SEMANTIC URBAN MESH ENHANCEMENT UTILIZING A HYBRID MODEL

Patrick Tutzauer*, Dominik Laupheimer, Norbert Haala

Institute for Photogrammetry, University of Stuttgart, Germany
- (patrick.tutzauer, dominik.laupheimer, norbert.haala)@ifp.uni-stuttgart.de

ICWG II/III: Pattern Analysis in Remote Sensing

KEY WORDS: Urban Scene Understanding, Semantic Segmentation, Mesh Features, 1D CNN

ABSTRACT:
We propose a feature-based approach for semantic mesh segmentation in an urban scenario using real-world training data. There are only few works that deal with semantic interpretation of urban triangle meshes so far. Most 3D classifications operate on point clouds. However, we claim that point clouds are an intermediate product in the photogrammetric pipeline. For this reason, we explore the capabilities of a Convolutional Neural Network (CNN) based approach to semantically enrich textured urban triangle meshes as generated from LiDAR or Multi-View Stereo (MVS). For each face within a mesh, a feature vector is computed and fed into a multi-branch 1D CNN. Ordinarily, CNNs are an end-to-end learning approach operating on regularly structured input data. Meshes, however, are not regularly structured. By calculating feature vectors, we enable the CNN to process mesh data. By these means, we combine explicit feature calculation and feature learning (hybrid model). Our model achieves close to 80% Overall Accuracy (OA) on dedicated test meshes. Additionally, we compare our results with a default Random Forest (RF) classifier that performs slightly worse. In addition to slightly better performance, the 1D CNN trains faster and is faster at inference.

1. INTRODUCTION

Virtual City Models are an integral part of our daily lives. Applications like navigation, urban planning, and computer games are based on 2D and 3D geodata. Map applications - no matter if they are crowd-sourced like Open Street Map or governmental - contain geometric information organized in several layers. Each layer corresponds to a specific object category like e.g. building, road, waterbody. For this reason, map applications contain semantic information to some extent in an implicit manner. Additional and explicit semantic information, on the other hand, is often still lacking. Colored triangle meshes have become one of the standard representations of virtual city models for 2.5D and 3D geodata [Bousaha et al., 2018]. In contrast to point clouds, meshes provide high-resolution textural information and explicit adjacency information. Meshes are the de-facto standard/deliverable in photogrammetric product pipelines since they require less storage than point clouds while providing more information like explicit topology of points. Recent work was concerned with image-based classification of building facades into different categories [Laupheimer et al., 2018]. This serves as a tool to semantically enrich map data. In order to classify individual buildings in a mesh in more detail, however, it is necessary to locate potential buildings first.

Deep Learning (DL) methods are the standard tool for semantic segmentation in image space nowadays [Zhao et al., 2017; Chen et al., 2018]. Point-based classifiers like PointNet++ [Qi et al., 2017] achieve good results on 3D point clouds. However, to the best of our knowledge, little work focuses on a DL framework that directly operates on meshes so far. In particular, the field of urban meshes has hardly been explored. Therefore, we address the task of semantic mesh segmentation - attaching semantic classes such as building mass/facade, roof, impervious surface, green space, mid and high vegetation, vehicle, chimney/antenna and clutter to each face. For this

*Corresponding author.
purpose we train a 1D multi-branch CNN. Ordinarily, CNNs are an end-to-end learning approach operating on regularly structured input data. Meshes, however, are not necessarily regularly structured. In order to make CNNs accessible to meshes, we calculate a multi-scale feature vector for each face in first place. Consequently, we artificially achieve regularly structured input data: each face is represented by a normalized feature vector. By these means, we end up with a hybrid model that combines explicit feature calculation and convolutional feature learning. Each branch of the 1D CNN is fed with the feature vector of corresponding scale (cf. section 4).

Labeled ground truth data is stringently required for supervised learning. Therefore, we manually attach a semantic label to each face of an urban mesh. Nevertheless, since generating training data is time-consuming and expensive, especially for semantic segmentation tasks, we explored the use of synthetically generated urban meshes to support the limited availability of real-world data. The big advantage of synthetically generated urban meshes is that the label is already explicitly attached to each face.

We describe data preparation including labeling and feature calculation in subsection 3.2. In Section 5 we evaluate our segmentation algorithm on real-world data and compare results to a semantic segmentation achieved by a default RF. Moreover, we compare the two classifiers’ results after being smoothed by a Markov Random Field (MRF). In summary, our contributions are: (1) We provide ground truth data by manually labeling a 2.5D mesh of a real-world urban area (cf. section 3). (2) We calculate a multi-scale feature vector for each face of the large-scale urban mesh (cf. subsection 3.2). Thereby, the subsequent semantic segmentation can be achieved by state-of-the-art classifiers. Furthermore, the approach is independent of the geometric structure of the mesh: 2.5D or 3D meshes can be processed. (3) We perform semantic mesh segmentation by training a tailored multi-branch 1D CNN (cf. section 4). We compare results to a RF baseline and after MRF-smoothing. (4) We perform an extensive ablation study in order to determine most influential features (cf. subsection 5.1).

2. RELATED WORK

For most tasks in the 2D domain, CNNs are state-of-the-art classifiers that outperform all other approaches. Several architectures have been proposed that are tailored (and thus directly applicable) to semantic segmentation in image space (Zhao et al., 2017; Chen et al., 2018). Ideally, DL frameworks for 3D semantic segmentation would directly use 3D data (point clouds or meshes) as input. However, CNNs require Euclidean or grid-like structures. Hence, they cannot be applied to 3D data like point clouds or meshes. In case of point clouds, a natural solution is to migrate to voxel space (Landrieu, Simonovsky, 2017; Hackel et al., 2016; Huang, You, 2016). This comes along with memory overhead. Therefore, much effort is put into networks that use sparse 3D convolutions (Graham et al., 2018). This approach has been successfully applied to urban 3D point clouds (Schmohl, Sörgel, 2019). Another common approach is to make use of well-performing semantic segmentation in image space and back-project the results to 3D space again (Boukh et al., 2017; Lawin et al., 2017; He, Uferott, 2013). Due to the non-grid like structure of point clouds, classical machine learning approaches still compete with DL approaches. (Blomley, Weinmann, 2017) compute contextual features on multi-scale and multi-type neighborhoods and feed a RF for supervised point-based classification. Basically, we take the same steps, but we work with meshes. Furthermore, we leverage a DL approach. Recently, several works deal with classification of point clouds using a DL approach like (Qi et al., 2017; Su et al., 2018). PointNet++ is a hierarchical neural network that directly operates on the point cloud using different abstraction levels of the point cloud for feature learning (Qi et al., 2017).

Similar to semantic point cloud segmentation, approaches that aim to segment meshes semantically usually do not operate on the mesh directly (PointNet++ is an exception here). Common approaches make a circuit to 2D image space to take advantage of aforementioned image segmentation. Instead of directly operating on the mesh, those approaches rather render 2D views of the 3D scene, learn the segmentation for different views and finally, back-project the segmented 2D images onto the 3D surface (Kalogerakis et al., 2017). Main drawback of such approaches is that they do not make use of geometric information. Moreover, they suffer from occlusions in 2D space. On the contrary, the few existing approaches working on meshes directly, however, often do not use textural information and merely perform mesh segmentation. Therefore, (Valentin et al., 2013) train a cascaded boosting classifier using both geometric features (extracted from the mesh) and textural features (extracted from imagery). The classifier’s output is used as unary term for a sub-sequential MRF.

To the best of our knowledge, there are little or no applications of 3D DL to urban meshes yet. Almost all approaches known to us that directly operate on meshes are merely dealing with small-scale toy data sets such as the Princeton Shape Benchmark (PSB) (Shilane et al., 2004). (Theologou et al., 2015) list several methodologies for mesh segmentation in the domain of computer vision. Most of the presented methods are unsupervised methods applied to small-scale toy data sets focusing on the detection of coherent parts. Without exception, commonly used features are purely geometric. Graph Convolutional Neural Networks (GCNNs) have been introduced for classification purposes (Bronstein et al., 2016). (Qi et al., 2017) use GCNNs for (semantic) 3D shape segmentation. However, these approaches also operate on point clouds only. A joint approach to refine geometry and semantic segmentation of meshes is presented by (Blaha et al., 2017). The mesh shape is deformed with variational energy minimization, while semantic labels are updated with MRF inference. Initial semantic segmentation maps are obtained from a MultiBoost classifier. Most similar to our work is the semantic mesh segmentation proposed by (Rouhani et al., 2017). They gather faces of a 3D textured mesh into so-called superfacets to reduce the calculation effort. In other words, they use segmentation as preprocessing before performing the semantic segmentation on the superfacets. The prediction is done via a RF using geometric and photometric features. In contrast to the work of (Rouhani et al., 2017), we do inference on each face instead of superfacets. However, our features are multi-scale considering several Spherical Neighborhoods (SPNs) and thus, the achieved labeling is smoothed implicitly. We refer to this as implicitly smoothed per-face classification through a larger support region. Using this implicit smoothing we avoid treating each face as independent of all others. We assume that this implicit smoothing reduces the need for an explicit smoothing via a MRF as done in their approach. Nonetheless, we perform explicit MRF smoothing and evaluate the performance (cf. section 5). Our presented work adapts the approach of (George et al., 2017), which uses a multi-branch
1D CNN for 3D mesh segmentation of the PSB. Their proposed network aims to extract coherent parts of a 3D mesh. They do not provide explicit semantic information. On the contrary, we leverage this network in order to achieve a semantic segmentation of urban meshes. For each triangle, a multi-scale feature vector is computed and serves as input for the CNN. Different scales are fed to respective branches.

The negligence of semantic mesh segmentation in the domain of urban scenes may be due to the absence of labeled benchmarks. Several mesh benchmarks or public data sets exist for indoor scenes (Armeni et al., 2017) or for single objects (Shilane et al., 2004). Available labeled urban data sets, however, are 3D point clouds (Wichmann et al., 2018, Hackel et al., 2017, Niemeyer et al., 2014) where research focuses on semantic segmentation of 3D point clouds. The work presented in (Rouhani et al., 2017) bases on an 3D point clouds.

3. DATA PREPARATION

Our real-world data set is obtained by fusing Airborne Laser Scanning (ALS) data and aerial oblique imagery. Both are captured simultaneously for the purpose of monitoring the lock in Hessigheim (Cramer et al., 2018). The capturing Unmanned Airborne Vehicle (UAV) is a RICOpter multi-copter platform equipped with a RiegV VUX-1LR LiDAR and two Sony Alpha 6000 oblique cameras. The LiDAR point cloud is used for creating a 2.5D mesh (also known as DSM mesh). The resulting mesh is textured with images captured by the two cameras. The triangle mesh is generated and textured with software SURE from nFrames (Rothermel et al., 2012). Although we use a 2.5D mesh, the applicability of the pipeline to a 3D mesh is exactly the same. A Ground Sampling Distance (GSD) of 5 cm is used for meshing. SURE provides a Level Of Detail (LOD) mesh model. We use the second most detailed level in order to retain a good trade-off between computation effort and (LOD) mesh model. We use the second most detailed level in order to retain a good trade-off between computation effort and 2.5D mesh, the applicability of the pipeline to a 3D mesh is exactly the same. A Ground Sampling Distance (GSD) of 5 cm is used for meshing. SURE provides a Level Of Detail (LOD) mesh model. We use the second most detailed level in order to retain a good trade-off between computation effort and preserving details in the mesh. Therefore, we create our own labeled ground truth data (cf. section 3).

3.1 Semantic Classes

Our considered classes are inspired by the ISPRS 3D semantic labeling contest (Niemeyer et al., 2014). Their relative frequency is given in parentheses: building/mass/facade (9.28 %), roof (6.34 %), impervious surface (5.67 %), green space (5.97 %), mid and high vegetation (63.38 %), vehicle (0.83 %), chimney/antenna (0.31 %) and clutter (8.22 %). Building mass and roof are mutually exclusive, since roof extraction is a quite common task. Impervious surface includes rocky areas, streets, sidewalks, parking lots and other man-made surfaces. In accordance with the closed world assumption, class clutter gathers all faces that do not match the other class labels.

3.2 Features

Prior to training the CNN, for each face of the mesh, a multi-scale feature vector is calculated. To do so, we implemented a pipeline that parses waveform OBJ files with their texture maps. There are two modes for parsing: texture and label. The former is responsible for parsing the actual textured mesh and generates features for each face. Features can be distinguished into geometric and radiometric features (cf. Table 1). To calculate either of them a local neighborhood for each face has to be established. We do this by computing the Center of Gravity (COG) for every triangle. Thus, we essentially obtain a point cloud equal to the number of triangles in the mesh. This point cloud serves as input for a k-d tree. For each face, multiple SPNHs are queried in the k-d tree - in our experiments we use SPNHs of radii 0.5 m, 1.0 m and 1.5 m. All COGs within the respective neighborhood are selected. If the number of selected candidates is below a threshold (e.g. five faces), the selection is extended to the required minimum number of COGs, regardless of their distance to the query point, i.e. the COG of the considered face. Figure 2 gives an impression of the SPNHs. Table 1 lists the features we calculate for every face. Features are calculated by means of (Zhou, 2015). We will brieﬂy introduce some of the features and would like to refer the reader to the reference for further details. Since we work with triangle meshes, vertex features account for 3 or 9 components in the feature vector each, depending on whether the feature is scalar or vectorial, respectively. We obtain the Laplacian for each vertex i within the current face f. Mean Curvature H is the mean of the maximum and minimum principal curvature κi and κ2, respectively: \( H = 0.5(κ_1 + κ_2) \). Accordingly, Gauß Curvature K is deﬁned as: \( K = κ_1κ_2 \). Valance deﬁnes the number of edges incident to the current vertex, hence its degree of connectivity. Dihedral Angle of vertex i represents the maximum dihedral angle of vertex i. A dihedral angle of vertex i is the angle between two faces that are adjacent to an edge that incidents at vertex i. Face features are straight-forward. Area is the area spanned by the triangle. Normal depicts the normal vector computed from the cross-product of the triangle edges. The Voronoi area of each corner of the face is encoded in Voronoi. Verticality, Horizonality and Inclination are computed for the face itself, as well as for each SPNH. For the latter case, we extract mean, median and standard deviation. Inclination i is defined as the angle between the face normal n and vertical/up vector \( v = [0 \ 0 \ 1]^T \). Horizonality is simply \( |n_z| \), i.e. the absolute vertical component of the face normal n. Correspondingly, Verticality is deﬁned as \( 1 – |n_z| \). The difference in \( Z \) between lowest and highest vertex within the triangle is described by Vertical Difference.
To obtain terrain-specific features, we incorporate DTM information for the geolocated mesh area. Hence, nDSM describes the height above ground terrain for all 3 vertices and the COG. (Kölle et al., 2019) showed that this feature is essential in the case of point cloud classification. (Kouchan et al., 2017) also encourage such kind of elevation feature, which is no height above ground however and therefore less expressive. For each SPNH, similarly to Hackel et al., 2016, covariance features are calculated: Lineararity, Planarity, Variation, Curvature, Omnidirection, Anisotropy, Eigenentropy and Sum Of Eigenvalues. The Face Density \( \frac{\text{face}}{\text{area}} \) is defined as the total number of faces within the neighborhood, divided by the distance to the farthest face in the SPNH. The higher the face density, the more triangles are agglomerated in this area. Similarly to the face feature Vertical Difference, the maximum vertical range for the complete neighborhood is captured by Max Vertical Difference. This feature gives information about the vertical extent of the current neighborhood. To inject additional terrain information, we calculate \( \sigma_{\text{DSM}} \), a measure for the vertical geometry variation within the neighborhood. To obtain radiometric features, we extract texture patches for each triangle. The predominant color information for each face is captured by its median value. We use RGB color information, as well as its HSV color space transformed pendant, in order to ensure lighting independent features. We capture these medians for each SPNH, too. However, to obtain more color feature expressiveness, we calculate several histograms. The histograms are depicted by different levels in Table 1. Each level depicts a different histogram bin discretization, with 9, 20, 64 bins for level 0, 1 and 2 respectively. We use three discretization steps in order to obtain different levels of radiometric information granularity while reducing memory footprint with respect to vanilla 256-himining. However, the more bins, the sparser the histograms and thus the less meaningful information within the feature vector. Calculating three different histograms gives us more flexibility in crafting the feature vector, since all of them can be omitted easily in the final feature vector composition. In general, we expect from the radiometric features that they are beneficial in distinguishing classes that are geometrically similar. In particular in the case of green space and impervious surface. The features of all data splits (train, validation and test set) are normalized according to statistics of the train set. All features are zero-centered and the standard deviation is normalized to one. Formally speaking, by calculating feature vectors, the mesh segmentation basically turns into a data-point-based classification task. In total, we calculate 749 features for each SPNH and each face. However, only a subset of the feature vector is affected by the chosen neighborhood. Vertex and face features as given in Table 1 only refer to the current face of interest and thus are independent of their neighbors. Our training set consists of 1.74 million (valid) faces. By calculating \( \sigma_{\text{DSM}} \) for every face, we can be sure that the labeled mesh only uses materials instead of texture patches to encode semantic classes. Using a data set specific Lookup Table (LUT), the semantic label for each face can be retrieved from the RGB triplet. The 1-to-1 correspondence for faces between the textured and labeled mesh is accomplished via identical COGs. To give an impression of the actual values within the feature vector we report them here briefly. For the roof example on the left of Figure 2, \( \text{Face Area} \) is 0.305 m\(^2\), Median Inclination of the complete neighborhood is 45\(^\circ\) and Face Density in the SPNH is 20.2 faces m\(^{-2}\). For the vegetation example on the right on the other hand, \( \text{Face Area} \) is 0.001 m\(^2\), Median Inclination of the neighborhood 90\(^\circ\) and the Face Density is at 68.7 faces m\(^{-2}\). It is clearly conceivable through the Inclination that the main direction of the faces for the vegetation example is vertical.

### 4. FEATURE BASED SEMANTIC MESH SEGMENTATION

Given a textured triangle mesh \( M = (V, E, F) \), with \( V, E, F \) its vertices, edges and faces respectively, our goal is to process \( M \) in a way that every face \( f \in F = \{1, \ldots, N_f\} \) is assigned to a semantic label \( l_f \in L = \{1, \ldots, N_l\} \). Here, \( N_l \) is the total number of faces within the mesh of interest and \( N_S \) is the number of semantic target classes. In our fine-grained case, \( N_S \) equals 8. To do so, we pursue a supervised learning scheme. As described in the previous section, we set up a training data set, which contains multi-scale feature vectors \( X_f^{\text{multi}} \) for each face \( f \) and an associated label for each face \( l_f \). For many Photogrammetry and Remote Sensing related classification and regression tasks, RFs are still state of the art and on par with DL approaches (Weinmann et al., 2015; Kölle et al., 2019). Thus, we use the concatenated multi-scale feature vectors \( X_f^{\text{multi}} \) to train a default RF classifier as a baseline. The concatenation of all SPNHs (level 0, 1 and 2) considers the fact that some features do not depend on the SPNHs (cf. Table 1). Therefore, the concatenated feature vectors have a length of 1901 (instead of 2247 when stacking naively). We perform a grid search for parameters number of trees and depth of trees to obtain the optimal parameter configuration for the RF. Details can be found in section 5. To explore the potential of CNNs for semantic mesh segmentation, we adapt the architecture of (George et al., 2017). The gist is described in section 2. This network consists of three input branches. Each branch is fed with the respective feature

<table>
<thead>
<tr>
<th>per Vertex</th>
<th>per Face</th>
<th>per SPNH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laplacian</td>
<td>Area</td>
<td>Covariance Features</td>
</tr>
<tr>
<td>Mean Curvature</td>
<td>Normal</td>
<td>Verticality</td>
</tr>
<tr>
<td>Gauß Curvature</td>
<td>Voronoi</td>
<td>Inclination</td>
</tr>
<tr>
<td>Valance</td>
<td>Verticity</td>
<td>Face Density</td>
</tr>
<tr>
<td>Dihedral Angle</td>
<td>Horizontality</td>
<td>Max Vertical Difference</td>
</tr>
<tr>
<td>Vertical Difference</td>
<td>Inclination</td>
<td></td>
</tr>
<tr>
<td>( \sigma_{\text{DSM}} )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Calculated features for each face. SPNH-based features are computed for SPNHs of radii 0.5 m, 1.0 m, 1.5 m. Histogram levels 0, 1, and 2 use 9, 20 and 64 bins respectively to map color values originally discretized in 256 steps.
vector of the SPNH, depicted on the left in Figure 3. Our feature vectors $X_f$ per SPNH have a dimension of $749 \times 1$ (with the first dimension denoted as $N_f$), whereas the feature vectors of (George et al., 2017) are slightly larger with dimensions of $800 \times 1$. Although using more features their implementation only considers geometric features. The first convolutional layer $conv0$ of the original model has a filter size $F = 15 \times 1$. However, our implementation differs here. In our opinion, the filter size should be as large as $X_f$ itself in order to capture context within the whole feature vector. To highlight the difference to common CNNs, spatial/contextual information is provided via SPNH-dependent features and not gathered by the convolutional kernels themselves. The convolutional kernels are used as cheap feature embeddings. If filter size of $conv0$ is very small, only local correlations within the feature vector $X_f$ can be captured. For image processing tasks, where pixels are the fastest units, this does not hold true since we construct the feature vector ourselves. The initial arrangement of single features within $X_f$ is de facto random but fixed after initialization. In order to decouple the feature correlation from the feature ordering, we use a kernel with the same dimensions as $X_f$ (i.e. $N_f$) with a stride $S = 1$ and padding $P = \left\lceil \frac{S}{2} \right\rceil$. In addition, we define an unnormalized Gaussian weighting function $W_G$ with $\mu = 0$ and $\sigma = 0.4$ in the range $[-2, 2]$, discretized in $N_f$ steps. Using the mentioned $\sigma$ results in weights very close to 1 near the maximum. This normal distribution is centered at $\left\lceil \frac{S}{2} \right\rceil$. The output of $conv0$ is element-wise multiplied by $W_G$. The Gaussian Weighting enforces the importance of activations generated by incorporating the majority of features. In other words, central parts of the feature map are multiplied with values close to 1, whereas the tails of the feature map are less important since increasingly less features contributed to the convolution and hence are multiplied by values closer to 0. The padded convolution is needed as otherwise feature maps would consist of mere zeros (i.e., no value for each kernel). This would be a massive feature embedding. Here, we use padded convolution instead of fully connected layers as parameter sharing is only possible for convolutional layers. We use leaky ReLU as activation function and incorporate batch normalization. After each convolutional layer, we perform max pooling with $F = 2$, $S = 1$ and $P = 0$. The three scale branches are merged using depth concatenation, resulting in a dimension of 17856 (output dimensions of each branch: [93, 641]). Subsequently, the feature maps are passed through two dense layers with a dropout layer in-between, with the output size of the last dense layer being $N_c$. Finally, the output is passed through the softmax layer and delivers class probabilities in the range $[0, 1]$. To deal with the imbalanced classes in the training set, we incorporate Focal Loss (FL) (Lin et al., 2017). This loss was originally introduced for object detectors to tackle the extreme foreground-background class imbalance. FL is defined as:

$$FL(p_i) = -(1 - p_i) \gamma \log(p_i)$$

where $p_i$ is the predicted probability of the ground truth label. $\gamma$ is a non-trainable hyperparameter and was empirically found to work best for $\gamma = 2$ in the original paper. Intuitively, if $\gamma = 0$, FL degenerates to the cross-entropy loss. Initially, we used a weight-scaled cross-entropy loss with weights obtained from the class distribution of the training samples. However, FL consistently produced better results. With the network architecture depicted in Figure 3 we perform several runs with different hyperparameter settings for learning rate, batch size, L2 regularization, and optimizers. Generally, our real world is spatially smooth, i.e. adjacent faces are more likely to belong to the same semantic class than to others. This observation can be used as prior knowledge in order to spatially smooth the semantically enriched mesh. With increasing LOD the importance of smoothness increases because of higher intra-class variability. To give an example, more details will be visible and there will be higher spectral variability in high-resolution imagery when LOD increases. Due to the SPNH dependence of our feature vectors we already implicitly introduced the smoothness assumption to some extent. In order to explicitly smooth the achieved labeling, we employ a MRF. For each face, a label smoothing is performed. This smoothing is based on the local neighborhood defined by the value of feature Face Density. We report our results in section 5.

5. RESULTS

In this section, we discuss the results obtained from the RF and CNN-based semantic segmentation. Both, CNN and RF, are trained on the training set partially shown in Figure 1. Tests are performed on a machine with an NVIDIA GeForce GTX 1080 Ti GPU and 64 GB RAM and a 12-core CPU. Table 2 registers achieved accuracies and inference times for all test tiles of both classifiers. The shown RF results correspond to a RF consisting of 250 trees, each having a depth of 25. These parameters have been picked by a grid search, as they offer a good trade-off between accuracy and training time. To give an example, increasing number of trees by 5 comes along with an increased training time (inference time) of 68 min (3 s) while merely increasing accuracy by 0.016 %. RF trains 6.6 h (for best parameter configuration) and achieves an accuracy of 79.009 % for Tile A. Subsequent MRF makes the result visually more appealing but decreases accuracy to 78.578 %. The best performing 1D CNN configuration uses a feature vector that is composed of all geometric features but only uses Median HSV per face and Hue Hist Level 0 per SPNH as textural features. We use SGD as optimizer, a batch size of 50, and an initial learning rate of 0.001. Training time is less than 15 min. Inference time for Tile A is 14.69 sec and the achieved accuracy is 79.868 %. Subsequent MRF smoothing slightly decreases the result (79.732 %). As expected, MRF does not improve results as we smooth implicitly via contextual features. It has to be noted, that running time of RF and CNN is not fully comparable, since the scikit-learn implementation of RFs is running on the CPU and the PyTorch implementation of our 1D CNN is running on the GPU. (Liao et al., 2013) report a RF GPU implementation that is ten times faster than scikit-learn. However, even with such an improvement, the CNN training currently is still three times faster. DL frameworks are specifically relying on cuDNN enabled GPUs with cudNN that use highly optimized implementations for learning routines such as forward/backward convolutions and pooling. Coupled with efficient CPU based batch handling, this is expected to scale better on large data sets and models than RFs, which is important when processing entire cities. We are aware of the fact that OA is not as expressive as other classification metrics such as per-class precision and recall when dealing with highly imbalanced data sets. Since we experienced similar behavior throughout all our experiments, we therefore report per-class precision and recall in Table 3 once and limit ourselves to OA for the remaining tests. Using OA still has its justification because usually train and test splits...
This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.

https://doi.org/10.5194/isprs-annals-IV-2-W7-175-2019 | © Authors 2019, CC BY 4.0 License.

### Figure 3
Architecture of the multi-branch 1D CNN. After each convolutional layer (input dimensions denoted in square brackets), a max pooling with $F = 2$ and $S = 2$ is performed. 16, 32 and 64 denote the number of used kernels. For better visibility, please refer to the digital version.

### Figure 4
From top to bottom: Textured, ground truth labeled Tile A, RF prediction, 1D CNN prediction. In the ground truth you can see label noise: a flat roof (black circle) is labeled as green space and one face on the street is labeled as building mass/facade (black ellipse). Please note that the RF visually might indicate better performance on green space.

---

### Table 2
Achieved accuracies (in %, first row) and inference times (in s, second row) per test tile. There is no time depicted for the MRF smoothing as its processing time is independent on provided predicted label distribution. The weighted arithmetic mean is given for accuracies only. Best performances are marked in bold. Please note, the class distribution from Tile D differs significantly from the training tiles leading to bad performance. Class distributions of residual test tiles are similar to training distribution.

<table>
<thead>
<tr>
<th>Tile</th>
<th>RF</th>
<th>RF$_{MRF}$</th>
<th>CNN</th>
<th>CNN$_{MRF}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tile A</td>
<td>79.009</td>
<td>78.578</td>
<td>79.732</td>
<td>79.688</td>
</tr>
<tr>
<td>295884 faces</td>
<td>45.83</td>
<td>–</td>
<td>14.69</td>
<td>–</td>
</tr>
<tr>
<td>Tile B</td>
<td>76.584</td>
<td>76.513</td>
<td>76.425</td>
<td>76.824</td>
</tr>
<tr>
<td>226168 faces</td>
<td>36.14</td>
<td>–</td>
<td>11.43</td>
<td>–</td>
</tr>
<tr>
<td>Tile C</td>
<td>72.366</td>
<td>72.270</td>
<td>74.761</td>
<td>74.929</td>
</tr>
<tr>
<td>133932 faces</td>
<td>23.50</td>
<td>–</td>
<td>6.83</td>
<td>–</td>
</tr>
<tr>
<td>Tile D</td>
<td>48.559</td>
<td>48.279</td>
<td>48.336</td>
<td>47.766</td>
</tr>
<tr>
<td>116883 faces</td>
<td>19.82</td>
<td>–</td>
<td>5.82</td>
<td>–</td>
</tr>
<tr>
<td>Weighted Arithmetic Mean</td>
<td>72.543</td>
<td>72.298</td>
<td>73.209</td>
<td>73.213</td>
</tr>
</tbody>
</table>

### Table 3
Per-class precision and recall (in %) for predictions of the 1D CNN (top row) and RF (bottom row) for Tile A with best performing feature configuration (all geometric features but only Median HSV per face and Hue Hist Level 0 per SPNH as textural features). Class definitions are according to Table 1.

<table>
<thead>
<tr>
<th>Class</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>62.237</td>
<td>77.110</td>
<td>66.418</td>
<td>40.997</td>
</tr>
<tr>
<td>Recall</td>
<td>63.600</td>
<td>78.206</td>
<td>36.755</td>
<td>14.446</td>
</tr>
<tr>
<td>Precision</td>
<td>58.767</td>
<td>78.063</td>
<td>30.433</td>
<td>12.740</td>
</tr>
<tr>
<td>Recall</td>
<td>96.877</td>
<td>9.199</td>
<td>2.317</td>
<td>1.668</td>
</tr>
<tr>
<td>Precision</td>
<td>97.468</td>
<td>8.262</td>
<td>0.000</td>
<td>1.403</td>
</tr>
</tbody>
</table>

### 5.1 Influence of Feature Vector Composition
As described in section 4, our full feature vector is 749-dimensional. In the following, we refer to this as $X_{i,full}$. Training on $X_{i,full}$ delivers good results in terms of the predicted class coverage, however, suffers from smoothness. Due to the sparseness of the histogram features, especially for level 1 and 2, we examined different feature vector compositions and their influence on predictions. Table 2 lists training runs with different configurations of the feature vector. Testing was performed on Tile B. Although both classification metrics, OA and Weighted Precision (WP), deliver similar results for all runs, the visualization of the results in Figure 5 depicts the impact of different compositions of the feature vector. Please note, how config 2 (all textural features removed;
on the lower left) can no longer distinguish between impervious surface and green space and consequently only predicts the former. Removing all geometric features (config 3, cf. Figure 5 in the middle on the bottom) results in failure of identifying buildings (building mass and roof) correctly.

<table>
<thead>
<tr>
<th>Features Dropped</th>
<th>OA/WP in [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config 1: None</td>
<td>74.9/71.4</td>
</tr>
<tr>
<td>Config 2: All Textural Features</td>
<td>75.4/71.8</td>
</tr>
<tr>
<td>Config 3: All Geometric Features</td>
<td>72.6/67.1</td>
</tr>
<tr>
<td>Config 4: All Textural Features except: Median HSV, Hue Hist Level [0,1]</td>
<td>76.8/74.3</td>
</tr>
</tbody>
</table>

Table 4. Prediction results of Tile B for different compositions of the feature vector. The first column denotes all features that were removed with respect to $X_{\text{full}}$. OA - Overall Accuracy, WP - Weighted Precision.

Figure 5. Impact of different feature vector compositions with respect to Table 4. From left to right, top to bottom: Textured Tile B, ground truth labeled mesh, predictions for config 1, 2, 3 and 4, respectively. The figure is best viewed digitally.

6. CONCLUSIONS AND OUTLOOK

Within this paper, we presented an approach that combines feature engineering and feature learning for the semantic segmentation of urban triangle meshes. For each face, a multi-scale feature vector is computed and serves as input to a 1D CNN. For the fine-grained distinction of 8 classes, we achieve accuracies close to 80% for the used data set. This is slightly better than a default RF. Moreover, training the CNN is faster than training the RF. Inference is faster as well. This is important when processing entire cities or even larger areas. The detection of buildings is an important application-oriented task in geodata processing. Both classifiers deliver encouraging results for roof and building mass extraction although being not explicitly trained for this task. Moreover, CNN detects a fraction of class chimney/antenna whereas RF completely fails. Applying a MRF further enforces smoothness, yet does not consistently improve our results. This might be due to contextual features that achieve implicit smoothing. The class distribution of our data set is quite imbalanced - for example, we currently have very few samples for the chimney/antenna and vehicle classes. Normally, class imbalance is cured using data augmentation, which is not trivial for our approach. In general, there is a lack of large-scale labeled real-world data and benchmarks that represent well the complexity and variability of urban scenes. Thus, it is not possible to publicly evaluate mesh-based semantic segmentation approaches. As a first step, we labeled an urban scene manually on our own. With this in mind, one big challenge is to provide annotated data from a variety of scenes. Crowdsourcing seems to be a suitable approach to tackle this problem. We tested the incorporation of procedurally generated data by means of ESRI’s CityEngine. While exclusively training and testing on synthetic data works quite well, this does not transfer to the real word data. Potential reasons are: a strict Manhattan world assumption, geometric and textural simplifications of the synthetic data and LOD misalignment of the real world and synthetic mesh data.

Currently, we do not exploit the full potential of texture information as we rely on 1D feature representation (i.e. histogram of colors). In the future, additional branches could be introduced for explicit texture processing. Hence, the number of features can be highly reduced when color histogram features are replaced by explicit 2D branches. These branches could either explicitly extract features from the texture patches or even perform a classification only based on the texture. The per-triangle classification could serve as input for a voting scheme for the final semantic class decision for each triangle. Additionally, random search or Bayesian optimization could be used for better performance of the feature vector composition. Using LiDAR as base for the mesh geometry preserves the option of using LiDAR features per point, additionally in future. We anticipate further improvement when ground truth does not suffer from label noise and/or using a 3D mesh instead of 2.5D mesh. For example, the feature nDSM is more expressive in a 3D scenario (becoming Relative Height Above Ground consequently). To further increase capacity, a CNN ensemble could be employed where each network within the ensemble processes a different composition of the feature vector. In the longer term, it is more desirable to avoid feature engineering and to design an end-to-end (Graph-)CNN pipeline instead.

ACKNOWLEDGEMENTS

The urban mesh is a result of a research project in collaboration with the German Federal Institute of Hydrology (BfG) in Koblenz. We would like to thank the German Research Foundation (DFG) for financial support within the project D01 of SFB/Transregio 161.

REFERENCES


