Traversability Estimation with Dynamic Graph CNN

Francesco Saverio Zuppichini[1]
[1] *Universita della Svizzera Italiana (USI)*
zuppif@usi.ch

Recent deep learning breakthrough allows to effective classify and segment point clouds by converting them into graphs and feed them to *Geometric Deep Learning* models. Point clouds are able to represent any shape in space making them one of the most versatile data-structure for non-euclidean data. The architecture that archives *state of the art* results in booth classify and segment those cloud is Dynamic Graph CNN [1]. It utilizes a special Convolution operator applied directly on the graph's edge called *EdgeConv* using booth local and global information. The aim of this project is to first reproduce the results on the famous ModelNet40 [2] dataset with more than ten thousand meshed. Reproducing the paper's results ensure our architecture's correctness. Then, we test DGCNN against a vanilla CNN on a dataset composed by heightmaps, images where each pixel is the height value of a terrain region, labeled as traversable or not traversable. Those images were generated letting a legged crocodile-like robot, *Krock*, walking into a simulated environment on synthetic maps and cropping the corresponding terrain patch around each stored trajectory pose. This dataset represents a really interesting playground to explore which architecture is able to extract the most information from the geometry of the terrain and correctly predict their traversability.

I. INTRODUCTION

Graphs can express a wide array of data structures using a set of nodes and the relationship between them, edges. Graphs are non-euclidean data, thus standard deep learning methods, like CNN [3], fails. Recently, a new set of machine learning models, *Graph Deep Learning* have flourished and successfully applied on a wide array of graphs structure such as social networks **To do (??)**, proteins **To do (??)**, and on point clouds. A Point cloud is a set of data points in space that often represents a specific 3D object by defined its points coordinates. They are extremely flexible and can be easily gather using consumer hardware such as a Kinect or a Lidar. Even if they are not graphs by definition, we can easily convert them connecting to each point k neighbors.

As with images, we are interested in classify of segment those point clouds. The current architecture that achieves the *state of the art* in those tasks is *Dynamic Graph CNN* proposed by Wang et all [1]. It utilizes a special convolution layer, *EdgeConv*, to classify or segment point clouds.

This project will focus on explaining, test and evaluate this architecture. First, we will reproduce the results of the original paper on the classification task. Then to test its effectiveness on predicting traversability estimation using ground's patches for a legged robot by comparing it to a classic CNN approach. The discussion will only scrape the surface of the topic and will not treat in detail other architectures. If interested, we suggest to the reader the review by Zhou et all [4].

A. Model

In this section we summarized the original paper's architecture, starting by the backbone of DGCNN, the *EdgeConv*.

B. Dynamic Graph CNN

Dynamic Graph CNN is composed of multiple layers of *EdgeConv* stacked one after the other similar to how convolutions are integrated into CNN. Figure 1 shows the two architecture proposed by the Authors, for classification and segmentation.

1. EdgeConv

Given a point cloud with n denoded as $X = \{x_1, \dots, x_n\} \subseteq R^F$ and a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ representing the local structure of the cloud, the i-th output of *EdgeConv* operation is defined as

$$x'_i = \square_{j:(i,j)\in\mathcal{E}} h_\Theta\left(x_i, x_j\right) \tag{1.1}$$

Where $\square$ is a channel wise aggregation function and h_Θ is the edge function. The authors choose $\square = max$ and $h_\Theta = h_\Theta(x_i, x_j - x_i)$. Where x_i and x_j are the coordinates of the center of the patch and its neighbors respectively. Intuitivelly, this allows the layer to use booth global and local information about the mesh. In addition, the edge function has a learnable parameter Θ that is a classic feed-forward neural network.

2. Dynamic EdgeConv

The authors shows that dynamically recompute the graph using k-nearest neighbors algorithm improves performance. Thus, the *DynamicEdgeConv* i-th output at layer l is defined as:

$$x_i^{(l+1)} = \square_{j:(i,j)\in\mathcal{E}^{(l)}} h_\Theta^{(l)}\left(x_i^{(l)}, x_j^{(l)}\right) \tag{1.2}$$

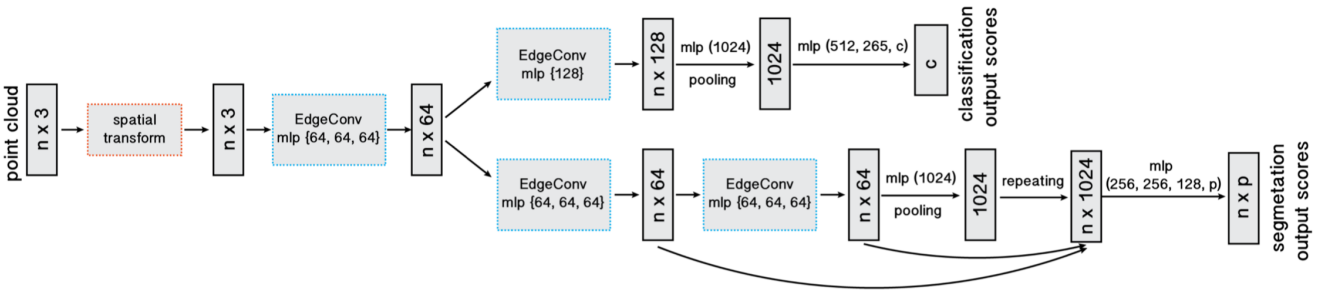


FIG. 1: The two Dynamic Graph CNN architectures divided into an upper and lower branch for classification and segmentation respectively, image from the original paper [1]. In each edge conv, mlp$\{n_0, ..., n_i\}$ represents the number of output neurons of each dense layer. For example, in the first edge conv there are three dense layers with outputs size of 64.

where $\square$ and h_Θ are the same used in the vanilla EdgeConv layer.

C. Similarties to other architectures

The authors shown that DGCNN is related to booth two categories of approaches: PointNet and graph CNN. First, PointNet is just a special case of DGCNN where $k = 1$ resulting in an empty graphs where the edge convolution functions utilises only the global information of each patch, $h_\Theta = h_\Theta(x_i)$ PointNet++ [5] tries to compensate the lack of local informations by appling PointNet in a local manner, it uses a farthest point sampling algorithm to sample from the graph at each layer reducing its size at each step.

The commmon denominator with graph CNN methods, MoNet [6], ECC [7] and Graph Attention Network [8] and DGCNN is the notion of local patch. Hovewer, one crucial different is that all previous models work on a *static* graph.

II. EXPERIMENTS

In this section we summarized the setup and the results of DGCNN on two dataset, one for 3d meshed classification and one composed by images for traversability estimation for a legged robot. We compare DGCNN with a classic CNN on the latter. Code can be found here.

A. Setup

We train the DGCNN classifier, up branch in figure 1, on Ubuntu workstation with an nvidia TITAN-X GPU kindly borrowed by the course stuff. We used PyTorch and PyTorch Geometric to implement the tested networks.

B. Mesh Classification

Exactly as in the original paper, we train the model on the *ModelNet40* dataset, a collection of more than ten thousand CAD meshed. The samples are divided into $9,843$ for training $2,468$ for testing with 40 categories. Figure 2 shows some of the meshed for the category *chair*. We keep the same batch size used in the original paper, randomly sampled 1024 points from per batch and data augment the points by randomly rotate and scale them. We tested the DGCNN classification architecture, upper branch figure 1, with DynamicEdgeConv and $k = 20$. We minimized the Cross-Entropy loss using Adam [9]. We run a total of 50 epochs with a starting learning rate of 0.001 reducing it each 10 epochs by a factor of 0.2. We monitor the accuracy and store the best performing model. The feed-forward weights are initialized using Xavier normal [10] and the bias to 0. The batchnorm's [11] weights and bias to 1 and 0 respectively.

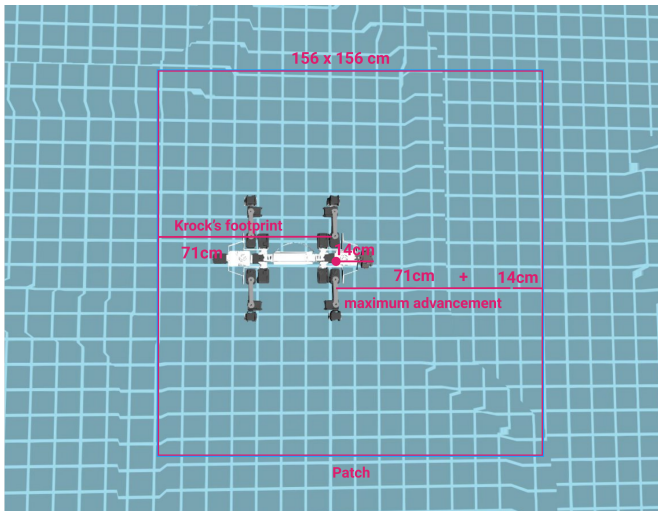
FIG. 2: Some of the *chairs* CAD meshed in ModelNet40 [2].

C. Traversability Estimation

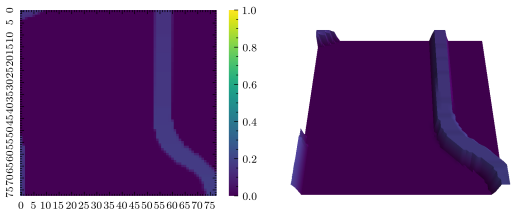
In our Master Thesis work, we trained a convolution neural network to predict the traversability of a

ground region for a legged crocodile-like robot called *Krock*. Our approach follows the pipeline proposed by R. Omar Chavez-Garcia et all [12] where the dataset is entirely generated using synthetic maps in a simulated environment. Figure 4 shows some of the thirty synthetic heightmaps used in the data generation process. We load those maps into a simulator and spawn the robot on them. Then, we let it walk for a certain amount of time, around $10 - 20$s, while storing its pose, position and orientation, at a rate of $50hz$. Later, those pieces of information are used to crop around the robot a region of ground, a patch, and compute the advancement for each one of them by projecting the robot pose in the future using a time window, Δt, of two seconds. Finally, we select a threshold, tr, of twenty centimeters to label each patch as *traversable* or *not traversable* if the advancement is major or less respectively.

The following image shows the patch cropping operation, in each region, we include the robot's footprint and the ground ahead corresponding to the maximum distance it can travel in the selected Δt.



(a) Robot in the simulator.



(b) Cropped patch in 2d. (c) Cropped patch in 3d.

(d) Patch extract process. Each patch includes the robot's footprint and the maximum ground's region it can traverse in a selected Δt.

III. RESULTS

1. *ModelNet40*

The following table shows the result of the ModelNet40 dataset comparing the original DGCNN and our implementation

	ACCURACY
Original	92.2%
Ours	92.1%

TABLE I: Test accuracy score of the original DGCNN and ours implementation on the MonelNet40 dataset.

Pretrain weights can be downloaded here **To do (??)**

2. *Traversability Estimation*

Roba a caso

IV. CONCLUSION

We successfully reproduced the results of the original on the MonelNet40 reimplemmenting DGCNN entirely in PyTorch. Roba a caso

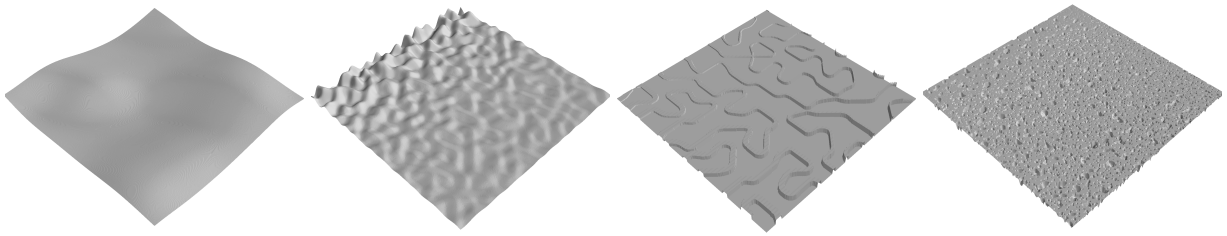


FIG. 4: Some of the thirty synthetic generated heightmaps, they include bumps, walls and holes. Those maps are loaded into the simulator and we store the robot interactions on them by let it walk forward for a certain amount of time.

[1] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," (2018), arXiv:1801.07829.

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

[3] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, (1989).

[4] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," (2018), arXiv:1812.08434.

[5] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," (2017), arXiv:1706.02413.

[6] F.Monti, D.Boscaini, J.Masci, E.Rodola, J. oda, and M. M. Bronstein, (2017).

[7] M. Simonovsky and N. Komodakis, (2017).

[8] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, , and Y. Bengio, (2017), arXiv:1710.10903.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," (2014), arXiv:1412.6980.

[10] U. the difficulty of training deep feedforward neural networks, "Xavier glorot and yoshua bengio," (2006).

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," (2015), arXiv:1502.03167.

[12] R. O. Chavez-Garcia, J. Guzzi, L. M. Gambardella, and A. Giusti, "Learning ground traversability from simulations," (2017).