Semantic Segmentation on Point Clouds PointNet++ with Attention and Additional Features

Tim Pfeifle
tim.pfeifle@tum.de

Maximilian Rieger
max.rieger@tum.de

Abstract

A popular deep learning model to process 3D data in the form of point clouds is PointNet++. In this work we modify PointNet++ by introducing an attention mechanism and adding more features. We analyze the impact of these changes on the models performance at the semantic segmentation task of ScanNet.

1. Introduction

In comparison to the widely used voxel format for 3D data, *PointNet++* [4] works on computationally less expensive point clouds. Despite the computational advantage, point cloud based methods do not yield as strong performance as state-of-the-art voxel based methods. We tried to improve the performance of PointNet++ on the *ScanNet* [1] semantic segmentation task by modifying the architecture and adding more features.

2. Related Work

2.1. ScanNet

The ScanNet data set provides colorized 3D surface reconstructions of over 700 unique indoor scenes and provides labels for semantic segmentation. We can easily derive point clouds and colors for each point from this format. Additionally, the surface reconstruction also allows to estimate the orientation of the surface at each point.

2.2. PointNet

PointNet [3] was a milestone for deep learning on point clouds. As described in Equation 1, it uses an MLP $h(x)$ to distinctly transform the features of each point x_i, then combines the resulting feature vectors by a $\max$ pooling function.

$$f(\{x_1, ..., x_n\}) = \gamma\left(\max_{i=1,...,n}\{h(x_i)\}\right) \tag{1}$$

The resulting vector is fed into another MLP $\gamma(x)$. In addition, PointNet++ allows to extract features hierarchically by dividing point clouds into groups, on which PointNet can be used. The results of these groups form a new point cloud, which is used analogously in the next step.

2.3. Attentional ShapeContextNet

Xie et al. replaced the $\max$ pooling function of PointNet by an attention mechanism [7]. The self attention formulation introduced by Vaswani et al. [6] can be used, because it is a symmetric function, i.e. it works on an unordered set of variable size. However, this method is inferior to PointNet++, which was published at a similar time.

3. Method

We followed two approaches to improve the performance of PointNet++: introducing an attention mechanism and adding additional features.

3.1. Attention

In contrast to [7], we used PointNet++ as our baseline and combined its hierarchical structure with an attention mechanism. Each layer of PointNet++ includes an instance of PointNet. Similarly to Attentional ShapeContextNet, we replaced the $\max$ pooling function of these instances as described in Equation 2. To create a combined feature vector, we used a slightly modified version of self attention introduced in [6]:

$$f(\{x_1, ..., x_n\}) = \mathit{Attention}(q(x_0), k(x_i), v(x_i)) \tag{2}$$

For each group, $q(x)$ computes a single query vector from the central point in the group. $k(x)$ transforms each point in a group to a key vector. This vector is used to compute an attention score for each point by computing the dot product of key and query: $s_{x_i} = k(x_i)^T q(x_0)$. The feature vectors $v(x_i)$ are scaled by the scores s.

This form of attention has still a significant drawback. By assigning only a single weight to each feature vector, it is not possible for the model, to emphasize certain features of a point. To overcome this shortcoming, we use *Multi-Head Attention* [6]. Instead of only assigning one weight to the whole feature vector of a point, $q(x_0)$, $k(x_i)$ and $v(x_i)$ are split in multiple vectors, so called heads. For these, the

result is computed exactly as above, and the resulting feature vectors for each point are concatenated.

The functions q, k and v are implemented by a single dense layer in our model.

3.2. Features

The reference implementation of PointNet++ uses only the coordinates of points as an input. We worked with two additional features, color and surface orientation. These can be fed to PointNet++ as point embedding for the first layer.

Color features are already given in ScanNet data for each point and can be simply extracted.

Surface orientation for points cannot be extracted from the dataset directly. However, the mesh reconstruction of ScanNet can be used to estimate the surface orientation at each point. We used an angle weighted average $\hat{\boldsymbol{N}}_v$ of the normals of all adjacent surfaces to a point v as described in Equation 3 and proposed in [5].

$$\hat{\boldsymbol{N}}_v = \sum_{\boldsymbol{N}_i \in A_v} \alpha_{\boldsymbol{N}_i} \boldsymbol{N}_i \tag{3}$$

$\alpha_{\boldsymbol{N}_i}$ denotes the angle between the two edge vectors of $\boldsymbol{N}_i$, which start at v. This method relies on a mesh, but there exist solid methods to reconstruct a mesh from a point cloud, such as [2].

3.3. Dataset

A large part of our work went into efficiently loading the data, because the new dataset with the additional features did not fit into memory anymore. Consequently, we implemented a new data loader based on the Tensorflow dataset API, wrote precomputation scripts for extracting the random scene subsets and wrote new training and validation methods. This dataloader, in contrast to simply loading each scene from disk and then computing the scene subset, reduced the training time from 3:25min to 1:25min by 59%.

3.4. Scene Subsets

To predict the large ScanNet scenes ($\geq$ 100k points each), following PointNet++, we feed subsets of points to the model by sampling n points from a cuboid region. Instead of sampling those points from exactly the region $c = (d \times d \times d')$ which we want to predict, we sample points from the larger concentric region $c_{larger} = (d + x \times d + x \times d')$ with $x = 0.4$ and only keep the predictions of the points inside of c. Thereby we make sure that points at the border of the smaller cube c still get an embedding that considers their neighboring points. We then advance the cuboid region c gridwise until we predicted all points of the scene.

4. Experiments and Results

To evaluate each modification made, we started experiments with only one change at a time. The first part of this section covers the experiments with using the attention mechanism. Then the results of our experiments with additional features and different scene subsets are given. In the final paragraph, we give a short analysis of the performance of PointNet++ without any changes.

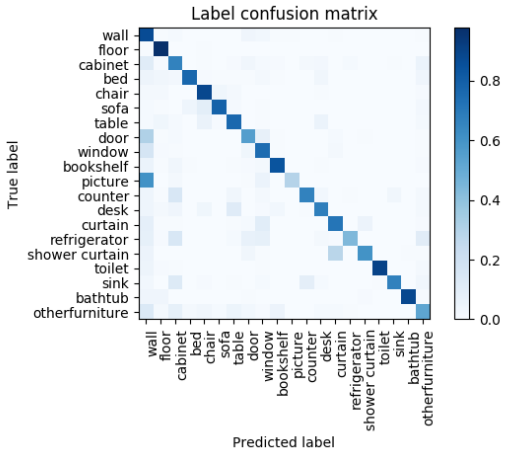

Figure 1. Confusion matrix of our model with additional features

4.1. Attention

Replacing pooling with attention led to significantly worse performance. We further investigated this, by starting four different experiments, for which we replaced the pooling operation in only one layer of PointNet++. To evaluate a variety of models more quickly, we executed these experiments on a reduced dataset, containing only $1/3$ of the scenes of ScanNet.

Method	train mIoU	val mIoU
Baseline	**0.480**	**0.361**
Att-All-Layers	0.223	0.178
Att-Layer-1	0.387	0.293
Att-Layer-2	0.426	0.326
Att-Layer-3	0.439	0.325
Att-Layer-4	0.436	0.337

Table 1. Mean IoU of models with attention instead of $\max$ pooling for certain layers. Trained on reduced dataset.

As visible in Table 1 the attention mechanism decreased the performance of the model in every single case. One can also see, that the train performance decreases just as well as the validation performance. Hence, the model does not overfit, but just performs worse overall.

4.2. Features

With the additional color information we overfitted stronger on the training dataset as seen in Table 3, but our validation performance increased only slightly. In contrast,

Method	mIoU	Cabinet	Chair	Floor	Picture	Sofa	Table	Wall	...
PointNet++	0.339	0.256	0.360	0.677	0.117	0.346	0.232	0.523	...
Baseline (PointNet++ reproduced)	0.481	0.340	0.686	0.931	0.102	0.580	0.470	0.711	...
Additional Features (ours)	**0.557**	**0.491**	**0.744**	**0.946**	**0.205**	**0.643**	**0.497**	**0.756**	...

Table 2. Test scores from ScanNet benchmark of initial PointNet++ submission, our version without changes and our version with normals and color

when using the additional normals we were able to considerably increase the validation performance by 5.3%. By using colors as well as normals, our model's performance increased even more. This suggests that our model has learned to combine color and normal information in a meaningful way. It still has difficulties to differentiate classes with very similar normal vectors, e.g., pictures and doors from walls (see Figure 1).

Method train mIoU	train mIoU	val mIoU
Baseline	0.592	0.456
Colors	0.610	0.461
Normals	0.649	0.509
Normals & Colors	**0.671**	**0.533**

Table 3. Comparison of models using different additional features

Method	val mIoU
Baseline without context	0.490
Baseline (with context)	**0.517**

Table 4. Comparison of models with and without using the context of the wrapping cube c_{larger} for the scene subsets

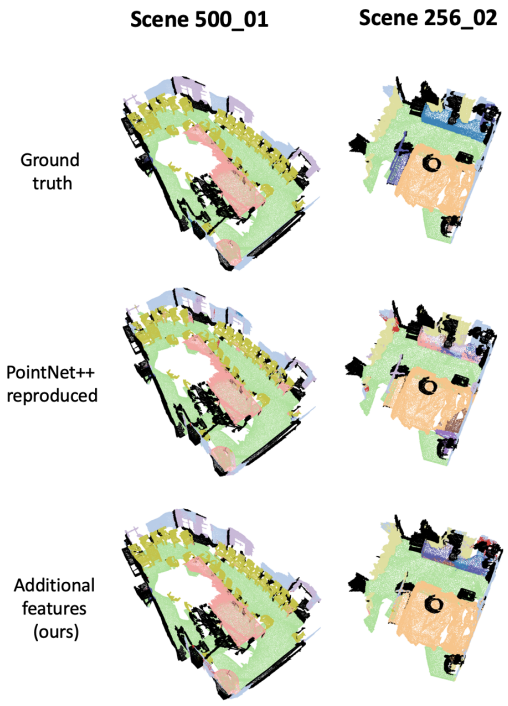

Figure 2. Qualitative results on two exemplary scenes

4.3. Scene Subsets

By additionally using the larger concentric cube c_{larger} to support the predictions for each cube c, the models performance increased by 2.7% on the ScanNet validation dataset, as seen in Table 4.

4.4. Baseline

The validation and test scores for our feature model were rather similar. Surprisingly, we found that the validation scores for our baseline implementation were significantly better that the test scores of PointNet++ by Qi et al. at the ScanNet benchmark. As we did not make substantial changes to the reference implementation, we expected the scores to be very similar. To verify this discrepancy, we also uploaded test predictions for our baseline. As you can see in Table 4.1, also the test scores of our implementation are considerably better than the reference. Unfortunately, we do not know the reason of this for sure.

One explanation is, that the model for the original submission was not trained until convergence. We suspect, that the model of the original submission was trained on the older ScanNet-v1 dataset, because ScanNet-v2 was released after the submission. This would also explain our model, which was trained on ScanNet-v2, having better performance.

5. Conclusion

We introduced a modified PointNet++ architecture, which uses an attention mechanism and showed, that PointNet++'s $\max$ pooling is superior to that. Using bigger cuboids to provide context for points at the border of subsets increases the performance. We also showed, that the use of surface normals as input features can improve the performance of PoinNet++ significantly. PointNet++ can reach considerably better performance than the scores of the ScanNet benchmark suggest.

References

[1] A. Dai, A. X. Chang, M. Savva, M. Halber, T. A. Funkhouser, and M. Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. *CoRR*, abs/1702.04405, 2017.

[2] H.-W. Lin, C.-L. Tai, and G.-J. Wang. A mesh reconstruction algorithm driven by an intrinsic property of a point cloud. *Computer-Aided Design*, 36(1):1 – 9, 2004.

[3] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[4] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, abs/1706.02413, 2017.

[5] G. Thürrner and C. A. Wüthrich. Computing vertex normals from polygonal facets. *Journal of Graphics Tools*, 3(1):43–46, 1998.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[7] S. Xie, S. Liu, Z. Chen, and Z. Tu. Attentional shapecontextnet for point cloud recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.