VUE-net: Semantic Segmentation of Urban Environment

Artico Giovanni, Toffolon Mattia

Introduction

Semantic segmentation of 3D point clouds of urban environment is a particularly important task since it is crucial for the development of autonomous vehicles and can also support urban planning and help performing environmental monitoring. With this work we address this problem by introducing a novel Deep Neural Network that we called *VUE-net*.

The name has multiple origins. Firstly, it is an acronym that stands for *Voxelized Urban Environment network*. Secondly, *vue* is a french word, commonly used also in english, that means "view", which in a sense is the input of our model. Lastly, *VUE-net* is phonetically similar to *U-net*, which has been the major inspiration for this project.

Dataset

To build and train our network we used 1000 samples taken from the SemanticKITTI dataset [1]. The data is in the form of pointclouds. We used the functions provided by the colab notebook on PointNet for reading and sampling the pointclouds. It must be noted that these function combined do not read the full pointcloud but return a random subset of points of fixed size (20000 points).

Although, we decided not to use these sets of points as input for our model. As shown by PointNet, a model based on pointclouds is particularly heavy and therefore unusable for real-time applications. We instead decided to perform voxelization on the pointclouds and therefore our model's input is a grid of voxels. A voxel is a volumetric unit of space, so it can be considered as a pixel in 3D space. In order to transform the data we used the Open3D library api [6]. We decided to use voxelgrids of size 32^3 to have simple calculations, as it is a power of 2, and at the same time a manageable number of voxels to process. The voxels are encoded into a $(32 \times 32 \times 32 \times 1)$ tensor, where the element $(X, Y, Z, 0)$ is set to 1 if and only if there's at least a point in the pointcloud that corresponds to the voxel with coordinates (X, Y, Z), or 0 otherwise.

For the labels encoding we decided to use $(32 \times 32 \times 32 \times \#classes)$ tensors which were generated as follows. For each point in the pointcloud, the value of the tensor $labels(X, Y, Z, :)$ is updated as $labels(X, Y, Z, :)$+=l, where (X, Y, Z) are the coordinates of the corresponding voxel and l is the one-hot encoded version of the point's label. After iterating over all points labels, the obtained tensor was normalized along the appropriate dimension in order to obtain a probability distribution over the possibile classes for each voxel. As for the empty voxel, they were assigned to an additional (w.r.t. SemanticKITTI) fictitious class with total probability to allow our model to predict a voxel as empty. This encoding was initially performed at run-time but it introduced a considerable over-head in the prediction time. Our tests showed that it takes about $1s$ in order to convert the pointcloud using the adopted Open3D function, and $2s$ at most to convert the labels. This amount is also not ideal to have at training time as it would take the most time out of a training round. To solve this issue we resorted to creating a new version of the dataset by computing and saving the voxelgrids, so that to access the voxelized samples at training time, only their loading is needed. This also allowed to have a stable dataset, since the sampling of the pointclouds is not deterministic. Although, in real-time applications this process must be performed on-the-fly and so, to assure a fast response, the transformation should be performed at a lower level with respect to our implementation.

Model

Given the similarity of our task with image encoding, we decided to take inspiration for our model architecture from many different networks used for image segmentation. Our main inspiration was U-net [4]. This network exploits an autoencorder-like architecture to obtain the segmented image. In particular, it uses convolution to extract features and maxpooling to downsample the image. Another U-net element that inspired us is the use of concatenation in order to provide residual connections between the "down" and "up" convolutions. It must be noted that the authors of the paper used cropping in order to perform these operations, as the feature dimensions between layers do not exactly match. It will be shown later that our architecture does not suffer from this issue since we decided to simplify this process by having hidden states of equal size. Another model architecture we took inspiration from was VoxSegNet [5]. This architecture performs sequential extraction of features that are combined toghether, across different levels of extraction, to obtain a prediction. This is analogous to the residual but with some additional complexity that won't be further explained in this paper. The specific VoxSegNet element we included in our model architecture is the use of convolutional layers with

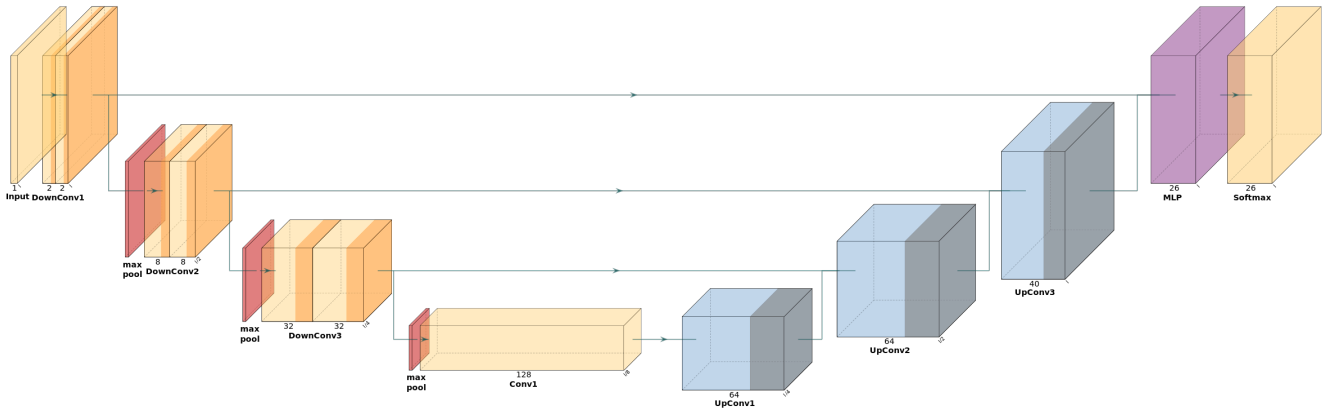


Figure 1: VUE-net architecture

different dilation rates (atrous convolutions) in order to obtain features through kernels with different receptive fields.

Figure 1 illustrates the complete scheme of our model architecture, which is now going to be discussed. For the down-convolution steps we used a custom *DownConv* block that consists in two different convolution layers, one with a dilation rate of 2 and a traditional one, which both double the number of features. The output of this layer is the concatenation of the results of the two convolutions. Between steps, max pooling has been adopted to extract the more relevant features and to compress the representation. At the "bottom" of the "convolutional ladder" a standard convolution that doubles the number of features has been used. Then, each one of the up-convolution (also known as transposed convolution) steps is executed by a *UpConv* block that halves the number of features and increases the other dimensions, therefore expanding the data representation until layer *UpConv3* matches the input dimensions. As shown in the figure, this leads to a features number unbalance towards the ones coming from up-convolutions. This was purposly done to give more importance to features of the new data representation that is expected to be more contiguos with respect to the original one that is affected by sparsity caused by the pointclouds sampling mentioned earlier. In the end, a multilayer perceptron (*MLP*), made of two fully connected layers, has been used to shrink the features dimensions to the number of classes (26) combined with a SoftMax layer that allows to have the desired probability distribution. Additionaly, between every subsequent convolutions Batch Normalization layers and ReLU activation functions were used. This choice was made to stabilize the outputs of each layer and avoid overfitting. The ReLU in particular was chosen for its general reliability and simplicity, as no particular properties were required for the values of the in-between layers.

To obtain more accurate predictions, we created an additional ensemble-like model that exploits the previous one robustness to input rotations which, as it will be explained later, is embedded at training time. This network feeds the original model with the four possibile rotated versions of the given sample, aligns the received labels and returns their mean values.

Training

The dataset made of 1000 sample was split into subsets of 750, 150 and 100 samples that correspond respectivly to training, validation and test sets. In addition, samples from the training set were shuffled before being used. The training phase was set such that for a determined number of epochs, the samples from the proper set were loaded in batches of 10, rotated by 90 degrees around the z-axis by a random number of times in the range 0-3 (to embedd robustness to such rotations) and then given as input to the model, which performed the forward and backward propagation processes. For this last operation, a loss function is needed. We initially opted for the CrossEntropy loss, but we soon encountered poor results. This was due to the class imbalance of the samples towards the "empty voxel" class. This happened for two reasons: first, by using voxelgrids instead of pointclouds we introduced lots of empty space in our samples that had to be encoded too; second, the sampling of the pointclouds may lead to creating empty voxels that actually contain points. We tried to address this issue by using the weighted version of CrossEntropy loss. The weights were computed as as the inverse of the normalized sum of the class membership distribution probabilities over all voxels in the training set samples. Even if this approach showed better performance, the class imbalance was too severe and the obtained results were not acceptable. This lead us to our final choice for the loss function, Dice Loss. This function allowed for fairly good model performance after training. This result is caused by the fact that Dice Loss considers the intersection over the union (IoU) of the predicted and true positive regions, which makes it particularly sensible to small and underrepresented classes. After each epoch, validation was performed through the ensemble version of the model to assess the current model performance and save it only if it was the best one seen so far. As for the Validation Loss we decided to use "one minus" the accuracy metric computed on the validation set samples. This value is computed by a function that confronts each true and computed label tensors in their one-hot encoded version (argmax set to 1, others to 0) and returns the normalized count of pairs of different tensors. By

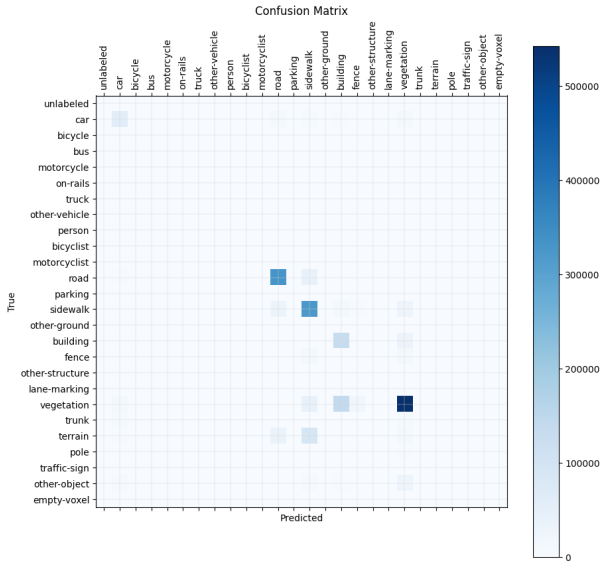
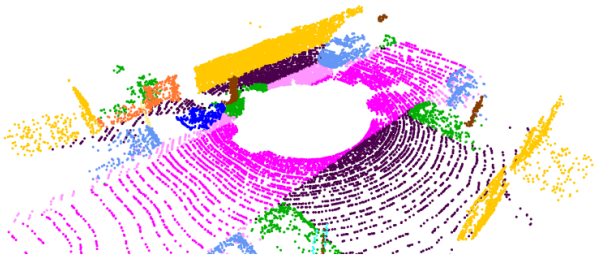


Figure 2: Confusion matrix

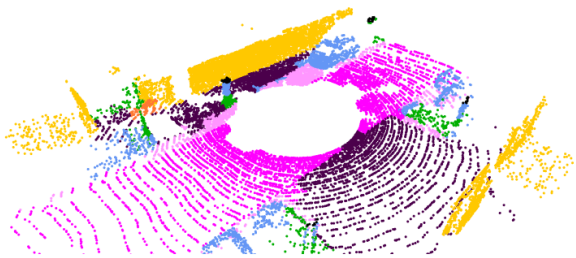
VUE-net

$mAccuracy$	0.692	$2.197s$	$Vox.\ time$
$mPrecision$	0.427	$0.011s$	$Pred.\ time$
$mRecall$	0.327	$0.911s$	$Back.\ time$
$F1-score$	0.371	$3.119s$	$Tot.\ time$

Table 1: Model performance



(a) True labels



(b) Predicted labels

Figure 3: True and predicted labels for pointcloud 55

training our network for 50 epochs while relying on Adam optimizer with learning rate 0.001, we achieved a validation loss value of 0.280, which means that the model correcly labels 72.0% of the samples voxels in the validation set.

Performance

Afterwards, the test phase was performed. The test set used, composed by the last 100 samples, was not voxelized in order to test model performance on data in its original format. To be able to achieve this, ad-hoc backprojection functions were implemented. Given a pointcloud, these functions attribute directly to each point the predicted label of the corresponding voxel in the voxelized pointcloud (voxel grid). With these operations, it was possibile to build the confusion matrix shown in figure 2. Using the data in this matrix, we were able to compute the following performance metrics over the 26 classes: mean accuracy, mean precision, mean recall and consequently F1-score. Also, the average time needed by the model to perform a complete prediction (*Tot. time*) was measured along the average partials relative to pointcloud voxelization (*Vox. time*), model prediction on the voxelgrid (*Pred. time*) and label backpropagation (*Back. time*). The times have been taken over 30 random pointclouds predictions. These two sets of data are reported in table 1.

From these results, the following observation can be made. Firstly, only a few classes appear among test set pointclouds. Secondly, it can be noted from the confusion matrix and guessed by the performance metrics, that our model is able to correctly classify the most popular classes such as *car*, *road*, *sidewalk*, *building* and *vegetation* but lacks in performance for the others. This happens because the latters appear a significantly lower amount of times and are possibly lumped together in a differently labeled voxel. This fact is reflected in the recall value, as two out of the classes present in the dataset are never predicted (yielding each one zero recall) causing the mean metric to be much lower than the other ones. However, the obtained results are still acceptable thanks to the great unbalance among point classes. Lastly, the total prediciton time of our model is particularly high, making it unsuitable for real-time applications. Although, it must be noted that the "true" prediciton time of *VUE-net* is of only $11ms$ and the phases that generate the overhead in the total time are the voxelization and backprojection ones.

Thefore, for a possibile deployment of this model, only the implementation of these two operations shall be changed.

Using other Open3d functions, we were able to visualize the pointclouds with different colors depending on their labels. Figure 3 shows one of the pointclouds with the correct colors (true labels) and the ones choosen by *VUE-net* (predicted labels). The observations made previously on performance agree with what can be seen from the images. In fact, the model correcly predicts road, sidewalks, cars, buildings and vegetation but is not able to recognize smaller, unpopular areas as the one in orange that corresponds to a fence.

Conclusions

VUE-net was proven to be a model capable to perform semantic segmentation of 3D pointclouds with good performances. In fact, it was able to classify correctly an average of almost 70% of the pointclouds points. Although, it still struggles to recognize unpopular classes. Also, the current version of the model is unsuitable for real-time applications but by lowering the implementation level of the voxelization and backpropagation phases, the network would be ready for deployment. In conlusion, to further improve *VUE-net*, a better way to make the network pay particular attention to the previously described classes must be found and the data transformation phases should be made more time efficient.

References

[1] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

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

[3] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[5] Zongji Wang and Feng Lu. Voxsegnet: Volumetric cnns for semantic part segmentation of 3d shapes. *CoRR*, abs/1809.00226, 2018.

[6] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing, 2018.