

Parsing Hierarchical Document Structure with Graph Neural Networks

Thomas Reiser, Veronika Vasileva, Volker Riediger, Jan Jürjens
0009-0007-5452-7800
0009-0006-2024-4653
0000-0003-1461-7442
0000-0002-8938-0470
University of Koblenz
Email: {treiser, vvasileva, riediger, juerjens}@uni-koblenz.de

Alexander Esser, Petra Steiner
0000-0002-5974-2637
Bundesinstitut für Berufsbildung,
Email: {alexander.esser, steiner}@bibb.de

Abstract—Document hierarchy parsing is used to identify hierarchical relationships between layout elements in documents, such as headings, paragraphs, and list items. This information makes documents that are not born-digital easier to navigate for human readers and can support automated processing of these documents. Similar to existing approaches, we insert a sequence of nodes that represent layout elements into a layout tree to incrementally reconstruct the original document hierarchy. Different to existing research, our approach uses a GraphSAGE-based feature extraction network and a deep neural network based scorer to predict parent relationships between layout element pairs using text and layout features. Our data set contains 50 historical German documents and we apply k-fold cross validation with five splits to validate the model's ability to generalize from a small data set to unseen data. The model was able to obtain an average F1-score of over 91% on the selected data set.

I. INTRODUCTION

DOCUMENT HIERARCHY PARSING describes the identification of hierarchical relationships between layout elements like headings and paragraphs in documents [1]. The document hierarchy thus organizes elements like sections and subsections in full documents. An example of such hierarchical relationships in a single-paged document is shown in Figure 1.

A document hierarchy that is known to machines and researchers can support at many tasks, e.g., table of content reconstruction, better document navigation for human readers, or better chunking for retrieval augmented generation with LLMs with a limited number of tokens.

In this paper, we aim to identify the document hierarchy in historical documents given text and layout features that are obtained by a self-trained document layout analysis (DLA) model based on DocLayout-YOLO [2] and machine recognized text in the detected bounding boxes using the Tesseract OCR engine.

Our data set is based on an excerpt of a set of historical vocational education and training (VET) regulations maintained in an occupational archive maintained by the Federal Institute for Vocational Education and Training (BIBB). Because legal documents in particular are frequently organized in sections, subsections and paragraphs, these document corpora represent

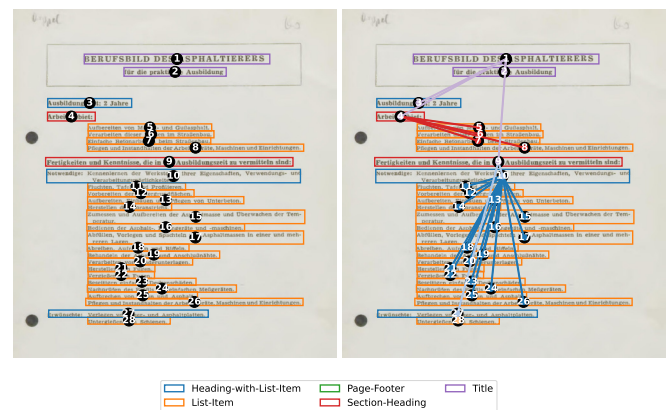


Fig. 1. Example of hierarchical structure in a document layout graph. Outgoing edges are the parent of the element with the ingoing edge. Ingoing edges are visualized by a dot in the center of the bounding box.

a good foundation of hierarchy parsing methods as they often also follow deeper document hierarchies.

A common approach for parsing document hierarchy is the insertion of a node sequence into a layout tree, where pairs of nodes are scored to decide how likely a node in the tree is a parent node of the new inserted node [3, 4]. Our approach also follows this idea but uses GraphSAGE [5] to pass information of neighboring nodes to the parent candidate node in order to help the scorer decide which node is the parent of the next inserted node.

Due to the underlying graph-structure of layout trees, we aim to investigate the effectiveness of graph neural networks in parsing document hierarchy by answering the following research questions:

- RQ1 How can graph-based neighborhood aggregation be used to predict document hierarchy?
- RQ2 Which data set characteristics impact the prediction quality of document hierarchy parsing the most?

Compared to other studies, this does not only utilize node features of candidate parents and siblings, but also their respective parent nodes through a single feature extraction network based on aggregation of neighboring node features

with GraphSAGE rather than a combination of different scores. Another advantage coming from using graph neural networks like GraphSAGE is that nodes up to the number of additional GraphSAGE layers away from the candidate parent can be incorporated without adding further feature extractors for, e.g., grandparents and grandchildren of the candidate parent.

In the example of Figure 1, the order in which unconnected nodes are inserted into the generated tree is given by the reading order, that is visualized by the number in the center of the nodes on the left side of the plot. For each node in the sequence, a new edge is added from the selected parent node to the newly added node, until all nodes are added and a full layout hierarchy tree as seen in the right side of the plot is built.

To decide where nodes are added, a GraphSAGE feature extraction network is used to generate features for nodes that have already been added to the graph and a scorer that scores pairs of candidate parents and new nodes until the tree is completed like seen on the right side of the plot.

The remaining paper is structured as follows: Section II discusses related work and the current status of methods that address the problem of document hierarchy parsing. Section III presents the proposed methodology and the used model architecture. In Section IV, the results are presented and discussed using F1-score and describing common errors found in the predictions of our model. Finally, Section V summarizes our findings, followed by a critical discussion of our methods and an outlook for future research.

Our contributions are a new model architecture to extract features from document layout graphs and a new scorer architecture for scoring pairs of layout elements to decide where new nodes are added.

II. RELATED WORK

Document understanding is a broad research field that revolves around methods like document layout analysis [6, 7, 8] which describes methods that identify and classify layout elements in document images, named entity recognition [9, 10], aiming to extract information about, e.g., names and places in texts, optical character recognition (OCR) [11] to extract text from images, and form understanding [12, 13], used for question answering in forms like receipts or questionnaires.

Hierarchical structure analysis on the other hand focuses more on the logical and hierarchical structure of documents that have been previously processed with, for example, document layout analysis and OCR and frequently builds on top of these results. In [3], the Hierarchy Extraction from Long Document (HELD) model was introduced, where nodes are sequentially inserted into a tree structure. The decision where a new node is inserted is done by a put-or-skip module which computes the probability of the new node being inserted on a parent node of the graph under construction.

In [4], the Detect-Order-Construct approach is introduced that combines document layout analysis with reading order reconstruction followed by the construction of the document hierarchy tree. Similar to [3], they restrict candidate parents



Fig. 2. Features of the layout elements

to the rightmost path of the constructed graph. But instead of using a put-or-skip module, they utilize a Construct module which organizes a sequence of identified section headings. A relation prediction head scores how likely a new node is either a parent or a sibling of a candidate node in the table of contents tree. These scores are then used for a multi-class classification problem with softmax cross-entropy to select the parent node with highest score.

DocHieNet [1] is a large document hierarchy parsing data set that includes a variety of layouts rather than being reduced to a small specific domain like, e.g., PubLayNet [6] which focuses only on medical papers. To test their data set, they adapted the HELD approach with the put-or-skip module to compute probabilities where a newly added node must be inserted into the hierarchy tree.

Document hierarchy is also relevant for form understanding tasks. DocStruct [14] for example uses a multimodal method for form understanding to recognize the organizational structure of forms. Similar to parent-child relations in common document hierarchy parsing tasks, they define key-value pairs in forms. They use text, layout and image features to then compute a score that represents the probability of two layout elements having a key-value relationship.

The usage of graph neural networks (GNNs) for document understanding has also already been investigated. Paragraph2Graph [15] is a framework that uses GNNs for different document understanding tasks which is able to encode features at line-level but also encode recognized information in a graph using task-specific layers while not relying on language-specific encoders.

III. METHOD

In our approach, we encode page number, reading order, text, bounding box information and class features of layout elements and incrementally build a hierarchical document structure tree as seen in Figure 1 by inserting a sequence of nodes in their respective reading order. Figure 2 shows how the positional and textual features are represented. The green box around the text illustrates the bounding box that was previously detected by a layout analysis model. The x and y values represent the top left position of the input bounding box, whereas the h and w values represent height and width of the bounding box.

For each node, a set of candidate parents is selected, i.e., the rightmost path of the document tree at the stage where the next node is inserted. Our method assumes there is one layout element that can be considered to be the root element

for the document and all other layout elements have only one parent element. Similar to other works [3, 4], we only allow insertion into the rightmost path of the existing tree starting with a root node which is assumed to be the title element of the document. This ensures that nodes cannot be inserted into a section of the document that has already been completed in the tree.

The process can be summarized in four steps:

- 1) Select candidate parents for new node, i.e., the rightmost path
- 2) Extract features from candidate nodes and new node
- 3) Score pairs of candidate parent nodes and new node
- 4) Select parent with highest scoring pair

Due to size limitations in our data set, which consists of only 50 documents, we use k-fold cross validation with five splits and shuffling. This means, our data set is shuffled to reorder the documents in it and then divided into five equally-sized splits. Afterwards, our model is trained five times where four of these splits are used for training and the remaining one for testing. In the end, model performance is assessed using F1-score on predicted and true edges.

A. Data Set

For the training and evaluation of our method, a data set was manually annotated. The data set consists of 50 documents with two to 15 pages containing between 23 and 218 layout elements. Each document is structured in a graph structure with a set of nodes and edges. Nodes are represented by layout elements and edges which define the relationships between these layout elements. The left plot in Figure 3 shows the distribution of documents by the number of pages in our data set, the middle plot shows the distribution of layout elements by the number of pages, and the right plot the frequency of the deepest hierarchy level in each document. Here, the deepest hierarchy is defined as the length of the longest path from the root element to any leaf element.

The layout elements consist of page number, reading order position, bounding box, their text, and a class which was assigned by a document layout analysis model. Edges on the other hand are directed and each outgoing edge represents a parent relationship to the target node of that edge.

The bounding boxes are defined by horizontal and vertical position of their top left corner as well as width and height. These are normalized to a range between 0 and 1, representing relative position in the original document image. The same normalization was applied to page number and reading order position. Layout classes are one of 13 classes assigned during document layout analysis on given documents. These classes include section headings, paragraphs, list items, page headers, and several other classes. However, our approach is not limited to these classes and designed to allow different classes in future research. Nevertheless, the selection of layout classes will impact the document structure result because section headings, for example, are likely to be followed by a new paragraph. Floating text layout elements that are on the page images, but not part of the main text, like page headers, page

numbers, etc. are ignored for the hierarchy construction task and filtered out before training and inference, using the classes that were previously assigned by the DLA model.

A visualization of used features can be found in Figure 2.

B. Model Architecture

Our model consists of three components: A BERT encoder, based on XLM-RoBERTa [16] for text feature generation, a graph neural network based on GraphSAGE [5] for feature generation on the existing graph where new nodes are inserted, and a scorer that decides which node is most likely the parent node of a newly inserted node.

The original text is extracted from the document images using Tesseract OCR [17]. The XLM-RoBERTa encoder was selected because of its ability to handle German language due to its multilingual support.

The positional features are created using the bounding box coordinates from a document layout analysis (DLA) model that predicts the positioning on the original image. The layout classes are also predicted by the same DLA model but treated as a separate feature.

For the text feature encoder, we keep the pretrained weights. The feature extraction network and the pair scorer are described in the following sections.

1) *Feature Extraction Network:* The feature extraction network is visualized in Figure 4.

In the first step of the feature extraction network, textual and positional features are both processed by a linear layer, followed by a layer norm and a ReLU activation function. The class features are processed by an embedding layer and a layer norm. Embedding layers are particularly useful for categorical information to learn similarity between classes. For the given documents for example, paragraphs and list items are more similar than paragraphs and section headings because paragraphs are less likely to start a new section than section headings. For the non-categorical features, a ReLU activation function is applied before the resulting feature vectors are stacked in the fusion layer.

After feature fusion where the extracted features are concatenated, messages from neighboring nodes are passed through a GraphSAGE layer with an LSTM aggregator.

GraphSAGE is a special kind of graph neural network (GNN) that is able to learn graph structures inductively [5]. This enables these networks to learn the structure of unseen graphs, making them particularly useful in tasks like document hierarchy parsing where new graphs are built incrementally. In these networks, features of neighboring nodes are aggregated by an aggregation function that is learned during model training.

For this paper, an LSTM aggregator has been selected over mean or other convolutional aggregators because it can better learn relationships between different neighboring nodes, which can be particularly helpful for document hierarchy trees where relationships between sibling nodes can also exist. However, for GraphSAGE, the order of nodes is not known because nodes in a graph are not ordered by default. Therefore, a

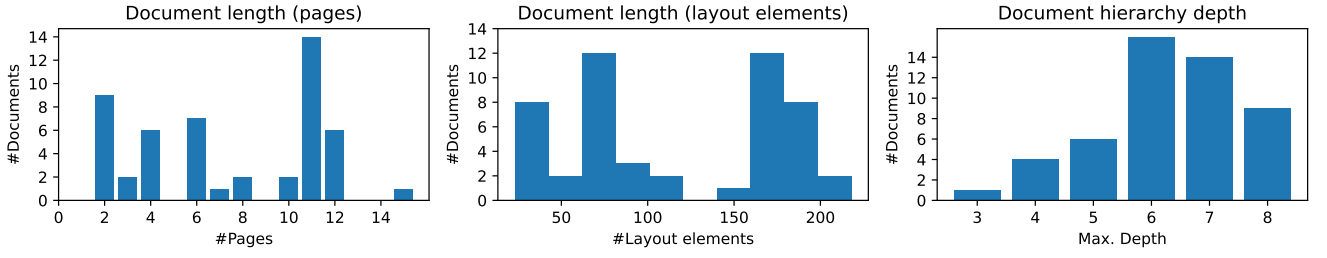


Fig. 3. Quantitative description of the used data set: Document length in number of pages, number of layout elements (nodes) and maximum hierarchy depths.

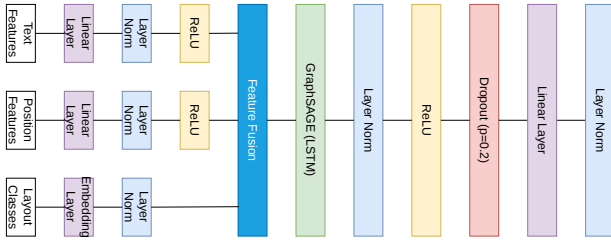


Fig. 4. Architecture of the proposed feature extraction network.

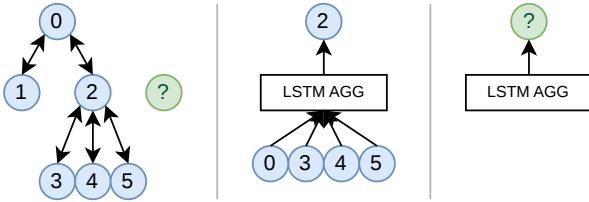


Fig. 5. Message passing on an example graph (left) for node 2 (middle) and node ? (right) in the GraphSAGE layer.

random sequence of nodes is used by the LSTM aggregator. The GraphSAGE layer passes information from the candidate's parent and children nodes to the candidate parent, as shown in Figure 5. In the given example, the disconnected green node has no messages to pass as it has no ingoing edges.

A single GraphSAGE layer was selected in particular to pass information from all children that were already added to a node and should help the pair scorer decide where the next node is inserted. In the rightmost tree of Figure 7, messages from nodes 0 and 3 would be passed to node 2 when $score(2, ?)$ is calculated.

This layer is then followed by another Layer Norm and ReLU layer before a dropout layer that is used to avoid overfitting. The dropout layer deactivates 20% of all neurons in the network randomly for the upcoming computation to avoid reliance on a few single nodes when making predictions.

In the last step, another linear layer is added, followed by a layer norm. Because cross entropy loss was used which applies softmax activation anyways, no other activation is applied afterwards.

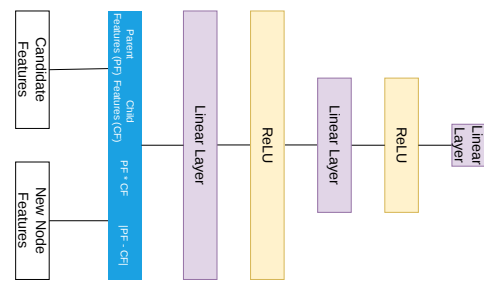


Fig. 6. Architecture of the pair scorer. The candidate features and new node features are both generated from the feature extraction network.

2) *Pair Scorer*: The pair scorer evaluates pairs of nodes to decide which node in the already existing layout tree is the parent of the next node in the insertion sequence. It takes the candidate parent's features, extracted by the feature extraction network, and the features of the next node. The node with highest score is then selected to be the parent of the new node.

Architecturally, the pair scorer is a deep neural network with three linear layers followed by a ReLU activation function each. The input of this network is the concatenation of the extracted parent and child features, as well as their product and the absolute value of their difference. This combination was selected to also include relative information between parent and child features.

Therefore, the first linear layer has four times the number of the feature extraction network's output dimension neurons. In the second layer, this is cut in half to two times the number of output neurons. In the final layer, this dimension is further reduced to a single scalar value.

The overall network architecture for the pair scorer is sketched in Figure 6.

C. Training and Inference

To simulate the incremental construction of a graph at training time, for each graph in the annotated data set, all prefix graphs are computed. In particular, the training data set consists of all subtrees of the document trees in the annotated data set where all nodes up to a certain reading order position are added. Because this problem is trivial for prefix graphs

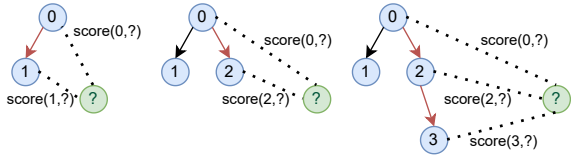


Fig. 7. Visualization of the training and prediction process. Red arrows mark the rightmost path containing the candidate parents. Blue nodes have already been assigned to the document tree. Green nodes with question mark are newly assigned to the document tree. Dashed lines mark pairs that are scored for classification.

containing only one node, those cases were removed from the training data set.

Examples of the prefix graph and new nodes that are added in each larger graph are visualized in Figure 7. During training, the GraphSAGE network learns which features to generate, whereas the scorer learns how to combine these features to generate higher scores for the correct parent.

Although edges are directed when it comes to sorting nodes in reading order, during training and inference, they are considered as undirected. This is particularly important for the message passing in the GraphSAGE network, so children that are already connected to the parent pass their information to it when a new node is added, ensuring they impact the decision for newly added nodes.

Only nodes on the rightmost path of the already constructed layout tree are considered as candidate nodes because we assume that once a new section at the same level started, all previous sections are completed. The rightmost path restriction models this assumption. To encode features on both the existing graph and the unseen node, the new node is added to the graph without any edges. Then, the pair scorer evaluates pairs of the nodes on the rightmost path. Cross entropy loss is used during training because the model is optimizing a multi-class classification problem with only one single class being the correct one for the new node. The candidate nodes correspond to the possible classes to which the new node can be assigned. The highest-scoring pair then indicates the two nodes between which the new edge is added.

On prediction time, the layout tree starts with the first node in the insertion sequence and then adds nodes sequentially, using the parent assignment of the trained models. A new link between each of the models is added after the parent node has been selected using the pair scorer.

IV. RESULTS

We applied k-fold cross validation with five splits. Each split contained 40 training documents and 10 test documents that the model has not seen during training.

In each test run, the number of edges that were both in ground truth and predictions (true positives), the number of edges that were in the predictions but not in the ground truth (false positives), the number of edges that were in the ground truth but not in the predictions (false negatives) and the number of edges that were neither in the predictions nor the ground

TABLE I
MACRO (PER-TREE AVERAGE) AND MICRO (PER-EDGE) F1-SCORES OF EACH SPLIT SPLIT (IN %).

Split	1	2	3	4	5	Mean
Micro						
GraphSAGE	92.13	94.34	90.21	87.95	92.82	91.49
Linear	91.78	91.78	88.43	87.66	88.45	89.62
Ablation	93.09	89.43	86.98	82.52	88.05	88.05
Macro						
GraphSAGE	94.26	91.86	90.85	87.63	92.02	91.32
Linear	93.88	89.33	88.15	88.11	85.29	88.95
Ablation	94.94	88.08	88.56	82.16	87.95	87.95

truth (true negatives) is counted to then compute F1-score on each of the test splits.

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

$$\text{F1-score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Precision depends on the number of true positives, recall on the number of false negatives. In our case, each false positive also implies a false negative since every edge has only one chance to be detected by the model, so that precision = recall in our particular approach. Therefore, we report results using the F1-score, the harmonic mean of precision and recall because it provides a standard metric that reflects both precision and recall which are equal in our context.

For better comparability, the experiment was conducted in 2 additional setups: We replaced the GraphSAGE layer by a linear layer of same dimensionality (Linear setup) and removed it entirely (Ablation setup). The results are shown in Table I. Bold printed numbers indicate the best result in each split and on average. The micro results refer to the evaluation on all edges in the split, the macro results refer to the average per-tree F1-score in the split.

While each of these other two architectures was able to outperform the GraphSAGE variant in singular splits, the GraphSAGE architecture still has the highest average micro and macro F1-score by roughly 1.8% for micro and 2.3% on macro, showing its better ability to generalize from the training data. It remains to be investigated how this difference changes on larger and more heterogeneous data sets.

However, since measuring only the recognition of all edges in a split does not necessarily give enough attention to the quality of individual document trees (edges that are added early are arguably easier to add), true and false positives and negatives are also reported. Both evaluations are combined because a single short document with low F1-score can heavily decrease the average of the F1-scores, even if all other trees were predicted successfully. The results of average F1-scores per tree in each split are also shown in Table I.

The average F1-score per tree in each test split is 91.32%. When counting all edges in each test split, the average F1-

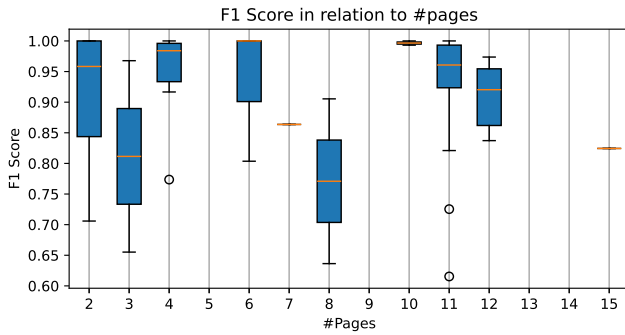


Fig. 8. Distribution of F1-scores for documents with different number of pages.

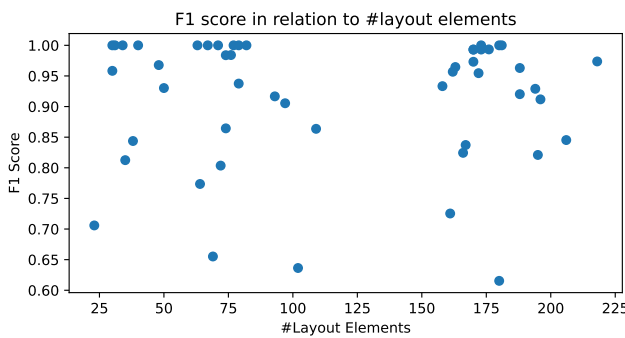


Fig. 9. Relationship between number of layout elements and F1-score.

score is 91.49%. This indicates that the high performance is not due to a few large trees with many correctly predicted edges, but that the model generally reconstructs the layout hierarchy trees well. Also, the model successfully learned to build layout trees on unseen data with overall high reliability. However, there were overall high fluctuations in the results, as seen for example in Figure 9 where each tree-wise F1-score is plotted on the y-axis and the document length measured in number of layout element is shown on the x-axis.

This plot also shows that the overall prediction quality is not directly influenced by document length, as also seen in the boxplot in Figure 8 which shows the distribution of F1-scores in relation to the number of pages in a document. While there are strong fluctuations in the average F1-score between different document lengths, these are expected due to the limited data set size.

There are only few predictions with low F1-score for both long and short documents and there seems to be no obvious relation between number of nodes and F1-score or number of pages and F1-score. Across all test splits, all but six of the 50 predicted trees achieved an F1-score of more than 80%, as seen in Figure 9. Even the document with the most nodes achieved a high F1-score of over 95%.

The relationship between depth of the document hierarchy and the F1-score is plotted in Figure 10. The single document that had a maximum hierarchy depth of three had an F1-

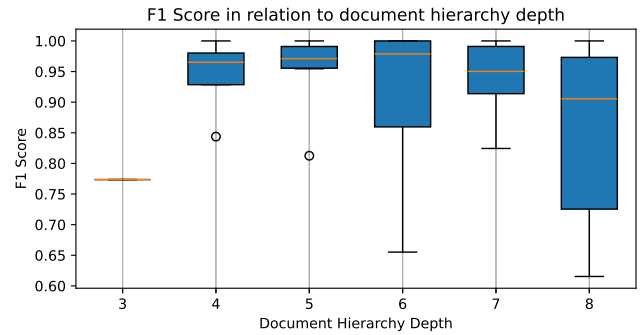


Fig. 10. F1-scores for all predictions per maximum hierarchy depth.

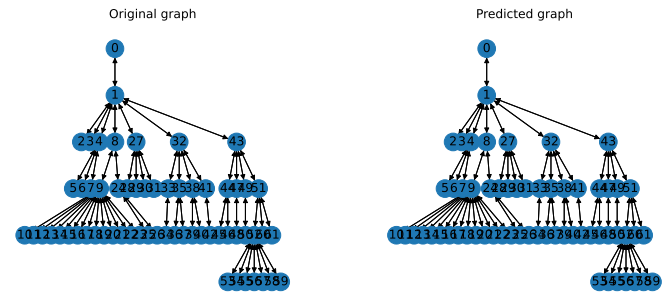


Fig. 11. Example of a tree in the ground truth compared to the predictions by the developed model (F1-score: 100%).

score of only 77%. Reason for this might be the lack of examples in the training data set and the model not being able to generalize from documents with deep hierarchies to flat ones. For documents with four to seven hierarchy levels, the average score was over 90%. While even the documents with deepest hierarchy have an average score of around 85%, there was a lot of inconsistency in the result of these trees compared to the less deep trees.

Other than the overall document length, it can be observed that deeper hierarchy causes more inconsistencies, as seen in the box plot. Because deeper hierarchies imply more possible classes for the given classification problem, F1-scores also more frequently decrease for these.

An example of a successfully predicted tree compared to its ground truth is shown in Figure 11. In this example, the basic document structure is captured, with the same nodes introducing new sections in both the ground truth and the prediction.

In example of Figure 12, the model failed to capture the actual layout tree depth. While many of the central nodes that introduce a new layer have been recognized correctly in the upper layers of the layout tree, errors like this can either propagate after a few wrong insertions or are just not fully learned by the model. This tree is also one of the challenging examples in the data set, where a paragraph is the parent node of following list items, which rarely occurs in the data set and is part of a data imbalance problem. This issue can likely be addressed by more similar examples in a larger data set.

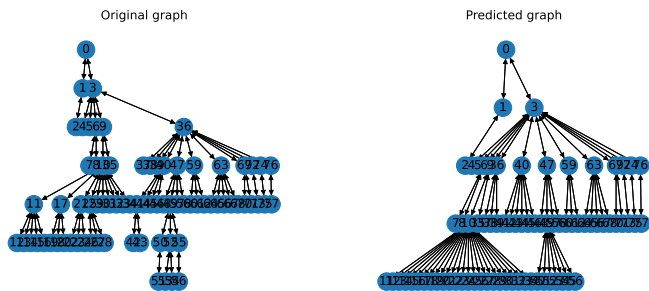


Fig. 12. Example tree where hierarchy prediction did not work as expected (F1-score: 63.6%).

Nevertheless, this would likely be a minor issue for readers and is also a consequence of ambiguous ground truth, also for humans.

This example also illustrates another challenge for document hierarchy parsing tasks: Well documented annotation guidelines are necessary for annotators because document hierarchy can be ambiguous, even for human readers. One could also argue that list items and paragraphs are on a same level in the hierarchy tree rather than a paragraph introducing a list item, no matter the layout format in the image.

V. CONCLUSION

This section discusses the contribution and findings of this paper as well as current limitations and future research.

A. Summary

We proposed a neural network architecture that is able to insert a sequence of nodes into a document layout tree to parse a document’s logical hierarchy. Our approach includes a GraphSAGE-based network that is used to extract meaningful features from an incrementally built layout tree and a pair scorer that takes candidate parent nodes and the next inserted node as input to find the parent of the new node. Both text and layout features have been used, including reading order, position on the document image and page number.

The model obtained an average F1-score of over 91%. We used k-fold cross validation with five splits to validate the model’s ability to generalize from different training sets, indicating that the model is able to learn successfully on the given document corpus.

Our neural network architecture was used to answer RQ1 by showing that the model successfully learned to predict the hierarchy relationships in a small test data set.

To answer RQ2, different data set characteristics have been investigated to identify relationships to prediction quality. While document length seems to have a lower impact on prediction quality, deeper document hierarchies are more difficult to reconstruct for the model as hierarchy depth is what actually determines the difficulty of the underlying classification problem. However, it remains to be tested on larger data sets with more layout variance across documents.

B. Limitations

While the proposed approach gave reasonable results on the given data set, there are some limitations that need to be considered. First of all, the data set consists of only 50 documents with similar structure. Future work should address the approach with more heterogeneous data for a stronger assessment of the approach’s reliability.

Due to the creation of prefix graphs, with two or more nodes at training time, the amount of training data increases fast. For example, a graph in the ground truth that has 200 layout elements will create subgraphs with two to 200 nodes before training, making it 199 additional subgraphs during training. For the same reason, smaller subgraphs are largely overrepresented during training and harder graphs with more nodes or deeper hierarchies might be underrepresented. For larger text corpora, the prefix graphs from training should be sampled in a balanced way.

Aside from this, other metrics like graph edit distance might also be considered during model evaluation because high-degree nodes, i.e. nodes with many outgoing edges in the ground truth, will have larger impact on precision and recall.

Additionally, layout classes during the layout analysis phase should be selected carefully to give meaningful information to the model. Lastly, we assume that there is a single root layout element for the entire document which might not always be true for all document corpora. Adding an artificial root node might resolve this limitation in future research.

C. Future Work

Future work may include the addition of image features to the model architecture which have been omitted to reduce network complexity. Alternatively to the hand-crafted features that are given to the feature extraction network, other multimodal encoders like LayoutLMv3 [18] could add more meaning to the features, further improving model performance. It also remains to test the model architecture on longer documents. This work focused on shorter documents and not on hundreds of pages like for example the HELD model [3], so the model’s performance on longer documents also remains to be investigated. Additionally, the output graphs also need to be encoded in a more standardized way, for example in the XML format of the Text Encoding Initiative (TEI) [19]. Because the model is targeted at working in automated document image processing pipelines, it also remains to see how well the model can handle noisy predictions of the DLA model that detects and classifies the layout elements’ bounding boxes.

ACKNOWLEDGMENT

During the preparation of this work, the authors used ChatGPT for generating the initial program code of the two designed network architectures in the experiment and code debugging. The code was reviewed and edited to match the described method. The authors take full responsibility for the content of this publication.

REFERENCES

- [1] H. Xing, C. Cheng, F. Gao, Z. Shao, Z. Yu, J. Bu, Q. Zheng, and C. Yao, "DocHieNet: A large and diverse dataset for document hierarchy parsing," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024. doi: 10.18653/v1/2024.emnlp-main.65 pp. 1129–1142. [Online]. Available: <https://aclanthology.org/2024.emnlp-main.65/>
- [2] Z. Zhao, H. Kang, B. Wang, and C. He, "Doclayout-yolo: Enhancing document layout analysis through diverse synthetic data and global-to-local adaptive perception," 2024. [Online]. Available: <https://arxiv.org/abs/2410.12628>
- [3] R.-Y. Cao, Y.-X. Cao, G.-B. Zhou, and P. Luo, "Extracting variable-depth logical document hierarchy from long documents: Method, evaluation, and application," *Journal of Computer Science and Technology*, vol. 37, no. 3, pp. 699–718, 2022. doi: 10.1007/s11390-021-1076-7. [Online]. Available: <https://doi.org/10.1007/s11390-021-1076-7>
- [4] J. Wang, K. Hu, Z. Zhong, L. Sun, and Q. Huo, "Detect-order-construct: A tree construction based approach for hierarchical document structure analysis," *Pattern Recognition*, vol. 156, p. 110836, 2024. doi: 10.1016/j.patcog.2024.110836. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0031320324005879>
- [5] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018. [Online]. Available: <https://arxiv.org/abs/1706.02216>
- [6] X. Zhong, J. Tang, and A. Jimeno Yepes, "Publaynet: Largest dataset ever for document layout analysis," in *2019 International Conference on Document Analysis and Recognition (ICDAR)*, 2019. doi: 10.1109/ICDAR.2019.00166 pp. 1015–1022.
- [7] B. Pfizmann, C. Auer, M. Dolfi, A. S. Nassar, and P. Staar, "Doclaynet: A large human-annotated dataset for document-layout segmentation," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '22. New York, NY, USA: Association for Computing Machinery, 2022. doi: 10.1145/3534678.3539043. ISBN 9781450393850 p. 3743–3751. [Online]. Available: <https://doi.org/10.1145/3534678.3539043>
- [8] M. Li, Y. Xu, L. Cui, S. Huang, F. Wei, Z. Li, and M. Zhou, "DocBank: A benchmark dataset for document layout analysis," in *Proceedings of the 28th International Conference on Computational Linguistics*, D. Scott, N. Bel, and C. Zong, Eds. Barcelona, Spain (Online): International Committee on Computational Linguistics, Dec. 2020. doi: 10.18653/v1/2020.coling-main.82 pp. 949–960. [Online]. Available: <https://aclanthology.org/2020.coling-main.82/>
- [9] J. Li, A. Sun, J. Han, and C. Li, "A survey on deep learning for named entity recognition," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 50–70, 2022. doi: 10.1109/TKDE.2020.2981314
- [10] Y. Wang, H. Tong, Z. Zhu, and Y. Li, "Nested named entity recognition: A survey," *ACM Trans. Knowl. Discov. Data*, vol. 16, no. 6, Jul. 2022. doi: 10.1145/3522593. [Online]. Available: <https://doi.org/10.1145/3522593>
- [11] R. Avyodri, S. Lukas, and H. Tjahyadi, "Optical character recognition (ocr) for text recognition and its post-processing method: A literature review," in *2022 1st International Conference on Technology Innovation and Its Applications (ICTIIA)*, 2022. doi: 10.1109/ICTIIA54654.2022.9935961 pp. 1–6.
- [12] G. Jaume, H. K. Ekenel, and J.-P. Thiran, "Funsd: A dataset for form understanding in noisy scanned documents," 2019. [Online]. Available: <https://arxiv.org/abs/1905.13538>
- [13] A. Abdallah, D. Eberharter, Z. Pfister, and A. Jatowt, "Transformers and language models in form understanding: A comprehensive review of scanned document analysis," 2024. [Online]. Available: <https://arxiv.org/abs/2403.04080>
- [14] Z. Wang, M. Zhan, X. Liu, and D. Liang, "DocStruct: A multimodal method to extract hierarchy structure in document for general form understanding," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, T. Cohn, Y. He, and Y. Liu, Eds. Online: Association for Computational Linguistics, Nov. 2020. doi: 10.18653/v1/2020.findings-emnlp.80 pp. 898–908. [Online]. Available: <https://aclanthology.org/2020.findings-emnlp.80/>
- [15] S. Wei and N. Xu, "Paragraph2graph: A gnn-based framework for layout paragraph analysis," 2023. [Online]. Available: <https://arxiv.org/abs/2304.11810>
- [16] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov, "Unsupervised cross-lingual representation learning at scale," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020. doi: 10.18653/v1/2020.acl-main.747 pp. 8440–8451.
- [17] R. Smith, "An overview of the tesseract OCR engine," in *International Conference on Document Analysis and Recognition (ICDAR)*, vol. 2, 2007. doi: 10.1109/ICDAR.2007.4376991 pp. 629–633.
- [18] Y. Huang, T. Lv, L. Cui, Y. Lu, and F. Wei, "Layoutlmv3: Pre-training for document ai with unified text and image masking," 2022. [Online]. Available: <https://arxiv.org/abs/2204.08387>
- [19] Text Encoding Initiative Consortium, "TEI P5: Guidelines for Electronic Text Encoding and Interchange," <https://tei-c.org/guidelines/>, 2023, version P5.