Shape-Graph Based Object Recognition Using Node Context Embedding

MARTON SZEMENYEI
Budapest University of Technology
Department of Control Engineering
Magyar Tudosok krt 2, Budapest
HUNGARY
szemenyei@iit.bme.hu

FERENC VAJDA
Budapest University of Technology
Department of Control Engineering
Magyar Tudosok krt 2, Budapest
HUNGARY
vajda@iit.bme.hu

Abstract: Graphical object representation is fequently used for visual object recognition and detection methods. Since most machine learning methods requira vectorial input, significant research has been done on assigning feature vectors to graphs - a process known as graph embedding. However, when one wishes to detect objects in a larger scene, it is a more viable strategy to assign feature vectors to graph nodes, and classify them individually. In this paper, we present a graph node embedding algorithm for 3D object detection based on primitive shape graphs. Our embedding algorithm encodes the local context of the selected node into the feature vector, thus improving the classification accuracy of nodes. The method also imposes no restriction on the structure of the graphs or the weights on the nodes and edges. The method presented here will be used as part of an intelligent object pairing algorithm for Tangible Augmented Reality.

Key–Words: Artificial Intelligence, Shape Recognition, Graph Embedding, Object Detection, Augmented Reality

1 Introduction

Visual object recognition and detection is one of the most intensely researched ares of computer vision. Most applications use common two dimensional images for recognition, still, with the increase in availability of 3D datasets, the interest in 3D object recognition has increased significantly. While most of these object detection algorithms take the whole appearance of the object into account, there is a great number of methods that make decisions based on 3D shape only. Object detection is a key step for achieving scene understanding [1], which has a number of applications in different fields, such as robotics [2] or augmented reality [3].

Scene understanding offers great potential for Tangible Augmented Reality applications (TAR) [4]. In TAR systems the virtual objects used by the system are attached to real ones, and the real-world objects serve as input devices for user manipulation. This idea allows for intuitive man-machine interaction, which makes these systems easy to use. While most TAR systems use real objects with artificial markers [5, 6], there are a few that are able to use any object with natural features [7]. However, even these systems rely on the user to determine the pairing of the objects, making the setup of the scene time consuming.

In this paper we present an algorithm that performs the matching of virtual and real objects in a scene with natural features using 3D shape recognition. This way, virtual objects can be paired with real ones with similar shape, resulting in interaction techniques that are easy to master. Our method describes the shape of objects and scenes using graphs of primitive shapes [8]. This ensures that the actual segmentation of objects is also learnable. We use a support vector machine to classify the segments individually.

Our main contribution in this paper is a graph node embedding framework that aims to improve the node level classification. Our embedding method works on both directed and undirected graphs, without restrictions on the structure. Furthermore, the embedding is extended to graphs that have vector (or tensor) weights on their nodes and edges by using a feature transform function. We show, that this embedding significantly improves the segment-by-segment classification accuracy by producing a feature vector that also describes the local context of the given node. This local context is defined as the subgraph that is "close" to the given node, by using some distance measure. In the rest of the paper we assume that the edge features of the graph have a "distance" feature.

In the next section, we discuss relevant results of other workshops in the areas of 3D shape recognition and the classification of graphs. Then, in section 3, we first shortly discuss our shape description method, while in the latter part of the section we elaborate on

the graph node embedding framework. Finally, in section 4, we present and evaluate the results of our methods on several datasets.

2 Previous Work

In this chapter, we discuss the research of other workshops related to our own work. We begin by elaborating on the various methods for 3D object recognition, while in the second part we discuss graph classification and detection in graphs.

2.1 Shape Recognition

Two dimensional shape recognition is a relatively common task in computer vision, for which a great variety of methods exist. Most of these, however, cannot be generalized easily to 3D shapes [9]. Still, there are a few methods, such as the Generalized Hough Transform [10] or the RANSAC algorithm [11], that work reliably for the 3D case as well. Nonetheless, these algorithms require a reference model for matching, which cannot be easily obtained, mainly due to high intra-class variation.

In order to perform shape recognition without a reference model, learning algorithms are recommended. There are numerous approaches for this, such as using local features [12, 13] to create a learning algorithm. Another approach is using shape distributions [9], or global features [14] to describe objects. These algorithms, however, require a segmentation step in order to find object candidates to classify in larger scenes. In 3D scenes with helpful prior information (such as urban scenes, where the ground is easy to segment [14]) this may be relatively straightforward to do. In complex, cluttered scenes (such as indoors scenes), however, segmentation might become unreliable, resulting in inferior detection performance.

In recent years, deep convolutional neural networks (CNN) have become increasingly popular amongst researchers working on object recognition and detection due to their superior performance [15, 18]. Unsurprisingly, there is significant work on 3D object detection using either RGBD [16, 17] or volumetric [18, 19] data. Since CNNs are able to perform classification for every (super)pixel or voxel, multi-object detection in larger scenes using CNNs is relatively straightforward. [16, 20] Nonetheless, CNNs are notoriously difficult to train [21, 22], since they are fraught with numerical difficulties. Moreover, training CNNs requires large amounts of training data and computational resources, since deep networks have millions of free parameters.

Schnabel et. al [23] presents a different approach, that may alleviate the difficulty of segmentation. They proposed a variant of the RANSAC algorithm to segment a scene into primitive shapes (such as plane, sphere, cylinder, etc.), which they treat as the "building blocks" of the objects and the scene. Their algorithm uses local sampling, in order to increase the chance of finding local shapes. Their solution also uses an octree grid for fast inlier counting, resulting in relatively low runtime, even on large point clouds. They describe the 3D shape by constructing a topology graph of the scene, where the nodes of the graph are the primitives, and edges represent the geometric relations between the nodes. The adjacency between the shapes is determined by their distance [8].

In order to detect objects in a larger scene, they construct a reference graph for each category and apply brute-force graph matching. Since a single reference graph has relatively low number of nodes, the matching algorithm remains feasible [8]. To further decrease the number of possible matches between the scene and the reference graphs, they introduce a three types of of constraints. Node constraints ensure that only nodes of the same primitive type are matched, while edge constraints enforce the similarity of the relations between adjacent nodes. A third type or constraints - graph constraints can be used to take the relationship of non-adjacent nodes into account as well.

Their method, however, still uses reference objects for each category, which might not be easy to obtain. Furthermore, they use a brute-force matching algorithm, which cannot easily handle segmentation errors. By employing a learning algorithm to find specific subgraphs in a larger scene, these limitations may be overcome.

2.2 Graph Recognition

Machine learning with graphical data has various applications, including bioinformatics [24] or network analysis [25]. It is also quite common to recognize objects visually using graph-based learning, since objects can usually be described using graph of (visual) features [32]. The difficulty of graph classification is that most standard learning algorithms require a vector (or tensor) of features as their input. Since these methods cannot take graphs as inputs, one needs a way to convert it to a vectorial representation - to embed the graph into a vector space. This, however, is not a simple task, since the ordering of graph nodes is arbitrary, and any simple method of vectorizing a graph would yield a vector that is not invariant to the ordering of nodes [26]. A related difficulty is, that graphs of different sizes yield vectors of different dimensions, while standard learning algorithms assume

that all data is in the same vector space.

A special class of learning algorithms, called kernel methods, present an elegant solution to this problem. These methods employ kernel functions, which allow the algorithm to use a higher dimensional representation of the data implicitly. Kernel functions are symmetric, positive semi-definite functions that can usually be interpreted as a similarity measure between objects [27]. When using kernel learning methods (such as SVM), one may simple define a kernel function between graphs. Since a kernel function does not require a vectorial input, nor does it explicitly produce a vectorial representation, the entire problem can be circumvented.

Perhaps the most widely-known graph kernel is the random walk kernel [27], which interprets the edge weights of the graph as the probabilities of taking that edge during a random walk. It performs simultaneous random walks on the two graphs, and derives a similarity score based on the probability of performing the same walk. The logic behind the random walk kernel is, that if two graphs are similar, then performing the same walk on the two graphs is likely. This probability is computed using the direct product of the two graphs, since performing the same walk on the two graphs is equivalent of performing a walk on the direct product. The kernel function is computed according to the following equation:

$$K(X,Y) = \sum_{i=1}^{N} w(i)\mathbf{e}^T\mathbf{A}^i\mathbf{s} \tag{1}$$

$$w(i) = e^{-\vartheta i}, \tag{2}$$

where $\mathbf{A}$ is the adjacency matrix of the direct product, $\mathbf{s}$ and $\mathbf{e}$ contain the probabilities of starting and ending the walk on a given node respectively, N is the maximum length of the walks considered, and ϑ is a hyperparameter controlling the slope of the weight w. The staring and ending probabilities may be set uniformly, or according to a priori information.

When not using kernel learning methods, however, then the graphs must be embedded explicitly. Perhaps the most widely used method for explicit vectorial embedding is the spectral representation [28]. The simplest version of spectral embedding is to compute the spectral decomposition of the adjacency matrix of the graph. (3) If the graph has weights on the nodes, these can be inserted in the diagonal of the adjacency matrix [28].

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T \tag{3}$$

If the eigenvalues and the corresponding eigenvectors are ordered, then this representation will be partially invariant to node ordering. Since this invariance is only partial, and alignment step is still needed [29]. In order to handle graphs of different sizes, smaller graph is enlarged by adding dummy nodes to it [30].

Aside from the adjacency matrix, other matrices may be used for the spectral decomposition. One such instance is the Laplacian matrix of graphs, which is computed (4) using the adjacency ($\mathbf{A}$), and the degree ($\mathbf{D}$) matrices of the graph. One other method for embedding graphs is the heat kernel (5). The t parameter heat kernel controls the trade-off between local and global representation of the graph. According to Zhu and Wilson [31] the heat kernel outperforms the other two. It is also possible to mix different spectral representations [30] in order to create a more robust method.

$$\mathbf{L} = \mathbf{D} - \mathbf{A} \tag{4}$$

$$\mathbf{H} = \mathbf{V}e^{-t\mathbf{\Lambda}}\mathbf{V}^T \tag{5}$$

In our method, however, we intend to classify a graph on a node-by-node basis, which means, that instead of embedding entire graphs, we need to embed nodes into a vector space. In contrast with embedding graphs, there has been very little work done on the topic of embedding nodes. For instance, Demirci et. al. [32] has used a low-distortion node embedding framework to perform many-to many feature matching using the earth movers distance. Riba et. al. [33] use binary embedding to produce hash keys for fast graph retrieval.

These methods, however, place limitations on the structure of the graphs or the weights of nodes and edges. One assumes the graphs are trees, while the other assumes labeled graphs. Also, Riba et. al. produce hash keys, not feature vectors, which makes it hard to base a learning algorithm on their work. Since our shape description method yields full graphs with vectorial weights on both the nodes and edges, the previous methods are insufficient for our application. To our best knowledge, no work has been done yet on embedding graph nodes of vectorial weighted graphs, with no restrictions on the topology.

3 Embedding Graph Nodes

Object detection algorithms capable of detecting multiple instances usually employ a segmentation procedure, in order to produce object candidates for a subsequent classification method. This is a viable method of shape recognition, especially when applied to scenes where segmentation is relatively straightforward. In urban scenes, for instance, one can easily remove the ground, resulting in most of the objects becoming disjoint in the point cloud [14].

Segmentation, however, becomes significantly more difficult in indoors scenes, since objects much more likely to be cluttered in this context. For this reason, we use a different approach: we segment our scene into primitive shapes, which we interpret as the "building block" of the scene. Then, we classify primitive shapes individually, and determine objects based on the segment labels. Since primitive shapes have several features (depending on the primitive type), it would be straightforward to use these features to classify each primitive.

While the simplicity of this method is alluring, it ignores the geometric relations between the primitives and the local context of each primitive. This could lead to high classification errors, especially if two classes contain very similar shapes, albeit in different contexts. In order to avoid this, we construct a graph from the primitive shapes, and use a graph node embedding procedure to produce a feature vector for each node that encodes the local context of the primitive as well.

In order to construct shape graphs, we segment the 3D point cloud of the scene using the algorithm proposed by Schnabel et. al. [23] Their implementation is able to detect five different primitive shapes: planes, cylinders, cones, spheres and tori. We then construct a graph using the primitive shapes as nodes, while the edges represent the geometric relations between the nodes. (Fig.1)

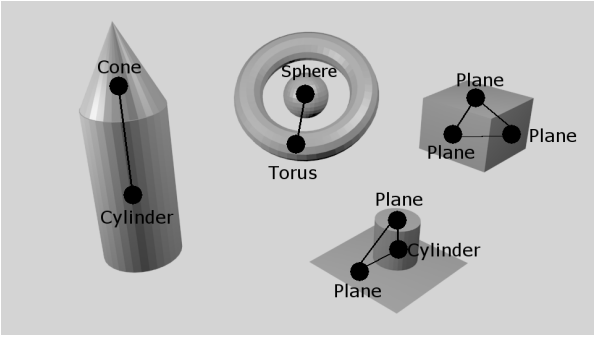

Figure 1: The graph constructed from primitive shapes (only close edges are drawn).

Each primitive shape type has a few distinct features that further define the exact shape of the point cloud they represent. (Table 1) By computing these features we are able to assign a feature vector to each primitive shape. Since the features of the different primitive types are incompatible, it makes sense to construct a unified feature vector for each primitive by concatenating the individual feature vectors. Of course, for every primitive shape the features of the other types will be set to zero.

Furthermore, each primitive shape can be easily assigned with a well-defined coordinate system, consisting of an origin and at least a single direction in the 3D space. (The only exception is the sphere, where one cannot find a special direction.) This means, that we may describe the geometric relations between the primitives by computing the rigid transform between their coordinate systems. (Table 1) However, since we want our algorithm to be invariant to rotation, we only consider the distance between the origins, and the angle of the rotation between their special directions. Since spheres do not have a special direction, the angle between spheres and other primitives is always set to zero.

Since primitive shapes are rigid transforms are described by more than one parameter, the nodes and the edges of the constructed graph have vectorial weights. Furthermore, the edges of the graph have two different types of weights. The first type is the traditional "distance" type, meaning, that if this feature is larger, then the two nodes are less connected. The second type is the "feature" type, which describes other qualities of the connection, but its magnitude does not influence the strength of the connection. While constructing graphs, we do not make explicit decisions on the adjacency of the nodes, we simply store the distance between them amongst the features. This means, that we always produce full graphs, which allows us to treat adjacency as a continuous measure instead of a binary one, thus avoiding loss of information.

3.1 The Embedding Framework

In this subsection, we present our graph node embedding framework in detail. Our goal is to create vectorial descriptors for nodes in graphs that have vectorial node and edge weights. We also wish to place no restrictions on the graph structure, that is, we propose a framework that is applicable to full, directed graphs. Our embedding method aims to describe "what the graph looks like" from the perspective of the node that is being embedded (the central node). Therefore the framework needs to include information on the features of the central node, as well as the surrounding ones. It also needs to incorporate information on the geometric relations between the nodes.

The first step of the embedding process is to order the nodes of the graph in based on the distance from the central node. Since the spectral embedding is only partially invariant to the node ordering, this step alone ensures that the feature vectors are different for separate nodes. If the ordering is ambiguous due to some nodes being too close, then two separate embeddings may be made with the different orderings and averaged. In order to create descriptors of the same size

Primitive	Plane	Cylinder	Sphere	Cone	Torus
Features	Area Diameter Bounding Box Area	Radius Height	Radius	Radius Height Angle	Inner Radius Outer Radius
Origin	Centroid	Centroid	Centroid	Peak	Centroid
Direction	Normal	Axis	N/A	Axis	Normal

Table 1: Features and reference frames for every primitive shape type

for all nodes, the maximum number of nodes included must be set. Distant nodes are clipped from larger graphs, while smaller ones are padded with zero nodes and edges.

Padding the graph with zero nodes may introduce a notable difficulty with embedding methods. For most learning algorithms the training data is normalized to zero mean and unit variance in order to avoid numerical difficulties. However, if features are normalized, then by padding the graphs with nodes with all zero features meas we are adding "average" nodes to "average" distance from the central node. Since padding should not affect the shape of the scene, nor the result of the embedding, this is problematic.

Luckily, in the case of primitive shape graphs, a relatively simple solution presents itself. Since our node and edge features are non-negative, we divide the features with the standard deviation, but do not subtract the mean. Thus, the non-negative property of our features is preserved, and adding zero nodes to the graph is equivalent with leaving the shape of the scene unchanged.

With the preprocessing steps completed, we construct a descriptor matrix for the node, called the node feature matrix $\mathbf{F}$. The matrix is unique for every node, while it contains information on the neighboring nodes as well. Is is computed according to the equation below.

$$\mathbf{F} = \begin{bmatrix} T_{1,1} & T_{1,2} & \cdots & T_{1,N} \\ T_{2,1} & T_{2,2} & \cdots & T_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ T_{N,1} & T_{N,2} & \cdots & T_{N,N} \end{bmatrix} \tag{6}$$

$$T_{ij} = T(n_i, n_j, e_{1i}, e_{1j}, e_{ij}), \tag{7}$$

where T is a feature transform function, n_i is the i_{th} node of the graph, while e_{ij} is the edge pointing from the i_{th} to the j_{th} node. N is the maximum number of neighboring nodes considered in the embedding. While the node feature matrix is unique for every node, its spectrum might be the same, especially or close nodes (provided that eigenvalues are ordered by magnitude). Still, because of the node ordering step, the eigenvectors will be different even if the spectra are not. One may introduce further differences in the descriptors of close nodes by choosing the feature transform function well.

In order to finalize the embedding process, we compute the singular value decomposition of the graph feature matrix, and concatenate the first couple singular values and vectors (8). It is possible to use feature transform functions, which ensure that the resulting node feature matrix is symmetric. In this case, the eigendecomposition may be used. Still, we do not wish to place such restrictions on the embedding framework, so using SVD is recommended.

$$\mathbf{w} = \begin{bmatrix} \sigma_1 \mathbf{u}_1 \\ \sigma_1 \mathbf{v}_1 \\ \sigma_2 \mathbf{u}_2 \\ \sigma_2 \mathbf{v}_2 \\ \vdots \\ \sigma_k \mathbf{u}_k \\ \sigma_k \mathbf{v}_k \end{bmatrix}, \tag{8}$$

where σ_i, $\mathbf{u}_i$, and $\mathbf{v}_i$ are the i_{th} singular value, left and right singular vectors respectively, and k is the maximum number of singular values considered. In the case of a symmetric node feature matrix, it is unnecessary to add both the left and right eigenvectors to the descriptor, since they are the same, thus it is possible to reduce the size of the feature vector by a factor of two.

3.2 Feature Transform Functions

Previously, we have said very little about the feature transform function. This was deliberate, since we do not intend to pose unnecessary restrictions that limit our framework's generality. In this subsection, we discuss our choices for the feature transform for *this specific problem*. For different graphs, however, other choices may be more appropriate. Notwithstanding, some of the principles established here may be useful for other applications as well.

Since our goal is to produce a feature vector that contains information on the *local* context of the node, the influence of nodes farther from the central node

Metric	e_{ho}				e_{cv}				e_{train}			
Embed	No	Linear	Quad	RWK	No	Linear	Quad	RWK	No	Linear	Quad	RWK
Synth	16.5	5.2	3.4	**0.2**	17.7	5.0	3.7	**1.9**	17.0	4.8	2.8	**0.0**
Synth2	67.8	1.4	0.9	**0.0**	68.1	1.6	0.9	**0.2**	67.8	1.2	0.8	**0**
Images	44.6	36.9	**35.9**	36.0	45.3	36.6	**36.0**	36.6	45.2	36.4	36.1	**35.9**
Images2	13.7	8.9	**7.7**	8.0	13.9	**8.5**	9.1	10.9	12.8	7.2	6.8	**4.95**

Table 2: Node-by-node classifiaction errors

must be less than that of the immediate neighbors. This means that if the edge features of the graph include a parameter that can be interpreted as "distance" or "connection strength", then this parameter may be used to weight the influence of the nodes. Since shape graph edges have a distance parameters, we will use this for our discussion without loss of generality, since connection strength may be understood as the inverse of distance.

That being said, we divide edge features into distance e_d and feature e_f types. We treat feature type values the same way we treat node features, therefore we concatenate them to the node feature vectors according to equation (9). Second, we use the distance type features to scale the result of the feature transform, therefore distant nodes will not affect the graph feature matrix significantly. Our choice for the feature transform function is shown in the equation below.

$$T_{ij} = w_{ij} \begin{bmatrix} n_i & e_{1if} & e_{ijf} \end{bmatrix}^T \begin{bmatrix} n_j & e_{1jf} & e_{ijf} \end{bmatrix} \quad (9)$$

$$w_{ij} = \frac{1}{1 + \mu(e_{1id}^2 + e_{1jd}^2 + e_{ijd}^2)}, \quad (10)$$

where μ is a hyperparameter controlling the distance scaling. It is important to note, that we do not only scale the feature transform using the distance from the central node, but also the distance between the two interacting nodes. Thus, the significance of the interaction between distant nodes will be reduced in the resulting feature vector.

There are two desirable properties of this transform function. First, it produces a higher dimensional transform of the original features, which makes it possible for linear learning methods to learn decision functions that are nonlinear in the original features. Moreover, T_{ij} returns a quadratic matrix, and satisfies the requirement $T_{ij} = T_{ji}^T$, which means that the node feature matrix will be symmetrical as well. Consequently, the eigendecomposition may be used for embedding, resulting in a smaller feature vector. It is also possible to define a linear feature transform function to use with nonlinear learning methods using the following choice of feature transform function.

$$T_{i,linear} = \frac{1}{1 + \mu e_{1id}^2} \begin{bmatrix} n_i & e_{1if} \end{bmatrix}, \quad (11)$$

where T_i is the feature transform function. Note, that the index j is omitted, since the feature transform is the same for all j values. This creates a node feature matrix with a rank of one, meaning that the resulting feature vector will simply be a concatenation of T_i values. This seems relatively straightforward, however, this method cannot add information on the interactions *between* nodes into the embedded feature vector.

3.3 The Random Walk Node Kernel

Aside from explicit node embedding, using implicit embedding via kernel methods is a viable way of graph node classification as well. In this subsection we briefly discuss the modification of the random walk kernel (RWK) to compare nodes instead of entire graphs. The underlying idea is that the local context of the graph node is equivalent with the set of nodes you are likely to get to through *short* random walks starting from the given node. This means that two contexts are similar, if simultaneous random walks can be performed with high probability, starting from the two nodes being compared.

This idea can be easily introduced into the random walk kernel by setting the starting probability vector $\mathbf{s}$ from Eq. (1) so that the walks would always start from the central nodes. The size of the local context explored by the walks can be influenced by setting the maximum length of the walks N or the slope of the weight function ϑ from Eqs. (1) and (2) respectively.

Note, that the random walk kernel requires the direct product of the two graphs. However, in order to compute the direct product of graphs with vectorial node and edge weights, kernel functions between the nodes and edges are needed [27]. A sensible choice for the node kernel is a relatively simple RBF similarity measure between the feature vectors of the nodes.

$$K_n = e^{-\nu(n_1 - n_2)^T(n_1 - n_2)}, \quad (12)$$

where n_1 and n_2 are the two nodes compared, and ν is the hyperparameter controlling the kernel. Choosing the edge kernel is requires somewhat more consideration. Arguably, defining an RBF similarity measure between the features of the two edges is sensible. However, edges also posses a "distance" type feature, which may be to weight the probability of taking the given edge during a random walk. In order to encourage the random walks to explore the local structure we set these weights to decrease the probability of taking long edges that take far from the starting nodes, resulting in the following equation:

$$K_e = \frac{1}{1+\gamma(e_{1,d}^2+e_{2,d}^2)} e^{-\xi(e_{1,f}-e_{2,f})^2}, \qquad (13)$$

where e_1 and e_2 are the two edges compared, γ and ξ are the hyperparametes controlling the kernel.

4 Experimental Results

In this chapter we present the result of the experiments based on our embedding method. We test the embedding using our recommendations for feature transform function, as well as the case where the vectorial node descriptors are used as feature vectors with no embedding. We use a Support Vector Machine (SVM) to perform the classification using a linear kernel, and compare the results achieved using the random walk node kernel.

There are four different datasets used for training the classification algorithms. The first dataset consists of synthetic shape graphs. This dataset is meant to be easily to learn, for the five classes use different types of nodes, with some random noise added. Some instances have additional noise nodes, while others are missing some nodes, to represent errors of the segmentation. The second dataset is also synthetic, however, the five classes use the same pool of nodes, albeit in different combinations. This dataset is meant to demonstrate that the embedding algorithm significantly outperforms the simple node descriptor based classification in cases where the different classes have very similar nodes, but in different configurations.

The last two datasets contain series of images that can be used for 3D reconstruction. The third dataset consists of images of synthetic objects created in Blender. There are five classes in ten variations each, with series of images taken with a moving camera. We created hundreds of partial 3D reconstructions for each category using VisualSfM [34]. The last database uses real images in four categories and 4-10 variations for each category. Here, the categories are relatively simple: box/book, mug/can, sphere, and horizontal surface).

We evaluated the performance of our node embedding method using support vector machines with both linear and random walk kernels. We applied Bayesian optimization to find good hyperparameter values, using 20% holdout validation error e_{ho} as the objective function. We then compute the training e_{train} and 10-fold cross-validation errors e_{cv} with the acquired hyperparameters. We compare the classification and validation errors with different feature transform functions used.

The results clearly show that the graph node embedding algorithm significantly decreases the loss of the node-by-node classification in most cases. Arguably, the node embedding method is most helpful for the second synthetic dataset, that contains classes with similar nodes in different configurations, which is in line with our expectations. It is worth noting, that the quadratic feature transform tends to outperform the linear version.

Apparently, the random walk node kernel outperforms the explicit node embedding using a linear kernel. Still, the random walk kernel has a significant drawback, namely that it is computationally expensive. While it takes approximately 30 seconds to compute the explicit embedding and train the SVM for a dataset containing about ten thousand nodes, computing the random walk kernel takes 30 *minutes* on the same dataset. This is especially problematic for hyperparameter optimization, since the kernel has to be reevaluated with every change in the hyperparameters, while the explicit embedding has to be computed only once.

5 Conclusion

In this paper, we have presented a 3D object detection algorithm that uses graph node embedding to classify segments of a larger scene individually. Our method builds primitive shape graphs from the 3D point clouds, and uses a graph node embedding method to allow for superb node-level classification. The embedding method aims to improve the classification by embedding the local context of the nodes into the feature vector as well. The framework presented is applicable to graphs with any structure, and the feature transform functions allow us to extend it to graphs with vectorial node or edge weights.

We performed experimental results and compared different choices for the feature transform function. We demonstrated that node embedding improves both training and validation accuracies significantly, especially when there are similar nodes in different

classes, albeit in different configurations. The proposed method may serve as basis for a high-level optimization procedure aimed to determine the setup of virtual objects for the TAR system.

Acknowledgements: The research was supported by the University of ABC and in the case of the first author, it was also supported by the Grant Agency of DEF (grant No. 000/05/ 0000).

References:

[1] L.-J. Li, R. Socher and L. Fei-Fei, "Towards Total Scene Understanding:Classification, Annotation and Segmentation in an Automatic Framework", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[2] C. Wojek, S. Walk, S. Roth, K. Schindler and B. Schiele, "Monocular Visual Scene Understanding: Understanding Multi-Object Traffic Scenes", *IEEE Transactions on Pattern Analysis and Machine Intelligence* **35**(4), pp. 882-897 (2012).

[3] O. Pauly, B. Diotte, P. Fallavollita, S. Weidert, E. Euler and N. Navab, "Machine learning-based augmented reality for improved surgical scene understanding", *Computerized Medical Imaging and Graphics* **41**(1), pp. 55-60 (2015).

[4] M. Billinghurst, H. Kato and I. Poupyrev, "Tangible Augmented Reality", in *ACM SIGGRAPH ASIA*, 2008.

[5] M. Billinghurst, H. Kato and S. Myojin, "Advanced Interaction Techniques for Augmented Reality Applications", in *Lecture Notes in Computer Science*, Springer, 2009, pp. 13-22.

[6] G. A. Lee, M. Billinghurst and G. J. Kim, "Occlusion based Interaction Methods for Tangible Augmented Reality Environments", *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, 2004.

[7] W. Broll, E. Meier and T. Schardt, "The Virtual Round Table - a Collaborative Augmented Multi-User Environment", *Proceedings of the ACM Collaborative Virtual Environments*, 2000.

[8] R. Schnabel, R. Wahl, R. Wessel and R. Klein, "Shape Recognition in 3D Point Clouds", *Proceedings of the 16-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2008.

[9] R. Osada, T. Funkhouser, B. Chazelle and Dobkin David, "Matching 3D Models with Shape Distributions", *SMI 2001 International Conference on Shape Modeling and Applications*, 2001.

[10] F. Tombari and L. Di Stefano, "Object recognition in 3D scenes with occlusions and clutter by Hough voting", *Fourth Pacific-Rim Symposium on Image and Video Technology*, 2010.

[11] M. A. Fishler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,", *Magazine Communications of the ACM*, **24**(6), pp. 381-395 (1981).

[12] R. Rusu, N. Blodow and M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D Registration", *Proceedings of the IEEE International Conference on Robotics and Automation*, Kobe, 2009.

[13] S. Lazebnik, C. Schmid and J. Ponce, "A sparse texture representation using local affine regions", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **27**(1), pp. 1265-1278 (2005).

[14] A. Golonivskiy, V. G. Kim and T. Funkhouser, "Shape-based Recognition of 3D Point Clouds in Urban Environments", *IEEE 12th International Conference on Computer Vision*, 2009.

[15] C. Szegedy, W. Liu and Y. Jia, "Going deeper with convolutions", *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[16] C. Wu, I. Lenz and A. Saxena, "Hierarchical Semantic Labeling for Task-Relevant RGB-D Perception", *Robotics: Science and Systems*, 2014.

[17] M. Schwarz, H. Schulz and S. Behnke, "RGB-D Object Recognition and Pose Estimation based on Pre-trained Convolutional Neural Network Features", *Proceedings of the IEEE International Conference on Robotics and Automation*, Seattle, 2015.

[18] Z. Wu, S. Song, A. Khosla, L. Z. F. Yu, X. Tang and J. Xiao, "3D ShapeNets: A Deep Representation for Volumetric Shape Modeling", *Proceedings of 28th IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[19] S. Bai, X. Bai, Z. Zhou, Z. Zhang and L. J. Latecki, "GIFT: A Real-time and Scalable 3D Shape Search Engine", *Proceedings of 29th IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[20] S. Gupta, R. Girshick, P. Arbelaez and J. Malik, "Learning Rich Features from RGB-D Images for Object Detection and Segmentation", *European Conference on Computer Vision*, 2014.

[21] Y. Bengio, "Practical Recommendations for Gradient-Based Training of Deep Architectures", *Neural Networks: Tricks of the Trade*, Springer, 2012, pp. 437-478.

[22] L. Bottou, "Stochastic Gradient Descent Tricks", *Neural Networks, Tricks of the Trade, Reloaded*, Srpinger, 2012, p. 430445.

[23] R. Schnabel, R. Wahl and R. Klein, "Efficient RANSAC for Point-Cloud Shape Detection", *Computer Graphics Forum*, **26**(2), pp. 214-226 (2007).

[24] Roded Sharan and Trey Ideker, "Modeling cellular machinery through biological network comparison", *Nature Biotechnology*, **24**(4), pp. 427433 (2006)

[25] Ravi Kumar, Jasmine Novak, and Andrew Tomkins, "Structure and evolution of online social networks", *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006

[26] R. C. Wilson, E. R. Hancock and B. Luo, "Pattern vectors from algebraic graph theory", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **27**(7), pp. 11121124 (2005).

[27] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor and K. M. Borgwardt, "Graph Kernels", *Journal of Machine Learning*, **11**(1), pp. 1201-1242 (2010).

[28] F. Chung, *Spectral graph theory*, American Mathematical Society, 1997.

[29] M. Ferrer, F. Serratosa and A. Sanfeliu, "Synthesis of median spectral graph", *Lecture Notes in Computer Science*, **3523**(1), pp. 139146 (2005).

[30] D. White and R. C. Wilson, "Mixing Spectral Representations of Graphs", *Proceedings of the 18th International Conference on Pattern Recognition*, 2006.

[31] P. Zhu and R. C. Wilson, "Stability of the Eigenvalues of Graphs", *11th International Conference, CAIP 2005*, 2005.

[32] M. F. Demirci, Y. Osmanlioglu, A. Shokoufandeh and S. Dickinson, "Efficient many-to-many feature matching under the l1 norm", *Journal of Computer Vision and Image Understanding*, **115**(7), pp. 976-983 (2011).

[33] P. Riba, J. Llados, A. Fornes and A. Dutta, "Large-scale Graph Indexing using Binary Embeddings of Node Contexts", *Proceedings of the 10th IAPR-TC-15 International Workshop*, 2015.

[34] Changchang Wu, "Towards Linear-time Incremental Structure From Motion", *International Conference on 3DTV*, 2013