

000
001
002
003054
055
056
057

Supplementary Material for OctNet: Learning Deep 3D Representations at High Resolutions

004
005
006
007
008
009
010
011
012
013058
059
060
061
062
063
064
065
066
067

Anonymous CVPR submission

014
015068
069

Paper ID 1319

016
017
018
019
020
021070
071
072
073
074
075

1. Introduction

In the following supplemental material we present details regarding our operations on the hybrid grid-octree data structure, additional experimental results and all details on the used network architectures. In Section 2 we provide details on the data index used to efficiently access data in the grid-octree data structure. Section 3 yields more insights into the efficient implementation of the convolution operation on octrees. We present further quantitative and qualitative results in Section 4 and in Section 5 we specify all details of the network architectures used in our experiments.

022

076

2. Data Index

023
024
025
026077
078
079
080

An important implementation detail of the hybrid grid-octree data structure is the memory alignment for fast data access. We store all data associated with the leaf nodes of a shallow octree in a compact contiguous array. Thus, we need a fast way to compute the offset in this data array for any given voxel. In the main text we presented the following equation:

$$\text{data_idx}(i) = 8 \underbrace{\sum_{j=0}^{\text{pa}(i)-1} \text{bit}(j) + 1}_{\#\text{nodes above i}} - \underbrace{\sum_{j=0}^{i-1} \text{bit}(j)}_{\#\text{split nodes pre i}} + \underbrace{\text{mod}(i-1, 8)}_{\text{offset}}. \quad (1)$$

032
033
034
035
036
037086
087
088
089
090
091

As explained in the main text the whole octree structure is stored as a bit-string and a voxel is uniquely identified by the bit index i , i.e., the index within the bit string. The data is aligned breadth-first and only the leaf nodes have data associated. Consequently, the first part of the equation above counts the number of split and leaf nodes up to the voxel with bit index i . The second term subtracts the number of split nodes before the particular voxel as data is only associated with leaf nodes. Finally, we need to get the offset within the voxel's neighborhood. This is done by the last term of the equation.

038
039092
093

Let us illustrate this with a simple example: For ease of visualization we will consider a quadtree. Hence, each voxel can be split into 4 instead of 8 children. The equation for the offset changes to

$$\text{data_idx}_4(i) = 4 \underbrace{\sum_{j=0}^{\text{pa}_4(i)-1} \text{bit}(j) + 1}_{\#\text{nodes above i}} - \underbrace{\sum_{j=0}^{i-1} \text{bit}(j)}_{\#\text{split nodes pre i}} + \underbrace{\text{mod}(i-1, 4)}_{\text{offset}}, \quad (2)$$

045
046099
100

with

047
048
049
050
051
052
053101
102
103
104
105
106
107

$$\text{pa}_4(i) = \left\lfloor \frac{i-1}{4} \right\rfloor. \quad (3)$$

Now consider the following bit string for instance: 1 0101 0000 1001 0000 0100. According to our definition, this bit string corresponds to the tree structure visualized in Fig. 1a and 1b, where s indicates a split node and v a leaf node with associated data. In Fig. 1c we show the bit indices for all nodes. Note that the leaf nodes at depth 3 do not need to be stored in the bit string as this information is implicit. Finally, the data index for all leaf nodes is visualized in Fig. 1d. Now we can verify

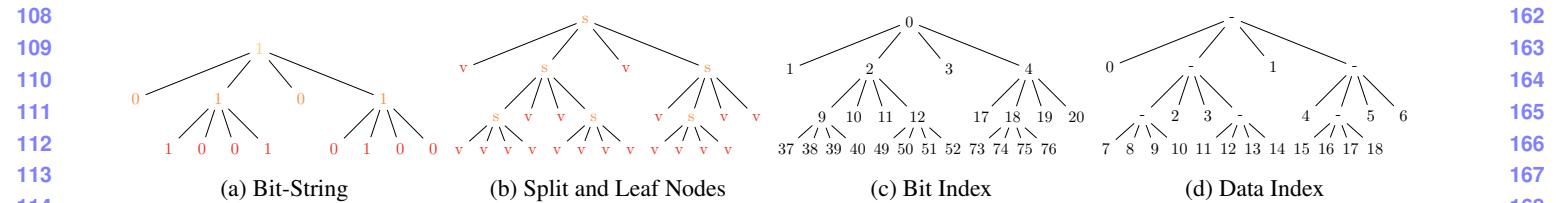


Figure 1: Data Index.

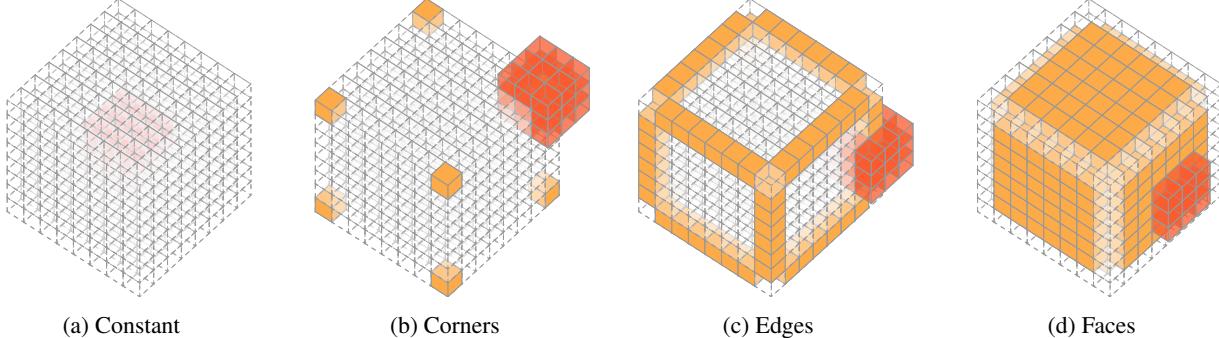


Figure 2: Efficient Convolution.

equation (2) using a simple example. Assume the bit index 51: The parent bit index is given by equation (3) as 12. To compute the data index we first count the number of nodes before 49 as it is the first node within its siblings (first term of equation), which is 17. Next, we count the number of split nodes up to 49 (second term of equation), which is 6. Finally, we look up the position 51 within its siblings (last term of equation), which is 2. Combining those three terms yields the data index $17 - 6 + 2 = 13$.

3. Efficient Convolution

In the main text we discussed that the convolution for larger octree cells and small convolution kernels can be efficiently implemented. A naïve implementation applies the convolution kernel at every location (i, j, k) comprised by the cell $\Omega[i, j, k]$. Therefore, for an octree cell of size 8^3 and a convolution kernel of 3^3 this would require $8^3 \cdot 3^3 = 13,824$ multiplications. However, we can implement this calculation much more efficiently as depicted in Fig. 2. We observe that the value inside the cell of size 8^3 is constant. Thus, we only need to evaluate the convolution once inside this cell and multiply the result with the size of the cell 8^3 , see Fig. 2a. Additionally, we only need to evaluate truncated versions of the kernel on the corners, edges and faces of the voxel, see Fig. 2b-d. This implementation is more efficient, as we need only 27 multiplications for the constant part, 8 · 19 multiplications for the corners, 12 · 6 · 15 multiplications for the edges, and 6 · 6² · 9 multiplications for the faces of the voxel. In total this yields 3203 multiplications, or 23.17% of the multiplications required by the naïve implementation.

4. Additional Results

In this Section we show additional quantitative and qualitative results for 3D shape classification, 3D orientation estimation and semantic 3D point labeling.

4.1. 3D Classification

In the main text of our work we analyzed the runtime and memory consumption of OctNet compared with the equivalent dense networks on ModelNet10 [3]. Additionally, we demonstrated that without further data augmentation, ensemble learning, or more sophisticated architectures the accuracy saturates at an input resolution of about 16^3 , when keeping the number of network parameters fixed across all resolutions. In this Section we show the same experiment on ModelNet40 [3]. The results are summarized in Fig. 3. In contrast to ModelNet10, we see an increase in accuracy up to an input resolution of 32^3 . Beyond this resolution the classification performance does not further improve. Note that the only form of data augmentation

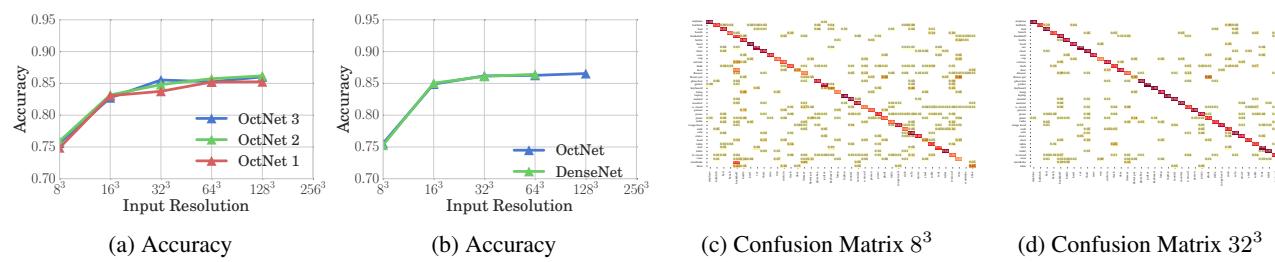


Figure 3: ModelNet40 results.

we used in this experiment was rotation around the up-vector as the 3D models in this dataset vary in pose. We conclude that object classification on the ModelNet40 dataset is more challenging than on the ModelNet10 dataset, but both datasets are relatively easy in the sense that details do not matter as much as in the datasets used for our other experiments.

4.2. 3D Orientation Estimation

To demonstrate that OctNet can handle input resolutions larger than 256^3 we added results for the 3D orientation estimation experiment with an input resolution of 512^3 . The results are presented in Fig. 4. As for the orientation experiment in the main paper, we can observe the trend that performance increases with increasing input resolution.

We evaluated our OctNet also on the Biwi Kinect Head Pose Database [?] as an additional experiment on 3D pose estimation. The dataset consists of 24 sequences of 20 individuals sitting in front of a Kinect depth sensor. For each frame the head center and the head pose in terms of its 3D rotation is annotated. We split the dataset into a training set of 18 individuals for training and 2 individuals for testing and project the depth map to 3D points with the given camera parameters. We then create the hybrid grid-octree structure from the 3D points that belong to the head (In this experiment we are only interested in 3D orientation estimation, as the head can be reliably detected in the color images). As in the previous experiment we parameterize the orientation with unit quaternions and train our OctNet using the same settings as in the previous 3D orientation estimation experiment. Fig. 5 shows the quantitative results over varying input resolutions. We see a reasonable improvement of accuracy from 8^3 up to 64^3 . Beyond this input resolution the octree resolution becomes finer than the resolution of the 3D point cloud. Thus, further improvements can not be expected. In Fig. 6 we show some qualitative results.

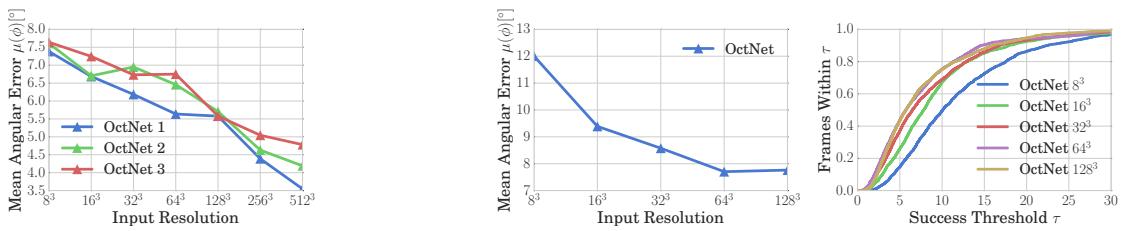


Figure 4: Additional Chair Orientation Results.

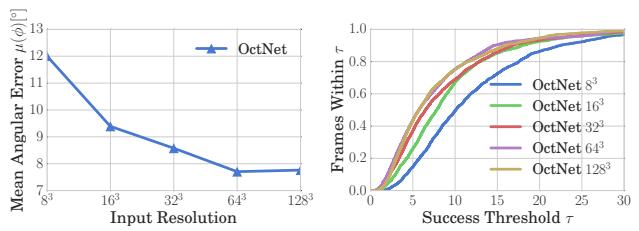


Figure 5: Head Pose Results.

4.3. 3D Semantic Segmentation

In this Section we present additional qualitative results of the semantic 3D point labeling task in Fig. 7 to 11. In these visualizations we show the color part of the voxelized input, the result of the labeling in the voxel representation, and the result back-projected to the 3D point cloud for different houses in the test set of [2].

5. Network Architecture Details

In this Section we detail the network architectures used throughout our experimental evaluations. We use the following notation for brevity: $\text{conv}(x, y)$ denotes a 3^3 convolutional layer with x input feature maps and y output feature maps. Similarly, $\text{maxpool}(f)$ is a max-pooling operation that decreases dimensionality by a factor of f along each axis. All convolutional and fully-connected layers, except the very last one, are followed by a ReLU as activation function.

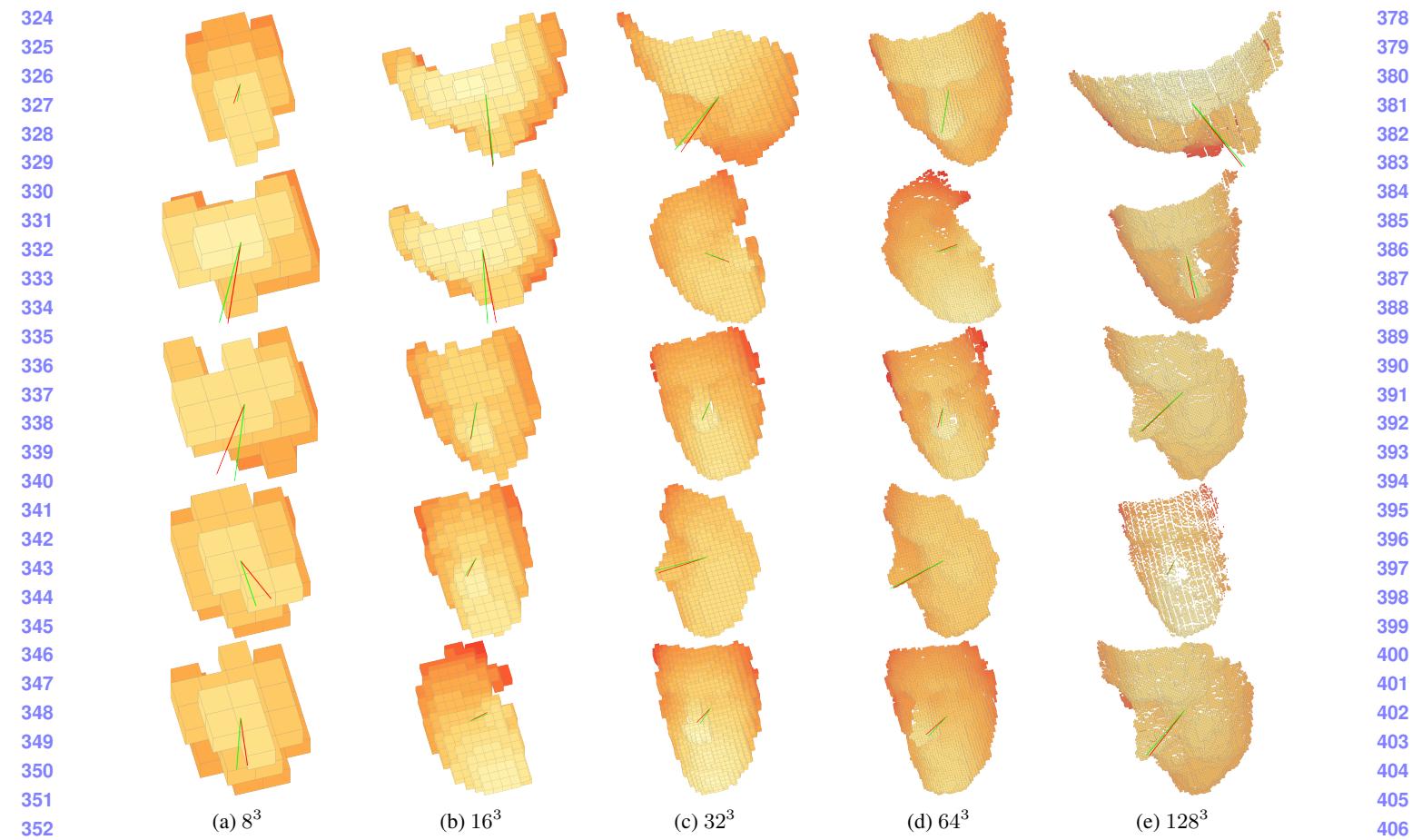


Figure 6: Qualitative Results for Head Pose Estimation.

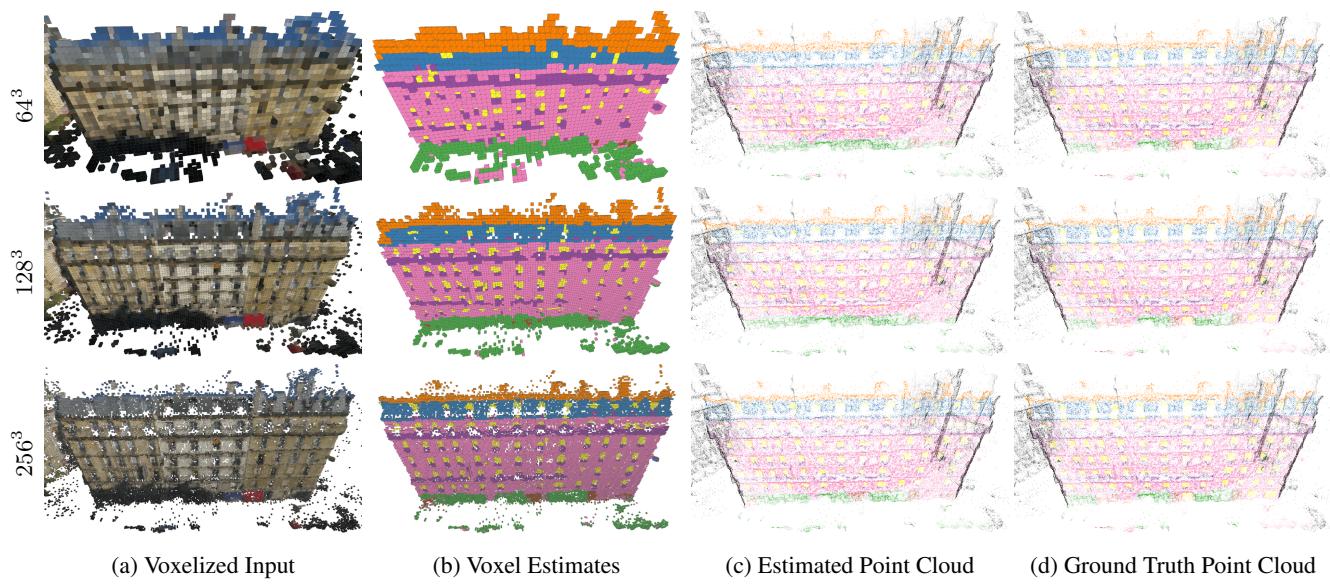
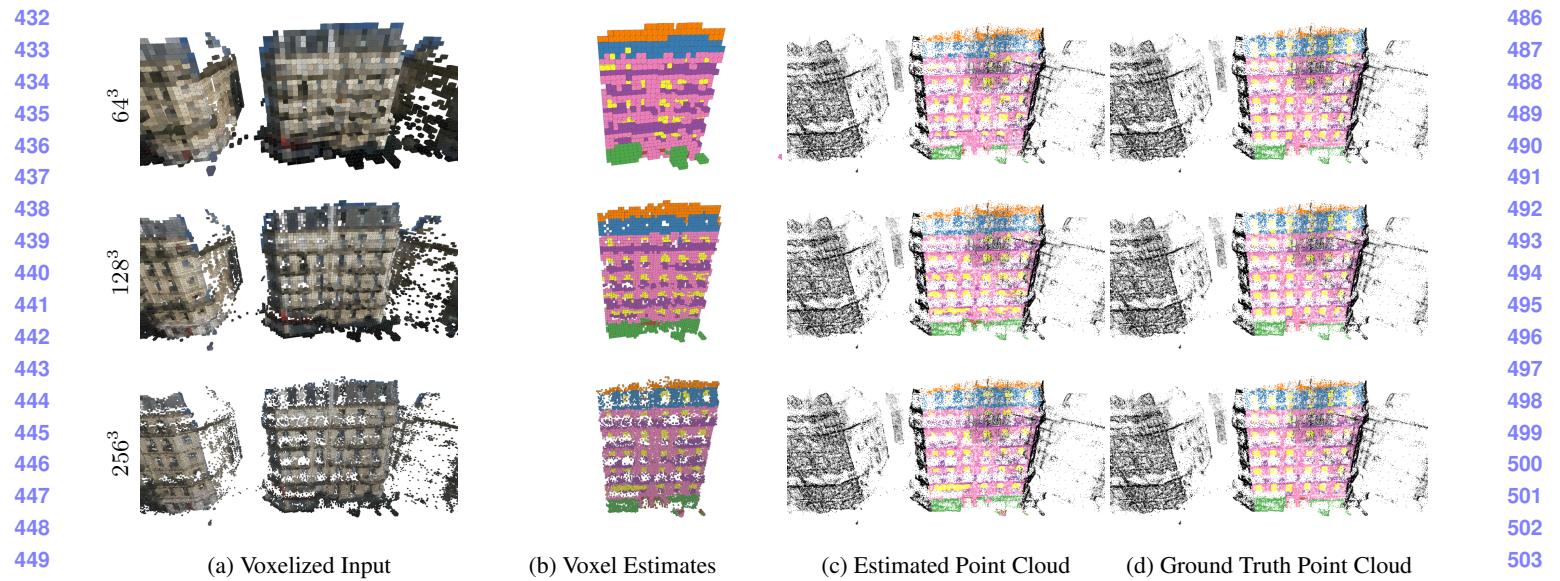
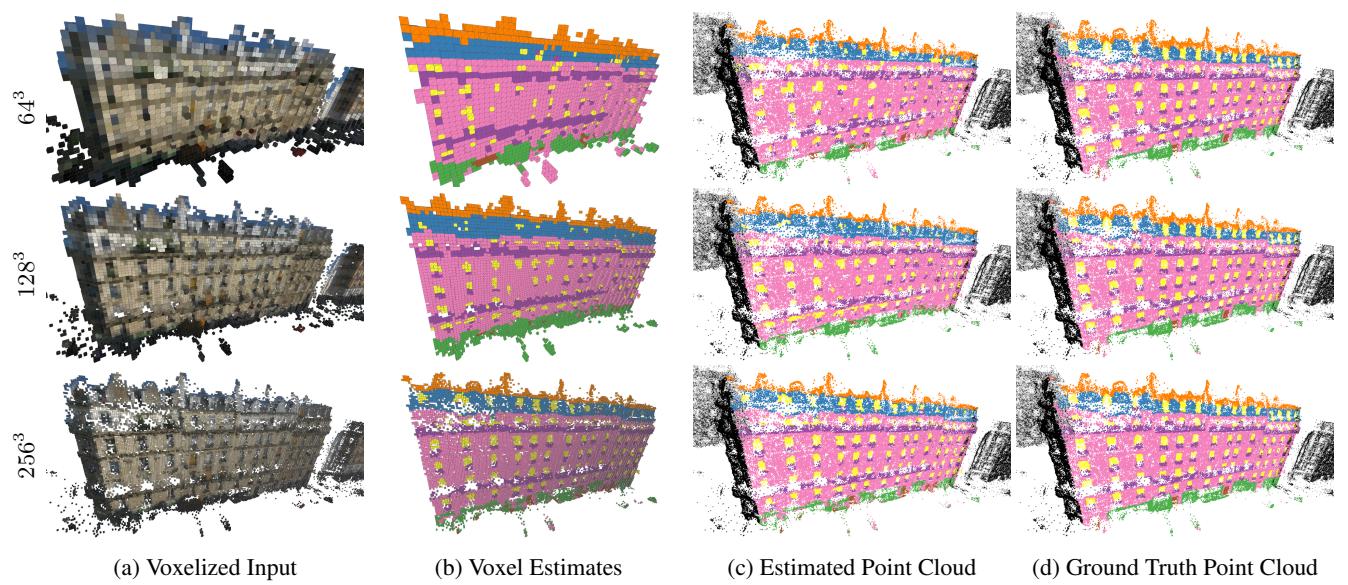


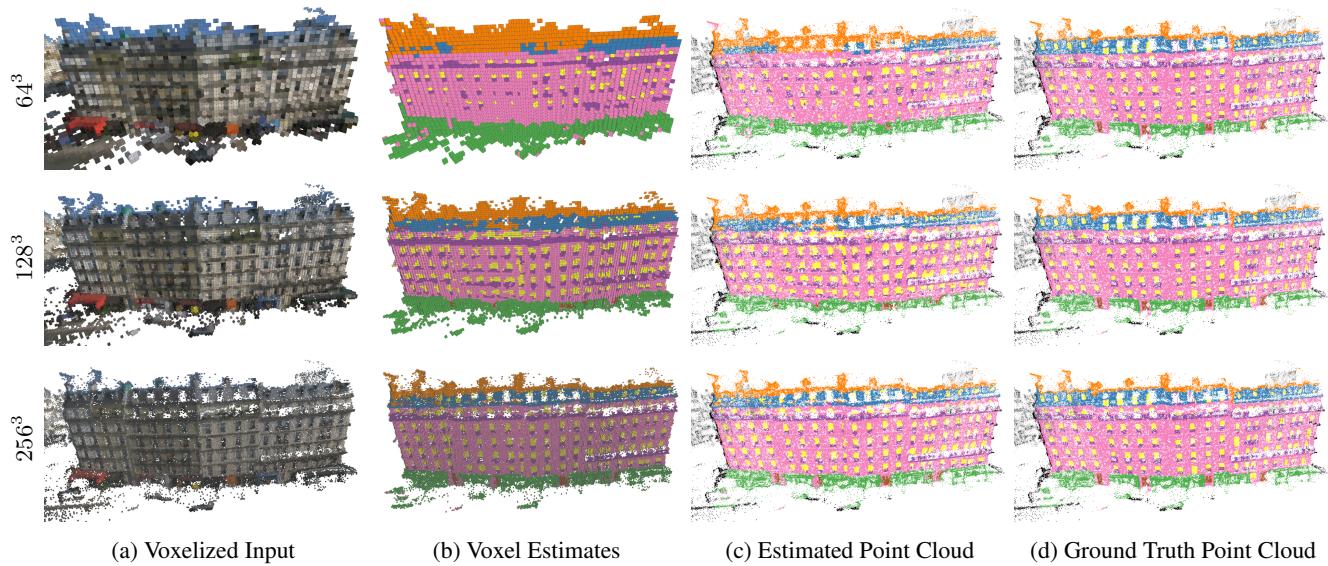
Figure 7: Facade Labeling Results. Zoom in for details.

Figure 8: **Facade Labeling Results.** Zoom in for details.Figure 9: **Facade Labeling Results.** Zoom in for details.

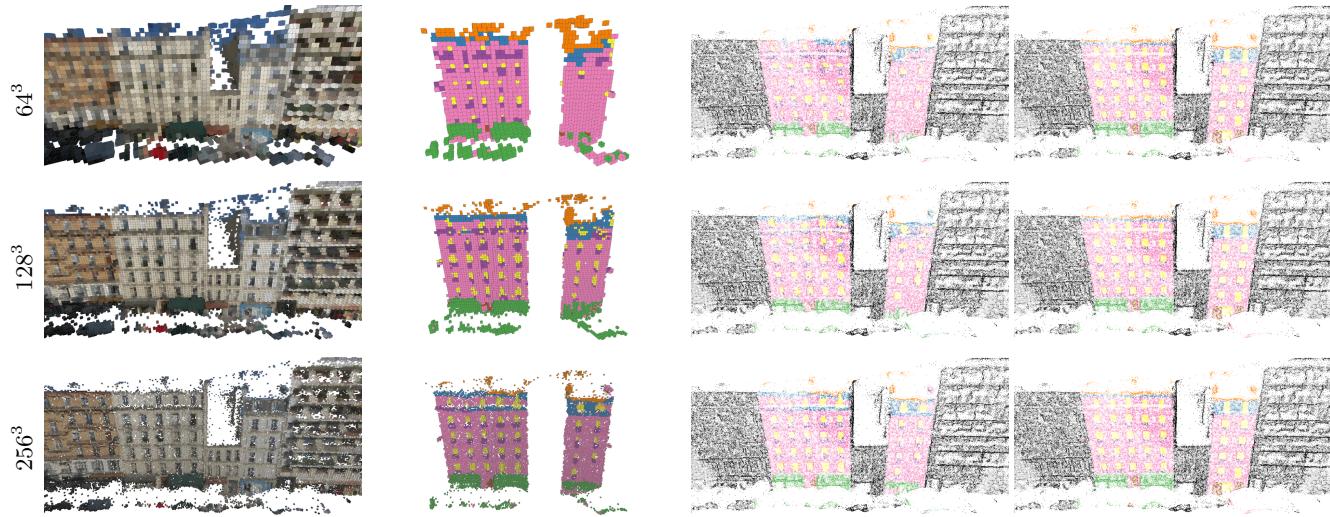
In the first two experiments, 3D classification and 3D orientation estimation, we show two different classes of architectures. In the first one we keep the number of convolution layers per block fixed and add blocks depending on the input resolution of the network. We call those networks OctNet1, OctNet2, and OctNet3, depending on the number of convolution layers per block. Therefore, the number of parameters increases along with the input resolution. The detailed architectures are depicted in Table 1, 2, and 3 for the classification task and in Table 5, 6, and 7 for the orientation estimation tasks, respectively. Second, we trained network architectures where we keep the number of parameters fixed, independently of the input resolution. The detailed network architectures for those experiments are presented in Table 4 and 8.

Finally, for semantic 3D point labeling, we use the U-Net type architecture [1,4] shown in Table 9. We use a concatenation layer $\text{concat}(\cdot, \cdot)$ to combine the outputs from the decoder and encoder parts of the networks to preserve details.

CVPR 2017 Submission #1319. CONFIDENTIAL REVIEW COPY. DO NOT DISTRIBUTE.



(a) Voxelized Input (b) Voxel Estimates (c) Estimated Point Cloud (d) Ground Truth Point Cloud

Figure 10: **Facade Labeling Results.** Zoom in for details.

(a) Voxelized Input (b) Voxel Estimates (c) Estimated Point Cloud (d) Ground Truth Point Cloud

Figure 11: **Facade Labeling Results.** Zoom in for details.

648						702	
649						703	
650						704	
651						705	
652	8³	16³	32³	64³	128³	256³	
653	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	706
654	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	707
655		conv(8, 16)	conv(8, 16)	conv(8, 16)	conv(8, 16)	conv(8, 16)	708
656		maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	709
657			conv(16, 24)	conv(16, 24)	conv(16, 24)	conv(16, 24)	710
658			maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	711
659				conv(24, 32)	conv(24, 32)	conv(24, 32)	712
660				maxpool(2)	maxpool(2)	maxpool(2)	713
661					conv(32, 40)	conv(32, 40)	714
662					maxpool(2)	maxpool(2)	715
663						conv(40, 48)	716
664							717
665						Dropout(0.5)	
666						fully-connected(1024)	718
667						fully-connected(10)	719
668						SoftMax	720

Table 1: Network Architectures ModelNet10 Classification: OctNet1

669						721	
670						722	
671						723	
672						724	
673						725	
674						726	
675						727	
676	8³	16³	32³	64³	128³	256³	728
677	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	729
678	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	730
679	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	731
680		conv(8, 16)	conv(8, 16)	conv(8, 16)	conv(8, 16)	conv(8, 16)	732
681		conv(16, 16)	conv(16, 16)	conv(16, 16)	conv(16, 16)	conv(16, 16)	733
682		maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	734
683			conv(16, 24)	conv(16, 24)	conv(16, 24)	conv(16, 24)	735
684			conv(24, 24)	conv(24, 24)	conv(24, 24)	conv(24, 24)	736
685			maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	737
686				conv(24, 32)	conv(24, 32)	conv(24, 32)	738
687				conv(32, 32)	conv(32, 32)	conv(32, 32)	739
688				maxpool(2)	maxpool(2)	maxpool(2)	740
689					conv(32, 40)	conv(32, 40)	741
690					conv(40, 40)	conv(40, 40)	742
691					maxpool(2)	maxpool(2)	743
692						conv(40, 48)	744
693						conv(48, 48)	745
694							746
695						Dropout(0.5)	747
696						fully-connected(1024)	748
697						fully-connected(10)	749
698						SoftMax	750

Table 2: Network Architectures ModelNet10 Classification: OctNet2

699						751
700						752
701						753
702						754
703						755

756	8 ³	16 ³	32 ³	64 ³	128 ³	256 ³	810
757	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	811
758	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	812
759	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	813
760	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	814
761							815
762	conv(8, 16)	conv(8, 16)	conv(8, 16)	conv(8, 16)	conv(8, 16)	conv(8, 16)	816
763	conv(16, 16)	conv(16, 16)	conv(16, 16)	conv(16, 16)	conv(16, 16)	conv(16, 16)	817
764	conv(16, 16)	conv(16, 16)	conv(16, 16)	conv(16, 16)	conv(16, 16)	conv(16, 16)	818
765	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	819
766							820
767	conv(16, 24)	conv(16, 24)	conv(16, 24)	conv(16, 24)	conv(16, 24)	conv(16, 24)	821
768	conv(24, 24)	conv(24, 24)	conv(24, 24)	conv(24, 24)	conv(24, 24)	conv(24, 24)	822
769	conv(24, 24)	conv(24, 24)	conv(24, 24)	conv(24, 24)	conv(24, 24)	conv(24, 24)	823
770	maxpool(2)						824
771							825
772							826
773							827
774							828
775							829
776							830
777							831
778							832
779							833
780	Dropout(0.5)						834
781	fully-connected(1024)						835
782	fully-connected(10)						836
783	SoftMax						837

Table 3: Network Architectures ModelNet10 Classification: OctNet3.

784	8 ³	16 ³	32 ³	64 ³	128 ³	256 ³	838
785	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	839
786	conv(8, 14)	conv(8, 14)	conv(8, 14)	conv(8, 14)	conv(8, 14)	conv(8, 14)	840
787	maxpool(2)						841
788							842
789	conv(14, 14)	conv(14, 14)	conv(14, 14)	conv(14, 14)	conv(14, 14)	conv(14, 14)	843
790	conv(14, 20)	conv(14, 20)	conv(14, 20)	conv(14, 20)	conv(14, 20)	conv(14, 20)	844
791							845
792	maxpool(2)						846
793							847
794							848
795	conv(20, 20)	conv(20, 20)	conv(20, 20)	conv(20, 20)	conv(20, 20)	conv(20, 20)	849
796	conv(20, 26)	conv(20, 26)	conv(20, 26)	conv(20, 26)	conv(20, 26)	conv(20, 26)	850
797							851
798	maxpool(2)						852
799							853
800	conv(26, 26)	conv(26, 26)	conv(26, 26)	conv(26, 26)	conv(26, 26)	conv(26, 26)	854
801	conv(26, 32)	conv(26, 32)	conv(26, 32)	conv(26, 32)	conv(26, 32)	conv(26, 32)	855
802	maxpool(2)						856
803							857
804	Dropout(0.5)						858
805	fully-connected(512)						859
806	fully-connected(10)						860
807	SoftMax						861
808							862
809							863

Table 4: Network Architectures ModelNet10 Classification.

CVPR 2017 Submission #1319. CONFIDENTIAL REVIEW COPY. DO NOT DISTRIBUTE.

864						918	
865						919	
866						920	
867						921	
868	8³	16³	32³	64³	128³	256³	
869	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	922
870	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	923
871		conv(8, 16)	conv(8, 16)	conv(8, 16)	conv(8, 16)	conv(8, 16)	924
872		maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	925
873			conv(16, 24)	conv(16, 24)	conv(16, 24)	conv(16, 24)	926
874			maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	927
875				conv(24, 32)	conv(24, 32)	conv(24, 32)	928
876				maxpool(2)	maxpool(2)	maxpool(2)	929
877					conv(32, 40)	conv(32, 40)	930
878					maxpool(2)	maxpool(2)	931
879						conv(40, 48)	932
880							933
881						Dropout(0.5)	
882						fully-connected(1024)	934
883						fully-connected(4)	935
884						Normalize	936
885							937
886							938
887							939
888							940
889							941
890							942
891							943
892							944
893							945
894	8³	16³	32³	64³	128³	256³	
895	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	946
896	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	947
897	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	948
898		conv(8, 16)	conv(8, 16)	conv(8, 16)	conv(8, 16)	conv(8, 16)	949
899		conv(16, 16)	conv(16, 16)	conv(16, 16)	conv(16, 16)	conv(16, 16)	950
900		maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	951
901			conv(16, 24)	conv(16, 24)	conv(16, 24)	conv(16, 24)	952
902			conv(24, 24)	conv(24, 24)	conv(24, 24)	conv(24, 24)	953
903			maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	954
904				conv(24, 32)	conv(24, 32)	conv(24, 32)	955
905				conv(32, 32)	conv(32, 32)	conv(32, 32)	956
906				maxpool(2)	maxpool(2)	maxpool(2)	957
907					conv(32, 40)	conv(32, 40)	958
908					conv(40, 40)	conv(40, 40)	959
909					maxpool(2)	maxpool(2)	960
910						conv(40, 48)	961
911						conv(48, 48)	962
912							963
913							964
914							965
915							966
916							967
917							968

Table 5: Network Architectures Orientation Estimation: OctNet1

892							946
893							947
894							948
895							949
896							950
897							951
898							952
899							953
900							954
901							955
902							956
903							957
904							958
905							959
906							960
907							961
908							962
909							963
910							964
911							965
912							966
913							967
914							968
915							969
916							970
917							971

Table 6: Network Architectures Orientation Estimation: OctNet2

972	8 ³	16 ³	32 ³	64 ³	128 ³	256 ³	1026
973	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	1027
974	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	1028
975	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	conv(8, 8)	1029
976	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	1030
977							1031
978		conv(8, 16)	conv(8, 16)	conv(8, 16)	conv(8, 16)	conv(8, 16)	1032
979		conv(16, 16)	conv(16, 16)	conv(16, 16)	conv(16, 16)	conv(16, 16)	1033
980		conv(16, 16)	conv(16, 16)	conv(16, 16)	conv(16, 16)	conv(16, 16)	1034
981		maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	1035
982			conv(16, 24)	conv(16, 24)	conv(16, 24)	conv(16, 24)	1036
983			conv(24, 24)	conv(24, 24)	conv(24, 24)	conv(24, 24)	1037
984			conv(24, 24)	conv(24, 24)	conv(24, 24)	conv(24, 24)	1038
985			maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	1039
986				conv(24, 32)	conv(24, 32)	conv(24, 32)	1040
987				conv(32, 32)	conv(32, 32)	conv(32, 32)	1041
988				conv(32, 32)	conv(32, 32)	conv(32, 32)	1042
989				maxpool(2)	maxpool(2)	maxpool(2)	1043
990					conv(32, 40)	conv(32, 40)	1044
991					conv(40, 40)	conv(40, 40)	1045
992					conv(40, 40)	conv(40, 40)	1046
993					maxpool(2)	maxpool(2)	1047
994						conv(40, 48)	1048
995						conv(48, 48)	1049
996						conv(48, 48)	1050
997							1051
998							1052
999							1053
1000							1054
1001							1055
1002							1056
1003							1057
1004	8 ³	16 ³	32 ³	64 ³	128 ³	256 ³	1058
1005	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	1059
1006	conv(8, 14)	conv(8, 14)	conv(8, 14)	conv(8, 14)	conv(8, 14)	conv(8, 14)	1060
1007						maxpool(2)	1061
1008		conv(14, 14)	conv(14, 14)	conv(14, 14)	conv(14, 14)	conv(14, 14)	1062
1009		conv(14, 20)	conv(14, 20)	conv(14, 20)	conv(14, 20)	conv(14, 20)	1063
1010						maxpool(2)	1064
1011		conv(20, 20)	conv(20, 20)	conv(20, 20)	conv(20, 20)	conv(20, 20)	1065
1012		conv(20, 26)	conv(20, 26)	conv(20, 26)	conv(20, 26)	conv(20, 26)	1066
1013						maxpool(2)	1067
1014		conv(26, 26)	conv(26, 26)	conv(26, 26)	conv(26, 26)	conv(26, 26)	1068
1015		conv(26, 32)	conv(26, 32)	conv(26, 32)	conv(26, 32)	conv(26, 32)	1069
1016						maxpool(2)	1070
1017		conv(32, 32)	conv(32, 32)	conv(32, 32)	conv(32, 32)	conv(32, 32)	1071
1018		conv(32, 32)	conv(32, 32)	conv(32, 32)	conv(32, 32)	conv(32, 32)	1072
1019		maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	1073
1020							1074
1021							1075
1022							1076
1023							1077
1024							1078
1025							1079

Table 7: Network Architectures Orientation Estimation: OctNet3.

1004	8 ³	16 ³	32 ³	64 ³	128 ³	256 ³	1058
1005	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	conv(1, 8)	1059
1006	conv(8, 14)	conv(8, 14)	conv(8, 14)	conv(8, 14)	conv(8, 14)	conv(8, 14)	1060
1007						maxpool(2)	1061
1008		conv(14, 14)	conv(14, 14)	conv(14, 14)	conv(14, 14)	conv(14, 14)	1062
1009		conv(14, 20)	conv(14, 20)	conv(14, 20)	conv(14, 20)	conv(14, 20)	1063
1010						maxpool(2)	1064
1011		conv(20, 20)	conv(20, 20)	conv(20, 20)	conv(20, 20)	conv(20, 20)	1065
1012		conv(20, 26)	conv(20, 26)	conv(20, 26)	conv(20, 26)	conv(20, 26)	1066
1013						maxpool(2)	1067
1014		conv(26, 26)	conv(26, 26)	conv(26, 26)	conv(26, 26)	conv(26, 26)	1068
1015		conv(26, 32)	conv(26, 32)	conv(26, 32)	conv(26, 32)	conv(26, 32)	1069
1016						maxpool(2)	1070
1017		conv(32, 32)	conv(32, 32)	conv(32, 32)	conv(32, 32)	conv(32, 32)	1071
1018		conv(32, 32)	conv(32, 32)	conv(32, 32)	conv(32, 32)	conv(32, 32)	1072
1019		maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	maxpool(2)	1073
1020							1074
1021							1075
1022							1076
1023							1077
1024							1078
1025							1079

Table 8: Network Architectures Orientation Estimation.

1080		1134
1081		1135
1082		1136
1083		1137
1084		1138
1085		1139
1086		1140
1087		1141
1088		1142
1089		1143
1090		1144
1091		1145
1092		1146
1093		1147
1094		1148
1095		1149
1096		1150
1097		1151
1098		1152
1099		1153
1100		1154
1101		1155
1102		1156
1103		1157
1104		1158
1105		1159
1106		1160
1107		1161
1108		1162
1109		1163
1110		1164
1111		1165
1112		1166
1113		1167
1114		1168
1115		1169
1116		1170
1117		1171
1118		1172
1119		1173
1120		1174
1121		1175
1122		1176
1123		1177
1124		1178
1125		1179
1126		1180
1127		1181
1128		1182
1129		1183
1130		1184
1131		1185
1132		1186
1133		1187

Table 9: Network Architecture Semantic 3D Point Cloud Labeling.

1188	References	1242
1189		1243
1190	[1] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. <i>arXiv.org</i> , 1511.00561, 2015. 5	1244
1191		1245
1192	[2] H. Riemenschneider, A. Bódis-Szomorú, J. Weissenberg, and L. V. Gool. Learning where to classify in multi-view semantic segmen- tation. In <i>Proc. of the European Conf. on Computer Vision (ECCV)</i> , 2014. 3	1246
1193		1247
1194	[3] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In <i>Proc.</i> <i>IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)</i> , 2015. 2	1248
1195		1249
1196	[4] Özgün Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation. <i>arXiv.org</i> , 1606.06650, 2016. 5	1250
1197		1251
1198		1252
1199		1253
1200		1254
1201		1255
1202		1256
1203		1257
1204		1258
1205		1259
1206		1260
1207		1261
1208		1262
1209		1263
1210		1264
1211		1265
1212		1266
1213		1267
1214		1268
1215		1269
1216		1270
1217		1271
1218		1272
1219		1273
1220		1274
1221		1275
1222		1276
1223		1277
1224		1278
1225		1279
1226		1280
1227		1281
1228		1282
1229		1283
1230		1284
1231		1285
1232		1286
1233		1287
1234		1288
1235		1289
1236		1290
1237		1291
1238		1292
1239		1293
1240		1294
1241		1295