Roof Segmentation based on Deep Neural Networks

Regina Pohle-Fröhlich[1], Aaron Bohm[2], Peer Ueberholz[2], Maximilian Korb[1] and Steffen Goebbels[1]
[1]*Institute for Pattern Recognition, Faculty of Electrical Engineering and Computer Science, Niederrhein University of Applied Sciences, Reinarzstr. 49, 47805 Krefeld, Germany*
[2]*Institute for High Performance Computing, Faculty of Electrical Engineering and Computer Science, Niederrhein University of Applied Sciences, Reinarzstr. 49, 47805 Krefeld, Germany*

Keywords: Building Reconstruction, Deep Learning, Convolutional Neural Networks, Point Clouds.

Abstract: The given paper investigates deep neural networks (DNNs) for segmentation of roof regions in the context of 3D building reconstruction. Point clouds as well as derived depth and density images are used as input data. For 3D building model generation we follow a data driven approach, because it allows the reconstruction of roofs with more complex geometries than model driven methods. For this purpose, we need a preprocessing step that segments roof regions of buildings according to the orientation of their slopes. In this paper, we test three different DNNs and compare results with standard methods using thresholds either on gradients of 2D height maps or on point normals. For the application of a U-Net, we transform point clouds to structured 2D height maps, too. PointNet and PointNet++ directly accept unstructured point clouds as input data. Compared to classical gradient and normal based threshold methods, our experiments with U-Net and PointNet++ lead to better segmentation of roof structures.

1 INTRODUCTION

For a variety of simulation and planning applications, 3D city models are used. These models typically are exchanged in the XML based description standard CityGML. Each polygon represents a single wall, roof facet or other building element according to the chosen Level of Detail (LoD). Due to available data, most CityGML models are given in LoD 2, i.e., they include roof and wall polygons but no further details such as windows or doors. Often such models are based on roof reconstruction using airborne laser scanning or photogrammetric point clouds obtained from oblique areal images. There are two main approaches to roof reconstruction that might be used in combination on point clouds, cf. (Wang et al., 2018b): In the first approach model driven methods fit parameterized roof models to building sections. This leads to simple geometries. In some cases they differ significantly from the real roof layout because typically catalogues of parameterized roof models are quite small. In addition, parameters like slope might be estimated wrongly due to dormers or other small building elements. The second approach is data driven, where plane segments are fitted to the point cloud and combined to a watertight roof. This allows modeling even sophisticated roof geometries that, for example, can be found with churches. However, this approach is sensitive to noise. Whereas ridge lines can be detected quite reliably by intersecting estimated planes, step edges are difficult to reconstruct if point clouds are sparse, as in the case of airborne laser scanning, or noisy, as in the case of photogrammetric point clouds.

In our previously developed data driven modeling workflow (see (Goebbels and Pohle-Fröhlich, 2016) and subsequent papers), we first segmented areas in which the roof's gradients homogeneously point into the same direction. Only within such areas, we then used RANSAC to find planes. Without segmentation prior to RANSAC, one would find planes which might be composed from many not connected segments or even planes that just cut larger structures. We computed gradient directions on a height map. This is a 2D greyscale image which is interpolated from the z-coordinates of the point cloud. We used linear interpolation on a Delaunay triangulation of the points. Small gradients belong to flat roofs. Gradients with length above a threshold value point into a significant direction. In order to gain segmentation we classified these directions. There are some shortcomings with this method: Classification depends on computed threshold values. Moreover, to obtain the height map we had to interpolate sparse or noisy point cloud data. Results depend on the choice of resolution and

Pohle-Fröhlich, R., Bohm, A., Ueberholz, P., Korb, M. and Goebbels, S.
Roof Segmentation based on Deep Neural Networks.
DOI: 10.5220/0007343803260333
In *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2019)*, pages 326-333
ISBN: 978-989-758-354-4

the interpolation method. By interpolating, we lose sharpness of ridge lines and step edges. In addition, this is a 2D method. We cannot distinguish between points with same x- and y- coordinates that for example occur with walls, chimneys, antennas or overhanging trees.

Now the idea is to replace the segmentation step by a deep neural network (DNN). In recent years, DNNs have been applied successfully to many classification and even segmentation problems. DNNs could be better suited to handle outliers due to antennas and chimneys as well as occlusion by trees. Building reconstruction algorithms typically use threshold values that depend on point cloud density. DNNs might be able to learn such threshold values. Furthermore, due to normalization of input data, cloud density becomes less important.

However in contrast to images, point clouds are unstructured, the amount of data is often much higher, and there might be no color information. Due to the lack of structure, convolution of point clouds with kernels is more elaborate than convolution of images. Nevertheless, in recent years neural networks, also including convolutional networks, have been developed to directly work on point clouds.

In our paper, we evaluate 2D segmentation with U-Net based on interpolated height maps as well as 3D segmentation directly on the point cloud with PointNet and PointNet++. The tested networks classify 3D points to belong to six classes describing walls, flat roofs, roofs with a main slope facing north, roofs with gradients pointing east, south and west. Finally, we compare the results with our previous methods using thresholds either on the gradient values of the 2D height map or on point normals.

2 RELATED WORK

DNNs are very popular in the area of computer vision for object classification and semantic segmentation. Especially, neural networks are used for urban area classification with airborne laser scanning data based on height maps, see (Hu and Yuan, 2016; Yang et al., 2017). In this context Boulch et. al. (Boulch et al., 2017) used a neural network with encoder-decoder architecture for segmentation of colored, dense point clouds. They applied SegNet and U-Net for semantic labelling. In our context, 2D segmentation of roof regions can also be done on the previously described height maps using U-Net (Ronneberger et al., 2015), a convolutional DNN that has successfully been applied to many image segmentation tasks. The U-Net architecture consists of extraction and extension parts in which multi-channel feature maps are organized. In the extraction part, feature maps are connected to realize either convolutions or max pooling steps to decrease the size and increase the number of feature maps. The expansion part of the network is symmetric to the extraction part and realizes convolution steps, too. Here, instead of max pooling, upconvolutions are used to increase resolution. Additionally, high resolution feature maps of the extraction part directly contribute to upsampling steps. However, some of the above mentioned shortcomings of working with 2D height maps remain. Therefore, it seems to be a good idea working directly on 3D data.

Generally, CNNs require structured data. In the most simple case, one can achieve this by rasterization of point clouds and transfer into voxel representation. Unfortunately, this is memory and time consuming and might also imply the need of interpolation. On this account, in most cases the resolution of objects has to be decreased. A popular CNN for 3D applications is VoxNet (Maturana and Scherer, 2015).

An alternative approach to structuring point clouds is the use of indexing structures. Input data of the Kd-network are kd-trees that are computed from point clouds (Klokov and Lempitsky, 2017). Apart from point's coordinates, other point-wise attributes can be considered. In OctNet (Wang et al., 2017) the 3D points are represented with octrees. Typically, the hierarchy of octants is only sparsely filled. 3D CNN operations are performed only on octants occupied by boundaries of the 3D shapes under investigation. Due to using index structures, both Kd-network and OctNet are not rotation independent. In addition, generation of index structures might be time consuming. A further disadvantage ofthese methods is that new convolution and pooling operations are necessary.

In recent years, deep learning methods for irregular data have been researched. One prominent representative is PointNet (see (Qi et al., 2017a)). In its basic form only the coordinates of the points are used for the classification process. PointNet processes every point identically and independently in the first step with a T-Net. This network is a CNN with convolution layers, max pooling layers and two layers with fully connected neurons. As a result, we receive a transformed point cloud with uniform orientation and size, see Figure 1. In a next step, local features are computed for every point.

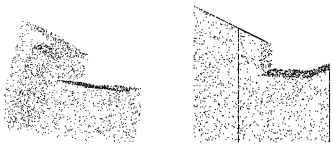

Figure 1: The T-Net transforms the left input point cloud to the aligned right point cloud.

In contrast to neighborhood relations, only global positions of points are important for recognition with PointNet. Since slopes of roofs and vertex normals typically are obtained using nearest neighboring points, this might be disadvantageous with respect to our application. PointNet's successor PointNet++ (Qi et al., 2017b) appears to be better suited for this task. PointNet++ is a multi-scale point-based network that considers neighborhood information by applying PointNet on nested partitioning of the input point set.

In (Rethage et al., 2018) a hybrid approach for a fully convolutional point network was chosen that is based on multi-scale feature encoding by the use of 3D convolutions in combination with direct processing of the point clouds. The network runs on unorganized input clouds and uses PointNet as a low-level feature descriptor. Internally, the input is transformed into an ordered representation. This transformation is followed by 3D convolutions to learn shape-dependent relationships of the points at multiple scales. With regard to semantic point segmentation, published results are slightly worse than for PointNet++.

In their work (Hua et al., 2018), Hua et. al proposed a new convolution operator for CNN, called point-wise convolution. This convolution operator can be applied at each point in a point cloud to learn point-wise features. Compared to other methods which used Tensorflow's optimized convolution operators, running time was slower. According to (Hua et al., 2018), per class segmentation accuracy was reported to be slightly worse than for PointNet.

In (Ben-Shabat et al., 2017) a 3D modified Fisher vector was used as hybrid representation of the point cloud. The 3D Fisher vector representation describes data samples from the point cloud in varying sizes by their deviation from a Gaussian Mixture Model. Published results obtained on a test data set were similar to those of PointNet.

Furthermore, 3D data is often represented as a mesh. Then a CNN for semantic segmentation can be applied to a graph derived from the mesh (Wang et al., 2018a). In this case, special pooling operations are necessary to coarsen the graph. We were not able to use this method because automated mesh generation failed for a significant number of buildings. Although some points should become connected, distances between them were probably too large due to outliers and shading effects.

Because of easy handling and good results in other studies we did 3D segmentation with PointNet and PointNet++. For comparison we also applied the U-Net and classical gradient segmentation on 2D height maps.

3 TRAINING DATA

DNNs require a huge amount of training data. We directly obtained ground truth by sampling point clouds from already existing 3D city models and received annotations at nearly no cost by mapping face normals to our six classes, `Wall`, `Flat Roof`, `North`, `South`, `East` and `West`.

We worked with a 3D model of four square kilometers of the center of the city of Krefeld that consists of more than 10,000 buildings. We converted each single CityGML building model into OBJ-format representation such that each triangle had a label (color) referring to its roof or wall polygon. Per definition, each building's ground plane lay in the x-y-plane. Additionally, we rotated the buildings representation in such way that the largest edge of the building's cadastral footprint pointed into the direction of x-axis of the coordinate system. We then scaled the model to fit within the cube $[-1,1] \times [-1,1] \times [-1,1]$. By allowing three different component-wise scaling factors, directions of normals did change slightly. This was tolerable because, as a preprocessing step to plane estimation, we still got useful segmentation into roof's gradient directions. If we did not scale to completely cover the interval $[-1,1]$ in all three dimensions then PointNet's preprocessing step performed geometric transformations that changed classification.

With the tool CloudCompare, we randomly sampled 2048 or 4096 labeled surface points for efficient processing of the data by PointNet. To obtain ground truth, we computed face normals of all wall and roof planes belonging to the transformed building model, see Figure 2. According to the normals, we classified the planes into the classes `Wall` (z-component of normal is zero), `Flat Roof` (x- and y- components of normals are (approximately) zero), `North` (roofs with slope to the north, i.e. in direction of the positive y-axis), `East` (positive x-axis), `South` (negative y-axis) and `West` (negative y-axis). According to the points' labels, we annotated them with their

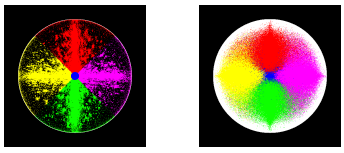

Figure 2: Distribution of face normals of roof and wall planes (left) and point normals obtained from the point neighbors (right): The normals were projected to the x-y-plane by removing their z-coordinate. Each dot represents the normal vector associated with a point of the cloud. Clearly, four major directions are visible, denoted as classes `North`, `East`, `South` and `West` in this paper. Normals of flat roofs were marked in blue, white pixels represent normals of walls.

corresponding class, see Figure 3. We ensure that all classes contain approximately the same number of points. PointNet++ allows for additional point-wise

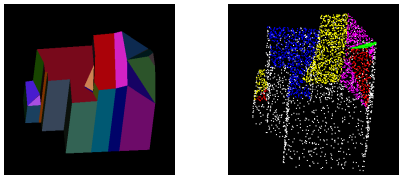

Figure 3: Left: 3D model of the building, each facet is marked with a different color. Right: Ground truth classification of a building's point cloud; Facets with different slope but same gradient direction are not distinct.

input data, and PointNet can be executed with derived data instead of original points. To this end, we equipped the points with point (vertex) normals computed from their nearest neighbors. Only for network training we additionally considered face normals. A point's face normal is the normal vector of the building's plane that is associated with the point Also, we generated multi-view density images by orthogonally projecting all points to the x-y-, x-z– and y-z-planes, see Figure 4. Such density maps were already used for object classification in (Minto et al., 2018). On each plane, we accumulated the number of points

Figure 4: Density image on the x-z-plane.

using a 256×256 raster. Thus, the data set contains 3D points, two (often different) corresponding normal vectors and also triples of density values according to a point's positions in the three raster images.

For the application of U-Net on height maps, we again turned each building so that the longest footprint edge matched the x-axis. Then, we initialized a greyscale height map image covering the buildings's footprint with height values taken from z-coordinates of the point cloud. We completely filled the image using linear interpolation, see greyscale pictures in Figure 5. This served as the net's input. To find corresponding ground truth segmentation, we used the face normals of triangles from the building model, which were projected into the x-y-plane. If a pixel was outside a building's footprint then we classified it as background.

Figure 5 shows examples for height maps and associated ground truth maps. The class `Wall` is not included in ground truth maps because walls are covered by roof points.

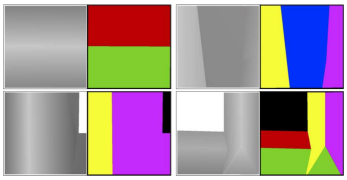

Figure 5: Height maps with ground truth annotation: blue regions represent flat roofs, black areas do not belong to the building, the other colors represent the classes `North`, `East`, `South` and `West`.

We used data derived from existing building models both for training and testing. Thus, we trained the network with 90% of sampled buildings and used 10% as test data. Because of computing time, we did not apply cross-validation. But we checked that there is no over-fitting. Also, the ratio 90% : 10% led to best recognition results. However, in order to evaluate networks' performance on real point clouds obtained by airborne laser scanning, we extracted the points of 21 buildings from such a cloud. Then we removed outliers because such points do not occur in training data. Finally, we manually classified the points.

4 EXPERIMENTS

To measure quality of results, we use the established intersect over union metric (IoU). For a given class and a given building, TP (true positive) is the number of points or pixels that are correctly identified as being members of this class: TN (true negative) is the number of input points that are correctly classified as not belonging to the given class. In turn, FP (false positive) and FN (false negative) are numbers of wrongly classified points. Then precision is defined via $\frac{TP+TN}{TP+TN+FP+FN}$ and $IoU := \frac{TP}{TP+FP+FN}$. The number TN does not occur in this definition because in general most points do belong to other classes (background) and one wants to avoid a good rating for networks that just classify background. To get a quality measure for a given class and all buildings of the test data set, we restrict ourselves to those buildings only, that include this class according to ground truth. For each of these buildings, we separately compute an IoU value and then determine the arithmetic mean over all such buildings. We do not set $IoU := 1$ if a class is not contained in a building's ground truth as it is done in PointNet. Thus, presented IoU-values might be smaller but more meaningful than measured with unchanged PointNet code. Because of computing IoU values separately for each building and not weighting with the building's class size, we amplify errors of small classes.

4.1 U-Net on Height Maps

We applied U-Net to 25,000 patches of 2,800 greyscale height maps with varying parameters. Each height map was an image with 492×492 pixel. All tests were performed using cross entropy error function. Nets might learn training data very well but fail with other input data. This effect is called overfitting. To avoid overfitting, we generally used regularization by adding a constant fraction of the absolute sum of the net's weights to the error function.

We selected the Adam optimizer (Kingma and Adam, 2015) as a statistical gradient descent method. We also tested with Adagrad (Duchi et al., 2011) and RMSProp but, as expected, the Adam method performed best by far. It reached high IoU values around 0.8 within 15 epochs in some initial tests. Simple gradient descent and Adagrad optimizer did not reach values above 0.3 after 200 epochs, and RMSProp tended towards a decrease in IoU after 100 epochs.

We worked with a learning rate of 0.0001, as smaller rates did not yield better IoU results after the same number of epochs. To the contrast, higher rates resulted in significantly smaller IoU numbers.

We initialized the nets with random weights following a Gaussian distribution. Thus it was desirable to get stable results independent of the initial values. By repeating experiments ten times under same conditions, we tested stability. Thereby, we observed two outliers with IoU values that were about 20% lower than the values obtained for the other eight experiments within the same magnitude, see Figure 6.

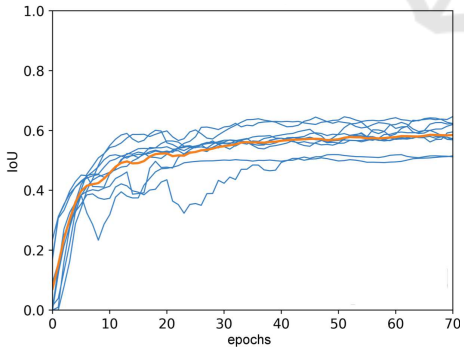

Figure 6: IoU values for learning class `Flat Roof` with U-Net: Blue curves belong to independent trainings with same net configuration, orange curve shows arithmetic mean.

Stability can be improved either by using the dropout method (that was applied in previously mentioned stability tests) or batch normalization. But both methods should not be used at the same time, cf. (Li et al., 2018). With stable networks, all further results were obtained using a single set of initial weights.

Table 1: IoU values after 200 training epochs: Numbers in the first column refer to the amount of activated neurons.

drop-out	`Flat Roof`	`North`	`East`	`South`	`West`	back-round	arith. mean
full							
90%	**66.8%**	**86.1%**	**90.2%**	**88.0%**	**93.9%**	**98.6%**	**87.3%**
80%	59.7%	80.3%	88.6%	81.0%	91.6%	98.5%	83.3%
70%	59.2%	76.3%	87.2%	76.9%	89.4%	98.5%	81.3%
half							
90%	61.5%	83.0%	90.0%	83.4%	92.3%	98.5%	84.8%
80%	56.3%	78.5%	88.3%	78.1%	90.5%	98.4%	81.7%
70%	52.0%	70.4%	84.6%	71,3%	86.5%	98.4%	77.2%
no	60.3%	82.2%	89.2%	81.5%	92.1%	98.4%	84.0%

By using the dropout method, a predefined number of randomly chosen neurons are deactivated. The desired effect is that results become more robust and not dependent on single neurons. We analyzed two different configurations. In the "full dropout" variant, dropout was applied to all convolution layers. The "half dropout" configuration applied dropout to every second convolution layer starting with the first. Table 1 shows that a 90% "full dropout" performed best, i.e., 10% of neurons were randomly deactivated.

The networks were trained with data grouped into batches. For each activation layer's input, batch normalization subtracts the batch's mean and divides the result by the batch's standard deviation. It also multiplies it with a trainable parameter γ and adds another trainable parameter β such that gradient descent can modify them instead of having to change many weights. As expected, with batch normalization, IoU values were higher for all tested learning rates than without use of batch normalization and dropout. With dropout, IoU numbers were alike, but variance was greater (cf. Figure 6), thus stability was worse.

The architecture of U-Net allows modification of two specific parameters: The number of top-layer feature maps and the size of the convolution kernel. Table 2 shows the results of four trainings that only differ in the number of feature maps. Due to running time, we decided to work with 16 feature maps. We

Table 2: Influence of the number of top-level feature maps.

number of feature maps	32	16	8	4
running time	23h	11.8h	5.2h	2.7h
median IoU	85%	83%	62%	48%

applied 3×3 and 5×5 kernels and found that, with using batch normalization, 5×5 kernel led to equally good IoU values as 3×3 kernel with 90% "full dropout" or with batch normalization. But due to the larger kernel size, less convolution layers were needed, and running time decreased by about 30%.

We compare U-Net results with following classical segmentation approach on the 2D greyscale height map in Table 3: On the height map, we computed gra-

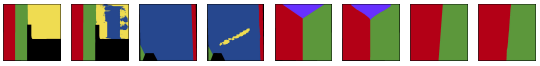

Figure 7: Ground truth and prediction: Black pixel are background, blue denotes `Flat Roof`.

dients via convolution with 3×3 or 5×5 Sobel kernel. Pixels belonging to gradients with length below a threshold value 0.1 were classified to be part of a flat roof. Otherwise, a gradient's direction in x-y-plane determines the class `North`, `East`, `South` or `West`. U-Net clearly leads to better segmentation results than thresholding of gradients computed with Sobel kernels. Figure 7 compares ground truth with network results. The image pairs in the upper row show quantization errors in areas with low slope.

Table 3: IoU values achieved with U-Net in comparison to classical segmentation.

	`Flat Roof`	`North`	`East`	`South`	`West`	arith. mean
U-Net	72.8	**94.6%**	**93.2%**	**94.5%**	**90.9%**	**89.2%**
3×3 Sobel	**84.4%**	77.4%	65.4%	73.8%	65.8%	73.4%
5×5 Sobel	80.6%	81.1%	72.2%	78.1%	72.4%	76.9%

4.2 PointNet

PointNet is designed only to handle point clouds in which each point consists of three coordinate values. We evaluated both, with the points' x- y- and z-coordinates but also with replacing the three values by the components of corresponding vertex normal vectors that are computed from the point's nearest neighbors. Face normals typically are not available as input and were only used to define ground truth.

To select parameter values, we trained the network with the points' coordinates. We chose the learning rate 0.001 that, with regard to IoU, performed as well as lower rates 0.0005 and 0.0001, whereas higher rates 0.01 and 0.005 resulted in significantly lower IoU values. The choice of using batch normalization improved results slightly.

Within PointNet, T-Net serves as a preprocessing step. This step can be followed by a similar T-Net-Feature network. We observed slight improvements in all tests when choosing to use both transformations. Probably, the improvement was only small because our input data was rotated according to the longest footprint edge. Hence, a suitable transformation was in place already.

With regard to stability, by repeating a training with random initial weights and batch normalization we did not observe outliers as we did for U-Net tests based on the dropout method. However, Table 4 and Figure 8 show that PointNet did not perform well. The number of wall points was higher than the number of points in other classes. Therefore, wall points were classified better.

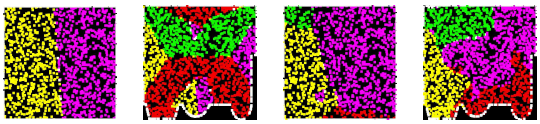

Figure 8: Two examples of ground truth segmentation (left) and prediction (right) with PointNet on x, y, z coordinates.

One would expect that results should improve if points were replaced with their corresponding point normal vectors. The direction of a normal vector immediately implies the class. Only differences between face normals (used to determine ground truth) and point normals should lead to errors. With exception of class `West`, PointNet applied to point normals indeed yields better but still not convincing results, see Table 4. Applying trained networks to manually classified real point clouds changed results for the worse.

4.3 PointNet++

In contrast to PointNet, its successor PointNet++ accepts six values per point as input. That allowed evaluating PointNet++' performance with two very different inputs: We used 3D points in connection with normal vectors as well as 3D points in combination with three projected density values, see Section 3. Since face normals can be generated only from 3D building models and not from point clouds directly, we tested our configurations with point normals. Anyhow, for training the networks with model data we can use either point normals or face normals detected on planes, see Figure 9. Table 5 summarizes corresponding results. All class sizes were chosen to be approximately equal for training but not for testing with real world data. It is not surprising that the network performs better, if training and test data matched, i.e., one should train with point normals. This also holds true for testing with real world data. However, segmentation results were a good deal poorer, see Table 6. Independent of chosen segmentation method, IoU values of walls were low. That is due to the fact that in airborne laser scanning data walls are mostly invisible. Thus, only 3% of points belong to walls.

PointNet++ trained on point normals and tested on model data even performed significantly better than outcomes of classical point normal-based segmentation using threshold values (Table 5). In this situation the network apparently learned thresholds more easily. In case of real world point clouds, the picture was not as clear (see Table 6). Reality and learning data seemed to differ too much, e.g., real roofs

Table 4: On the one hand, PointNet was trained and tested with points and on the other hand the network was trained and tested with point normals. For comparison, IoU values of classical segmentation are given in Table 5.

training variant	Wall	North	East	South	West	Flat Roof	arithmetic mean
x, y, z coordinates	69.4%	34.8%	31.8%	27.3%	**25.1%**	34.9%	37.2%
point normals	**92.5%**	**50.1%**	**33.7%**	**51.1%**	18.4%	**75.7%**	**53.6%**

Table 5: PointNet++ trained over 200 epochs and tested with points and normals of generated data as well as with density values from three points of view: To train the network with normals, we either used point or face normals obtained from 90% of 3D models. For testing on remaining 10% of models, points were equipped with point normals. For comparison, IoU values of segmentation on point normals (using same thresholds as for ground truth generation on face normals) are listed.

training variant	Wall	North	East	South	West	Flat Roof	mean
point normals	**96.3%**	84.3%	83.5%	84.1%	83.5%	84.6%	86.1%
face normals	88.1%	70.8%	74.6%	68.3%	73.8%	69.5%	74.2%
density data	95.1%	**85.4%**	**84.7%**	**85.9%**	**85.4%**	**85.9%**	**87.1%**
classical segmentation	88.1%	69.8%	76.6%	69.9%	76.6%	68.8%	74.9%

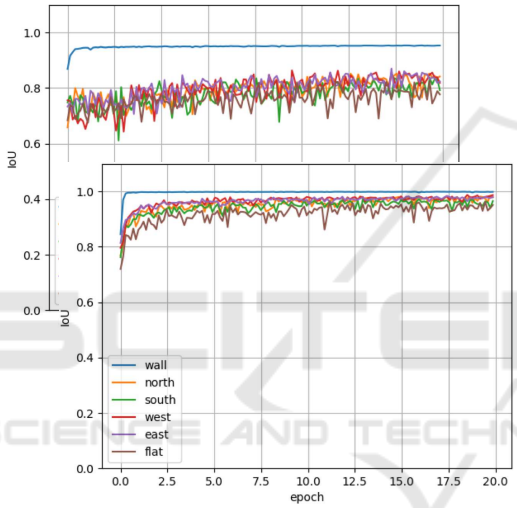

Figure 9: IoU values during training on points and normal vectors with PointNet++: Upper plots belong to training with point normals. Second plots summarize training with face normals.

were not exactly planar.

When applying PointNet++ with point coordinates and three corresponding density values instead of normals results were similar good than for training with point normals. Both variants were better than results for training with face normals, see Table 5. Ridge lines and step edges were clearly visible in density images. But surprisingly, segmentation results along these lines appeared noisier when using density values than point normals, see Figure 10.

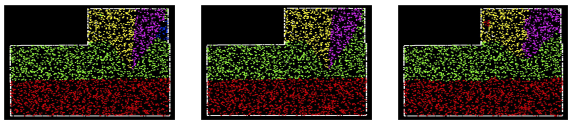

Figure 10: Difference between PointNet++ segmentation based on point normals and segmentation on density values: ground truth (left), point normal segmentation (middle), density value segmentation (right). To avoid error coming from e.g. small dormers higher resolution is required. This can be obtained by splitting up buildings into parts, cf. Figure 11.

5 CONCLUSION

Deep learning shows potential to improve existing algorithms for 3D building reconstruction. Both U-Net trained on height maps and, if trained on point normals or density images, PointNet++ are able to yield better segmentation results than we could obtain with classical gradient or normal based approaches. At least this holds true for testing with point clouds that are sampled from 3D city models. For application to real world point clouds, we have to improve our training data. Recently, we trained with annotated real 3D data and got similar results as on training data described in this paper.

In our tests, 3D networks did not perform better than 2D U-Net. Again, the main reason might be that our training data for roofs can be represented in 2.5D.

Most non-flat roofs possess only four main gradient directions. Therefore, classification into flat roofs and four directions generally serves well. However, in future work we will increase the number of classes to improve segmentation results for sophisticated roof topologies. To this end, we also have to split up buildings into small processable parts instead of using non-uniform scaling. This also solves the problem that our currently used networks only accept quite small point clouds due to memory restrictions. As tested with PointNet++, segmentation results of parts can be merged without significant loss of quality, see Figure 11. Suitable strategies for split-up procedures have to be developed.

Table 6: PointNet++ tested with real world point clouds: To train the network, we either used point or face normals obtained from 90% of 3D models as in Table 5. For testing, we used real point clouds equipped with point normals. Results were compared with classical threshold segmentation as in Table 5.

training variant	Wall	North	East	South	West	Flat Roof	mean
point normals	**65.2%**	**64.3%**	52.0%	70.7%	56.8%	**55.7%**	**60.8%**
face normals	53.3%	56.2%	57.0%	**73.2%**	57.1%	47.1%	57.3%
classical segmentation	52.1%	60.3%	**59.5%**	71.6%	**59.4%**	44.2%	57.9%

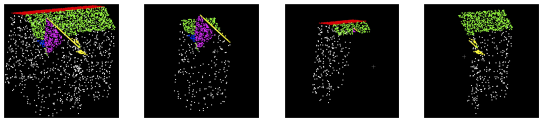

Figure 11: Stable segmentation results on cloud subsets.

ACKNOWLEDGEMENT

This work was supported by a generous hardware grant from NVIDIA.

REFERENCES

Ben-Shabat, Y., Lindenbaum, M., and Fischer, A. (2017). 3D point cloud classification and segmentation using 3D modified fisher vector representation for convolutional neural networks. *arXiv preprint arXiv:1711.08241*.

Boulch, A., Le Saux, B., and Audebert, N. (2017). Unstructured point cloud semantic labeling using deep segmentation networks. In *3DOR*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:12:2121–2159.

Goebbels, S. and Pohle-Fröhlich, R. (2016). Roof reconstruction from airborne laser scanning data based on image processing methods. *ISPRS Ann. Photogramm. Remote Sens. and Spatial Inf. Sci.*, III-3:407–414.

Hu, X. and Yuan, Y. (2016). Deep-learning-based classification for dtm extraction from als point cloud. *Remote sensing*, 8(9):730.

Hua, B.-S., Tran, M.-K., and Yeung, S.-K. (2018). Pointwise convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 984–993.

Kingma, D. and Adam, J. B. (2015). A method for stochastic optimization. In *International Conference on Learning Representations*, pages 1–15, San Diego, CA.

Klokov, R. and Lempitsky, V. S. (2017). Escape from cells: Deep kd-networks for the recognition of 3D point cloud models. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 863–872.

Li, X., Chen, S., Hu, X., and Yang, J. (2018). Understanding the disharmony between dropout and batch normalization by variance shift. *CoRR*, abs/1801.05134.

Maturana, D. and Scherer, S. (2015). Voxnet: A 3D convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE.

Minto, L., Zanuttigh, P., and Pagnutti, G. (2018). Deep learning for 3d shape classification based on volumetric density and surface approximation clues. In *VISIGRAPP (5: VISAPP)*, pages 317–324.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85.

Qi, C. R., Yi, L., Su, H., and Guibas, L. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*.

Rethage, D., Wald, J., Sturm, J., Navab, N., and Tombari, F. (2018). Fully-convolutional point networks for large-scale point clouds. *arXiv preprint arXiv:1808.06840*.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597.

Wang, P., Gan, Y., Shui, P., Yu, F., Zhang, Y., Chen, S., and Sun, Z. (2018a). 3D shape segmentation via shape fully convolutional networks. *Computers & Graphics*, 70:128–139.

Wang, P.-S., Liu, Y., Guo, Y.-X., Sun, C.-Y., and Tong, X. (2017). O-CNN: Octree-based convolutional neural networks for 3D shape analysis. *ACM Transactions on Graphics (TOG)*, 36(4):72:1–72:11.

Wang, R., Peethambaran, J., and Dong, C. (2018b). LiDAR point clouds to 3D urban models: A review. *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, 11(2):606–627.

Yang, Z., Jiang, W., Xu, B., Zhu, Q., Jiang, S., and Huang, W. (2017). A convolutional neural network-based 3D semantic labeling method for als point clouds. *Remote Sensing*, 9(9):936.