Graph Neural Networks for Point Cloud Segmentation

Niklas Kaspareit

Data Science, University of Potsdam, Potsdam, Germany niklaskaspareit@gmail.com

1 Introduction

In various fields like robotics or other object representations point clouds are being used to provide a 3D representation of the data. Point clouds are convenient to use as they are sparser than for example a tensor representation as empty spaces can be left out. However to train a neural network using point clouds the point clouds need to be transformed into a representation that the neural network can work with. There are plenty of works, that transform point clouds into a tensor representation using voxelization. This enables the use of convolutional neural networks. However due to the mentioned challenges of using a tensor to represent a point cloud this method can be computationally less efficient. Qi et al. [5] came up with a new approach which trains a neural network directly on the set of points that the point cloud consists of. In this work we will show an alternative approach to directly using point clouds by transforming the point cloud into a graph and using a graph neural network for segmentation.

2 Dataset

Yi et al. [12] have created a version of the *Shapenet* dataset in a point cloud format, which is conveniently provided through Pytorch geometric as well. The dataset contains a representation of different 3d shapes of objects. Wu et al. [11] came up with the the idea of using a convolutional deep belief network as a "probability distribution of binary variables on a 3D voxel grid". The original dataset provides about 17.000 shapes in the format of a point cloud with 16 different categories with a total of 50 segmentations. Each category has between 2 to 6 individual segments.

In this work we will only focus on a subset of this dataset due to limited computational budget. The subset is reduced to the *airplane* class. This class, with a total of $N = 2690$ graphs, is split into a train dataset with 1958 graphs, a validation dataset with 391 graphs and a test dataset with 341 graphs. For the actual training and evaluation batches are used to increase processing speed and enable parallel computation. Besides that a custom visualization dataset with only 3 manually selected graphs is used for the simple visualization of the results of the trained models. For each point p in the point cloud P of there exists a p_x-value with the encoded probabilistic representation of the corresponding point of the airplane. Secondly the dataset provides the positional data p_{pos} in terms of the point in the $\mathbb{R}^3$ space. And for the segmentation there exists a p_y-value with the corresponding category per point. There exist 4 different classifications for this subset of airplanes: the "body", the "wings", the "turbines" and the "fins". We will call this point cloud P with $P = \{x_i, pos_i, y_i : i \in 1, \cdots, N\}$.

Since we do not work on point clouds, but actually on graphs, it is required to create a graph $G = (V, E)$ with $V = \{p_x : p \in P\}$. One can either generate a *radius graph* or a *k-NN Graph* (or k nearest neighbors graph). In this work we focused on a radius graph without any more detailed discussion. The radius graph is computed using the method of a KDTree which has been introduced by Maneewongvatana et al. in [4].

Henceforth some preprocessing steps are used while loading the dataset. The number of sampled points is fixed to 2048 to guarentee equal sizes of the point clouds. The distance between two connected nodes and the degree as a one hot encoding are added as features to the node. This results in an input graph with 104 dimensions in the node features.

3 Methodology

3.1 selection of hyper parameters

There are multiple hyper parameters that have to be optimized. A collection of all hyper parameters and their search spaces can be found in 1. We select the hyper parameters using Tree Parzen Estimators. It is a technique of bayesian optimization that has been described as an algorithm for hyper parameter optimization by Bergstra et al. [1]. The idea is to use $P(\theta|l)$ with θ being a single hyper parameter and l being the loss, instead of $P(l|\theta)$. So this is interesting because usually we would model one probability density function (pdf) using $P(l|\theta)$ and then use this to find the highest expected improvement. For the TPE a new hyper parameter λ is introduced that is used as a threshold to split measurements into a "good" and a "bad" group. Then for each group a new pdf is determined and based on the two pdfs the expected improvement can be determined to find the maximum expected improvement. We will not go deeper into detail in this work, however there is a nice and intuitive explanation video by the youtube channel AIxplained which can be accessed via https://www.youtube.com/watch?v=bcy6A57jAwI&ab_channel=AIxplained.The parameter λ has been chosen by the default function of *optuna*.

$$\lambda = \min\left(\{\lceil 0.1n \rceil, 25\}\right) \tag{1}$$

with n being the number of completed or pruned trials.

3.2 evaluation metrics

The code basis provides the option of choosing two different options for the evaluation metric. This work has mainly been focused on using *accuracy*, as it has been used in the PointNet paper [5] for better comparability. The code base also provides the option to use the Area Under the Receiver Operating Characteristic Curve (AUROC) as a metric for hyper parameter optimization and comparison between the PointNet baseline and the suggested architecture.

3.2.1 accuracy

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \tag{2}$$

For the calculation we determine the correct predictions per batch, sum them up and then divide the total correct predictions by the total overall predictions.

3.2.2 Area Under the Receiver Operating Characteristic (AUROC)

So the idea is to first compute the Receiver Operating Characteristic (ROC) curve. This curve is simply the trade off between the true positive rate and the false positive rate. These values are determined by a parameter. So a parameter is used to weight whether the point should rather be classified as one class or the other. Moving this parameter from 0 to 1 will result in the ROC curve.

For the multi class case this will not directly work as this is a one class to another comparison. For this one can either compare one vs the rest (OvR) or one vs one (OvO). [8]

4 Implementation

4.1 Baseline PointNet

As the baseline for comparison the PointNet [5] is been used with their suggested hyper parameters and on the subset of only airplanes of the original subset that it has been trained on. The overall training is done over 200 epochs just as it has been trained on the whole dataset as well. This provides a fair comparison on the new reduced dataset.

4.2 Graph segmentation network

As an architecture different building blocks have been used to setup the general architecture 1. first block contains a graph convolutional layer, followed by an activation layer and a dropout layer. This block works as a embedding block. After this block we use n hidden layer building blocks, which work with n being a hyper parameter that can be used to define the number of hidden blocks that are supposed to be used in the hidden layer. Last there is a block for transforming the output of the hidden blocks into a class prediction with a linear and a softmax layer.

Now some of the blocks are not uniquely identified like the graph convolutional layer and the normalization layer. These layer types can be chosen as hyper parameters. The type of graph convolutional layer can be chosen as one of the following types: GCN layer [3], GraphSAGE layer [2] or Graph attention layer [10]. All graph convolutional layers are of the same type for one model. The normalization layer can either be removed or it can be either a batch normalization layer [6] or an instance normalization layer [9]. For the activation function the rectified linear unit (ReLU) is used.

The network uses the cross entropy loss for multiclass classifications.

$$l(x|\theta) = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{3}$$

This architecture will be called Graph Segmentation Network (GSegNet) in this work.

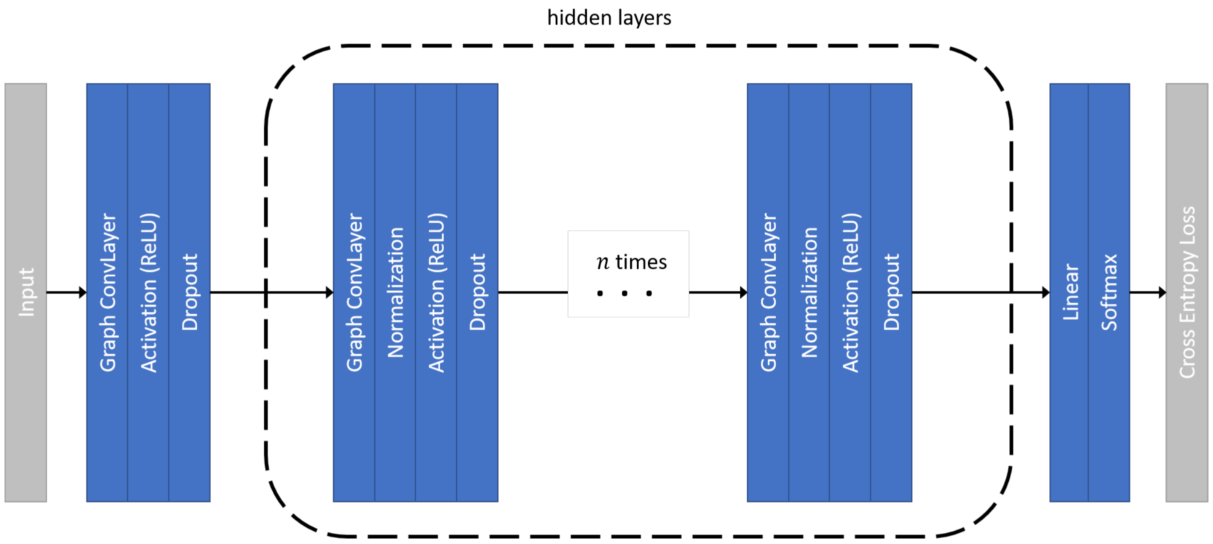


Figure 1: representation of the architecture

4.3 hyper parameters

The general architecture still requires a specification of all the hyper parameters. The search spaces for the hyper parameters were chosen based on experience. They are specified in table 1. To find the best hyper parameters, the model has been trained in 20 trials using the Tree-structured Parzen Estimator as a bayesian optimization approach.

category	search space
radius threshold	$U(0.02, 0.06)$
batch size	$\in \{16, 32\}$
epochs	$\in \{20, 40, 60\}$
embedding dimension	$\in \{64, 128\}$
n layers	$\in \{2, 3, 4, 5\}$
graph convolutional layer type	$\in \{'SageConv',' GATConv',' GCNConv'\}$
hidden dimensions	$\in \{128, 256\}$
learning rate	$log\ U(1e-5, 1e-2)$
dropout rate	$U(0, 0.4)$
slope (Leaky ReLU)	$U(0, 0.2)$
norm layer type	$\in \{'None',' BatchNorm',' InstanceNorm'\}$

Table 1: Search spaces for hyper parameters. With U being the uniform distribution, *log* U being the uniform distribution in the logarithmic domain. Categories chosen by a set denoted by $\in \{\cdots\}$ means that one of the items in the set is chosen with equal weighting for each item.

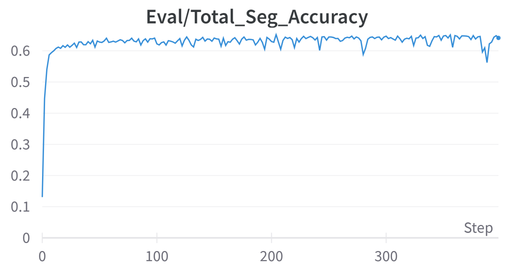
5 Results

After 20 trials of training different combinations of hyper parameters, the best combination of hyper parameters can be found in table 2.

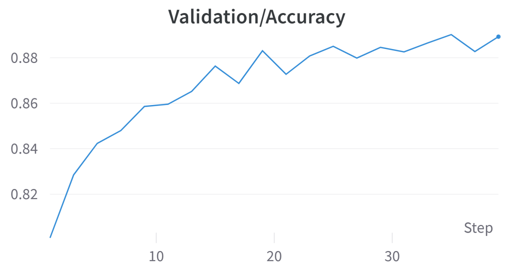
hyper parameter	value
epochs	20
dropout rate	≈ 0.008
embed dim	128
batch size	32
graph convolutional layer	GraphSAGE
hidden dimensions	256
norm layer	Instance Norm
hidden layers	3
learning rate	≈ 0.0005
radius threshold	0.04

Table 2: Best performing choice of hyper parameters after 20 trials of training different combinations.

The performance over the epochs is visualized in the figures 2a and 2b. The highest scored accuracy over 200 epochs for the PointNet architecture is 0.6521 while the highest scored accuracy over 20 epochs for the GSegNet network is 0.8902. This is a difference of 0.2381 in the accuracy. The visualization dataset has been used to show this difference directly on the 3D pointclouds. In figure 3 the reference point cloud with the classified points is shown as a ground truth and next to it the predicted classifications of the PointNet architecture as well as the GSegNet architecture are visualized.



(a) accuracy of pointnet



(b) accuracy of pointGCN

Figure 2: Comparison of the accuracies of the best performing runs.

6 Conclusion

Overall the GSegNet has performed very nicely and it outperformed the PointNet in the point classification task by a large amount and even in less epochs. Each training epoch is more complex due to the complexity of the GSegNet and hence takes more time. Therefore it was expected, that the GSegNet would produce better results than the PointNet in less epochs. However it is very promising, that the graph based approach outperforms the set of point approach by such a lot.

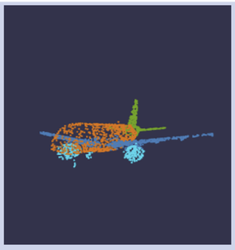
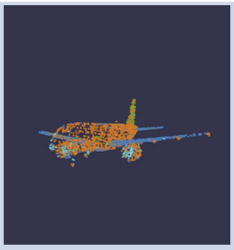
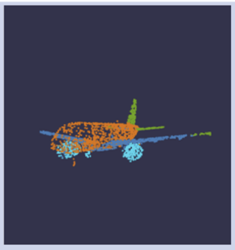
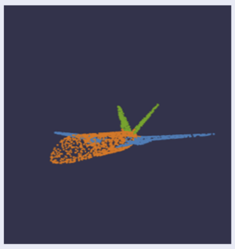
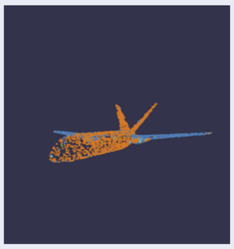
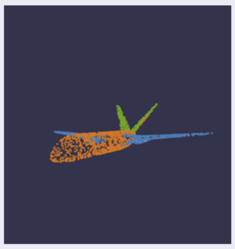
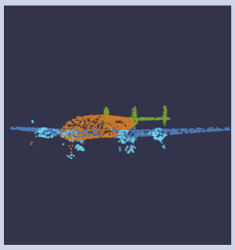
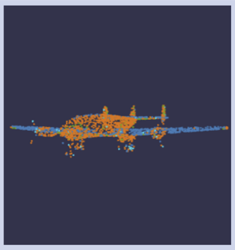
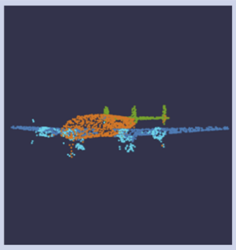
Modell	Reference	Prediction Point Cloud	Prediction GSegNet
1			
2			
3			

Figure 3: Visualization of the point clouds of airplanes with the classification of each point shown by the color. Left column shows the reference data of the true classification of each point. In the middle is the point cloud with the classification predicted based on the PointNet dataset after 200 epochs of training using the suggested hyper parameters. On the right side is the point cloud with the classification predicted based on the GSegNet after 20 epochs of training using the best parameters after hyper parameter optimization.

There are still some open challenges that are left for future work. Firstly the results are of the hyper parameter optimization are based on minimizing the test accuracy as a metric. This can have a negative impact on the actual selection of the hyper parameters as a post-hoc metric has been used. The code base has already been updated and needs to be rerun, however due to time and computational budget constraints it has not yet been possible to update the results in this work. Overall this should not result in a worse performing network as this only affect the choice of the hyper parameters not the performance of the chosen hyper parameters.

Secondly the used metric in this work is *accuracy* however the metric *AUROC* has been introduced in this work and is integretad in the code as well. Overall AUROC should be a better choice for the metric as it gives more depth into the interpretation. Accuracy can skew results easily. One example could be: imagine an airplane that has a huge body, but very small wings in comparison. A neural network that predicts all points to be classified as the body, it might still achieve an accuracy higher than 0.8. In this scenario the AUROC will be much better for the evaluation.

So far the work is only comparing to the PointNet architecture. However there has been newer work which tried to use a similar approach of using graph neural networks on pointnets to do segmentation and do point classification. One of these works is called

Point-GNN [7]. Also there are multiple approaches of using convolutional neural networks with point clouds using stronger voxelization. It would make sense to compare more network architectures using the same dataset and the same metrics as well.

Another point is that no real edge features have been used in the network. The distance between two points has been added to the node feature space. Maybe it is worth considering moving a few properties to the edges or add new features to the edges?

For very ambitious researchers the general architecture could also be still overworked. For example the ReLU activation functions could maybe be replaced by LeakyReLU functions. The hyper parameter for the slope of the LeakyReLU can already be optimized in the hyper parameter optimization.

Besides these open points, this work has been a great example of the application of graph neural networks on point clouds and it has shown very nice results.

References

[1] James Bergstra et al. "Algorithms for Hyper-Parameter Optimization". In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor et al. Vol. 24. Curran Associates, Inc., 2011. URL: https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

[2] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *CoRR* abs/1706.02216 (2017). arXiv: 1706.02216. URL: http://arxiv.org/abs/1706.02216.

[3] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *CoRR* abs/1609.02907 (2016). arXiv: 1609.02907. URL: http://arxiv.org/abs/1609.02907.

[4] Songrit Maneewongvatana and David M. Mount. *Analysis of approximate nearest neighbor searching with clustered point sets*. 1999. arXiv: cs/9901013 [cs.CG].

[5] Charles Ruizhongtai Qi et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *CoRR* abs/1612.00593 (2016). arXiv: 1612.00593. URL: http://arxiv.org/abs/1612.00593.

[6] Mengye Ren et al. "Normalizing the Normalizers: Comparing and Extending Network Normalization Scheme". In: *ICLR* (2017). URL: https://openreview.net/pdf?id=rk5upnsxe&uclick_id=6fa3be39-bd2e-41df-9d42-61d18b3bea1b.

[7] Weijing Shi and Ragunathan Rajkumar. "Point-GNN: Graph Neural Network for 3D Object Detection in a Point Cloud". In: *CoRR* abs/2003.01251 (2020). arXiv: 2003.01251. URL: https://arxiv.org/abs/2003.01251.

[8] Vinícius Trevisan. *Multiclass classification evaluation with ROC Curves and ROC AUC*. https://towardsdatascience.com/multiclass-classification-evaluation-with-roc-curves-and-roc-auc-294fd4617e3a. Accessed: 2023-10-16.

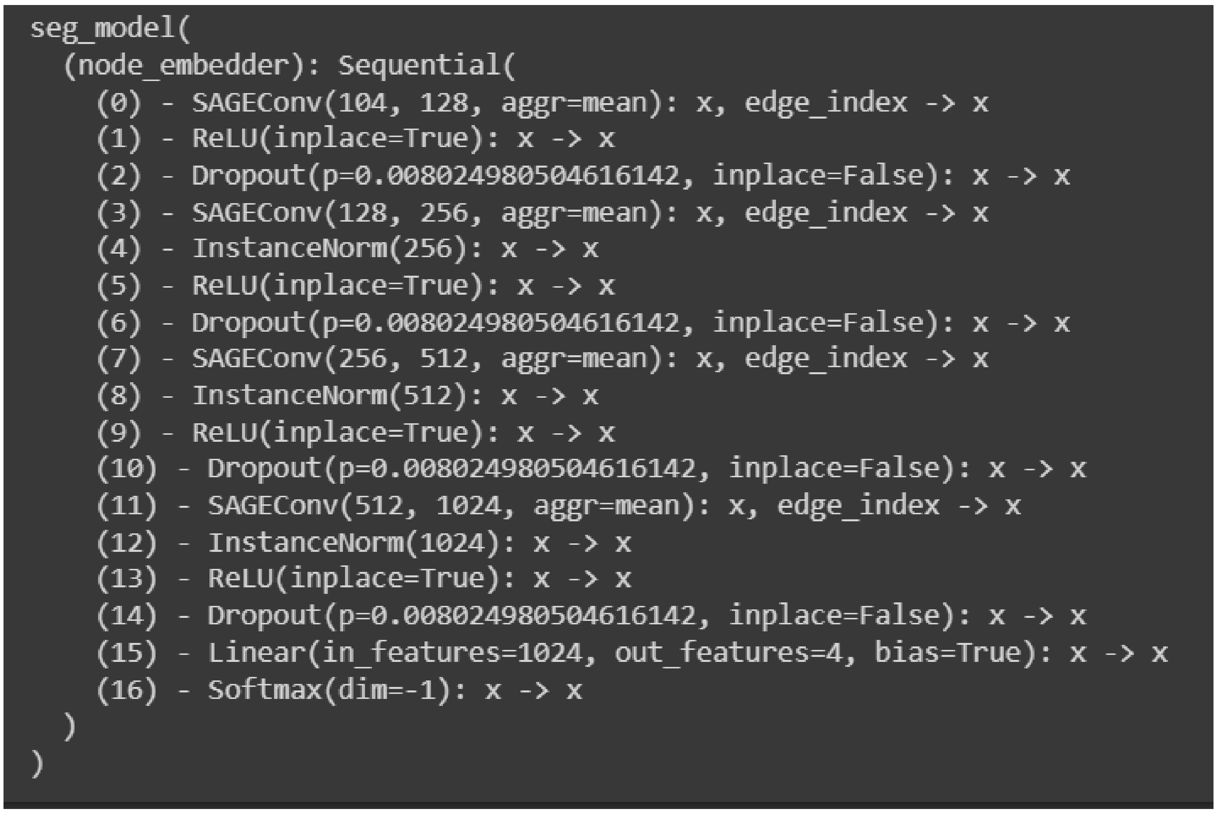
[9] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. "Instance Normalization: The Missing Ingredient for Fast Stylization". In: *CoRR* abs/1607.08022 (2016). arXiv: 1607.08022. URL: http://arxiv.org/abs/1607.08022.

[10] Petar Veličković et al. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML].

[11] Zhirong Wu et al. *3D ShapeNets: A Deep Representation for Volumetric Shapes*. 2015. arXiv: 1406.5670 [cs.CV].

[12] Li Yi et al. "A Scalable Active Framework for Region Annotation in 3D Shape Collections". In: *SIGGRAPH Asia* (2016).

7 Appendix



```
seg_model(
  (node_embedder): Sequential(
    (0) - SAGEConv(104, 128, aggr=mean): x, edge_index -> x
    (1) - ReLU(inplace=True): x -> x
    (2) - Dropout(p=0.008024980504616142, inplace=False): x -> x
    (3) - SAGEConv(128, 256, aggr=mean): x, edge_index -> x
    (4) - InstanceNorm(256): x -> x
    (5) - ReLU(inplace=True): x -> x
    (6) - Dropout(p=0.008024980504616142, inplace=False): x -> x
    (7) - SAGEConv(256, 512, aggr=mean): x, edge_index -> x
    (8) - InstanceNorm(512): x -> x
    (9) - ReLU(inplace=True): x -> x
    (10) - Dropout(p=0.008024980504616142, inplace=False): x -> x
    (11) - SAGEConv(512, 1024, aggr=mean): x, edge_index -> x
    (12) - InstanceNorm(1024): x -> x
    (13) - ReLU(inplace=True): x -> x
    (14) - Dropout(p=0.008024980504616142, inplace=False): x -> x
    (15) - Linear(in_features=1024, out_features=4, bias=True): x -> x
    (16) - Softmax(dim=-1): x -> x
  )
)
```

Figure 4: Generated Pytorch model using the best combinations of hyper parameters after 20 trials of optimization also showing dimensions of each layer.