Semantic Segmentation with DGCNN on ScanNet

Abstract

We present our work on point cloud semantic segmentation on ScanNet [2], using DGCNN [7]. Dynamic graph CNN [7] is a CNN-based model suitable for high-level tasks including classification and segmentation and particularly designed to deal with point clouds since they are an unordered and irregular 3D representation. In this work we conduct different experiments on DGCNN with different downsampling methods, different number of points and different set of features. We train PointNet++ [6] and use it as a baseline for comparison. In the end an evaluation of the experiments is conducted.

$$\tilde{v}_i = \psi_{\theta_i}(v_i)$$

1. Introduction

Point clouds are one of the most widely present shape representations and are the raw output of most 3D data capture devices. Learning on point clouds is challenging since they inherently lack topological structure. Using standard deep learning approaches such as CNN on point clouds is therefore not straightforward. A common approach to solving this problem is to convert point clouds to other formats such as 3D grids, but this method is inefficient because the conversion typically introduces quantization artifacts and requires more memory capacity. To tackle this issue many models have been developed to learn directly on point cloud such as PointNet [5], PointConv [8] and DGCNN [7]. In this project we perform the task of semantic segmentation on point clouds directly with the DGCNN [7] model. We use ScanNetv2 [2] as a benchmark dataset. ScanNetv2 [2] is an RGB-D video dataset containing 2.5 million views in more than 1500 scans annotated with 3D camera poses, surface reconstructions, and semantic segmentations. An example of one scene is shown in figure 1. In order to show how well DGCNN performs, we compare it to PointNet++ [6]. We briefly explain the two models in section 2. In section 3, we are explaining the experiment setup. Afterwards, in section 4 we are discussing the results in regards of their respective mIoU and accuracy.

2. Methods

In the following, we are giving a short overview of the methods which are used in the experiments.

2.1. DGCNN

Dynamic graph CNN model is based on the EdgeConv module.

In this module, for each point a graph of the k closest neighbours in the feature space is computed. The embedding of the point is updated based on its previous embedding and the embedding of its neighbours.

The graph is dynamically computed because the set of k-nearest neighbours of a point changes from layer to layer and is computed over the embeddings.

The model enables non local diffusion of information throughout the point cloud since the proximity in the feature space is different from the proximity in the input space. So, the model can capture semantic characteristics over potentially long distances in the original embedding.

The EdgeConv layer is permutation invariant and can be plugged into existing architectures. The architecture that we use for training is similar to the one proposed by the paper [7]. It is composed of a spatial transform block in the beginning, multiple interleaved EdgeConv layers and segmentation head in the end.

2.2. PointNet++

PointNet++ [6] is an extension of PointNet [5] which is more robust to non-uniformly sampled point sets. The architecture can be divided in hierarchical point set feature learning, segmentation and classification. The hierarchical structure consists of set abstractions which contains a sampling and a grouping layer.

In the sampling layer, farthest point sampling (FPS) is applied to get a better coverage for the point set. Afterwards, the grouping layer outputs sets of points each representing a local neighbourhood. Each region is then passed through a PointNet layer [5] to capture point-to-point relations in the local region.

3. Experiment

To test how well DGCNN performs on ScanNetv2, we are also running the experiments on PointNet++. ScanNetv2 consisting of 1513 scanned and reconstructed indoor scenes [2]. For the experiment, we follow the split in [6] with 1201 scenes in the training set and 312 scenes in the test set. In order to see how good DGCNN performs on ScanNetv2, we also train PointNet++ and compare it to DGCNN. Furthermore, we conduct experiments with different downsampling methods such as uniform sampling and farthest point sampling to see how it affects the performance but also the hardware limitation that a whole scene does not fit into the memory.

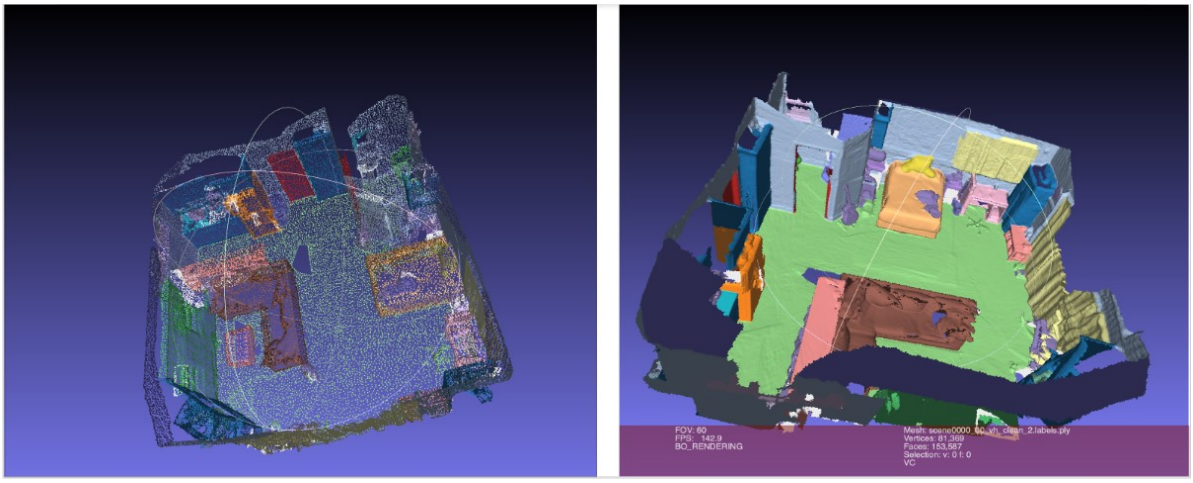

Figure 1. Example of a scene from ScanNetv2

3.1. Downsampling

In our experiment, we reduce the number of points for each scene with either point sampling methods or by using sub-volumes. In the following, we will describe the adopted methods.

3.1.1 Point sampling

Point sampling is done by selecting a subset of points to represent the original at a sparse scale. It can help improve the efficiency of 3D data processing and has many applications. There are several ways to do this task, some methods are handcrafted such as random sampling, farthest point sampling, and grid sampling. Other techniques based on learning have also been developed and proved to be better sampling methods such as S-NET [4], and SampleNet [3]. For simplicity, we use only uniform sampling and farthest point sampling. In the first method, points are sampled from a uniform distribution where each point is unique. The second is a greedy algorithm that samples points iteratively. First a random single point is sampled. Afterwards, in each iteration, the farthest point from the already sampled set is sampled. We compare these sampling techniques with different number of points ranging from 1024 to 16384 points.

3.1.2 Sub-volumes

By using sub-volumes we are able to process a scene by dividing it in multiple parts based on the number of points specified before.

1. We specify the size of the sub-volumes and the distance from the center to the next box and repeat the process for the rest of the scene.

2. Next, we merge two boxes if the number of points of two is less than the given threshold and divide a box if it is above the threshold.

3.2. Training

We train on both PointNet++ and DGCNN using the same experiment data to compare the results. We use the default models hyper-parameters given in the official repositories for our experiments and we try to fine tune the main hyper-parameters like learning rate and optimizer. Our training was limited by the GPU resources, so most of the experiment training was for around 4 hours, but with experiment that was still showing improvement we continue with the training for another 4 hours.

3.2.1 DGCNN

For the experiments we use an existing implementation of DGCNN[1] and extend it by our needs. As for the training, we begin with the hyperparameters proposed in [7]. Due to GPU resource limitations, we cannot use a batch size of 32, thus we choose an individual batch sizes for every dataset. We simply use the next smallest batch size that still fits into GPU memory. For our datasets we notice a high fluctuation in the validation loss. We consider three possible reasons for the high fluctuations: 1) low batch size, 2) overfitting, 3) high learning rate.

1) We cant use a higher batch size, due to the afore mentioned GPU resource limitations.

2) Overfitting can be tackled by regularization techniques like dropout or L2 regularization. These might smooth out the fluctuations. We dont want to further utilize dropout, as it is already part of the DGCNN model and we dont want to change the original implementation. The implementation of DGCNN, sets the L2 regularization to

[1] https://github.com/AnTao97/dgcnn.pytorch

Table 1. Experiment results for DGCNN and PointNet++

			DGCNN		DGCNN (Adam)		PointNet++	
Number of points sampled	Features	Down Sampling	avg. acc	mIoU	avg. acc	mIoU	avg. acc	mIoU
1024	xyz	US	0.29	0.26				
2048	xyz	US	0.28	0.25	0.31	0.29		
8192	xyz	US	0.28	0.26	0.27	0.26		
16384	xyz	US	0.31	0.28	0.35	0.30		0.23
1024	xyz+rgb	US	0.19	0.18				
2048	xyz+rgb	US	0.20	0.18				
8192	xyz+rgb	US	0.20	0.20				
16384	xyz+rgb	US	0.23	0.23	0.28	0.26		0.29
1024	xyz+rgb	FPS	0.19	0.21	0.24	0.24		
2048	xyz+rgb	FPS	0.21	0.22	0.28	0.25		
8192	xyz+rgb	FPS	0.21	0.22	0.37	0.31		0.32
16384	xyz+rgb	FPS	0.26	0.25	0.35	0.30		
1024	xyz+rgb	Sub-volume						
2048	xyz+rgb	Sub-volume						
8192	xyz+rgb	Sub-volume	0.21	0.21				0.30
16384	xyz+rgb	Sub-volume						

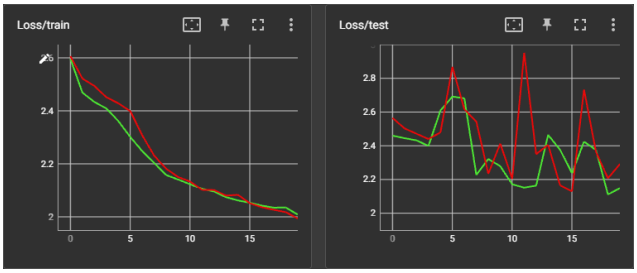

Figure 2. learning rate of 0.1 (red), learning rate of 0.01 (green)

0.0001. We experiment with higher and lower values, but they either lead to higher or similar fluctuations. We decide to keep the proposed value of 0.0001. Overfitting can also be prevented by adding more data. For example, we can add more points per sample or enrich the training set using more samples per scene (sub-volumes or data augmentation). We already consider more points per sample by having datasets with different amounts of sampled points per scene. Also, we already use sub-volumes as part of our experiments. We did not look into data augmentation because (just like sub-volumes) it will make the training set bigger, which results in longer training times, what again conflicts with the constrained time of GPU usage.

3) Lastly, we experiment with other learning rates. In figure 2 we show that with a smaller learning rate we can dampen the fluctuations in the validation loss.

To summarize, for our experiments we use the training configuration in [7] with a reduced learning rate of 0.01 and smaller (dataset individual) batch sizes. For better comparison we train every model for 100 epochs if not stated otherwise.

4. Results & Discussion

In this section we explain and discuss the results of the several experiments which we have conducted.

4.1. Quantitative results

Table 1 summarizes the quantitative results from our experiments. In the first three columns we show the dataset properties. Those are the number of sampled points, selected point features and the applied down sampling technique. In the other columns we show the results achieved on the validation set for DGCNN and PointNet++. Results are presented in mean intersection over union (mIoU) and average accuracy (avg. acc). In the following we mainly focus on mIoU score.

Firstly, we examine the DGCNN results. It is clearly visible that with the higher number of points, the higher the results. It can be observed that this applies for all selected features and sampling methods. Overall, this improves the mIoU score by up to +5%.

When moving from features that only include coordinate information (xyz) to features that also include color information (xyz+rgb), the performance drops. For the datasets using uniform sampling (US) the mIoU drops by -8% (1024 points), -7% (2048 points), -6% (8196 points), -5% (16384 points). From the results it can be observed, that with the higher number of points the smaller the performance drop. This can also be perceived when comparing with the FPS results. Because of higher number of features, we think that training the network longer might lower this gap. Due to time constraint, we could not look into this.

Using FPS instead of US we again can improve the re-

sults by up to +4% mIoU. An improved performance does not come with a surprise. Farthest points are often geometrically important points - points that shape objects in the scene. With FPS we can sample more meaningful points than with uniform sampling. This way the model can easier learn the representation, which results in a better score.

Lastly, we also evaluated the sub-volume technique on 8192 points with xyz+rgb features. With US and FPS one sample corresponds to one scene. With sub-volumes one sample is one sub-volume, hence for one scene we have now multiple samples, which leads to a overall bigger training set. Due to the increased training set size we could only run 29 epochs of training in four hours. From figure 4 we can assume, that with longer training the loss will most likely decrease, which will most likely improve the mIoU score. Thus, the difference of -1% mIoU score between FPS and sub-volume might be neglectable.

In DGCNN [7] they trained and evaluated the model on S3DIS [1] for the task of semantic segmentation. Although, we use different datasets, the model achieves significantly better results. We took this as an occasion to further tune our model. Therefore, we replaced the SGD optimizer with the Adam Optimizer. Adam is known for more effectively training, which is especially suitable for our case, where we are constraint in training time. With this change we achieve similar or better results. Most models improve by +3% mIoU, whereas one model improves up to +8% mIoU.

Due to limited time, we only compare DGCNN with PointNet++ on some datasets. The best PointNet++ model achieves an mIoU score of 0.32. This seems reasonable as the official PointNet++ mIoU score on the ScanNet benchmark is 0.33. Compared to the best performing DGCNN model, PointNet++ achieves a +1% higher mIoU score.

4.2. Qualitative results

Figure 3 shows two points clouds. One annotated by our best performing DGCNN model (left) and the other annotated with the ground truth labels (right). The color of the point reflects the assigned class. One can see that especially points that are part of the floor are well classified. On the other end, our method fails to distinguish i.a. the green, orange and purple points in the ground truth, which are all classified with the same class in our prediction (light green). We suspect that the class imbalance might cause this issue, because the model does better with points of high occurrence (floor) compared to points of low occurrence. We observed that more recent methods [8] implement a weighting in the loss function. It is possible that this weighting addresses the class imbalance problem.

5. Conclusion

In this work, we trained DGCNN on the ScanNet dataset for the semantic segmentation task. With a various number of experiments, we showed that varying the number of points per sample the performance increase with a greater number of points and the effectiveness of FPS over US. Additionally, we compared DGCNN to PointNet++ and the results are almost equivalent regarding the mIoU.

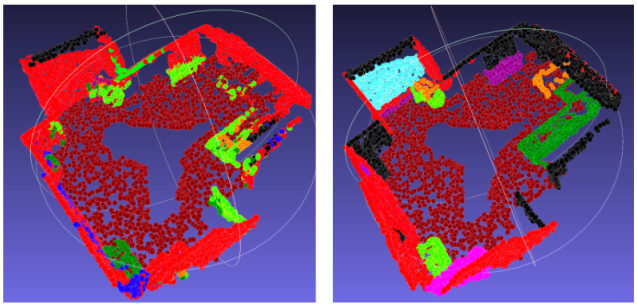

Figure 3. Qualitative results of DGCNN evaluated on 8192 points (left) compared to point cloud annotated with ground truth labels (right)

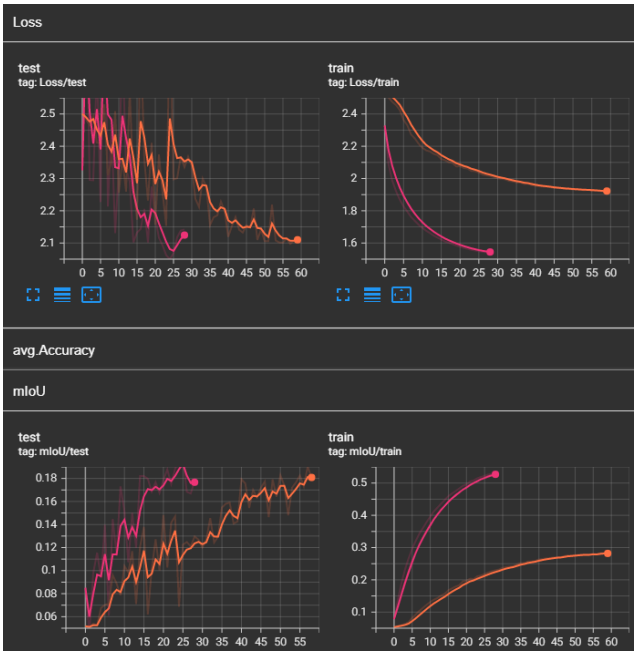

Figure 4. DGCNN loss and mIoU score while training on 8192 points with xyz+rgb features using sub-volumes (pink) and uniform sampling (orange)

References

[1] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2016. 4

[2] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-

annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017. 1

[3] Itai Lang, Asaf Manor, and Shai Avidan. Samplenet: Differentiable point cloud sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7578–7588, 2020. 2

[4] Yiqun Lin, Lichang Chen, Haibin Huang, Chongyang Ma, Xiaoguang Han, and Shuguang Cui. Beyond farthest point sampling in point-wise analysis. *arXiv preprint arXiv:2107.04291*, 2021. 2

[5] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 1

[6] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, 2017. 1

[7] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds, 2018. 1, 2, 3, 4

[8] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019. 1, 4