Employing graph neural networks for construction drawing content recognition

Andrea Carrara, M.Sc.[1], Stavros Nousias., PhD.[2] , and
André Borrmann, Prof. Dr.-Ing. [3]

[1] Ph.D. Candidate, Chair of Computational Modeling and Simulation Chair, Technical University of Munich, e-mail: andrea.carrara@tum.de

[2] Ph.D., Chair of Computational Modeling and Simulation Chair, Technical University of Munich, e-mail: stavros.nousias@tum.de

[3] Professor, Chair of Computational Modeling and Simulation Chair, Technical University of Munich, e-mail: andre.borrmann@tum.de

ABSTRACT

In this paper, we present a novel method for automatically detecting and classifying lines in technical drawings in PDF format. Our technique includes the processing of technical drawings in vector format and the appropriate transformation to a graph representation facilitating the generation of geometric design features. We achieve semantic segmentation at the line level by leveraging graph neural networks with graph attention layers. The model's efficacy in reading and classifying the delicate details of technical drawings is improved by optimizing hyperparameters using Bayesian hyperparameter optimization.

INTRODUCTION

Despite the growing prevalence of Building Information Modeling (BIM), traditional technical drawings remain the primary mode of communication between architects/engineers and constructors. For many existing buildings, technical drawings serve as the sole source of geometric and semantic information. This paper explores leveraging modern AI techniques to make this content computationally accessible.

In today's widespread practice, drawings are stored and exchanged via PDF, a standardized format capable of transporting vector and pixel information. In the proposed approach, we extract the vector data and reconstruct the semantic information at the line level creating an interconnectivity graph and using Graph Neural Networks (GNN) to learn the relationship between the vectors to recover semantic information of the drawing.

Since vectors are resolution-agnostic, achieving a prominent level of precision on the line elements is the main advantage of using a vector representation directly for semantic segmentation. In contrast, image-based neural networks face limitations such as fixed dimension patches and reduced image-patch quality, impacting intra-image relationships.

A crucial problem with complex drawings is that different architectural parts cross over and overlap with one another. These kinds of intricacies are easily filtered in layered formats, but they are a major problem in the raw, unprocessed PDF format. In this case, choosing a pixel-based methodology would decrease the precision of identifying these overlapping components, highlighting the necessity of more advanced analytical techniques to preserve accuracy.

In our approach, the geometrical data is extracted for each element and transformed into graph features. The drawings' vectors are connected by means of spatially inspired rules that compute geometrical relationships for graph edge feature-based descriptors. The spatial vector descriptor prepares the data for utilization within the GNN architecture. Specifically, we employ a Graph Attention-based Network architecture that consists of two layers of graph attention transformer layers that take as input the feature matrix, the adjacency matrix, and the edge features matrix to perform semantic segmentations using the message passing mechanism in graphs.

We present experiments using real-world data including well-defined metrics for assessing the performance of the developed approach.

RELATED WORK

Due to the information's complexities, automated technical drawing analysis is a particularly complicated task. The intricacy resulting from overlapping elements and symbols, such as text, measurement lines, and design features, and the ways that the same elements appear within the drawings, add to this complexity. Initial approaches employed in academic research to tackle these issues combined rule-based analysis of the drawings with the use of traditional computer vision techniques (Macé et al. 2010), (Ahmed et al. 2011).

Afterward, most of the research tackled the problem as a computer vision task of vectorization and semantic segmentation (Kalervo et al. 2019), (Liu et al. 2017), (Dodge, Xu, & Stenger 2017).

The attention has recently switched to using graph neural networks (GNNs) for element detection and segmentation in technical drawings. The authors of "Generalizing Floor Plans Using Graph Neural Network" (Simonsen et al., 2021) first extract floor plan graphs from CAD primitives. They classify graph nodes using GNNs to differentiate between doors and non-doors. to facilitate general room detection in office buildings. The authors show that employing a graph neural network considerably improves performance over earlier techniques such as those from Dodge, Xu, & Stenger (2017) or the Faster R-CNN (Girshick, R. 2015) baseline.

Fan et al. 2021 and 2022 propose graph neural networks for panoptic segmentation. In addition to detecting countable objects like furniture, windows and doors in CAD drawings, these works seek to establish the semantics of uncountable elements, for instance, walls. The authors propose a novel algorithm that combines GNNs with CNNs (Convolutional Neural Networks) for symbol spotting within the dataset.

Furthermore, Zheng et al. (2022) provide a symbol identification method called "GAT-CADNet: Graph Attention Network for Panoptic Symbol Spotting in CAD Drawings" that approaches the task as a subgraph search problem and leverages attention layers. However, it should be noted that while the studies focus on the analysis of line semantics in CAD drawings using computer vision techniques, they often overlook the prevalent issue of processing the technical drawing in the most popular formats.

Even though PDFs are the most often used format for sharing design data, problems related to their data representation are typically left out of the research. Furthermore, only geometric data from CAD is fed into the graph, hence excluding the potential of feature engineering using line geometric comparisons. This oversight points to a big chance for more research and development on graph neural networks in technical drawings.

METHODOLOGY

Objective. The objective is to automatically classify the elements of the floorplan drawings. The drawing elements will be categorized into distinct classes: *Wall, Filling Material, Opening, Measurement Element*, *Symbol*, and *Other*. For the *Wall* class, we focus exclusively on the external contours of the architectural feature, while the internal fill material is classified separately. The *Opening* class encompasses both doors and windows. *Measurement Elements* include lines indicating dimensions and details pertaining to room layouts. *Symbols* represent various icons found in technical drawings; in our specific analysis of the technical drawing, these primarily included icons for ventilation, electrical, and sanitary fixtures. The *Other* category serves as a catch-all for elements that do not align with the classes or those that appear only once across all the segments.

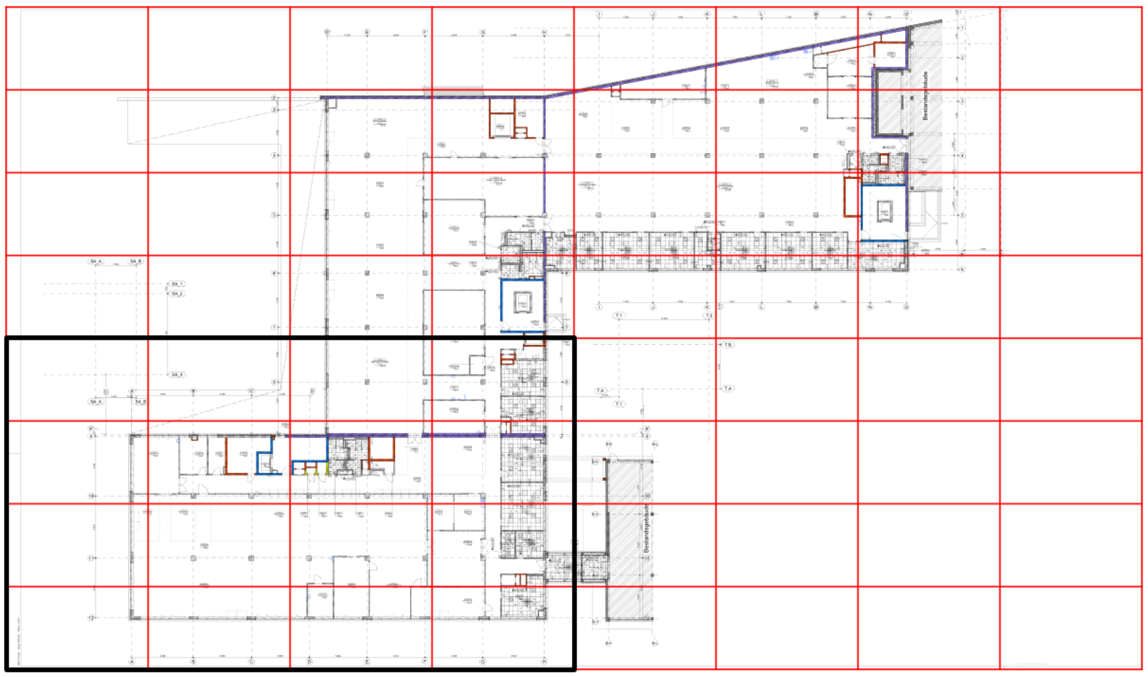

Figure 1. Technical Drawing used for analysis. The black rectangle contains the patches used for training.

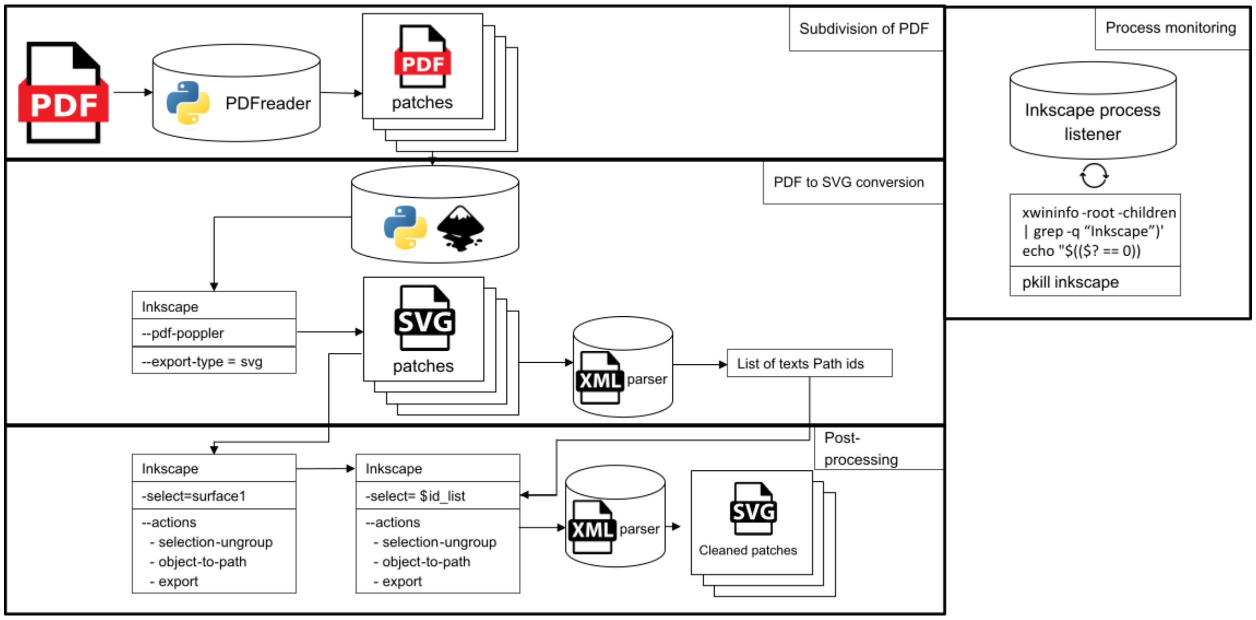

Figure 2. Diagram of conversion from PDF to SVG format

Conversion to SVG. To analyze the content of the technical drawings we developed a methodical approach to obtain Scalable Vector Graphics (SVG) patches from Portable Document Format (PDF) files, allowing for an efficient pipeline for graphics manipulation. The patches are small, distinct sections of the drawing processed or analyzed separately to improve the performance of the deep learning model. The initial phase involves the extraction of content from PDF file format using a PDF library to create different sections. Following the extraction, the patches undergo a transformation to SVG format for data extraction. In the last step, to ensure uniformity and editability, vector components found in the previous phase are ungrouped and turned into geometrical paths. The path is defined as a sequence of commands and attributes, mainly lines and Bezier curves, to represent complex geometries and shapes; instead of being represented as pixels each element is represented by the commands to recreate the geometry in the vector space.

The post-processing step is significant because it guarantees a simpler analysis of the geometric data and simplifies the vector information. The output of PDF-to-SVG converters is sub-optimal for our purposes, because they typically introduce structural complexities in the XML format, such as path grouping and matrix multiplications, which hamper the analysis of the vectors. To overcome this issue, we aim to create plain SVG. To this end, we obtain the plain SVG by applying matrix multiplication for linear transformation to the pathways. Subsequently, any type of lines grouping is removed from the data structure to simplify it and make it easier to process in later stages. It is easier to identify individual elements and their spatial relationships within the drawing when flat, ungrouped vector data is analyzed. The efficiency of our method depends on this simplified representation, which establishes the groundwork for the later use of Graph Neural Networks (GNN) for semantic segmentation.

The path and attributes data required for the graph-based analysis were extracted from the post-processed vector data. Each SVG path is converted to a node in the graph and linked to the neighbors based on the Euclidean distance from the geometrical center of the path. This method is augmented with a radius search that generates edges according to the path length of the nodes, generating interconnectivity for lines of the same length. Similarly, a random sample adds edges to the graph. This ensures that the graph is not only based on geometric proximity but also includes a diversity of path interactions. The three edge generation methods improve the data structure for the Graph Neural Network analysis.

Feature engineering. Feature engineering is an important step in our method. For each node, we create an extensive feature vector that includes normalized coordinates, size of the bounding boxes, and stylistic attributes (fill, stroke color, and stroke width) of the respective SVG element. This information provides insight on the geometric and semantic features of each path and is derived from the SVG style attributes. We additionally compute path closure at different points in area, length, and curvature along the path. We append to the feature vector the SVG command information converted to a fixed dimension tensor following the method defined in Carlier et al. (2020). To fully capture the geometric details of the technical drawing, we compute the edge interactions between the vectors that constitute the edge feature matrix of the graph. Among the features evaluated are the Boolean determination of proximity within a given threshold, vector congruence in length, and the minimum distance between the two pathways. We also assess the paths' contiguity, intersection sites, and uniformity in SVG properties like color, width, and style. We also analyze the inverse ratio of their lengths, the proportionality of their lengths, and the normalized angle between pathways, providing a descriptor of their geometric relationship.

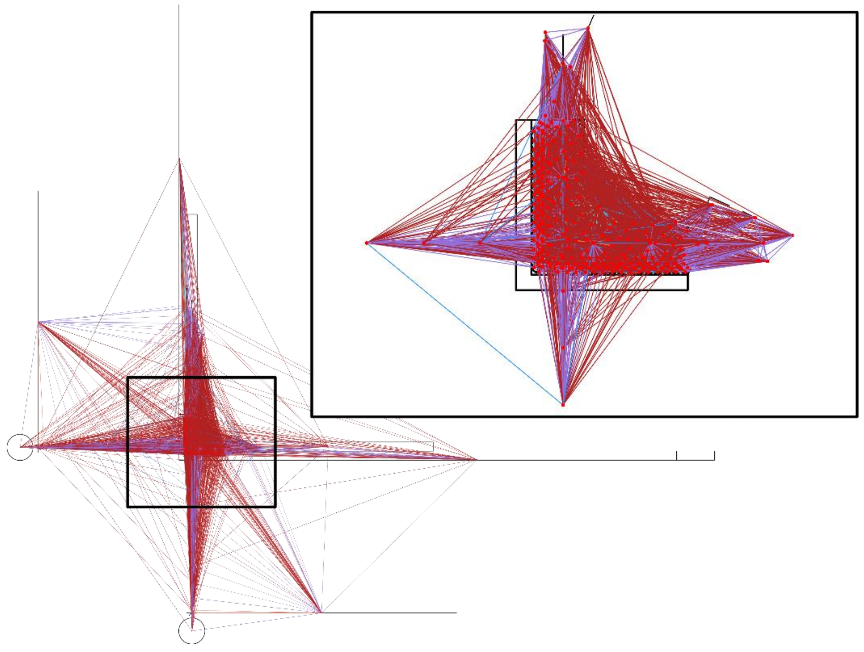

Figure 3. Example of graph from drawing, distinct colors of edges represent a different type of connection.

The drawing was segmented into 64 equally proportioned patches, of which 17 were selectively utilized for training the neural network. This selection process strategically excluded sections of the drawing that exhibited either a sparse concentration of lines or a redundancy in design patterns. These patches contain from 100 to 2800 lines each, on average having 950 lines per patch. The labelling of each training file has been carried out manually using Inkscape and associating each line in a new layer in the SVG file corresponding to the class predicted by the neural network. Unlike other datasets (Kalervo et al. 2019), (Liu et al. 2017), (Dodge, Xu, & Stenger 2017), this training data's main purpose is to use complex representation used in industry standard technical drawings.

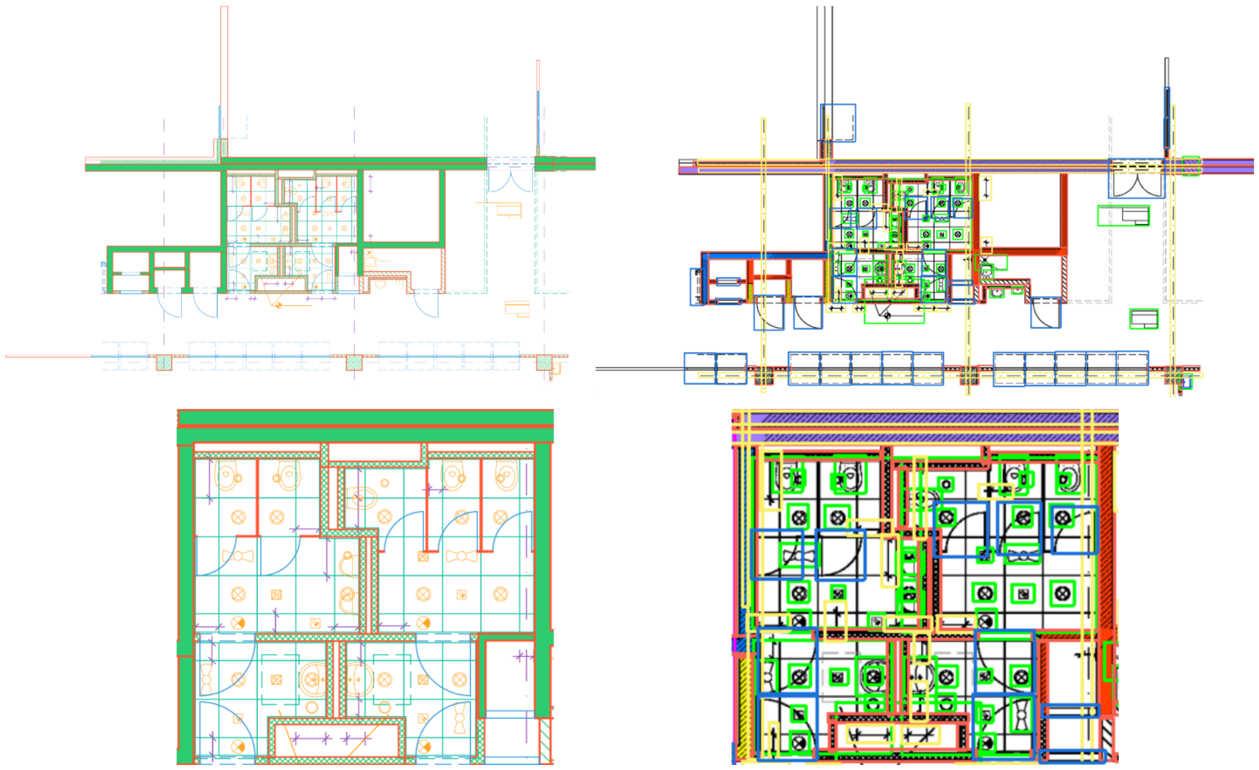

Figure 4. Comparison of labelling between vector labelling and image-based object detection.

Graph Neural Network. The graph, enriched with detailed node and edge features, is then processed using a Graph Neural Network. This network is trained to perform semantic segmentation on the graph, classifying each path into categories based on the learned relationships. The graph neural network algorithm learns the patterns in the graph structure and features. To feed this process, the input data for the deep learning model is formatted specifically for graph-based analysis. This data consists of a list of nodes with their corresponding attributes, an adjacency matrix that describes the relationships between nodes, and feature vectors for each matrix that represent the geometric relationships between nodes.

The model is composed by a sequence of two blocks of Dropout layer combined with Graph Attention v2 layers (Brody, Along & Yahav 2021). The Graph Attention Layer updates each node's features gathering information from its neighbors. This is weighted by adaptively learned coefficients, allowing the network to learn to select the most important neighbors following the formula:

$$\mathbf{x}_i' = \alpha_{i,i}\mathbf{\Phi}_s\mathbf{x}_i + \sum_{j\in\mathcal{N}(i)} \alpha_{i,j}\mathbf{\Phi}_t\mathbf{x}_j$$

The node's feature vector is defined by $\mathbf{x}_i'$ and this refinement is executed through aggregation of data from adjacent nodes, employing aggregation weights defined by attention coefficients.

Considering a node *i* with its feature vector $\mathbf{x}_i'$. The node's intrinsic features undergo a transformation via a learnable weight matrix $\mathbf{\Phi}_s$, and are then scaled by its self-attention coefficient $\alpha_{i,i}$. Parallelly, the feature vectors $\mathbf{x}_i$ of each adjacent node *j* are altered through a distinct learnable weight matrix $\mathbf{\Phi}_t$, and subsequently scaled by the attention coefficient $\alpha_{i,j}$.

The new feature vector $\mathbf{x}_i'$ is the result of the joint sum of these modified features together, including the node and all its neighbors.

$$\alpha_{i,j} = \frac{\exp(\mathbf{a}^\top \mathrm{LeakyReLU}(\mathbf{\Phi}_s\mathbf{x}_i + \mathbf{\Phi}_t\mathbf{x}_j + \mathbf{\Phi}_e\mathbf{e}_{i,j}))}{\sum_{k\in\mathcal{N}(i)\cup\{i\}} \exp\,(\mathbf{a}^\top \mathrm{LeakyReLU}(\mathbf{\Phi}_s\mathbf{x}_i + \mathbf{\Phi}_t\mathbf{x}_k + \mathbf{\Phi}_e\mathbf{e}_{i,k}]))}$$

The degree of influence that every neighbor, including the node itself, has on the updated feature vector is determined mostly by the attention coefficients. Using the properties of both nodes, the attention coefficient $\alpha_{i,j}$ is inferred for a pair of nodes, *i* and *j*. The weight matrices $\mathbf{\Phi}_s$ and $\mathbf{\Phi}_t$of node *i* and its neighbor *j*, respectively, initially alter its attributes. To add non-linearity, these modified features are then combined and passed through a LeakyReLU activation function. Furthermore, the edge characteristics between nodes *i* and *j* are considered by integrating an edge-specific term $\mathbf{\Phi}_e\mathbf{e}_{i,k}$. This combination is translated into a scalar value by a trainable vector *a*, which is then exponentiated. Normalizing these exponentiated values across all the node *i* neighbors (including *i* itself) generates the final attention coefficient, which ensures a cumulative sum of 1.

Bayesian hyperparameter optimization. We perform a Bayesian hyperparameter optimization (Wu et al. 2019), an optimization technique to obtain the best possible sets of parameters for our machine learning task. This method converts the task of hyperparameter tuning into an optimization problem, to find the optimal hyperparameters for the machine learning algorithm. It uses accuracy results from prior hyperparameter definitions to direct the search and applies Bayes' rule to predict performance based on hyperparameter combinations, utilizing a substitute model that represents the probability of performance given the hyperparameters. Based on the predicted improvement, it selects the next hyperparameters.

We run the Bayesian optimization iteratively 350 times, defining priorly a custom set for each individual hyperparameter in the deep learning model and for the optimizer. The objective of the optimization is to identify simultaneously the best model and optimizer parameters; the list is composed of dropout probabilities, epochs, batch size, learning rate, number of input channels of the first attention layer, weight decay and number of attention heads.

k-fold cross-validation. Given the limited size of our dataset for the extensive time required for proper labeling, we adopted a complete analysis method involving k-fold cross-validation. In each iteration of the process, the model is trained on k-1 folds, while the remaining one-fold serves as the validation (or test) set. This cycle is repeated seventeen times (k), each time selecting a different fold for validation, ensuring that every data segment is used exactly once as the test set. We run the cross-validation using the optimal hyperparameters derived from the Bayes Hyperparameters Optimization experiments.

Results

The linear relationship between a hyperparameter and accuracy is evaluated by correlation, an increase in the hyperparameter value usually increases the metric and the reverse is true. Nevertheless, when inputs vary in range (comparing probability and attention heads), it can be difficult to capture complex interplays between them.

Table 1. Results of the Bayesian hyperparameter optimization sorted by "importance" metric

Hyperparameter	Parameter Space	Optimal Value	Importance	Correlation
Probability Dropout 1	[0, 0.1, 0.2, ..., 0.9]	0.3	0.484	-0.39
Epochs	[50, 100, 150, ..., 600]	450	0.127	0.227
Batch Size	[1, 2]	2	0.112	-0.048
Learning rate	[min: 1e-4, max: 1e-3]	0.0004787	0.069	0.173
Probability Dropout 2	[0, 0.1, 0.2, ..., 0.9]	0.3	0.068	0.146
Input Channels	[8, 16, 32, 64, ..., 1024]	1024	0.05	0.204
Weight Decay	[gaussian, μ:5e-4,σ: 1e-5]	0.0005149	0.048	-0.035
Attention heads	[2, 4, 8, 16, 32, 64, 128]	8	0.043	0.04

We compute a significance metric to supplement this. Using the metric as the response variable and the hyperparameters as predictors, a random forest is trained to achieve this. The feature important scores from this random forest are then examined, providing insight into the relative influence.

The most important metrics to optimize are having a low dropout of probability as first layer of the model, increasing the number of epochs and having a low batch size. From the importance analysis the number of attention heads had the least influence on the accuracy metric.

We use the optimal parameters from the Bayesian hyperparameter optimization for training the model.

Given the reduced dimension of the dataset we validate the results with k-fold validation, in which each patch extracted from the technical drawing is used to test the model and the remaining patches are used as training data. The model is trained 17 times in this way to validate the learning capabilities. Qualitative results are presented training the model on all the patches and predicting unseen patches.

Figure 5 shows results of the trained network prediction in an unseen patch. It is possible, for example, to predict lines that are hidden beneath another one like the dot in 4.a that for image-based networks would have been impossible to differentiate.

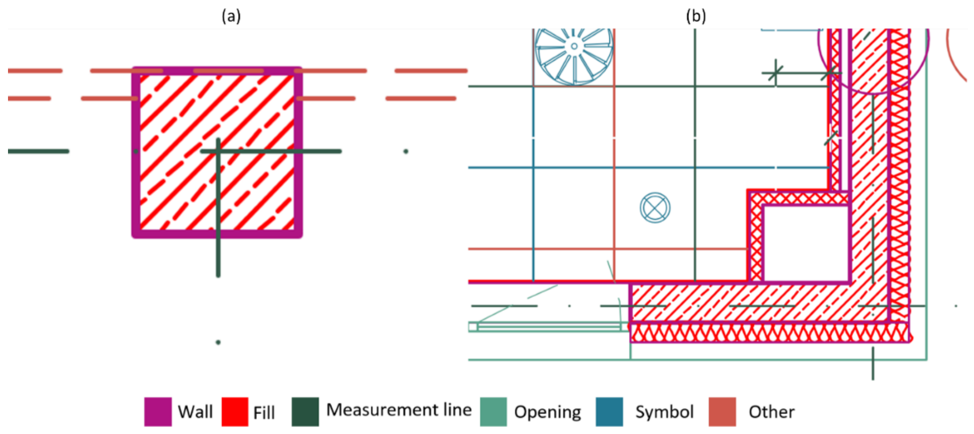

Figure 5. Comparison of labelling between vector labelling and image-based object detection.

Table 2. Results for k-fold-cross validation, k=17

Fold Index	Number of Nodes	Test Accuracy	Precision	Recall	F1
1	101	1.0	1.0	1.0	1.0
2	187	0.989	0.99	0.989	0.989
3	342	0.985	0.986	0.985	0.985
4	405	0.998	0.998	0.998	0.998
5	491	0.992	0.992	0.992	0.992
6	516	0.981	0.982	0.981	0.981
7	589	0.988	0.988	0.988	0.988
8	617	0.989	0.989	0.989	0.989
9	725	0.982	0.983	0.982	0.982
10	775	0.991	0.991	0.991	0.991
11	894	0.993	0.993	0.993	0.993
12	1133	0.967	0.97	0.967	0.967
13	1675	0.916	0.921	0.916	0.916
14	2390	0.849	0.85	0.849	0.833
15	2431	0.994	0.994	0.994	0.994
16	2754	0.973	0.973	0.973	0.973
17	4511	0.869	0.891	0.869	0.875

The model demonstrates consistent performance in the line-wise classification task, with an average accuracy of 0.968, precision of 0.97, recall of 0.968, and an F1 score of 0.967. The precision, recall, and F1 score metrics are computed by averaging the results across individual classes. By weighing the accuracies depending on the amounts of nodes per graph, we obtain the following results for accuracy, precision, recall and F1 score: 0.938, 0.943, 0.937, 0.937.

CONCLUSION

In this paper, we presented a novel method that performs line-level segmentation on a technical drawing from a construction project. From the PDF drawing, we convert to a standard flat SVG format and create a graph extracting feature vectors for the drawing lines and edges connectivity information that connects the elements. We use a deep learning model based on graph attention transformer layers to perform semantic segmentation, fine-tuning the model parameters through Bayes hyperparameter optimization. We validate the predictions with k-cross validation, obtaining a mean accuracy above 0.95. These encouraging outcomes on a smaller dataset set the stage for future research, where our next objective is to test the method efficacy on larger and more diverse datasets. Some of the approach's limitations are the computationally intensive requirements for graph construction and the time-intensive and detail-oriented demands of dataset labelling. These factors significantly diminish the scalability of the dataset and the achievement of enhanced outcomes.

REFERENCES

Simonsen, C. P., Thiesson, F. M., Philipsen, M. P., & Moeslund, T. B. (2021). Generalizing floor plans using graph neural networks. In 2021 IEEE International Conference on Image Processing (ICIP) (pp. 654-658). IEEE.

Dodge, S., Xu, J., & Stenger, B. (2017). Parsing floor plan images. In Proceedings of the Fifteenth IAPR International Conference on Machine Vision Applications (MVA) (pp. 358-361). IEEE.

Macé, S., Locteau, H., Valveny, E., & Tabbone, S. (2010). A system to detect rooms in architectural floor plan images. In Proceedings of the 9th IAPR International Workshop on Document Analysis Systems (pp. 167-174).

Ahmed, S., Liwicki, M., Weber, M., & Dengel, A. (2011). Improved automatic analysis of architectural floor plans. In 2011 International conference on document analysis and recognition (pp. 864-869). IEEE.

Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1440-1448).

Fan, Z., Zhu, L., Li, H., Chen, X., Zhu, S., & Tan, P. (2021). Floorplancad: A large-scale CAD drawing dataset for panoptic symbol spotting. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 10128-10137).

Fan, Z., Chen, T., Wang, P., & Wang, Z. (2022). Cadtransformer: Panoptic symbol spotting transformer for CAD drawings. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 10986-10996).

Zheng, Z., Li, J., Zhu, L., Li, H., Petzold, F., & Tan, P. (2022). GAT-CADNet: Graph attention network for panoptic symbol spotting in CAD drawings. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 11747-11756).

Carlier, A., Danelljan, M., Alahi, A., & Timofte, R. (2020). Deepsvg: A hierarchical generative network for vector graphics animation. Advances in Neural Information Processing Systems, 33, 16351-16361.

Kalervo, A., Ylioinas, J., Häikiö, M., Karhu, A., Kannala, J. (2019). CubiCasa5K: A Dataset and an Improved Multi-task Model for Floorplan Image Analysis. In: Felsberg, M., Forssén, PE., Sintorn, IM., Unger, J. (eds) Image Analysis. SCIA 2019. Lecture Notes in Computer Science(), vol 11482. Springer, Cham. https://doi.org/10.1007/978-3-030-20205-7_3

Liu, C., Wu, J., Kohli, P., & Furukawa, Y. (2017). Raster-to-vector: Revisiting floorplan transformation. In Proceedings of the IEEE International Conference on Computer Vision (pp. 2195-2203).

Brody, S., Alon, U., & Yahav, E. (2021). How attentive are graph attention networks?. arXiv preprint arXiv:2105.14491.

Wu, J., Chen, X. Y., Zhang, H., Xiong, L. D., Lei, H., & Deng, S. H. (2019). Hyperparameter optimization for machine learning models based on Bayesian optimization. Journal of Electronic Science and Technology, 17(1), 26-40.