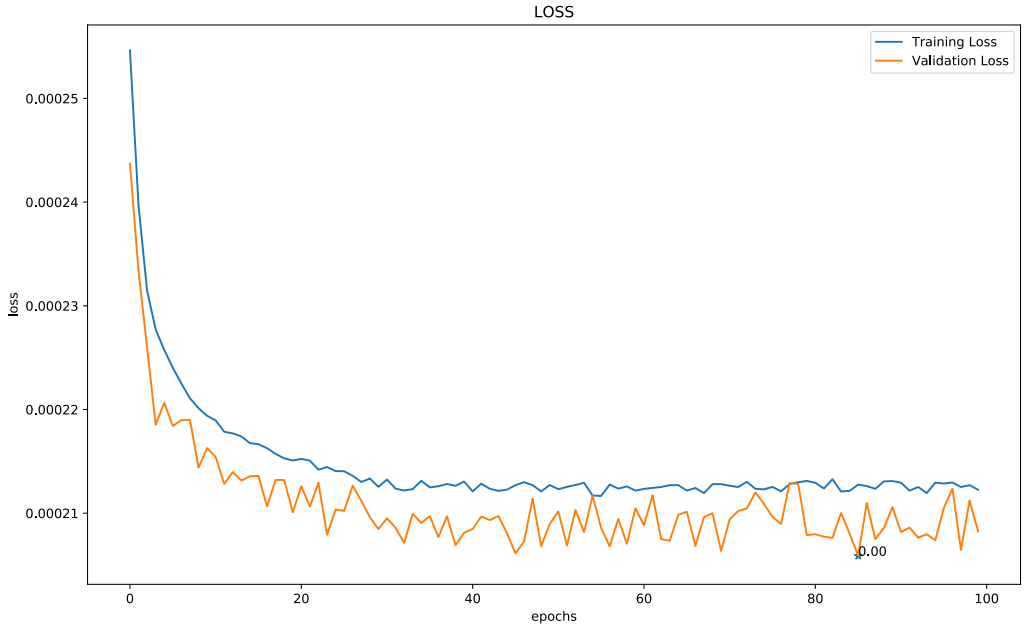


Figure 7: Training and validation loss being reduced in the training phase.

As can be observed in 8, for the point cloud example where it doesn't present many sharp features, max binning performs better than the CNN initially but quickly gets outperformed by the CNN as noise increases. The max binning performs good on surfaces without many sharp features and for this reason it has a lot more good points than the CNN. An additional point can be observed, max binning performance drops rapidly as noise increases while CNN's performance decreases slowly and is mostly stable, showing indications that it's robust to noise.

Performing the same experiment on a new example as is the case with a bumpy terrain with many sharp features, shown in Fig. 9, we can see that the CNN performs better from the start, both for rms and good point ration. This shows that the CNN is more robust to surfaces with sharp features while the max binning can not handle these cases too well. Once again, it can be observed that the CNN is more stable than the max binning method.

In Fig. 10 we can see, once again, that CNN is more stable with noisy data. The max bin method outperforms CNN when no noise is present. We can still observe that in the absolute edge the results have a big angle deviation from the actual ground truth than the corresponding points in the CNN.

5 Contribution

I would like to thank my teammate Nikolai. More than once he guided me towards the right direction with his helpful feedback. I would also like to acknowledge his hard work and his high quality code that I could easily integrate with mine.

I will use the following keywords:

1. **idea**: understood and/or came up with how to implement a certain idea from the paper or otherwise
2. **implementation**: implemented the idea to a functional and tested state
3. **testing**: worked further on testing, experimenting or visualising of a certain feature to verify correctness
4. **extensions**: worked on improving or extending some proposed method in the paper

Below are contributions broken by the main pieces of functionality.

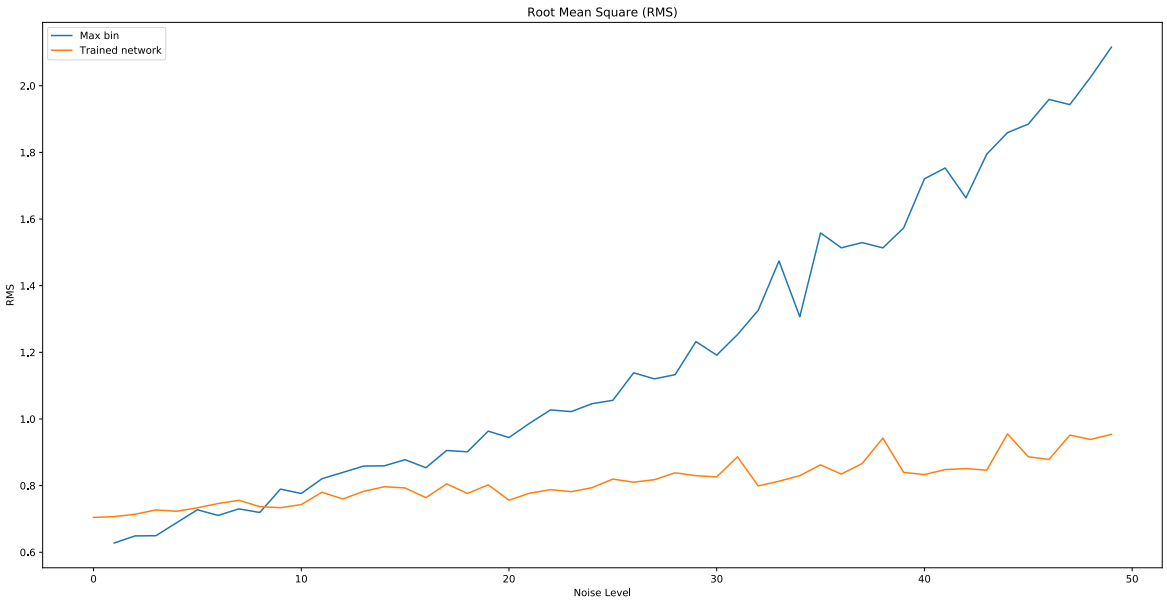
1. **Synthetic Data**
   - Nikolay (idea, implementation, testing, extentions)
   - Chrysostomos (testing)

2. **Hough Acummulator**
   - Chrysostomos (idea, implementation, testing)
   - Nikolay (idea, implementation, testing)

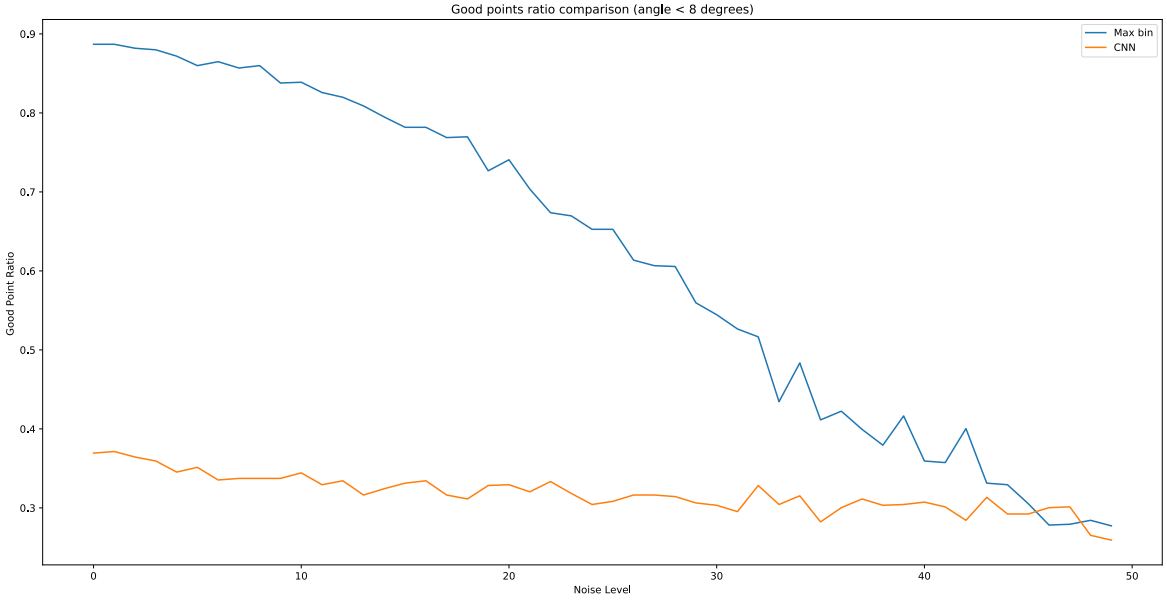
3. **CNN Architecture**
   - Chrysostomos (idea, implementation, testing)
   - Nikolay (implementation, testing)

4. **CNN Training**
   - Chrysostomos (idea, implementation, testing, extentions)
   - Nikolay (testing)

5. **Multiscale Extention**
   - Chrysostomos (idea, implementation, testing)
   - Nikolay (testing)

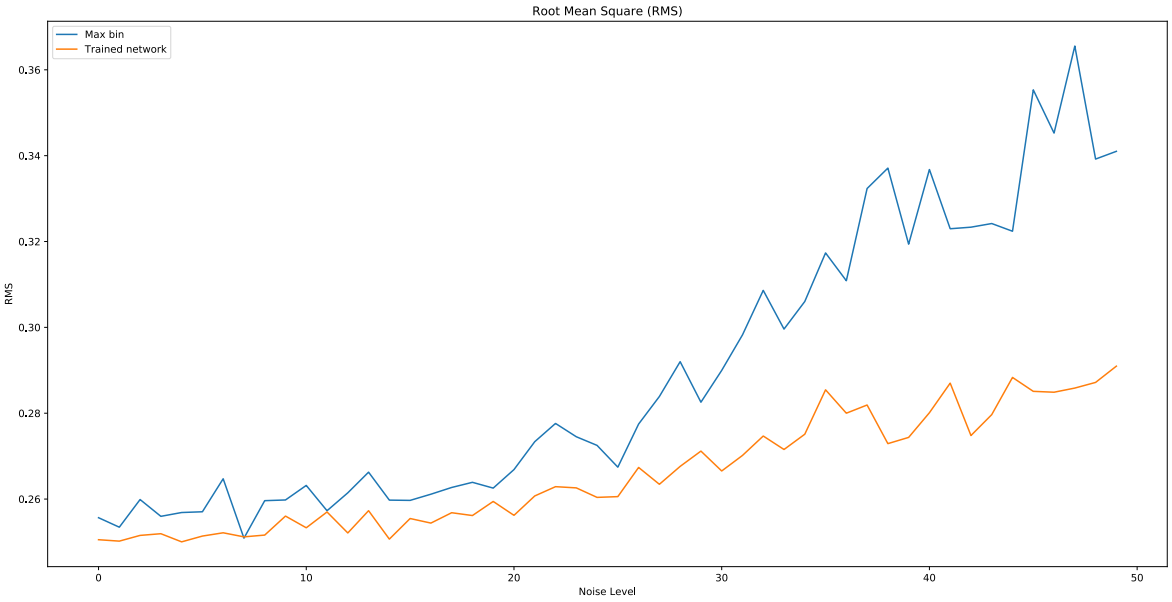


(a) Root Mean Square error as noise increases.

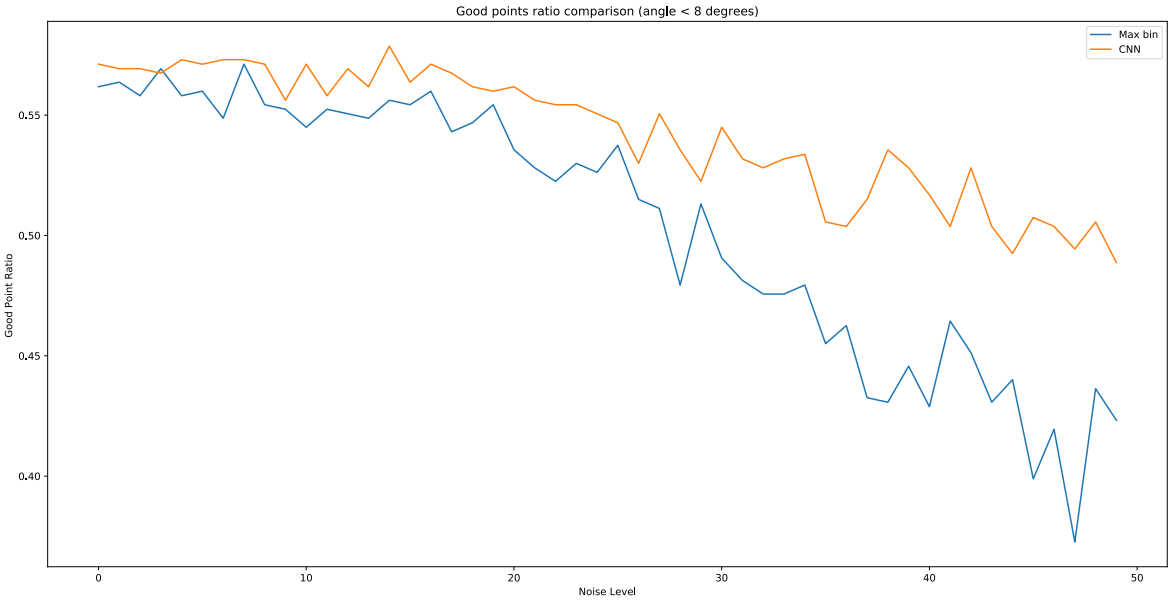


(b) Good point ratio as noise increases.

Figure 8: Performance comparison for max binning and CNN with 3 scales on a **pyramid point cloud** with increasing amount of noise.8a Initially, max binning performs better than the CNN but as the noise increases we can see that the CNN greatly outperforms it's competitor.8b The ratio of good points is bigger for max binning than for CNN until the noise is very big. Max binning is good for surfaces without many sharp features just like the pyramid. The ratio greatly decreases for max binning, as noise increases while for the CNN its mostly stable.



(a) Root Mean Square error as noise increases.



(b) Good point ratio as noise increases.

Figure 9: Performance comparison for max binning and CNN with 3 scales on a **bumpy terrain point cloud** with increasing amount of noise.9a CNN outperforms max binning from the start and is stable as noise increases.9b In contrast to 8b CNN outperforms max binning, this is because the terrain point cloud has many sharp features where the CNN is better than max binning.

6 Conclusion

The results were rewarding as the CNN seemed to be performing better than the max bin solution. In addition, the CNN seems to be robust to noise and could perform well when encountering surfaces with sharp features. We do not imply that our implementation was flawless as their could be potential oversights. In addition, there are many configurable parameters that could increase the performance such as training hyperparameters, CNN architecture, size of neighbourhood to name a few.

Overall, the project was a constructive experience as we had to face many challenges and come up with our own solutions to tackle them.

References

[BM12a] Alexandre Boulch and Renaud Marlet. "Fast and Robust Normal Estimation for Point Clouds with Sharp Features". In: *Computer Graphics Forum* 31.5 (Aug. 2012), pp. 1765–1774. ISSN: 01677055. DOI: 10.1111/j.1467-8659.2012.03181.x. URL: http://doi.wiley.com/10.1111/j.1467-8659.2012.03181.x (visited on 05/16/2020).

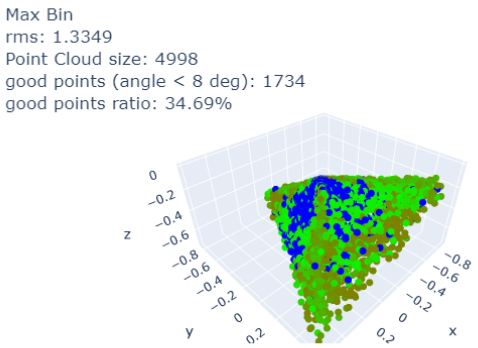
[BM12b] Alexandre Boulch and Renaud Marlet. "Fast and robust normal estimation for point clouds with sharp features". In: *Computer graphics forum*. Vol. 31. 5. Wiley Online Library. 2012, pp. 1765–1774.

[BM16] Alexandre Boulch and Renaud Marlet. "Deep learning for robust normal estimation in unstructured point clouds". In: *Computer Graphics Forum*. Vol. 35. 5. Wiley Online Library. 2016, pp. 281–290.

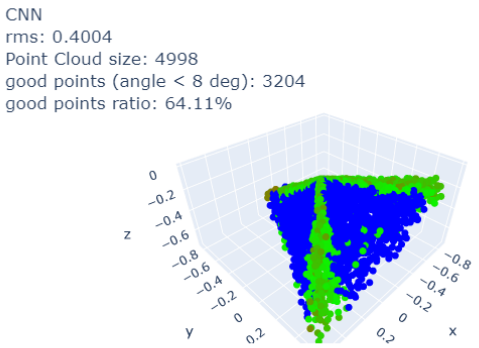
[LeC+89] Yann LeCun et al. "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4 (1989), pp. 541–551.

[Pas+19] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

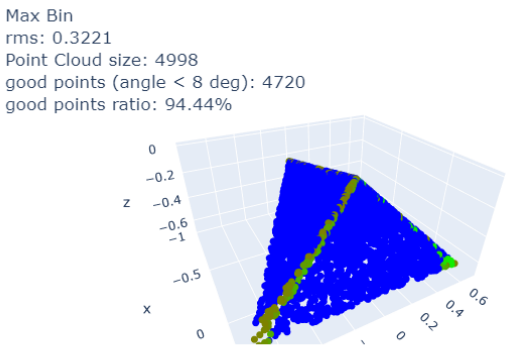
[Ped+11] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.



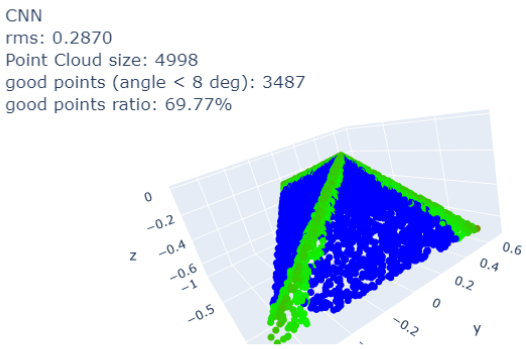
(a) Max bin on noisy data



(b) CNN on noisy data



(c) Max bin on not noisy data



(d) CNN on not noisy data.

Figure 10: Comparison of CNN and max binning for noisy and not noisy data. Blue points means that they are good points (angle from predicted normal is < 8 degrees). Bright green points are more correct than brownish and dark greens. 10a 10b On noisy data the CNN performs much better having both a lower rms and higher good point ratio. 10c 10b On not noisy data with flat surfaces max binning has a good performance except from the edges where we can see big angle deviations from the ground truth. The CNN is outperformed both in rms and good point ratio but the points at the edges do not deviate from the ground truth as much as the corresponding points in max binning. The performance of the CNN doesn't drop drastically, like max binning, when noise is applied showing robustness to noise.