BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

NEURAL NETWORKS FOR AUTOMATIC TABLE RECOGNITION

AUTOMATICKÉ ROZPOZNÁVÁNÍ TABULEK POMOCÍ NEURONOVÝCH SÍTÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR AUTOR PRÁCE	**Bc. LUKÁŠ PIWOWARSKI**
SUPERVISOR VEDOUCÍ PRÁCE	**Ing. MICHAL HRADIŠ, Ph.D.**

BRNO 2022

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2021/2022

Master's Thesis Specification

24864

Student:	**Piwowarski Lukáš, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Computer Vision
Title:	**Neural Networks for Automatic Table Recognition**
Category:	Image Processing

Assignment:

1. Familiarize yourself with convolutional and graph neural networks.
2. Survey the state-of-the-art in automatic table recognition including existing datasets, task formulations and evaluation.
3. Select or suggest a method that can automatically process tables from images.
4. Prepare a dataset suitable for experiments. You can adapt, extend or update an existing database.
5. Implement the proposed method and perform experiments on the dataset.
6. Compare the achieved results and discuss the possibilities of future development.
7. Create a concise video presenting your work, its goals and results.

Recommended literature:

- Harsh Desai, Pratik Kayal and Mayank Singh TabLeX: A Benchmark Dataset for Structure and Content Information Extraction from Scientific Tables. ICDAR, 2021.
- Pratik Kayal, Mrinal Anand, Harsh Desai, Mayank Singh. ICDAR 2021 Competition on Scientific Table Image Recognition to LaTeX, ICDAR 2021.
- Jose Ramón Prieto and Enrique Vidal: Improved Graph Methods for Table Layout Understanding. ICDAR, 2021.

Requirements for the semestral defence:

- Body 1 až 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Hradiš Michal, Ing., Ph.D.**
Head of Department:	Černocký Jan, doc. Dr. Ing.
Beginning of work:	November 1, 2021
Submission deadline:	May 18, 2022
Approval date:	May 3, 2022

Abstract

This thesis introduces the reader to the current table recognition techniques mainly used to extract information from historical handwritten and printed tables. We also introduce a method based on graph neural network, which is inspired by the presented techniques. The method consists of three phases: graph initialization, node/edge classification and graph to text transformation phase. In the graph initialization phase, we use the node visibility algorithm and OCR to create an initial graph representation of the input table. In the node and edge classification phase, the nodes and edges are classified, and in the graph to text transformation phase, we fit the graph's nodes into a grid which is then used to produce the final text representation of the table. The implemented model achieved horizontal neighbours detection precision of 68 %, vertical neighbours detection precision of 71 % and cell detection precision of 85 % on the ABP dataset.

Abstrakt

Tato práce seznamuje čtenáře se současnými technikami rozpoznávání tabulek, které se používají především k získávání informací z ručně psaných nebo tištěných historický tabulek. Představujeme také metodu založenou na grafové neuronové síti, která je inspirována představenými přístupy. Metoda se skládá ze tří fází: fáze inicializace grafu, fáze klasifikace uzlů/hran a fáze transformace grafu na text. Ve fázi inicializace grafu používáme algoritmus viditelnosti uzlů a OCR k vytvoření počáteční grafové reprezentace vstupní tabulky. Ve fázi klasifikace uzlů a hran jsou uzly a hrany klasifikovány a ve fázi transformace grafu na text zarovnáváme uzly grafu do mřížky, která je pak použita k vytvoření konečné textové reprezentace tabulky. Náš implementovaný model byl schopen dosáhnout přesnosti 68 % u detekce horizontálních sousedů, přesnosti 71 % u detekce vertikálních sousedů a přesnosti 85 % u detekce buněk na datové sadě ABP.

Keywords

table recognition, graph neural network, Transformer neural network, edge discover, node discovery, optical character recognition, table recognition datasets, graph initialization, table recognition evaluation

Klíčová slova

rozpoznávání tabulek, grafové neuronové sítě, neuronová síť Transformer, nalezení hran, nalezení uzlů, opticke rozpoznávání znaků, inicializace grafu, hodnocení rozpoznávání tabulek

Reference

PIWOWARSKI, Lukáš. *Neural Networks for Automatic Table Recognition.* Brno, 2022. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Michal Hradiš, Ph.D.

Rozšířený abstrakt

Tato práce se zabývá extrahováním informací z tabulek. Převážně však extrahováním informací z historických tištěných nebo historických ručně psaných tabulek. Cílem rozpoznávání tabulek je získat z obrázku tabulky její přepis v textové podobě. Textová podoba je většinou ukládána ve formátu HTML, XML nebo JSON a v ideálním případě zachycuje strukturální podobu tabulky doplněnou o text, který je obsažen v buňkách tabulky. Strukturální podobou tabulky rozumíme rozčlenění tabulky na jednotlivé její buňky. Každá taková buňka je doplněná o informaci o její logické pozici v tabulce (sloupec a řádek, ve kterém se buňka nachází) a o její pozici v obrázku (ohraničující rámeček, který určuje pozici buňky v obrázku).

Existuje několik přístupů k řešení tohoto problému. Jsou metody, které se snaží buňky v tabulce nalézt algoritmicky s využitím houghovy transformace, která nalezne v obrázku čáry definující strukturu tabulky. Další možný přístup je založen na morfologických operacích. V této práci se však zabýváme extrahováním informací z tabulek s využitím strojového učení. Speciálně se pak tato práce zaměřuje na rozpoznávání tabulek s využitím grafových neuronových sítí. Mezi další metody, které jsou založené na strojovém učení lze zařadit takové, které využívají neuronové sítě Transformer nebo další typy rekurentních neuronových sítí. Tyto metody jsou schopné ihned generovat cílový text reprezentující tabulku bez nutnosti složitě upravovat výstup sítě. Je však nutné podotknout, že takové metody nalézají uplatnění převážně u moderních tištěných tabulek, jejichž struktura je více předvídatelnější než u historických ručně psaných dokumentů.

Metoda založená na grafových neuronových sítích, která je v rámci této práce implementována, je inspirována metodami zmíněných v textu práce a skládá se ze tří hlavních části: inicializace grafu, klasifikace hran a uzlů grafu a transformace grafu do textové podoby. Počáteční graf, který je předán grafové neuronové síti, je vytvořen následujícím způsobem. Nejprve jsou v textu detekovány řádky textu za pomocí metody optického rozpoznávání znaků. Detekované textové řádky reprezentují uzly grafu. Následně najdeme hrany mezi uzly za pomocí algoritmu viditelnosti, který spojuje každý uzel s uzly, které jsou z jeho pozice viditelné (tzn. existuje úsečka, která spojuje centra detekovaných řádků a která není přerušena žádným jiným detekovaným řádkem).

Takto vytvořený graf je předán grafové neuronové síti, která klasifikuje uzly grafu do dvou kategorií: uzel reprezentující buňky záhlaví tabulky a uzel reprezentují datovou buňku tabulky. Hrany grafu jsou klasifikovány do čtyř kategorií: hrana spojující řádky náležící do stejné buňky tabulky, hrana spojující buňky stejného řádku, hrana spojující buňky stejného sloupce a konečně hrana spojující buňky mezi nimiž není vztah významný pro úspěšné rozpoznání tabulky.

Vytvoření cílové textové reprezentace tabulky je provedeno ve fázi transformace grafu na text. V této fázi graf podstupuje několik kroků, které postupně zjednodušují výstupní graf sítě. Uzly zjednodušeného grafu jsou pak zarovnány do mřížky, ze které je pak jednoduché vytvořit výslednou textovou reprezentaci tabulky. První krok zjednodušování grafu se skládá z odstranění hran, které byly klasifikovány jako hrany, které nemají pro rozpoznání tabulky žádný význam. Následně jsou v grafu spojeny uzly, které jsou spojené hranou, která spojuje uzly (řádky textu), které náleží do stejné buňky. Z grafu se spojenými uzly jsou pak odstraněny všechny tranzitivní hrany, a to ve dvou krocích. V prvním kroku jsou odstraněny tranzitivní hrany pro horizontální hrany a ve druhém kroku odstraníme tranzitivní hrany pro vertikální hrany. Ze zjednodušeného grafu jsou pak dále odstraněny hrany, které jsou napojeny na uzel, který má více než dva horizontální nebo vertikální sousedy. V závěru této části pak reprezentujeme každou souvislou komponentu grafu definovanou horizontálními/vertikálními hranami za pomocí ohraničujícího rámečku. Mezi získanými ohraničujícími rámečky pak postupně zmenšujeme prostor zvlášť v horizontálním a vertikálním směru. Po provedení všech zmíněných kroků by hodnota souřadnice x každého uzlu měla odpovídat sloupci, ve kterém se uzel (buňka tabulky) nachází, a souřadnice y každého uzlu by měla odpovídat jeho řádku.

Implementovaný model vyhodnocujeme pomocí námi navržené metriky a také podle částečně upravené metriky, která se používá na vyhodnocení metod na datasetu cTDaR. Navržené metriky sledují, kolik je námi vytvořená metoda schopna detekovat buněk s textem (85 %), dále kolik buněk se správně detekovanými horizontálními sousedy (68 %) a buněk se správně detekovanými vertikálními sousedy (71 %). U metriky, která se používá pro vyhodnocení na datasetu cTDaR bylo dosaženo hodnoty přesnosti 66 % a hodnoty výtěžnosti 43 %. Tato metrika sleduje mezi kolika dvojicemi bezprostředně sousedících buněk byl detekován správně vztah (horizontální nebo vertikální). Dle výsledků lze usoudit, že metoda má prostor pro zlepšení. Zejména se pak v budoucnu lze zaměřit na detekování buněk, které se rozprostírají přes více sloupců či řádků. Nicméně i nyní metoda v některých případech poskytuje dobré výsledky.

Neural Networks for Automatic Table Recognition

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Ing. Michal Hradiš, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

....
Lukáš Piwowarski
May 17, 2022

Acknowledgements

I would like to thank my supervisor Ing. Michal Hradiš, Ph.D. for his guidance and willingness. Also, I would like to thank my family, partner and friends who supported me and helped me to review this thesis.

Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA CZ LM2018140) supported by the Ministry of Education, Youth and Sports of the Czech Republic.

Contents

Chapter 1

Introduction

Data surrounds us in every corner of our lives, and we as humans tend to capture reality to the best of our abilities. This era provides us with all sorts of tools, from digital cameras to smartwatches, that enable us to collect data and store it in digital form. As time passes, we move all our collected data points to the digital space, and this trend seems to stay.

But what should we do with all the data we have already created and stored on paper? The answer is simple. We have to move it to the digital space as well. This will help us search through the already created data points more easily.

There is ongoing research in the field of document digitization that tries to solve the problem of moving data from paper to our hard drives. One of the challenges that the field of document digitization faces is the extraction of information from tabular data. Tables contain a lot of useful information that can be used either for historical research in case of historical handwritten tables or managing finances in case of paper invoices.

Despite the potential benefit, the task of digitization tabular data is not fully mastered, especially in the case of historical handwritten tables that may have a too complex structure. However, printed tables are no exception, and even with them, problems may occur when it comes to digitization.

The challenge of this task is to understand the table structure, which might seem easy at first glance when one has in mind a simple table with cells without a span. But the task gets increasingly difficult when cells that span across multiple columns or rows come into play. In addition, trying to classify table cells or digitize historical handwritten tables increases the difficulty even more.

The following pages contain a description of methods and approaches that try to tackle the task of tabular data digitization. In Chapter 2, the task of digitization of tabular data is formalized and described more thoroughly. Answers to questions such as "What are the inputs/outputs of a process that digitizes tabular data?" or "What are the existing datasets that can be used to train table recognition models?" can be found in this chapter.

The following Chapter 3 gives some theoretical background that is necessary to understand methods that are used for table recognition – mainly graph neural networks and Transformer neural networks. Chapter 4 then presents current methods that are used for table recognition. Finally, in Chapters 5 and 6 we describe and experiment with the implemented table recognition pipeline.

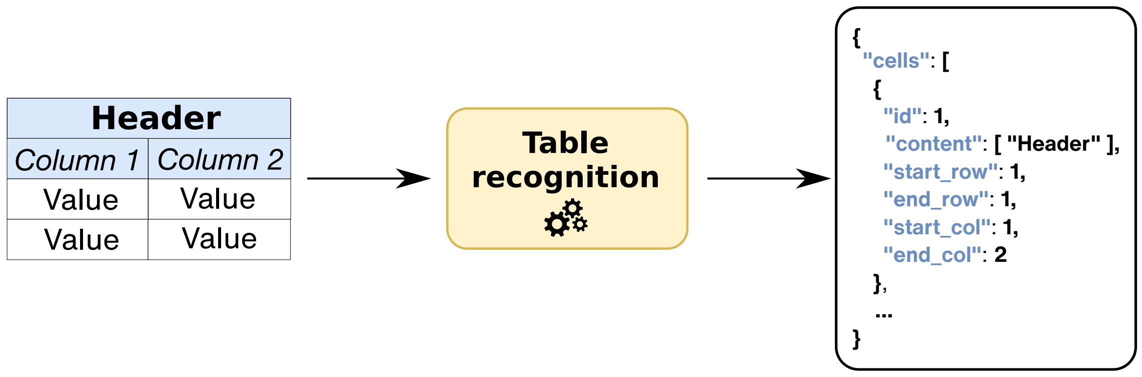
Chapter 2

Table Recognition

Table recognition is a process of extracting information from a table. Depending on the type of information we try to extract, we can further divide the process of table recognition into two sub-tasks: table structure reconstruction and table content reconstruction [19]. The goal of table structure recognition is to understand the topology of a given table. Precisely, locate each table cell and identify its location in the table by specifying the row and the column it occupies, including its row-span and col-span (Figure 2.1). On top of the mentioned information, we may want to classify each cell as a header or a data cell [18, 43].

When we obtain the regions of table cells, we can perform table content reconstruction. By table content reconstruction, we understand the process of extracting text from each cell. Finally, we can merge the extracted text with extracted table structure, thus ending the process of table recognition.

There are methods that need only one pass through the model to get the information from the table. These methods are usually based on Transformer neural networks or recurrent neural networks [18, 43]. These methods generate text representation of the table straight from the input image. On the other hand, there are some methods that require certain pre-processing of the input image, for example, those that use graph neural networks [34].



Header	
Column 1	*Column 2*
Value	Value
Value	Value

```
{
 "cells": [
  {
   "id": 1,
   "content": [ "Header" ],
   "start_row": 1,
   "end_row": 1,
   "start_col": 1,
   "end_col": 2
  },
  ...
}
```

Figure 2.1: An example of a table that we may want to try to extract information from. The table contains a header cell with the text „Header“ in the first row and column. The cell spans across two columns and two rows.

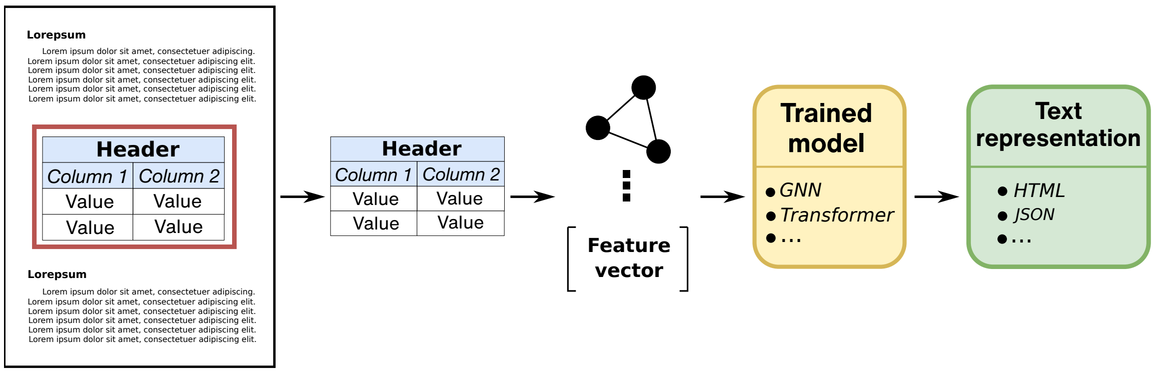


Figure 2.2: The process of table recognition using machine learning methods consists of the pre-processing phase in which the input table is represented as a graph, the table region is cut out of the image, etc. Then the model processes the table and outputs either a graph or a text representation of the table that can be further processed in the post-processing phase to generate the final text representation.

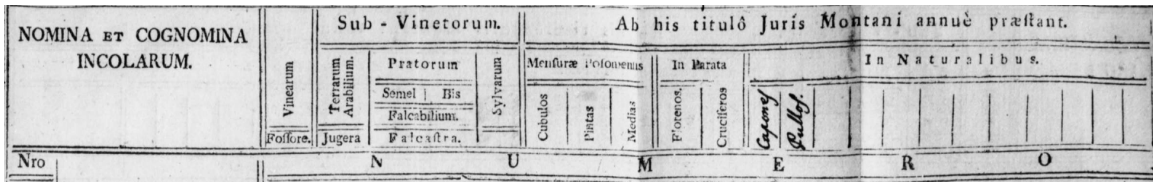
2.1 Description of Steps Involved in Table Recognition

Generally speaking, when it comes to methods that use machine learning to handle table recognition, we can observe that the extraction process can be split up into three typical steps that usually occur when working with neural networks: the pre-possessing of the input table, the inference, and the post-processing of the model's output.

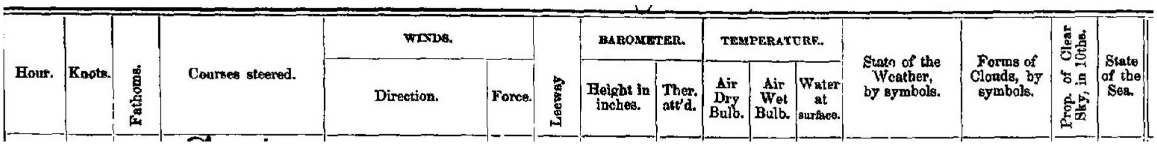
The pre-processing. Preparation of the input image is almost none in case a Transformer neural network (Section 3.4) is used. These methods take a table image as an input and produce the text representation of the table directly [9]. It depends on how we define the table recognition task, but one can also be concerned with detecting the table in a given image [14]. This is needed if we get an image that does contain other things except for the table. In this thesis, we expect that we are given images that contain only a table and nothing else.

The pre-processing is slightly more involved if we decide to use graph neural networks (abbrev. GNN) to solve the table recognition task [5, 34]. These types of neural networks work with graph representation of the input table. To give an idea of how such graph representation may look like we can imagine a graph whose vertices represent text lines, detected using an Optical Character Recognition model (abbrev. OCR), and edges represent potential connections between the text lines.

The inference. Once we are finished with the pre-processing phase, we pass the image or the graph to a model of our choosing that outputs either a graph with classified edges or nodes or the text representation of the table.


NOMINA ET COGNOMINA INCOLARUM.
Vinearum
Foſſore.
Sub - Vinetorum.
Terrarum Arabilium.
Jugera
Pratorum
Semel | Bis
Falcabilium.
Falcaſtra.
Sylvarum
Ab his titulô Juris Montani annuè præſtant.
Menſuræ Poſonienſis
Cubulos
Pintas
Medias
In Parata
Florenos
Cruciferos
Capones
Pullos
In Naturalibus.
Nro
N U M E R O

(a) Sample from the cTDaR dataset.


Hour.
Knots.
Fathoms.
Courses steered.
WINDS.
Direction.
Force.
Leeway
BAROMETER.
Height in inches.
Ther. att'd.
TEMPERATURE.
Air Dry Bulb.
Air Wet Bulb.
Water at surface.
State of the Weather, by symbols.
Forms of Clouds, by symbols.
Prop. of Clear Sky, in 10ths.
State of the Sea.

(b) Sample from the GloSAT dataset.

Figure 2.3: Images showing complicated table headers from GloSAT and cTDaR datasets [12, 30].

Comparison of datasets					
Dataset name	Type	Size	Cell type	pixel/logic. position	GT format
cTDaR [12]	Historic.	903	✗	✓/ ✓	XML ICDAR
GloSAT [30]	Historic.	500	✓	✓/ ✓	XML ICDAR
ABP [17]	Historic.	117	✗	✓/ ✓	XML
cTDaR [12]	Modern	600	✗	✓/ ✓	XML ICDAR
PubTabNet [43]	Modern	518,977	✓	✗/ ✓	HTML
SciTSR [5]	Modern	5,006	✗	✗/ ✓	JSON

Table 2.1: Comparison of existing datasets for table recognition.

The post-processing. Modification of the model's output is not always necessary, but for GNNs based methods, it is practically inevitable. The post-processing of a graph may require some computation time, and it can differ from one implementation to another. For example, some methods require identifying maximal cliques in the output graph to detect columns and rows in the table [36]. The time complexity of the algorithm for finding maximal cliques in a graph belongs to $O(3^{n/3})$, which indicates that the post-processing can take a non-negligible amount of time for large tables [37]. The post-processing is then finished by generating the final text representation of the table into HTML, XML or JSON text file.

2.2 Existing Datasets for Table Recognition

Neural network training requires a reliable dataset with enough data points. In the area of table recognition, there are many datasets that can be used. However, brief research shows that there are more big high-quality datasets with modern tables than big datasets with printed tables (Table 2.1).

There are a few attributes of a table cell that one may require from a dataset when training a model for table recognition: cell type, cell pixel position and cell logical position. Cell type determines whether a given table cell is a header or data cell.

Cell pixel position defines the position of each cell in the image, and logical position of the cell determines the row, the column, the row-span and the col-span of the cell.

When it comes to handwritten tables, the GloSAT dataset (Figure 2.3b) can be considered as the one that is best prepared for the training of a model for table recognition [30]. It consists of historical measurements from logbooks; therefore, the data contained in the tables are primarily numerical. The dataset consists of historical tables that combine both handwritten and printed text. Scanned images in the dataset sometimes contain other text surrounding the table.

Similarly, the cTDaR dataset (Figure 2.3a) provides both handwritten and printed historical tables [12]. The tables are more diverse compared to other datasets as the tables come from 23 different archives from all around the world. Its main disadvantage compared to the GloSAT dataset is that it does not provide information about the type of each cell.

From the printed datasets, PubTabNet stands out as the one with the most tables in it out of all the datasets mentioned here [43]. The tables are extracted from scientific publications, and most of the tables in the dataset are generated using LaTeX. The ground truth of this dataset has an HTML format. This format prevents determining each cell's pixel position, which may be necessary for some types of table recognition tasks. The next dataset with printed tables is the SciTSR dataset, which contains only LaTeX tables and is similar to PubTabNet dataset except that it contains fewer tables [5].

2.3 Metrics Used to Evaluate Table Recognition Methods

We face a problem when we want to evaluate the performance of table recognition methods. Choosing the correct metric is essential for a good comparison between the implemented models, yet there is no consensus on which metric for table recognition should be used. There are plenty of metrics and approaches that can be utilized for the evaluation. This section presents three metrics that can be encountered while researching table recognition.

To measure the performance of text output, a BLEU (BiLingual Evaluation Understudy) metric from the field of natural language processing can be borrowed to calculate the performance of the table recognition method:

$$P_n = \frac{\sum_{w_n \in \hat{Y}} \mathrm{CountClip(w_n)}}{\sum_{w_n \in \hat{Y}} \mathrm{Count(w_n)}}, \tag{2.1}$$

here the $\hat{Y}$ denotes the output of the model and w_n an n-gram from the output. CountClip counts how many times w_n appears at most between all the reference outputs and Count how many times w_n appears in $\hat{Y}$ [31]. The main disadvantage of BLEU is that it does not take into consideration the semantics of the output. It rewards the output with a high score even if the n-grams are mixed up and do not make together a sensible output.

Another metric that was designed especially with the evaluation of table recognition in mind is TEDS [43]. TEDS stands for Tree Edit Distance, and as its name suggests, it represents the table as an acyclic graph – tree. It calculates the edit distance between the ground truth tree and the output of the implemented method (Equation 2.2). Edit distance of two trees is defined as the minimum number of edit operations that need to be performed on one tree to turn it into another one. As edit operations, we consider insertion of a node, deletion of a node and renaming of a node [32]. We calculate the TEDS metric as follows:

$$\mathrm{TEDS}(T_a, T_b) = 1 - \frac{\mathrm{EditDist}(T_a, T_b)}{\max(|T_a|, |T_b|)}, \tag{2.2}$$

where T_x is a table represented in a tree structure, and EditDist calculates the edit distance of two trees. TEDs in this form does not allow to measure the performance of methods that also output a position of each cell in the input image. However, it probably could be modified in such a way that it would take the cell bounding box into consideration. For example, the EditDist operation could calculate some smooth score when renaming a node that would indicate how much the bounding box would have to change to have the same bounding box as a node in the ground truth. [43]

The last metric worth mentioning takes the bounding boxes of the table cells into consideration. It is used to evaluate methods on the cTDaR dataset. This metric views the tables as matrices of cells. To compare the two matrices, we first have to find cells in the model's output that have IoU with cells in ground truth higher than the chosen threshold. These matrices pass to the second step.

In the second step, a list of adjacency relations between the cells that passed the first step is created. The adjacency list contains for each cell a list of its neighbours (closest vertical neighbours and closest horizontal neighbours). The adjacency list is then used to calculate the precision and recall that is used to calculate the final F1 score [13]. The F1 score is calculated as:

$$\mathrm{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{2.3}$$

Chapter 3

Neural Networks and Table Recognition

The task of table recognition can be solved using several approaches from the field of machine learning, especially then transformer neural networks and graph neural networks (abbrev. GNN). This chapter focuses mainly on GNNs and offers their explanation on such a level that is needed to understand methods for table recognition in the following chapter.

3.1 Introduction to Graph Neural Networks

GNNs take as an input a graph $G = (V, E)$ with vertices $V = \{v_1, v_2, \ldots, v_n\}$ and edges $E = \{e_1, e_2, \ldots, e_m\}$. Graphs enable us to pass to the neural network data like social networks, molecules, text, and even graphs representing relationships between objects in an image. GNN can handle tasks like node classification, node regression, edge prediction, edge classification, clustering, graph classification, or graph matching [44]. The power of GNNs can be used in the case of table recognition for node classification (used for a decision whether a given node is a header or data cell) and edge classification (used for a decision whether two nodes belong to the same line/row or not) (Figure 3.1) [34, 44].

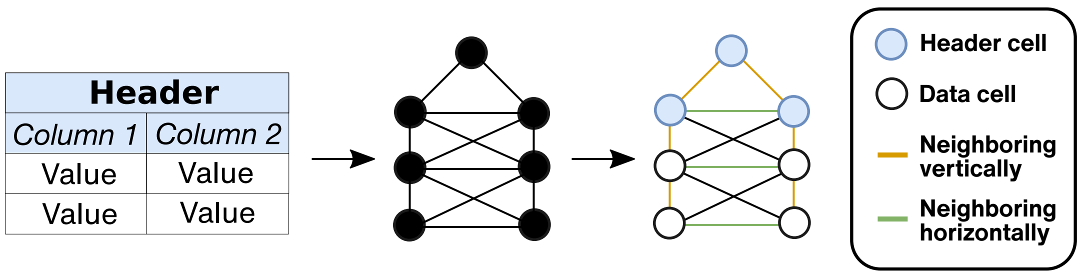


Figure 3.1: Picture showing how detected objects in an image (text lines) can be passed to a neural network as a graph. The detected text lines in an image are represented as nodes, and edges represent potential relationships between them. The GNN then tries to classify the edges and the nodes.

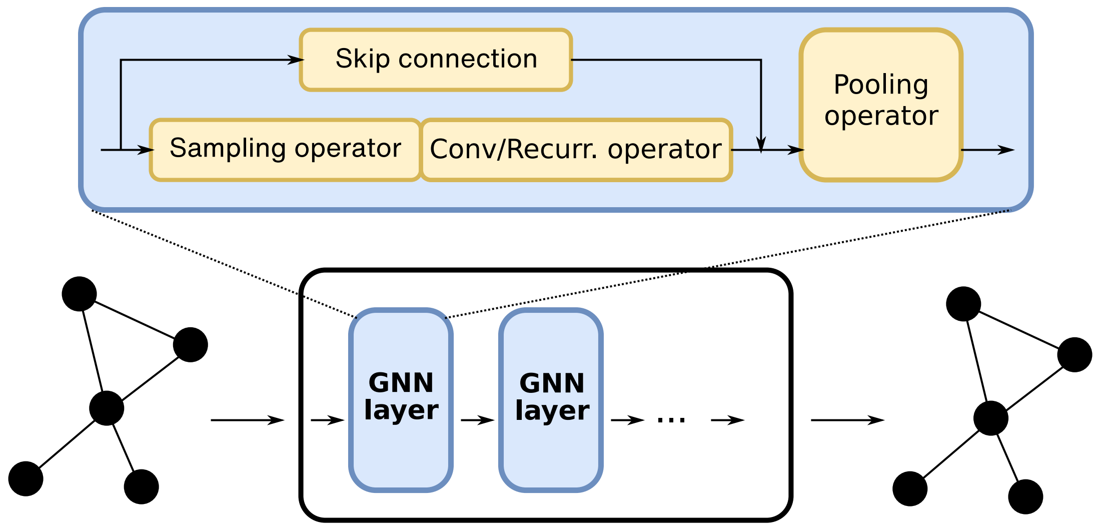


Figure 3.2: Scheme of a GNN. The input graph goes through a sequence of layers. Each layer can consists of: skip connection, sampling operator, convolutional/recurrent operator and pooling operator. At the end, the output of the GNN is another graph with each node/edge classified into one of the required classes. Image taken from [44].

When we focus on the task of node classification and edge classification using GNNs, then usually at the beginning, we obtain a graph in which each node n and each edge e has assigned a feature vector $h_n^{(l)}$, $h_e^{(l)}$ respectively. The GNN then iteratively propagates the feature vectors through the network among the graph's nodes. Hopefully, when the GNN is correctly trained, then after the iterative process, each node or edge is at the end classified to the correct class.

As with every neural network, the GNN consists of layers as well. Each layer of a GNN can (but does not have to) contain one of the following blocks: skip connection, sampling operator, convolutional/recurrent operator and pooling operator (Figure 3.2). [44]

The layer responsible for propagating the feature vectors through the network is the convolutional/recurrent operator (convolutional operator further). As the graph passes through the GNN, the convolutional operator transforms and aggregates information from surrounding nodes $N_l(n)$ of a given node n from the previous layer. The information of $N_l(n)$ is aggregated using specific aggregation function $AGG(N_l(n))$, which can be simple sum or mean. Examples of convolution layers are discussed in the following section. [44]

When the number of layers in GNN is increased with the goal of increasing the computational power, it is possible that the exact opposite may happen. This is caused by the fact that with the increasing number of layers, the nodes of a graph tend to have similar values. To solve this issue, which is sometimes called *smoothing problem*, the Skip Connection is introduced. The main goal of this section is, therefore, to enable deeper GNNs [2, 44].

Another problem that arises when the number of layers is increased is the so-called *neighbourhood explosion problem*. The neighbourhood explosion problem is the simple implication of the way in which GNNs work. Given a GNN with two layers and node n then in the final value of $h_n^{(2)}$ is computed from a set of nodes $N(N(\{n\})) - N(\{x\})$ takes as an input a set of nodes and returns set of all its neighbours. Or generally, when the average degree of a node in a graph is d then the number of nodes involved in the computation of

$h_n^{(l)}$ in GNN with l layers can be d^l. This exponential growth makes the calculation of gradient more complicated during the training of GNN. To tackle this issue, sample connections were introduced. Sample Connections simply said try to aggregate the information from only a subset of $N(n)$ using a heuristic that cleverly chooses the subset. This decreases the number of nodes involved in the computation of $h_n^{(l)}$ and therefore speeds up the learning process [29].

The pooling layer is the last type of layer we can find in GNNs. There is a connection between pooling layers in CNNs and GNNs. In CNNs, pooling layers down sample its input, and the same happens with GNNs. The pooling layer follows after the Convolutional layers, and its goal is to decrease the number of nodes in the current graph while preserving the most important information about the original graph. [15, 44]

3.2 Overview of Existing GNN Layers

As mentioned in the previous section, convolutional layers propagate information through the graph that is being processed. There are many types of Convolutional layers, and usually, they are classified into two groups: spectral convolutional layers and spatial convolutional layers. The spectral convolutional layer performs the convolution with a filter in a spectral domain of a graph [4, 23]. As the table recognition methods use the spatial convolution layers, we focus purely on them.

The spatial convolutional layers perform the convolutional directly on the graph. The challenge of creating a spatial convolutional layer is to make it order invariant. It should not matter in which order we aggregate the information from the surrounding nodes. The simplest spatial convolutional layers update the node feature vectors as follows:

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)}), \tag{3.1}$$

here $H^{(l)}$ refers to a matrix containing values of feature vectors in a given layer l and A is an adjacency matrix, and $W^{(l)}$ is a matrix of weights of the layer that are obtained by training the neural network [21]. The σ represent non-linear function for example, Rectified Linear Unit *(abbrev. ReLU)*. By multiplying matrices A and $H^{(l)}$, the surrounding nodes of each node n are added together and then transformed using $W^{(l)}$. This example of a convolutional layer is order invariant, as addition is a commutative operation. The main drawback of this approach is that each feature vector in $H^{(l)}$ is scaled differently based on the number of its neighbours [21]. To solve this issue, A can be normalized:

$$f(H^{(l)}, A) = \sigma(D^{-\frac{1}{2}}AD^{-\frac{1}{2}}H^{(l)}W^{(l)}), \tag{3.2}$$

here, D represents the degree matrix of the graph. GNNs that use this convolutional operation are often referred to as graph convolutional neural networks (abbrev. GCNs).

A more advanced convolutional neural networks try to identify the importance of each neighbouring node. To be precise, these approaches try to assign to each edge e_{ij} connecting nodes i and j weights α_{ij} and α_{ji} (Equation 3.3). Where α_{ij} states how much the value of h_i is dependent on the value of h_j [39]. The weights for each node pair are calculated in the following manner:

$$\alpha_{i,j} = \frac{\exp(\text{LeakyReLU}(\text{a}^\top[\mathbf{W}h_i^{(l)} \parallel \mathbf{W}h_j^{(l)} \parallel \mathbf{W}h_{ij}^{(l)}]))}{\sum_{k\in N(i)\cup\{i\}} \exp(\text{LeakyReLU}(\text{a}^\top[\mathbf{W}h_i^{(l)} \parallel \mathbf{W}h_k^{(l)} \parallel \mathbf{W}h_{ij}^{(l)}]))}, \tag{3.3}$$

where $a^\top$ and $\mathbf{W}$ are trainable parameters. The weights calculated using eq. 3.3 are then used for the computation of the resulting feature vector:

$$h_i = \alpha_{i,i}\mathbf{W}h_i + \sum_{j \in N(i)} \alpha_{i,j}\mathbf{W}h_j. \tag{3.4}$$

We can notice that in the Equation 3.3 the edge feature vector $h_{ij}^{(l)}$ is used. This is not necessary, and there are also graph attention layers that do not use the edge feature vector.

Another less known layer is the one from the work of *Diehl* [10]. This layer updates both node and edge feature vectors. The update of the feature vectors consists of two steps. In the first step, the edge feature vectors are updated using the following formula:

$$h_{ij}^{(l)} = f_e^{(l)}\left(h_i^{(l-1)} \parallel h_{ij}^{(l-1)} \parallel h_j^{(l-1)}\right), \tag{3.5}$$

where $h_{ij}^{(l)}$ is value of edge feature vector in the layer l, h_i^l is value of feature vector assigned to node i and $f_e^{(l)}$ is Multi-Layer Perceptron (abbrev. MLP) that processes the concatenated feature vectors from the previous layer to produce new feature vector for the edge which connects the node i with node j.

The new value of the edge feature vector is then used in the second step to update the node feature vector. To do so, the next formula is utilized:

$$h_i^{(l)} = f_1^{(l)}\left(h_i^{(l-1)} \parallel \sum_{(i,j)\in E} f_2^{(l)}\left(h_i^{(l-1)} \parallel h_{ij}^{(l)}\right)\right), \tag{3.6}$$

in which E is a set of edges, $f_x^{(l)}$ is MLP and the rest of the notation is similar to notation used in the Equation 3.5.

3.3 Sampling Operators and Skip Connections in Graph Neural Networks

Skip connections try to resolve the smoothing problem that occurs when we add more layers to a GNN. Layers in GNN that use skip connections generally approach this problem by propagating the history of $h^{(l)}$ throughout the network. The history of h is then used to calculate the value of a feature vector at the next layer ($h^{(l+1)}$). There are various types of skip connections that help to propagate the information through the network. The DenseGCN and ResGCN use sum and concatenation, respectively:

$$h_n^{(l+1)} = T(h_n^{(l)}, \mathbf{W}^{(l)}) + h_n^{(l)}, \tag{3.7}$$

$$h_n^{(l+1)} = T(h_n^{(l)}, \mathbf{W}^{(l)}) \parallel h_n^{(l)}, \tag{3.8}$$

here, T denotes function that calculates the transformations of feature vector using weights $W^{(l)}$ [3]. We can notice that in Equation 3.8 the h grows as it passes through the network.

On the other hand, there are sampling operators that help to solve the neighbour explosion problem (Figure 3.3). The sampling operator tries to find a subset of $N(n)$ such that contains vertices that influence the value of $h_n^{(l)}$ the most. The subset is then used to

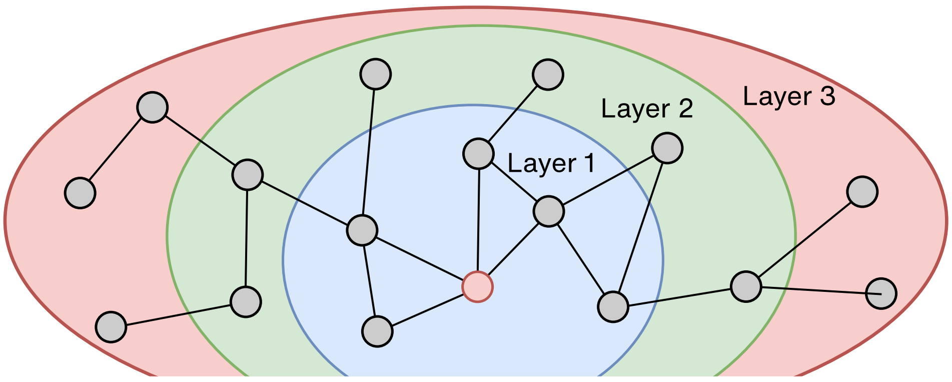


Figure 3.3: A picture displaying the issue that emerges when we stack multiple GNN layers together. As the number of layers grows, so does the number of nodes that are involved in the computation of the feature vector for each node *(the red node)*.

calculate the resulting $h_n^{(l+1)}$ using the subset of $N(n)$. A naive approach could be to select the neighbours randomly for each given node.

Another approach mentioned in [41] could be such that we use random walks to select the neighbourhood of a given node. First, we do random walks of fixed length starting from node n. After we have done a certain number of random walks, we obtain a multiset:

$$R = \{(r_n^1, r_n^2, ..., r_n^m), ..., (r_z^1, r_z^2, ..., r_z^m)\}. \tag{3.9}$$

In the set R, r_n^i represents how many times appeared node i in random walk n. We then sum together the visit count for each node, thus obtaining:

$$R' = \{r'_1, r'_2, r'_3, ..., r'_m\}. \tag{3.10}$$

In the set R', r'_i represents a number of how many times node i appeared in the random walks. The sampling is then done by choosing nodes that are represented by K highest values in R'.

3.4 Transformer Neural Network

The transformer architecture was first introduced by *Vaswani et al.* in [38]. In the paper, the authors use the transformer to solve the task from natural language processing; however, it turns out that it can also be utilized to solve computer vision tasks [28]. Moreover, what is relevant for this thesis is that the architecture can be used for the task of table recognition [9].

The idea of this architecture is based on three main blocks: encoder block, decoder block and attention block (Figure 3.4).

If we focus on the transformer architecture from [38], then we work with a model whose goal is to translate a sentence from one language into another. The model takes as an input matrix that encodes the input sentence (each column in the input matrix is a word represented using one-hot encoding). By recurrently calling the model, we obtain a series of vectors representing the translated sentence.

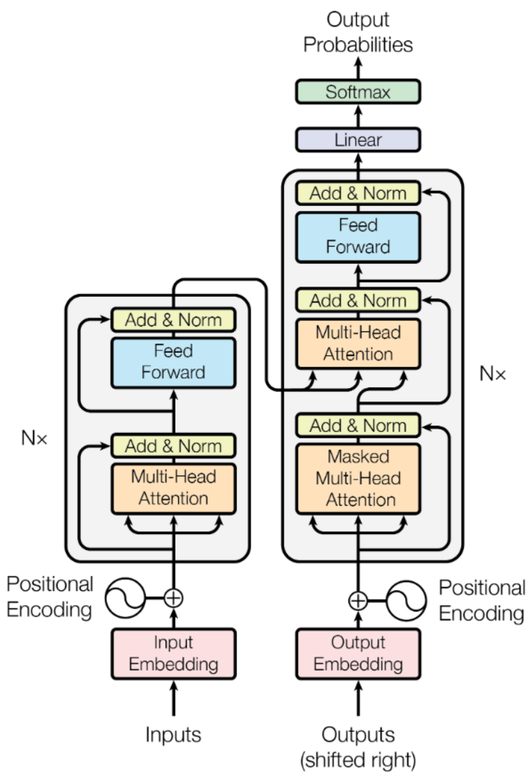


Figure 3.4: Image of the transformer architecture. It consists of two main blocks, encoder *(left section)* and decoder *(right section)*. Both encoder and decoder use *multi-head attention* block. The image is taken from [38].

Before the input matrix enters the encoder part of the model, it is first transformed so that each word in the matrix carries information about its position in the sentence. An option how to embed the word's position is to use the following equations:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right), \tag{3.11}$$

$$\mathrm{PE}_{(pos,2_{i+1})} = \cos\left(pos/10000^{2i/d_{model}}\right). \tag{3.12}$$

The equations produce vectors with a number of dimensions equal to the number of dimensions of the word embeddings that are being transformed. Generated vectors using these equations are then added to the input word embeddings (see positional encoding in Figure 3.4). In both Equation 3.11 and Equation 3.12 pos is index of the word in the input sentence and i is dimension index. Except for the mentioned equations, there are multiple other possibilities of how to embed the information about the word's position [42].

The matrix produced from the previous step enters the encoder. The task of this part of the neural network is to produce embedding for the input sentence that will help the decoder to produce the translated sentence. Multiple encoder blocks are usually stacked on top of each other. The number of encoders we use depends on the task we try to solve using the transformer.

The embedding produced by the encoder is passed to the decoder section. The decoder uses the embedding together with the already translated portion of the sentence to generate the next word of the translation. Similarly, as with an encoder, the decoder section can be stacked upon itself multiple times.

Both encoder and decoder use the attention block. In simple words, the attention block tries to compute the relationship of each word embedding with other word embeddings in the same sentence. Let us suppose that the attention block receives a series of word embeddings stored in a matrix, then the output of the attention block is a matrix in which the value in row i and column j defines how much important is the j-th word in the sentence for translation of the i-th word in the sentence. This idea can be used even when the input is an image. In such a case, the attention layer computes the relationship between sections of the input image [28]. The output of the attention is computed as:

$$Z = \text{softmax}(\frac{Q\dot{K}^T}{\sqrt{d_k}}). \tag{3.13}$$

In the equation (Equation 3.13) matrices, Q (Query), K (Key), V (Value) represent linearly transformed matrices with word embeddings of the input sentence, and d_k is the number of dimensions of one word embedding in either Q, K or V.

Chapter 4

A Survey of Table Recognition Methods

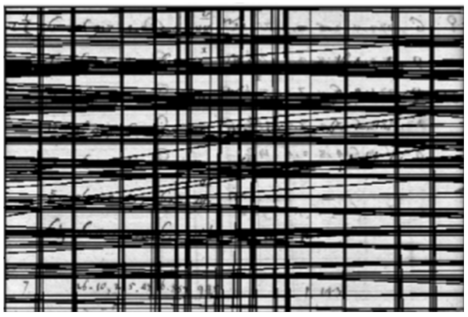
This chapter deals with already existing methods that use neural networks to solve the table recognition task. We focus mainly on the methods that use the GNNs as we implement GNN based table recognition pipeline, which focuses on historical documents (Figure 4.2). First, we give a brief introduction to table recognition approaches that do not rely on machine learning methods. Then we describe different strategies for the creation of the initial graph representation of the input table, and finally, different graph-based methods that can be used for table recognition are introduced.

4.1 Table Recognition Methods Without Machine Learning

Before looking at methods that use machine learning to solve table recognition tasks, it may be useful to understand at least one method that does not use a neural network to solve the problem. This may show the challenges that one can face when solving the table recognition task.

In [26] the authors rely on Hough transform to detect lines of the table. After receiving a set of lines out of the Hough transform, the authors further filter the obtained lines. The filtration is done so that lines that highlight pixels whose average colour is blacker than a certain threshold retain, and others are discarded. By calculating the intersection of the filtered lines, the resulting scheme of the table is obtained. This approach is clearly dependent on lines that outline the structure of the table. Also, this method does not solve the issue of cells that span across multiple columns or rows.

Another approach in [27] tries to identify the table lines using morphological operators. First, the thresholding is performed on the input image, and then the open operators are applied. By applying the open operator, we obtain the horizontal and the vertical table lines that can be used to reconstruct the table cells. Again this method relies solely on the presence of printed lines in the table.


(a) Lines detected using Hough transform.

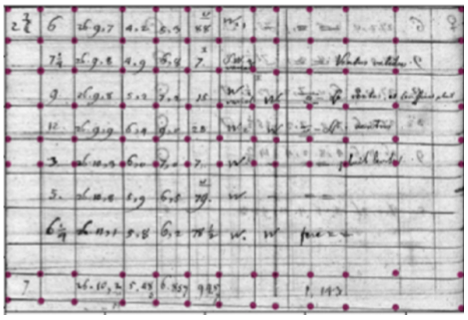

(b) Lines and points defining table cells.

Figure 4.1: Picture showing detection of table cells using the Hough transform. The black lines indicate lines detected by the Hough transform. The red circle represent the intersection points between the lines. Four corners then define each cell. Images taken from [26].

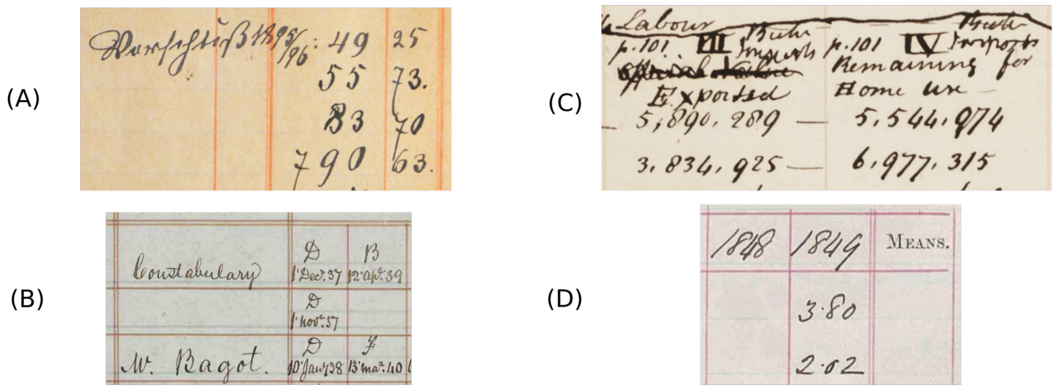


Figure 4.2: Samples of historical handwritten tables that represent issues that can be encountered when dealing with table recognition. In images *(A)* and *(C)* values that should be stored in single table cell span across multiple cells instead. Table cells with missing values are another common thing that can be encountered in historical handwritten tables (images *(B)* and *(D)*). Images taken from the cTDaR and GloSAT datasets [12, 30].

4.2 Comparison of Graph Initialization Approaches

As discussed in Section 2.1, the process of table recognition using a neural network can be divided into three sections: pre-processing, inference and post-processing. When it comes to GNNs, the importance of the pre and post-processing section slightly increases compared to other methods like [18, 43]. This section, therefore, introduces the reader to graph initialization more thoroughly.

Initialization of Edges in Graph

If the pre-processing section is done correctly, then it may decrease the time that is needed for inference and even more for the training of the GNN. In [34] the autor extends the approach from [33] by introducing a step by step method that can be used to initialize the input graph of the GNN. The method tries to reduce the average number of connections for each cell while keeping the most important ones untouched.

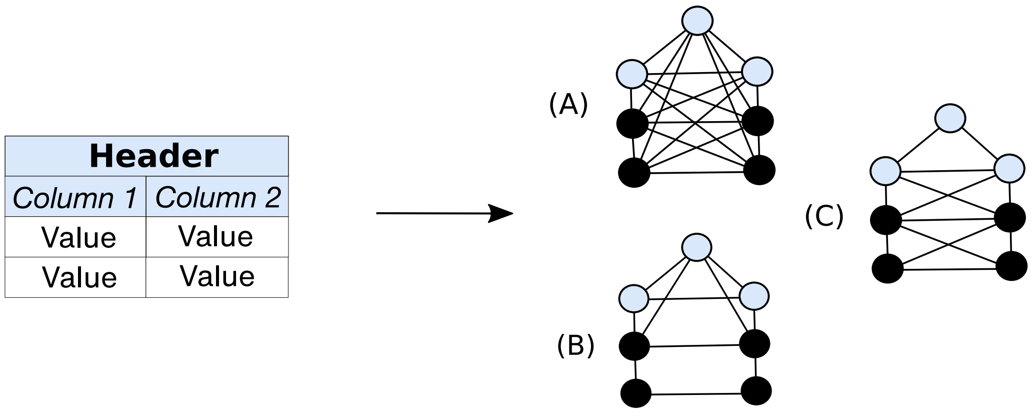


Figure 4.3: Example of three different approaches that can be used for an initial graph representation of the input table. In graph *(A)* all nodes are interconnected. In graph *(B)* each node is connected only to three of the nearest neighbours and in graph *(C)* only text lines that can see each other are connected.

By most important connection, we understand those that connect neighbouring cells. If we do not filter the connections, we may end up with an initial graph in which all cells are interconnected.

The first step. The main goal of this step is to filter out all the connections that connect non-neighbouring cells. At the end of the step, ideally, only cells that neighbour with each other, either horizontally or vertically, are connected. The cells are considered to be vertical or horizontal neighbours when their vertical respective horizontal spans of bounding boxes overlap and when they are in direct sight of each other (meaning there is no other bounding box between them) [34].

The second step. In this step, we try to tackle the issue that arises from the first step. When the table is slightly skewed, cells that neighbour with each other in the table may not be detected as neighbouring cells in the first step of graph initialization. To solve this issue, the authors pronounce cells as neighbouring when the Mahalanobis distance between the two cells is lower than the chosen threshold. This approach surely has its benefits, but one of the drawbacks of using the Mahalanobis distance is that we have to find parameters that are used to calculate it [1]. These parameters are handpicked by watching the method's performance [34].

The third step. Finally, we remove cells that span across multiple cells. When a cell that spans across multiple columns or rows is present in a table, it may prevent two cells from being connected even though they are vertical or horizontal neighbours. This is because the cells are not in direct sight of each other, and the distance between them can be too big to be considered neighbours by the second step. The idea of skip connection is introduced that solves this issue. Each cell is allowed to have a certain number of vertical or horizontal neighbours that are not in direct sight of it. Again the number of skip connections is handpicked by watching the method's performance [34].

These three steps demonstrate the issue of graph initialization. Clearly, this approach works, but one big disadvantage seems to be the need for parameter initialization in the second and third step. This approach needs to handpick these parameters every time we need to extract information from a new set of tables. Other more general approaches involve choosing k nearest neighbours to make the initial connections in the graph [5].

In [36] so-called interaction model is used to get rid of the graph initialization. Although the method does not initialize the graph at the beginning of the table recognition process, it does so at the end. It creates a complete graph in which it tries to classify every edge of the graph. The process of classification is time-consuming and clearly belongs to $\mathcal{O}(n^2)$. To reduce the time during the training of the neural network, it uses a sampling technique that randomly picks for each node a fixed number of edges that are removed from the graph. The GNN, therefore, learns only on a subset of the edges.

We can also use an algorithm which decides if there is an edge between two detected text lines based on whether the detected text lines are in the line of sight with each other [7]. Also, a simple neural network can be trained to decide whether the two text lines are connected using its feature vectors [8].

Node and Vertex Attributes

After we obtain a graph with defined nodes and edges, we need to assign attributes to each node and potentially to each edge. The feature vector assigned to the edges and nodes varies across existing methods. The straightforward approaches use spatial information about the detected text lines to create geometric feature vectors that represent them [5, 33, 34].

When it comes to geometric features of nodes, we can come across several possible options: position of bounding box specifies the position of the centre of the bounding box, height and width of the bounding box, number of neighbours that are adjacent to the cell either vertically or horizontally.

The possible geometric features that can be part of the edge feature vector are: distance between the two connected cells (nodes), axis that is intersected by the edge (either vertical or horizontal), horizontal/vertical overlap with neighboring cells in percent, the average of the bounding box centers of the two connected cells.

The feature vector, however, can also carry information about the appearance of the detected text line (Figure 4.4). The appearance of the detected text line is usually represented by a feature vector that is calculated using CNN. The visual feature vector is concatenated with the geometric feature vector [8].

4.3 Using Interaction Network for Edge Classification

In the work of *Quassim et al.* [36] authors do not create a graph that represents the relationships between the detected text lines. Instead, the detected text lines are passed to an interaction model that generates a graph that it uses to produce new feature vectors for every node. These feature vectors are then used to classify relationships between every pair of nodes (Figure 4.5).

The classification of the node relationship is done using fully connected neural networks. In the paper, the authors use three neural networks. One network is used to decide whether the nodes are in the same row, another one is to decide whether the nodes are in the same column, and the last one is whether two nodes are in the same cell.

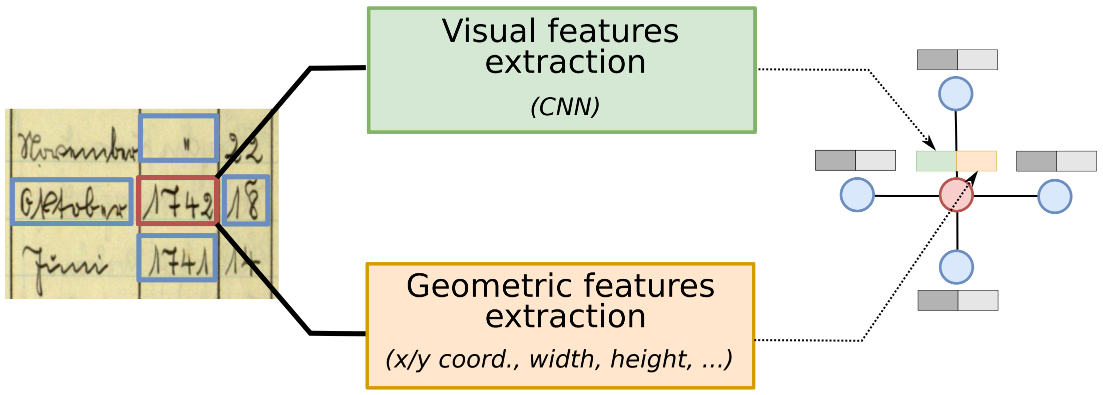

Visual features extraction
(CNN)
Geometric features extraction
(x/y coord., width, height, ...)

Figure 4.4: The example of a text line *(the red rectangle)* that is transformed to a node *(the red circle)*. The cell is represented with a node with geometric features *(the orange rectangle)* and visual features *(the green rectangle)* that are obtained by passing the corresponding section of the image through a CNN.

Interaction models that can be used to get the above mentioned feature vectors for each node are for example mentioned in work of *Wang et al.* [40] and *Qasim et al.* [35]. To give an idea of how such an interaction model may work, the model mentioned in [40] can serve as a good example. The process consists of three steps: in the first step, the graph is created; in the second step, the interactions between the graph's nodes are calculated; and in the last step, the final value of the feature vector is established.

The initial graph is created in **the first step** by first transforming each node's feature vector using a fully connected neural network. The output of the network is a new vector s that represents the position of the node in space and f_k that represents the transformed node's feature. The graph is created so that only nodes that are close to each other in space defined by the vector s are connected in the graph.

In **the second step** two new feature vectors are assigned to each node calculated using the following equations:

$$f_{jk} = f_j * V(d_{jk}), \tag{4.1}$$

$$f_k^2 = \max_j(f_{jk}), \tag{4.2}$$

$$f_k^1 = \sum_j f_{jk}, \tag{4.3}$$

here f_{jk} denotes auxiliary feature vector of node k that is calculated using feature vector of node j. d_{jk} is a distance between the nodes j and k calculated using vectors s. Function $V()$ is non-mandatory function that is used to scale the distance between two nodes and values f_k^1, f_k^2 are the new feature vectors calculated for node k [35].

The third step finally takes vectors f_k^1,f_k^2,f_k and produces the final feature vector f_k' that is an output of a neural network. The final feature vector f_k' is computed using a neural network that takes on its input the three mentioned vectors concatenated together.

Finally, when we have obtained feature vectors for each node, we can use them to classify relationships between nodes using a pre-trained neural network. The classified relationships are then used to create the final text representation of the input table. But to obtain the

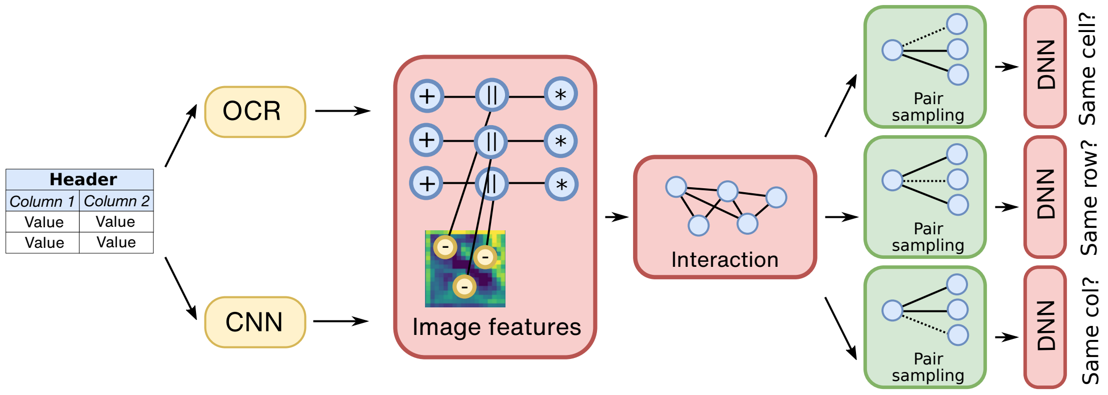


Figure 4.5: Scheme of table recognition method proposed in [36]. First, pieces of text are detected using an OCR. Then, the detected pieces of text are represented by feature vectors that are obtained by passing the corresponding regions of the input table to CNN *(image features block)*. The feature vectors are passed to an interaction model that creates a new set of modified feature vectors that are used in the last step to classify relationships between every pair of nodes. The image is taken from [36].

text representation, we have to first find the logical position of each cell in the table (its row index and column index).

Let's say we want to find all cells that belong to the same row. To do that, we have to first identify all maximal cliques in the sub-graph of our graph. The sub-graph consists of all nodes of the original graph and only of edges that connect nodes that belong to the same row.

Obviously, the question presents itself – Why is there a need to use the maximal cliques? The answer is that maximal cliques are necessary to solve the issue of row-span and col-span. If we defined the row in a graph as a continuous component in our sub-graph similarly to [34] it would be possible that two distinguishable rows would be identified as the same row. As already mentioned in Section 2.1 the computation of maximal cliques is computationally expensive.

4.4 Using Graph Convolutional Neural Networks for Table Recognition

In the work of *Prieto et al.* [34] an approach is described in which the authors use a graph convolutional neural network (abbrev. GCN) to detect columns, rows and cells in the image. The approach uses the 2-pass GCN introduced in this paper. The 2-pass GCN consists of two separate GCNs.

The first one, GCN_1, is used to prune the initial graph. It assigns to each edge probability that indicates how likely it is that nodes connected by the edge have any relationship with each other whatsoever. Edges with a probability lower than a given threshold are removed from the graph. The pruned graph is then passed to GCN_2. The goal of GCN_2 is to prune the graph's edges once again but this time it should output continuous components which can be interpreted either as a rows or columns of the table. This approach requires to have separate model for detection of column edges and model for detection of row edges.

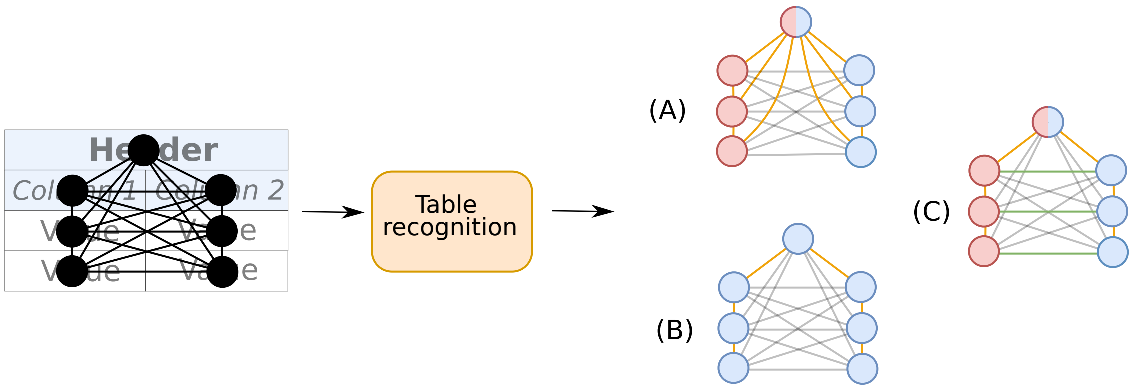


Figure 4.6: Example of different table recognition outputs for column detection. In graph *(A)* columns are represented by maximal cliques in the output graph. In graph *(B)* columns are represented by continuous components. Notice that this method does not allow to distinguish between columns that span across multiple columns and those that do not. In graph *(C)* edges indicate both vertical and horizontal relationships between nodes. When we merge these two information together we might be able to detect the column span of the detected column.

The detection process of rows and columns in a table goes as follows. First, rows are detected in the image and then the columns. By merging the detected rows and columns, we also are able to detect cells in the image.

A disadvantage of this method is that it is incapable of dealing with row-spanned and col-spanned cells. This can be viewed as a big disadvantage as the historical documents often contain cells that span across multiple columns/rows. There is potential to further improve the method by using visual features like in [36].

Chapter 5

Implementation

The model implemented within this thesis is based on Graph Neural Networks (abbrev. GNN), as it turns out that the GNN based approaches lead to good results on datasets consisting of historical tables [8, 33, 34].

The proposed table recognition pipeline consists of several stages (Figure 5.1). The first important step uses Optical Character Recognition (abbrev. OCR) to detect text lines in a given image. The detected text lines are used to represent nodes in the graph in the latter stages. We use an already existing OCR model – the Pero OCR[1] [22, 24, 25].

Once we obtain the detected text lines, we generate the feature vectors that are assigned to them. These feature vectors have two portions. The first portion of the feature vectors carries information like the size of the detected text line and its position in the image. The second portion is calculated using a small Convolutional Neural Network *(abbrev. CNN)*, similarly as in [8].

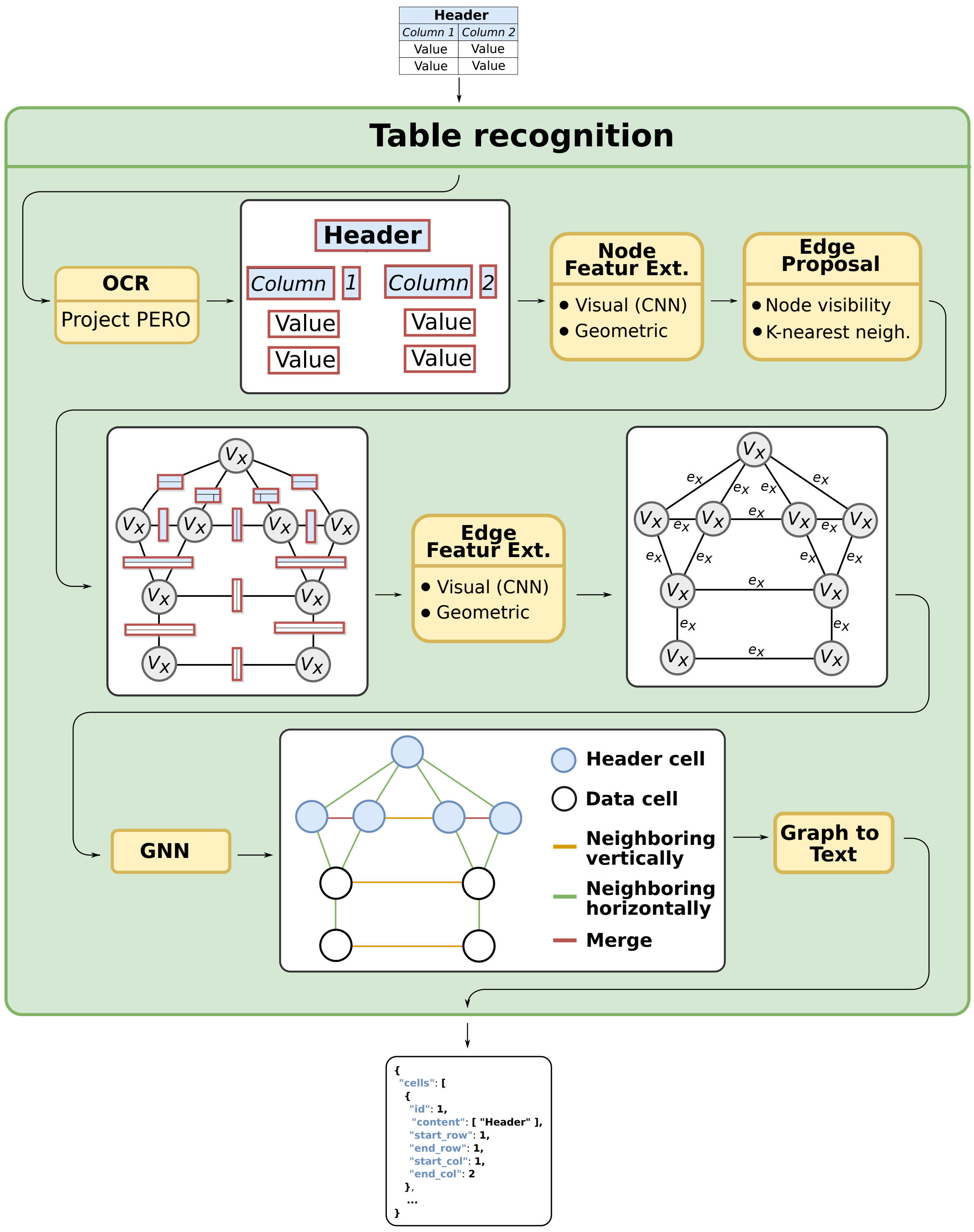
The edge proposal section of the pipeline tries to find the most important connections between the detected nodes. We experiment with the k-nearest neighbours algorithm and with the node visibility algorithm, which connects nodes that are visible from the point of view of a given text line [7, 33, 34].

Once the connections between the nodes are established, we follow a similar procedure as with the nodes – we create a feature vector that carries both geometric and visual features of a given edge. The geometric features of the given edge are, for example, the length of the edge or the angle at which the edge coincides with the x-axis.

The final graph representation of the input table is passed to a GNN that classifies both nodes and edges. We classify the nodes into two classes (header cell and data cell) and the edges into four classes (horizontal edge, vertical edge, edge connecting text lines belonging to the same table cell and so called no-relationship edge). The used GNN models are based on the GAT layer from [39] and the layer from the work of *Diehl*[2] [10]. The architecture of the GNN is also inspired by [8]. To test and train the GNN model, we use the cTDaR and GloSAT datasets [12, 30]. We also use the ABP dataset for the final evaluation of the implemented process [17].

Finally, the resulting graph with classified edges and nodes is then transformed into the text representation in the graph to text transformation stage. In this stage, we prune the graph and try to fit the nodes into a grid. Using this grid, it is then easy to transform the graph into its text representation.

[1]https://github.com/DCGM/pero-ocr
[2]https://github.com/fgerzer/gnn_acopf



Header	
Column 1	*Column 2*
Value	Value
Value	Value

```
{
 "cells": [
  {
   "id": 1,
   "content": [ "Header" ],
   "start_row": 1,
   "end_row": 1,
   "start_col": 1,
   "end_col": 2
  },
  ...
}
```

Figure 5.1: Scheme of the proposed system. The proposed system consists of several important blocks: *the OCR block* detects pieces of text in the input table, *the node feature extraction block* creates feature vector representing nodes in the initial graph, *the edge proposal block* proposes edges that should be part of the initial graph, *the edge feature extraction block* creates feature vectors representing edges in the initial graph, *the GNN block* classifies nodes and relationships between them, and finally the *the graph to text block* transforms the output graph of *the GNN block* to the text representation.

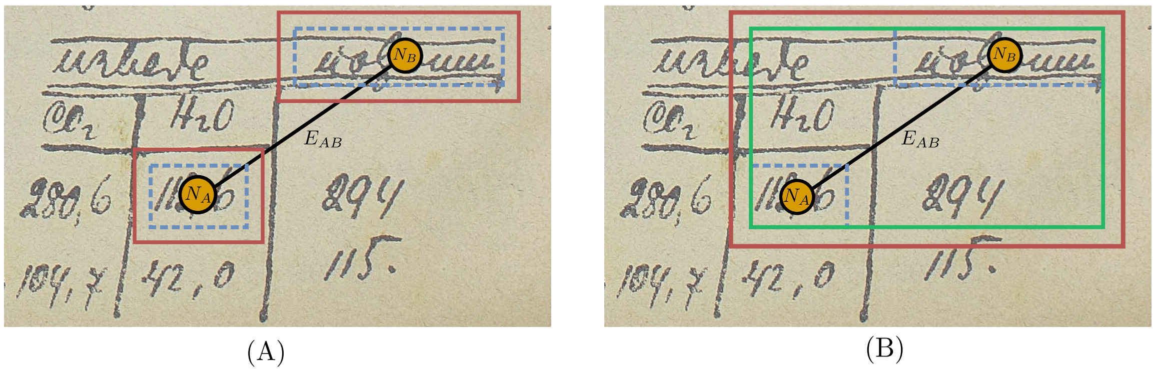


Figure 5.2: This picture depicts regions that are used for calculation of node and edge visual features. *(A)* Regions detected by the OCR are highlighted by the dashed blue bounding box, and the regions that belong to nodes N_A and N_B are depicted by the red bounding boxes. Similarly, in the picture *(B)* the region that is used to calculate the visual features for edge E_{AB} is depicted by the red bounding box. Idea taken from [8].

5.1 Graph Initialization

Graph initialization plays an important role in the process of table recognition. The more precise and relevant information we provide to the neural network, the more precise outcomes we might expect. The process of graph initialization consists of two steps – node discovery and edge discovery. The node discovery part is completed using an OCR model, which is used to find text lines in the table that is being processed. The graph is then completed by discovering a set of nodes that interconnect the discovered nodes. In this thesis, two methods were used for experimentation: k-nearest neighbours and node visibility.

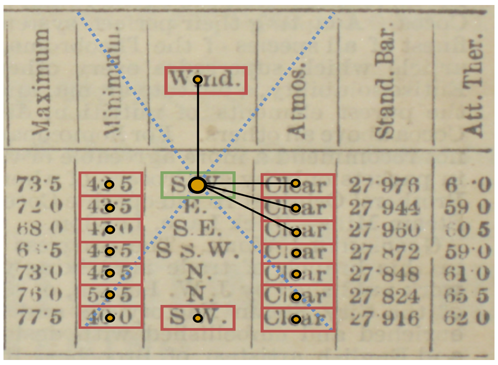
Node Discovery

Each node is represented by both geometric and visual features. To represent geometric properties of the detected text lines, the following attributes were utilized: the normalized position of the node and the normalized width/height of the bounding box. The normalized attributes are normalized with respect to the image dimensions, and the node position is the position of the bounding box centre of the detected text line.

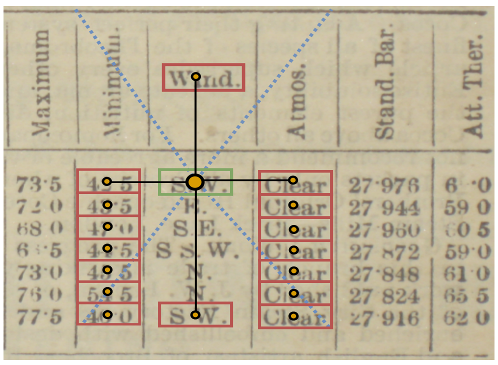
The region used for calculating the visual feature vector is defined by the padded bounding box with ten pixels on each side (Figure 5.2). The padded region is down-sampled using the ROI Align algorithm to image with a resolution of 10x10 [16]. And finally, the down-sampled image is used to obtain the final visual feature vector using a CNN.

Edge Discovery

To connect the discovered nodes, k-nearest neighbours algorithm and node visibility algorithm were used. The implementation of the k-nearest algorithm is straightforward and requires each node to find its k nearest neighbours and connect the given node with the discovered neighbours. The node visibility algorithm (inspired by [7, 33, 34]) is the more complicated of the two used algorithms. It connects each node with nodes that are visible from the point of view of the currently processed node. To be precise, the algorithm does



(A)



(B)

Figure 5.3: Example of usage of the node visibility algorithm for edge discovery with and without buckets. We use four buckets in this example that are denoted by the blue dotted lines and we also prevent each node from having more than four neighbours. The *red rectangles* denote all discovered text lines by OCR, which are visible from the point of view of the currently processed text line *(green rectangle)*. In the image *(A)*, the buckets are ignored, and in the image *(B)*, the buckets are used, which leads to a better distribution of the discovered edges.

not connect the node with every visible node but selects a specific number of visible nodes as connecting the node with every visible node can rapidly increase the number of edges in the graph.

The process of selecting the specific number of visible nodes goes as follows. First, rays are casted from the centre of a bounding box of the currently processed node. These casted rays intersect with a set of surrounding nodes. When a casted ray intersects a bounding box of another node, the angle the ray makes with x-axis is stored together with the discovered node. The discovered visible nodes are then split into buckets according to the angle of the casted ray which discovered them. The algorithm then iteratively goes through these buckets, and each time it visits a bucket, it picks the closest node to the currently processed node and adds an edge to the graph which connects these two nodes (Figure 5.3). The process of adding edges ends once we have discovered the specified maximum of edges or there are no edges left in the buckets.

The parameters that influence the outcome of the described algorithm are the number of buckets, the maximum number of edges each node can discover and the sampling rate that defines how many rays are casted from each node. The sampling rate plays an important role when it comes to the speed of the algorithm. The more rays are casted, the slower the algorithm is. When it comes to finding the correct parameters, it is essential to consider the bucket size and the number of edges each node can discover.

Usually, we want the number of edges each node can discover to be equal to multiples of the bucket number. Suppose this condition is not met, then in certain cases, it may happen that the algorithm will prefer to connect nodes that have a horizontal relationship over nodes that have a vertical relationship and vice versa (Figure 5.3). This is not welcomed as preferring one type of connection over another one can lead to an incomplete graph in which texts from neighbouring cells will not be connected.

If neighbouring text lines are not connected, then the relationship between them can not be established, which leads to incorrect output of the table recognition process. However, finding the correct parameters is not difficult, and it seems that once they are discovered, then they work with a wide variety of tables without the need to recalibrate them (Section 6.4). In our implementation, we always set the bucket number to be equal to the number of edges each node can discover. If not stated otherwise, we use six buckets in the following chapters.

Similarly to the nodes, once an edge is discovered, a feature vector that encompasses both geometric and visual features is assigned to it. The geometric features that are calculated for each edge are the normalized Euclidean distance between the connected nodes, the normalized positions of both nodes, the angle the edges make with the y-axis and the normalized overlap of bounding boxes in the direction of x and y-axis.

The visual features for each edge are calculated, similarly as with nodes, using pretrained CNN. The region passed to the CNN is the smallest rectangular region containing both nodes' bounding boxes (Figure 5.2). The region is also padded with ten pixels on each side and down-sampled using the ROI Align algorithm before passing it to the CNN. The region processed by the ROI Align algorithm has a resolution of 16x16 pixels. The idea for calculation of visual features for both nodes and edges comes from the work of *Davis et. al.* [8].

When it comes to edges, the idea behind the used visual features is that it might help a graph neural network to get some visual information about the space that is between the nodes. Suppose there is, for example, a line between them. In that case, the graph neural network might learn that the connected nodes occupy different table cells and similarly, when there is no significant visual mark between them, it might mean that the text lines are in the same table cell.

5.2 Models Description

A graph neural network (abbrev. GNN) was utilised to get the desired graph with classified edges and nodes. To find the best graph neural network, two main models were experimented with.

Graph Neural Network Based on GAT Layer

The first model is the less accurate one (further referred to as the baseline model). It is based on the GAT layer (Section 3.1) and lacks several important properties when compared with the second model. First, the model does not work with the visual attributes of nodes and edges at all. Secondly, the propagation of information that is necessary for edge classification can be questioned.

The first layer of the model is a GAT layer which processes the feature vectors assigned to nodes and edges to calculate new values for the node feature vectors. The second layer uses Multi-Layer Perceptron (abbrev. MLP) to classify edges using the newly calculated nodes' feature vectors. The last layer classifies the edges again using another GAT layer (Figure 5.4). The network works, but the propagation of the information from the edge feature vectors is not good as the GAT layer calculates the output only out of the nodes' feature vectors. The edge feature vectors are used to calculate only weights that help the GAT layer understand the importance of individual nodes' neighbours. This makes the model very ineffective.

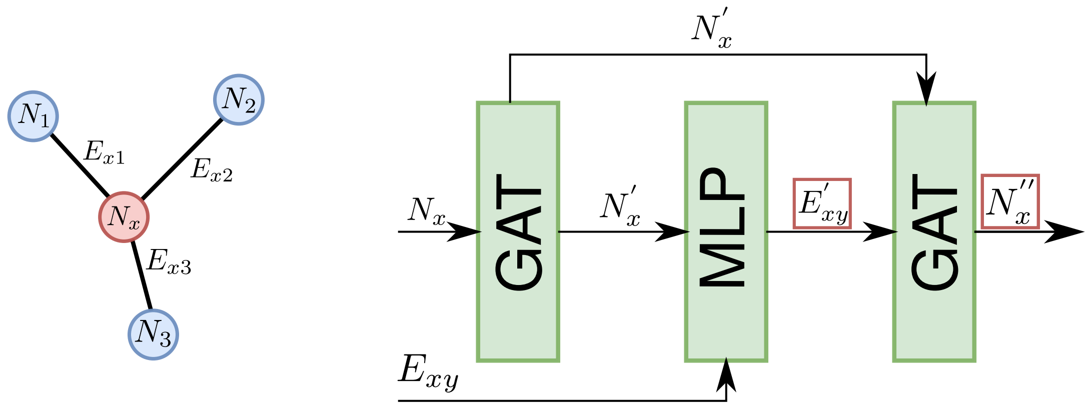


Figure 5.4: Computational graph of the GNN which is based on the GAT layer. The GNN consists of two GAT layers that propagate the information between nodes and one MLP section that is used to classify the edges. In the image the red rectangles denote output values of the GNN.

Graph Neural Network Using Visual Features

Another model that was experimented with consists of three main components (Figure 5.5):

- GNN from the work of *Diehl* [10, 11] that is used to propagate the information from the node and edge feature vectors throughout the network (Section 3.1). This component is further refered to as **GNN section**.
- CNN used to calculate the node and edge visual feature vectors which has the same architecture as the CNN in the work of *Davis et al.* [8]. This component is further referred to as **CNN section**.
- MLP that classifies the edges and nodes using the transformed node and edge feature vectors. This component is further referred to as **MLP section**. This section is also inspired by [8].

The **GNN section** promises better propagation of the information between both nodes and edges (when compared with the first model). The propagation is better because the calculation of new node feature vectors is not dependent only on the neighbouring node feature vectors but also on the value of the neighbouring edge feature vectors.

As outlined in Section 5.1 node and edge feature vectors consist of visual and geometric features. The visual features are calculated out of the image regions in the **CNN section** and then concatenated with the geometric features. This section uses depthwise separable convolution to speed up the process of visual feature calculation.

Once the visual feature vectors are prepared, and the GNN section propagates the information throughout the graph, the transformed feature vectors are passed to **MLP section**. There are two MLP sections: the first one is used to classify the edges, and the second one is used for node classification. Both sections consist of two linear layers. The first one uses the ReLU non-linear function and the second layer ends with the softmax function that transforms the logits to probabilities of each node class.

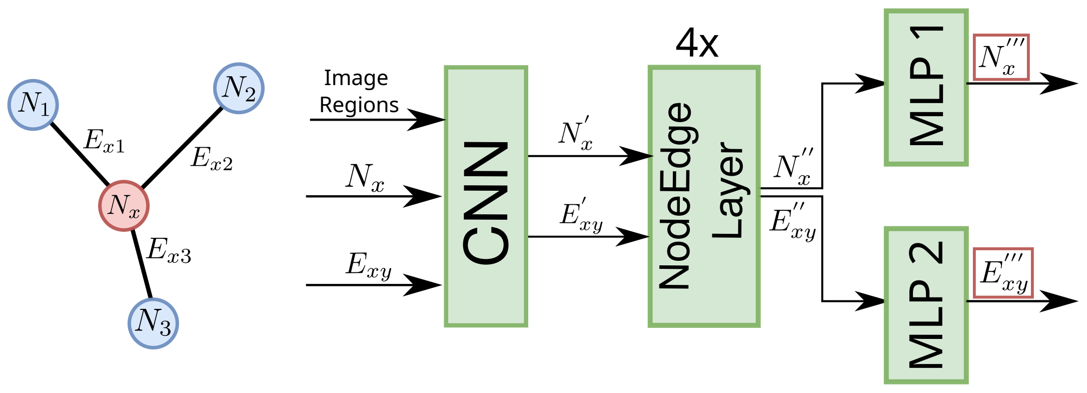


Figure 5.5: Computational graph of the graph neural network that uses the visual features. The network consists of the GNN section, CNN section and MLP section. The CNN section is used to get the visual features, the GNN section propagates the information in feature vectors throughout the network, and MLP classifies the edges and nodes. In the image, the red rectangles denote output values of the GNN and NodeEdge Layer is a name used for GNN layer from [10, 11].

5.3 Transformation of Graph to Text Representation

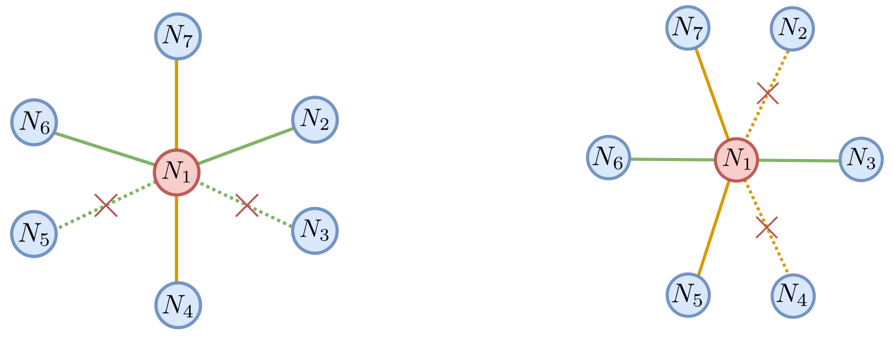
We encounter a problem once we have a model that can classify edges into the four mentioned classes we use (horizontal/vertical neighbours, no-relationship edge and same-cell edge). That is a transformation of the graph into a structured text that encaptures the table structure of the table in the image. To do this, a three-step process was used. The process first tries to get rid of unnecessary edges (in the edge removal phase), then straightens edges that connect nodes in the same row/column (in the straightening phase) and finally puts the node into a grid (in the graph to grid transformation phase). Using the created grid, it is then easy to transform the table to any desired format (HTML, ICDAR XML, ...). It is worth pointing out that this algorithm neglects the row-span and column-span of individual table cells. This means that tables with table cells that have cells that span across multiple rows/columns will never be transformed into text representation that resembles the table structure accurately.

Edge Removal Phase

In the edge removal phase, two types of edges are removed from the graph. First transitive reduction for both horizontal and vertical edges is performed to remove transitive edges. By transitive edges of graph G with vertices V and edges E, we understand edges $(n_1, n_2) \in E$ for which there exists a path from node n_1 to node n_2 whose length is greater than one [6].

Unfortunately, the transitive reduction algorithm is only defined for directed acyclic graphs, which implies the need to decide the direction of each edge in the output graph as our model's output is an undirected graph.

To decide the direction of each edge, the nodes' positions in the image are used. For example, when deciding about the edge direction of a horizontal edge, the direction of each edge that connects two nodes n_1 and n_2 is decided based on the position of nodes in the image.



(A) Horizontal Edges Removal (B) Vertical Edges Removal

Figure 5.6: Example of how excess vertical/horizontal edges are removed for nodes (N_1) with more than two vertical/horizontal edges connected to themselves. The green colour is used to highlight edges that are classified as horizontal edges, and the orange colour is used for edges classified as vertical edges. Horizontal edges of each node are divided into two groups: the edges to the left of the node ((N_1, N_5), (N_1, N_6)) and the edges to the right the node ((N_1, N_3), (N_1, N_2)). From the groups, edges are removed so that only one edge is left in each group. A similar process is applied to the vertical edges.

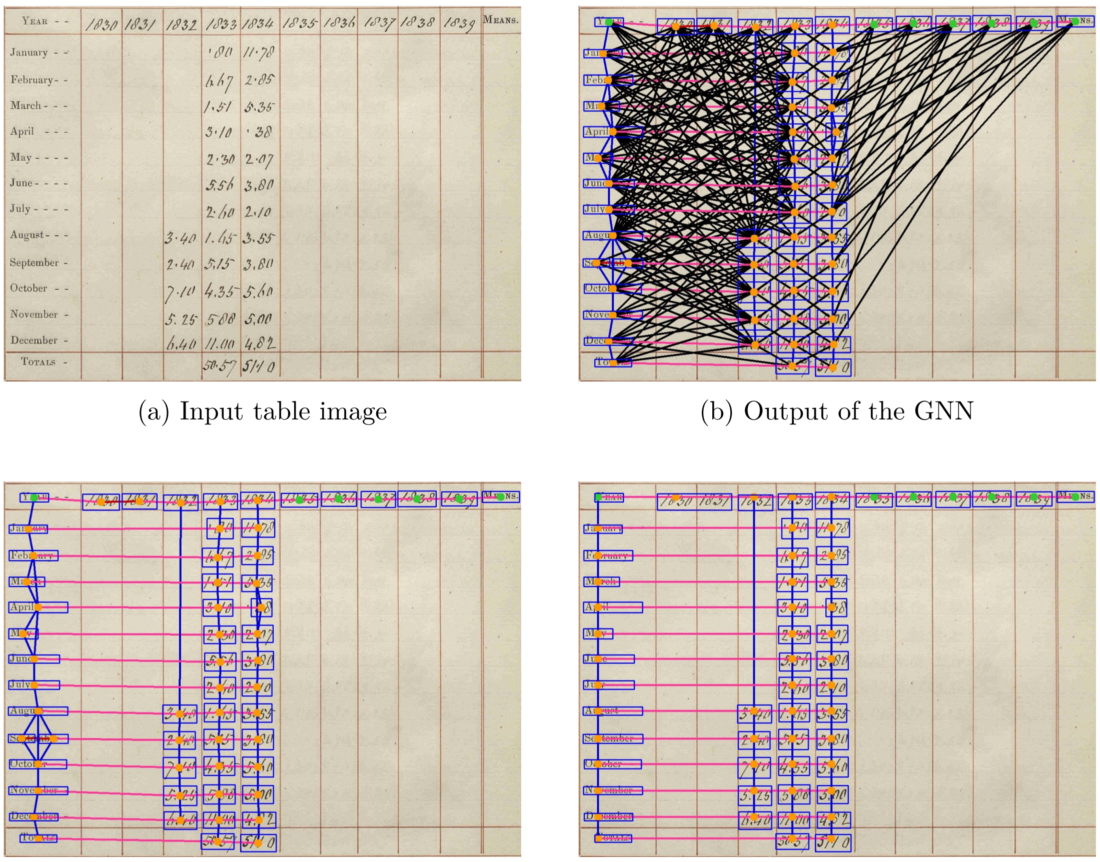
If n_1 is to the right of the node n_2 then the edge (n_2, n_1) is removed from the graph and (n_1, n_2) is kept. Similarly, when deciding about the edge direction for vertical edges and n_1 is above the node n_2, then the edge (n_2, n_1) is removed from the graph and (n_1, n_2) is kept.

Once the direction of each edge is decided, the transitive reduction is performed on horizontal and vertical edges separately. To perform the transitive reduction, we use a function from the Networkx library[3]. This process removes the most significant number of edges from the graph. After this step is finished, the graph should not contain any transitive edges, neither for columns nor for rows.

The next step of the edge removal phase tries to remove such edges that are connected to nodes that have more than two horizontal/vertical edges. Every node is visited once during this step, and for each node, horizontal and vertical edges are removed so that each node has only two vertical and horizontal edges. The question presents itself, how are the edges which are removed selected. The answer to this question can be seen in Figure 5.6.

Suppose we want to remove horizontal edges for a given node so that the node has only two horizontal edges. First, the edges are split into two sets. The first set contains edges located to the left of the node (meaning the edges are connected to a node that is to the left of the given node). The second set contains edges located to the right of the node (meaning the edges are connected to a node that is to the right of the given node). From the first set, we keep only the edge that is connected to a node whose y-coordinate is the greatest. From the second set, we remove edges in the same way. Similar process is followed for vertical edges removal.

[3]https://networkx.org



(a) Input table image

(b) Output of the GNN

(c) Removal of the no-relationship edges

(d) Resulting graph

Figure 5.7: Image depicting four stages of the implemented table recognition process. First, text lines are detected using an OCR *(blue bounding boxes)*. After this, the initial graph is created, which is then passed to a GNN *(a,b)*. The output of the GNN contains classified edges into four classes *(pink - vertical neighbours, blue - horizontal neighbours, red - nodes belonging to the same cell, black - text lines that have no relationship with each other)*. The output graph finally undergoes the edge removal phase and the straightening phase *(c,d)*.

Straightening Phase

The goal of the straightening phase is to take the graph from the previous step and straighten nodes in the same row and the same column. Ideally, at the end of the process, we want to have a graph in which nodes in the same column share the x-coordinate and nodes in the same row share the y-coordinate, as it is shown in the Figure 5.7.

To get the straightened graph, the columns are straightened first and then the rows. However, the order in which the straightening is done should not matter. To perform the straightening of rows, we remove vertical edges from the graph and then find all continuous components in the graph. Each continuous component should contain nodes that belong to the same row. The straightening is then done straightforwardly by choosing the node with the highest value of the y-coordinate and by setting the y-coordinate of every node in the line to the found maximum value. The straightening of columns is performed similarly.

Graph to Grid Transformation Phase

This phase focuses only on the continuous components of the graph. It aims to obtain nodes packed together as tightly as possible while preserving the relationship between the nodes. Hopefully, when the packing is completed, the nodes' x-coordinates should correspond to the column to which the nodes belong, and the nodes' y-coordinates should correspond to the row to which the nodes belong.

We pack the nodes in two runs. In the first run, we pack the nodes along the y-axis and in the second run along the x-axis. Again, the order in which the nodes are packed should not matter.

The packing along the x-axis is done by first removing every horizontal edge and finding all continuous components in the graph. And then by representing continuous components by bounding boxes, which are stacked up to each other as tightly as possible.

The dimensions of a bounding box for a given continuous component with x-coordinate value x are defined by two points (x_{min}, y_{min}) and (x_{max}, y_{max}) in the image plane. y_{min} and y_{max} are the minimal respective maximal values of y-coordinates from all nodes in the given continuous component. And x_{min} is equal to value $x - 1$ while the value of x_{max} is equal to value $x + 1$.

The defined bounding boxes are, as already outlined, stacked to the left as much as possible it is. Finally, when the packing along the x-axis is completed, the value of every node's x-coordinate equals its column position in the final table. We follow a similar process for the packing along the y-axis.

Chapter 6

Experiments and Evaluation

In this chapter we focus on the experiments with the trained model. Particularly we focus on the process of model training, preparation of dataset and ground truth graphs that were used for the training and finally on experiments that were carried out as part of this thesis.

6.1 Used Dataset and Ground Truth Preparation

To train the models, we use cTDaR and GloSAT datasets described in Section 2.2. Both datasets provide the necessary information needed to train a model that is able to perform table recognition.

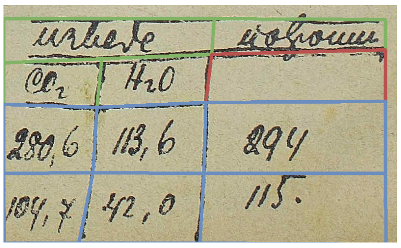
However, there was one thing that needed to be addressed, and that is the fact that the cTDaR dataset does not provide information about the cell type and therefore needs to be slightly modified before it is used to train a model that is capable of table cell classification.

It was experimented with dataset annotation tool CVAT[1] to classify the table cells in the cTDaR dataset. The tool at first seemed to be useful, but when editing large tables with many table cells, it slowed down significantly and sometimes did not react to user input. This was unbearable as it could take up to twenty minutes to modify a large table. Eventually, the best solution seemed to be to create a simple annotation tool that would be created with the only purpose in mind – to modify the table recognition datasets. This tool was created using the OpenCV library[2].

The user interface of the created tool is plain simple. It displays to the user a table and then iteratively highlights each table cell and waits for the user input. The user uses specific keys to classify the highlighted table cells into four categories: non-empty header/data cell, empty header/data cell and header/data cell containing symbols that are difficult to be classified by the used OCR. This created tool decreased the time needed for the table cell annotation. The modified cTDaR dataset consists of approx. 28.70 % empty cells.

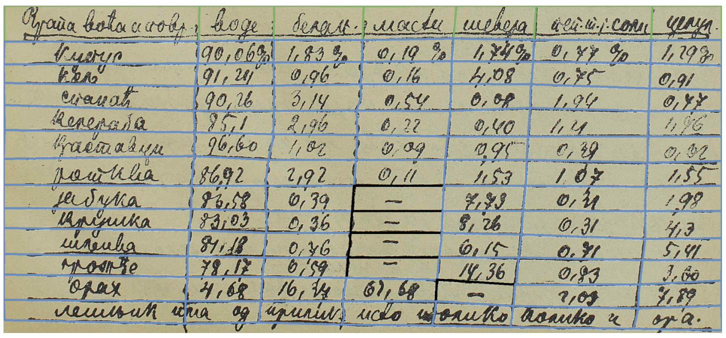
The modified cTDaR dataset and the GloSAT dataset were used together with the OCR outputs to generate ground truth graphs for the models' training. A ground truth graph G_t with nodes N_t and edges E_t is created in the following manner. Each detected text line in the OCR output represents a node in the graph. The class of each node is decided based on the class of the ground truth table cell with which the node has the highest value of IOU. The edges are discovered either with k-nearest neighbors method or node visibility method (Section 5.1).

[1]https://cvat.org
[2]https://opencv.org



угљене		водоник
CO_2	H_2O	
280,6	113,6	294
104,7	42,0	115.

(A)



Врста воћа и повр.	воде	беланч.	масти	шећера	ацет. ј. соли	целул.
Купус	90,06%	1,83%	0,19%	1,74%	0,77%	1,29%
Кељ	91,24	0,96	0,16	4,08	0,75	0,91
спанаћ	90,26	3,14	0,54	0,08	1,94	0,77
келераба	85,1	2,96	0,22	0,40	1,4	1,96
краставци	96,60	1,02	0,09	0,95	0,39	0,92
ротква	86,92	2,92	0,11	1,53	1,07	1,55
јабука	83,58	0,39	–	7,73	0,31	1,98
крушка	83,03	0,36	–	8,26	0,31	4,3
шљива	81,18	0,76	–	6,15	0,71	5,41
грожђе	78,17	0,59	–	14,36	0,83	3,60
орах	4,68	16,34	61,68	–	2,09	7,89
лешник има	од	прилике	исто	толико	колико и	ора.

(B)

Figure 6.1: Sample images from the modified cTDaR dataset. The blue rectangles represent data cells, the green rectangles represent header cells, the red rectangles represent empty header cells, and the black rectangles represent cells that contain a symbol that is unlikely to be detected by the Pero OCR.

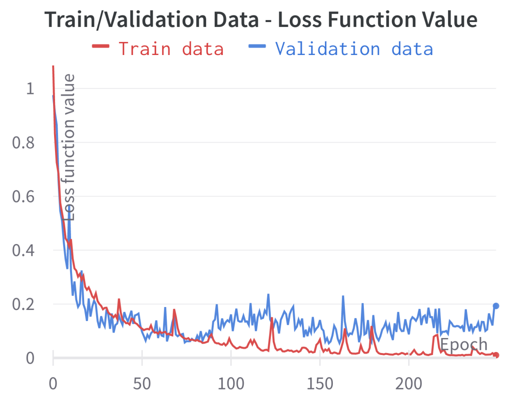
The class of each discovered edge is decided based on the relationship between the text lines it connects. To determine the type of relationship between two text lines $t1$ and $t2$ connected by edge e we find their corresponding cells c_1 and c_2 in the ground first. For a given text line a ground truth table cell with which the text line has the highest value of IOU is understood to be the matching ground truth table cell. Then the class of the edge e is decided as follows:

- If both c_1 and c_2 are in the same column but not in the same row then the edge e is called vertical edge.
- If both c_1 and c_2 are in the same row but not in the same column then the edge e is called horizontal edge.
- If $c_1 = c_2$ then the e is called same-cell edge.
- If neither of the mentioned rules applies then the edge e is called no-relationship edge.

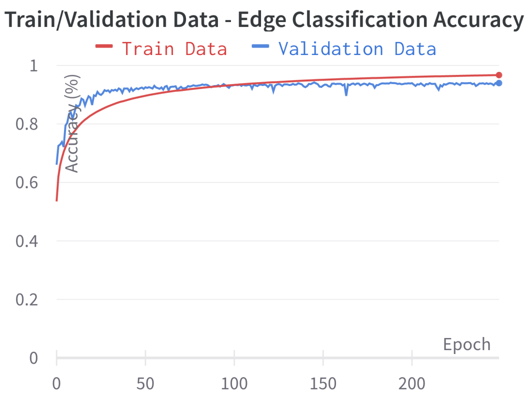
6.2 Models Training

The two mentioned models that scored the best on the test dataset were trained using both the cTDaR and GloSAT datasets. The training was done in two steps. First, initial graphs were created and stored on the disk so that the graph representation is not created from scratch at the beginning of every iteration, as for some tables, this task can be computationally heavy. Once the graphs had been prepared, the actual training started.

To train the model, the negative log-likelihood loss function was utilized. This function was used to minimize the error either of edge or node classification. It was experimented with the idea of training one model that would be able to perform the task of node and edge classification simultaneously, but this turned out to be a not good idea as the model was unable to reach the same accuracy as when it was trained to learn node and edge classification separately. Therefore two models were trained, one for node classification and another one for edge classification.



(a) Progress of loss function.



(b) Progress of edge classification accuracy.

Figure 6.2: Progress of both loss function and edge classification accuracy during training of the model for edge classification.

The model was trained on a test set that consists of 1277 tables in total (564 tables from the GloSAT dataset and 663 tables from the cTDaR dataset). Furthermore, the performance of the model during the training was measured on the validation dataset that consisted of 152 tables (70 tables from the GloSAT dataset and 82 tables from the cTDaR dataset). The performance of the model during training is calculated as the ratio of the number of correctly classified nodes/edges and the total number of nodes/edges in the graph.

It turned out that the batch size of 16 reached the best results. As an optimizer function, the standard ADAM optimizer was used [20]. It performed well during training; however, the correct learning rate needed to be found as for some learning rate values, the model was unable to escape the local minima of the cost function. Nevertheless, after a few experiments, the value $3e^{-4}$ lead to the best accuracy on the validation dataset.

We trained the model for 250 epochs. Longer training was unnecessary because there was no significant trend in the loss function that would indicate room for the neural network to continue to learn more from the train dataset (Figure 6.2).

6.3 Evaluation and Results Discussion

We evaluate the performance of the designed table recognition process using two sets of metrics. The first set of metrics consists of metrics introduced in this thesis which focus on the ability of the table recognition process to detect cells (cell precision, cell recall) and the ability to correctly detect horizontal and vertical relationships between table cells (horizontal/vertical relationships precision). We also evaluate how accurately is the model able to classify table cells. The second set of metrics consists of slightly modified metrics that are usually used to calculate the performance of a table recognition process on the cTDaR dataset.

Description of the Used Metrics

To calculate the table cell detection precision and recall, we have to find correctly detected cells. To do so, we calculate for each table cell intersection value I with all table cells in the ground truth. For a given detected table cell region r, the intersection value with each table cell $\bar{r}$ in the ground truth is calculated as follows:

$$I = \frac{intersectArea(r, \bar{r})}{area(r)}, \tag{6.1}$$

where $intersectArea$ is a function that calculates the intersection area of two bounding boxes and $area$ is a function that calculates the area of a given bounding box.

When we find $\bar{r}$ in the ground truth for which the value of I is higher than a selected threshold, we pronounce $\bar{r}$ to be correctly detected by the model. However, if we discover more than one region r that can be assigned to $\bar{r}$ during the evaluation, then the $\bar{r}$ is no longer counted as a correctly detected. We use I rather than Intersection over Union (abbrev. IOU) because sometimes the region of the detected text line takes up a small area with respect to the area of the table cell it occupies. This would lead to low IOU and therefore to the table cell being marked as wrongly detected even though the detection is correct.

When it comes to measuring how well the designed process is able to capture the table structure of a given table, the horizontal/vertical relationship detection precision was utilized. For example, the horizontal relationship detection precision is used to measure how well is the designed table recognition process able to detect horizontal relationships between the detected table cells. To do that, it focuses on how many correctly detected table cells exist in the output with correctly detected horizontal neighbours. To check whether a given cell c is a cell with correctly detected horizontal neighbours, we follow this two-step process:

1. We check whether the cell c is correctly detected cell using the intersection value I. If c is not a correctly detected cell, then we count it as a cell with incorrectly detected horizontal relationships.

2. We find for each horizontal neighbour of node c corresponding table cell in the ground truth using the intersection value I. If at least one neighbouring cell does not have a corresponding cell in the ground truth (meaning it is an incorrectly detected table cell), then c is pronounced to be a cell with incorrectly detected horizontal relationships. And finally, if every horizontal neighbour of the node c is mapped to the row column as the node c then the node is counted as a node with correctly detected horizontal relationships.

Once we complete the two-step process for every detected table cell, the final accuracy is calculated as the number of cells with correctly detected horizontal neighbours divided by the total number of detected cells. The same process is used to calculate the precision of detected vertical relationships.

To compare the implemented table recognition process wih other implementations we use the metric that is used for the cTDaR dataset [13]. To calculate the precision value we have to find how many correctly discovered vertical and horizontal neighbours there are in the output of the table recognition process and divide it by the total number of all horizontal and vertical relationships in the output.

Evaluation *(cell detection threshold = 0.80)*				
Dataset	*Horizontal Relationships Precision*	*Vertical Relationships Precision*	*Cell Precision*	*Cell Recall*
cTDaR	0.55 (0.27)	0.59 (0.31)	0.72 (0.44)	0.50 (0.40)
GloSAT	0.57 (0.44)	0.61 (0.46)	0.78 (0.64)	0.40 (0.36)
ABP	0.68 (0.07)	0.71 (0.12)	0.85 (0.21)	0.72 (0.32)

Table 6.1: Table containing the resulting values of used metrics for the model based on the visual features and the baseline model (values in the brackets).

	Header Cell	*Data Cell*
Header Cell	0.81 (0.01)	0.19 (0.99)
Data Cell	0.01 (0.47)	0.99 (0.53)

(a) cTDaR dataset

	Header Cell	*Data Cell*
Header Cell	0.68 (0.00)	0.32 (1.00)
Data Cell	0.02 (0.69)	0.98 (0.31)

(b) GloSAT dataset

Table 6.2: Normalized confusion matrices for the table cell classification. The results are calculated for both the model based on the visual features and the baseline model (values in the brackets). Rows indicate prediction by the model and columns ground truth value.

And similarly to calculate the recall value we divide the number of correctly detected relationships by total number of relationships in the ground truth. We modify the metric slightly by changing the rule for correctly detected cell. In the original metric cell is pronounced to be correctly detected when IOU of the detected cell with a ground truth cell is above a given threshold. Instead of using IOU we use I (Equation 6.1) similarly as with the previous metrics.

Results Discussion

We evaluated the baseline model and the model based on visual features using the cTDaR, the GloSAT and the ABP datasets [12, 17, 30]. The cTDaR dataset contains a few table images similar to images from the ABP dataset. Therefore, even though the models were not trained on the ABP dataset, they still could learn some patterns from it.

When observing the results from Table 6.1 we can notice that the model based on visual features scored the best on the ABP dataset. The reason is that the ABP dataset contains tables with relatively simple structures compared with the other two datasets. The simplicity lies in the fact that the majority of tables from the dataset do not contain table cells that span across multiple table cells.

If we take a look at the results of the baseline model, we can notice fairly low values for all metrics. After investigating the output of the model, it turns out that the baseline model was able to detect horizontal and vertical relationships quite well but where it lacks is the ability to detect text lines that belong to the same table cell.

On the other hand, the worst cell recall values were obtained for the GloSAT dataset. This can be explained by the following: a) the OCR often returns text lines that span across multiple table cells for the GloSAT dataset, and b) there are many empty table cells in the dataset (e.g., the cTDaR dataset consists of 28.70 % empty table cells and the GloSAT seems to contain tables that have more empty cells than the cTDaR dataset).

Comparison Of Table Recognition Approaches		
Implementation	*Precision*	*Recall*
NLPR-PAL [13]	0.69	0.75
Our results (*)	0.66 (0.36)	0.43 (0.29)
HCL IDORAN [13]	0.23	0.04

Table 6.3: Results calculated on the cTDaR dataset using the metric from [13]. Table cell must have IOU higher than 0.70 to be counted as a correctly detected one. (*) For our table recognition model we count table cell to be correctly detected when the value of I is higher than 0.80. Metric values for the baseline model are shown in the brackets.

When it comes to the classification of the detected table cells, the baseline model provides hardly any useful information (Table 6.2). The other model is able to classify the nodes more accurately, but there is definitely a space for improvement as the model is not very accurate at detecting header cells. This can be caused by the lower number of header cells in the dataset.

The usage of the cTDaR metric allows us to compare it with the other two methods that were evaluated on it (Table 6.3). Even though we use I instead of IOU to pronounce table cells to be correctly detected, it seems that our approach is not the worst. Where it definitely lacks, when compared with the approach that scored first, is the recall value.

Generally, the model based on the visual features gives more stable results. However, even the baseline model can give some insight into the table structure of some tables, especially then tables with not too complicated topology and tables that do not contain multiple text lines in a single cell.

Possible Future Development

Several potential improvements may lead to better metric values. As a first thing, we can deal with the low cell recall values by trying to estimate the position of the empty cell. We know the average spacing between table cells in x-direction and y-direction, and we are also able in some cases to find the column and the row position of an empty cell. This information, together with the information about the spacing in the x and y-direction, can be used to estimate the position of empty cells in the image. To increase the detection accuracy of header cells, we can expand existing datasets so that they contain information about the cell type.

Future development can also focus on handling cells that span across multiple rows and columns. This is a quite challenging task and requires making major changes in the graph to text transformation phase. This can be done by allowing each node to have more than two horizontal/vertical edges. This idea was experimented with very briefly and turned out not to be very accurate due to the relatively low accuracy of edge classification. We can increase the accuracy by improving the used GNN model. We may, for example, experiment with a multi-head attention aggregation function as in [8].

An interesting idea might be to make the process of table recognition semi-automatic (especially then for historical documents). The semi-automatic process would consist of allowing the user to modify the resulting pruned graph with classified edges. The user could interactively remove incorrectly classified edges. This could lead to better results; however, the question presents itself, whether there is a use for such an implementation.

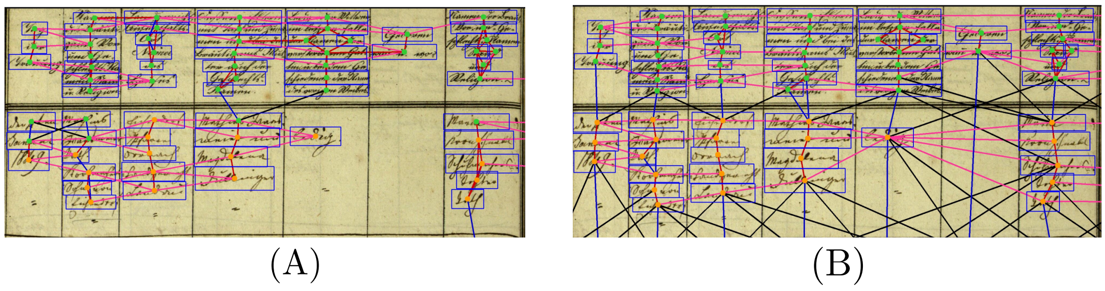


Figure 6.3: An image that shows the difference between the graph initialization with the k-nearest neighbours method *(A)* and the node visibility method *(B)*. The k-nearest neighbour method sometimes does not find an important connection between the text lines. In contrast, the node visibility algorithm connects text lines even when the distance between them is bigger than expected.

6.4 Graph Initialization and Models Training Experiments

Several experiments were conducted to find the best approach for the graph initialization and the model training. The first tries to empirically find the best value of the bucket size parameter of the node visibility algorithm *(Experimenting With the Bucket Size)*. The second experiment compares the graph initialization methods *(Comparison of Edge Discovery Methods)*. And finally, we focus on model training and model inference *(Node and Edge Classification in Single-Pass and Experimenting With Second Iteration)*.

Experimenting With the Bucket Size

The parameter that influences the output of the node visibility algorithm the most is the size of the bucket (the number of buckets we use). The bigger the bucket is, the fewer edges need to be classified by the model. The hypothesis is that if there are more edges than is necessary, the more difficult it is for the model to classify them as it has to find more subtle differences between the edges. This can lead to setting the bucket size to the biggest size possible. However, if the bucket is too big, then the initial graph will miss some edges, and therefore the output of the table recognition process will not be good enough.

Nonetheless, as seen in Figure 6.4, the hypothesis was declined as the model gave stable results for all the tested bucket size values. It was not experimented with bigger bucket sizes as sizes bigger than 45° will lead to the algorithm skipping edges in an either horizontal or vertical direction.

Comparison of Edge Discovery Methods

In this thesis, two algorithms for edge discovery were used. The k-nearest neighbours algorithm and the node visibility algorithm. Even though we do not present any quantified comparison of these two algorithms, a few experiments were conducted. The node visibility algorithm obviously outperforms the k-nearest neighbours algorithm.

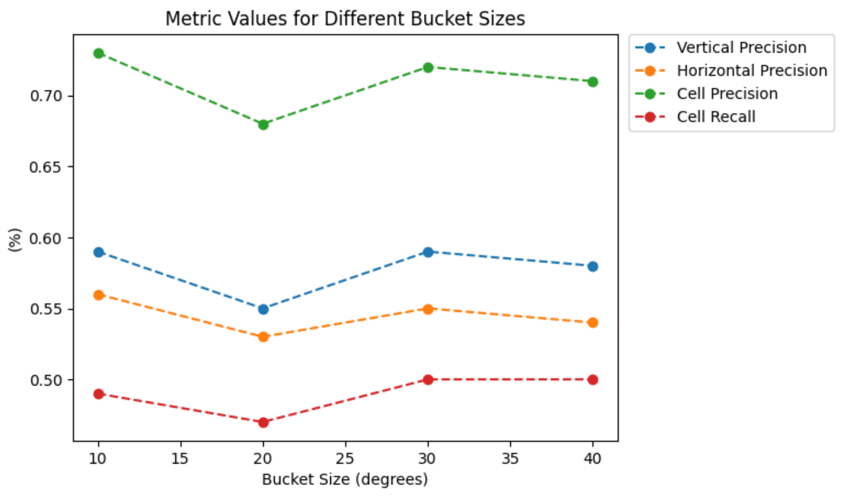


Figure 6.4: A graph displaying how the implemented table recognition process performs for different bucket size values (parameter of the node visibility algorithm). We can see that there is no significant difference between the tested sizes. The only noticeable difference is between the horizontal edge detection accuracy for the bucket sizes 20 and 30 degrees. Values in the graph are calculated on the cTDaR dataset [12].

The main drawbacks of the k-nearest neighbour algorithm are these two facts:

- if the table cells are not evenly spaced, then for lower values of k the algorithm tends to prefer to connect nodes in the horizontal direction than in the vertical direction or vice versa (Figure 6.3), and
- if there are two neighbouring text lines that are in the same column/row and there are a lot of empty cells between them, then the algorithm does not connect them (Figure 6.3).

However, there is one particular advantage of the k-nearest algorithm over the node visibility algorithm. The k-nearest algorithm is much faster than the node visibility algorithm. There is definitely space for the optimization as:

- the table recognition process using the node visibility algorithm can take up to 45 seconds on average in comparison with
- the table recognition process using the k-nearest neighbours algorithm that needs on average 12 seconds per table.

Even though there is space for optimization, there is still a big chance that the algorithm will still be slower even after the optimization. This is caused by the fact that the line casting from the centre of the bounding boxes and calculation of the intersection of the casted line with other bounding boxes is a more complex task than finding k nearest neighbours of a given node.

Node and Edge Classification in Single-Pass

We train two separate models for the table recognition pipeline (one for the edge classification and one for the node classification). This approach does not lead to the simplest processing of the table image as it needs to run through the GNN two times. It would be better if there was only one model that would be able to classify both nodes and edges.

We carried out an experiment in which it was tested whether a model can perform node and edge classification accurately after it was trained for both node and edge classification. We trained the model by trying to minimize the cost function that is calculated as the sum of loss function values for both the nodes and edges. It turned out, as already outlined in section 6.2, that the model learns to classify the edges and nodes, but in exchange, the model does not classify the nodes and edges as accurately as when the separate models are used.

The carried out experiment yielded the following results:

- When the model is trained to classify edges and nodes separately, then the edge classification accuracy on the validation dataset is 0.94.
- When the model is trained to classify both edges and nodes, then the edge classification accuracy on the validation dataset is 0.92.

Even though the difference between the accuracy values does not seem big, it can still play a big difference as even one incorrectly classified edge can lead to a completely different output of the table recognition process.

Experimenting With the Second Iteration

Another carried experiment is concerned with applying the trained model twice on the same graph. In the first iteration, we pass the graph to the model, which detects the no-relationship edges. These relationship edges are then removed from the graph, and the pruned graph is then passed to the model again. The hypothesis is that the model would be more accurate after the second iteration. However, this turned out to be false, as the metric values for the table recognition process without the second iteration are better than the metric values for the table recognition process with the second iteration.

- We obtained following metric values (on the ABP dataset) when applying the model twice on the same graph: *horizontal edge detection precision* - 0.65, *vertical edge detection precision* - 0.70, *cell detection precision* - 0.84, *class detection recall* - 0.76.
- And we obtained following metric values (on the ABP dataset) when applying the model once: *horizontal edge detection accuracy* - 0.68, *vertical edge detection precision* - 0.71, *cell detection precision* - 0.85, *class detection recall* - 0.78.

Chapter 7

Conclusion

This thesis aims to research the current trends in table recognition and then apply the learned information to create a table recognition pipeline that would be able to extract a structure of a given table. We focused mainly on the historical handwritten tables together with historical printed tables. There are many approaches that can be used to extract the information from a table image; however, when working with historical handwritten tables, the graph neural networks based approach seems to be currently the one which yields the best results.

Therefore, we decided to base our table recognition pipeline on graph neural network as well. The implemented pipeline consists of three stages: graph initialisation, edge and node classification and graph to text transformation stage. In the graph initialisation stage, we create an initial graph representation of the input table using an OCR model and edge discovery algorithms (k-nearest neighbours and node visibility algorithm). Then, in the edge and node classification stage, the nodes and edges are classified using the trained GNN. And in the last stage, the graph to text transformation stage, the graph with classified nodes and edges is transformed into a text representation.

We evaluated the implemented process with metrics that focus on how well the model is able to detect text belonging to a single cell and how it is capable of assigning correct horizontal and vertical neighbours to each correctly detected table cell. Using these metrics, the model turned out to give relatively good results. The implemented pipeline performs the best on the ABP dataset. On this dataset, the pipeline can detect horizontal neighbours with an accuracy of 0.68 and vertical neighbours with an accuracy of 0.71.

When it comes to cell detection, the pipeline reaches a precision of 0.85. However, the value for cell recall is lower – 0.78. The lower value of the cell recall is caused by the fact that the implemented pipeline ignores empty cells as the OCR cannot detect any text inside them. There is also space for improvement in the classification of the table cells. We tried to classify the table cells into two classes: header cell and data cell. The trained model is able to distinguish these two groups of table cells, but higher precision scores, especially then for header cells, would be welcomed.

There are many directions in which the future of the implemented pipeline can go. The most interesting one is probably the direction that leads to a pipeline that would correctly recognise the structure of tables that contain table cells that span across multiple rows and columns. Even though the implemented table recognition process has space for improvement, it still shows solid results.

Bibliography

[1] Mahalanobis Distance. In: *The Concise Encyclopedia of Statistics* [online]. New York, NY: Springer New York, 2008, p. 325–326 [cit. 2021-10-19]. DOI: 10.1007/978-0-387-32833-1_240. ISBN 978-0-387-32833-1. Available at: https://doi.org/10.1007/978-0-387-32833-1_240.

[2] AOMAR, A. A. *Over-smoothing issue in graph neural network* [online], 2021-06-07. 2016 [cit. 2022-01-15]. Available at: https://towardsdatascience.com/over-smoothing-issue-in-graph-neural-network-bddc8fbc2472.

[3] BYI, G. L., MÜLLER, M., THABET, A. and GHANEM, B. DeepGCNs: Can GCNs Go as Deep as CNNs? [online]. 2019, [cit. 2021-12-14]. Available at: https://arxiv.org/abs/1904.03751.

[4] CHEN, X. Understanding Spectral Graph Neural Network. [online]. 2012, [cit. 2021-12-14]. DOI: 10.13140/RG.2.2.27579.03364/1. Available at: https://arxiv.org/abs/2012.06660.

[5] CHI, Z., HUANG, H., XU, H.-D., YU, H., YIN, W. et al. Complicated Table Structure Recognition. [online]. 1.1. april 2019, [cit. 2021-12-14]. Available at: https://arxiv.org/abs/1908.04729.

[6] CLOUGH, J. R., GOLLINGS, J., LOACH, T. V. and EVANS, T. S. Transitive reduction of citation networks. *Journal of Complex Networks*. Oxford University Press (OUP). sep 2014, vol. 3, no. 2, p. 189–203. DOI: 10.1093/comnet/cnu039. Available at: https://doi.org/10.1093%2Fcomnet%2Fcnu039.

[7] DAVIS, B., MORSE, B., COHEN, S., PRICE, B. and TENSMEYER, C. *Deep Visual Template-Free Form Parsing*. arXiv, 2019. DOI: 10.48550/ARXIV.1909.02576. Available at: https://arxiv.org/abs/1909.02576.

[8] DAVIS, B., MORSE, B., PRICE, B., TENSMEYER, C. and WIGINTON, C. Visual FUDGE: Form Understanding via Dynamic Graph Editing. [online]. 2021, [cit. 2021-24-01]. DOI: 10.48550/ARXIV.2105.08194. Available at: https://arxiv.org/abs/2105.08194.

[9] DESAI, H., KAYAL, P. and SINGH, M. TabLeX: A Benchmark Dataset for Structure and Content Information Extraction from Scientific Tables. *Lecture Notes in Computer Science* [online]. 2021, p. 554—-569, [cit. 2022-01-16]. DOI: 10.1007/978-3-030-86331-9_36. ISSN 1611-3349. Available at: http://dx.doi.org/10.1007/978-3-030-86331-9_36.

[10] Diehl, F. Applying Graph Neural Networks on Heterogeneous Nodes and Edge Features. [online]. [cit. 2021-24-01]. Available at: https://grlearning.github.io/papers/6.pdf.

[11] Diehl, F. Warm-Starting AC Optimal Power Flow with Graph Neural Networks. [online]. [cit. 2021-24-01]. Available at: https://s3.us-east-1.amazonaws.com/climate-change-ai/papers/neurips2019/1/paper.pdf.

[12] Déjean, H., Meunier, J.-L., Gao, L., Huang, Y., Fang, Y. et al. ICDAR 2019 Competition on Table Detection and Recognition (cTDaR). [online]. april 2019, [cit. 2021-12-14]. DOI: 10.5281/zenodo.3239032. Available at: https://doi.org/10.5281/zenodo.3239032.

[13] Gao, L., Huang, Y., Déjean, H., Meunier, J.-L., Yan, Q. et al. ICDAR 2019 Competition on Table Detection and Recognition (cTDaR). In: *2019 International Conference on Document Analysis and Recognition (ICDAR)* [online]. 2019, p. 1510–1515 [cit. 2021-10-15]. DOI: 10.1109/ICDAR.2019.00243. Available at: https://ieeexplore.ieee.org/document/8978120.

[14] Gilani, A., Qasim, S. R., Malik, I. and Shafait, F. Table Detection Using Deep Learning. In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)* [online]. 2017, vol. 01, p. 771–776 [cit. 2022-01-05]. DOI: 10.1109/ICDAR.2017.131.

[15] Grattarola, D., Zambon, D., Bianchi, F. M. and Alippi, C. Understanding Pooling in Graph Neural Networks. [online]. 2021, [cit. 2022-24-01]. DOI: 10.48550/ARXIV.2110.05292. Available at: https://arxiv.org/abs/2110.05292.

[16] He, K., Gkioxari, G., Dollár, P. and Girshick, R. *Mask R-CNN*. arXiv, 2017. DOI: 10.48550/ARXIV.1703.06870. Available at: https://arxiv.org/abs/1703.06870.

[17] Hervé, D., Eva, L. and Florian, K. READ ABP Table datasets. [online]. 1.1. april 2018, [cit. 2021-12-14]. DOI: 10.5281/zenodo.1243098. Available at: https://doi.org/10.5281/zenodo.1243098.

[18] HTML, P.-V. S. for ICDAR 2021 Competition on Scientific Literature Parsing Task B: Table Recognition to. Jiaquan Ye and Xianbiao Qi and Yelin He and Yihao Chen and Dengyi Gu and Peng Gao and Rong Xiao. [online]. 2021, [cit. 2021-12-09]. Available at: https://arxiv.org/pdf/2105.01848.pdf.

[19] Kayal, P., Anand, M., Desai, H. and Sing, M. ICDAR 2021 Competition on Scientific Table Image Recognition to LaTeX. In: *Document Analysis and Recognition – ICDAR 2021* [online]. Cham: [b.n.], 2021 [cit. 2021-10-15]. ISBN 978-3-030-86337-1. Available at: https://arxiv.org/abs/2105.14426.

[20] Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. [online]. 2014, [cit. 2021-24-01]. DOI: https://doi.org/10.48550/arxiv.1412.6980. Available at: https://arxiv.org/abs/1412.6980.

[21] Kipf, T. N. and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. [online]. 2017, [cit. 2021-12-14]. Available at: https://tkipf.github.io/graph-convolutional-networks/.

[22] Kišš, M., Beneš, K. and Hradiš, M. AT-ST: Self-training Adaptation Strategy for OCR in Domains with Limited Transcriptions. In: *Document Analysis and Recognition – ICDAR 2021* [online]. Springer International Publishing, 2021, p. 463–477 [cit. 2022-05-09]. DOI: 10.1007/978-3-030-86337-1_31. Available at: https://doi.org/10.1007%2F978-3-030-86337-1_31.

[23] Knyazev, B. As part of the "Tutorial on Graph Neural Networks for Computer Vision and Beyond". *Spectral Graph Convolution Explained and Implemented Step By Step* [online], 16. august 2019. Available at: https://towardsdatascience.com/spectral-graph-convolution-explained-and-implemented-step-by-step-2e495b57f801.

[24] Kodym, O. and Hradiš, M. Page Layout Analysis System for Unconstrained Historic Documents. [online]. 2021, [cit. 2022-05-09]. DOI: 10.48550/ARXIV.2102.11838. Available at: https://arxiv.org/abs/2102.11838.

[25] Kohút, J. and Hradiš, M. *TS-Net: OCR Trained to Switch Between Text Transcription Styles*. arXiv, 2021. DOI: 10.48550/ARXIV.2103.05489. Available at: https://arxiv.org/abs/2103.05489.

[26] Lehenmeier, C., Burghardt, M. and Mischka, B. Layout Detection and Table Recognition – Recent Challenges in Digitizing Historical Documents and Handwritten Tabular Data. [online]. Cham: [b.n.]. 2020, p. 229–242, [cit. 2021-12-14]. DOI: 10.3934/mbe.2020182. Available at: https://rdcu.be/cDqJ7.

[27] Liang iaokang, Peng, J., Li, Z., Xie, D., Sun, W. et al. Robust table recognition for printed document images. *Mathematical Biosciences and Engineering* [online]. 2020, vol. 17, p. 3203, [cit. 2021-12-14]. DOI: 10.3934/mbe.2020182. ISSN 1551-0018. Available at: https://www.aimspress.com/article/id/5022.

[28] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y. et al. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. [online]. 2017, [cit. 2021-12-30]. Available at: https://arxiv.org/abs/2103.14030.

[29] Ma, Y. and Tang, J. *Deep Learning on Graphs*. Cambridge University Press, 2021. ISBN 9781108831741.

[30] Middleton, S. E. and Ziomek, J. GloSAT Historical Measurement Table Dataset. [online]. 1. september 2021, [cit. 2021-12-14]. DOI: 10.5281/zenodo.5363457. Available at: https://doi.org/10.5281/zenodo.5363457.

[31] Papineni, K., Roukos, S., Ward, T. and Zh, W.-J. BLEU: a Method for Automatic Evaluation of Machine Translation. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics* [online]. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, 2002, p. 311–318 [cit. 2021-12-14]. Available at: https://aclanthology.org/P02-1040.pdf.

[32] Pawlik, M. and Augsten, N. Tree Edit Distance. *Compare your trees with ease* [online]. 2016. Available at: http://tree-edit-distance.dbresearch.uni-salzburg.at/.

[33] Prasad, A., Déjean, H. and Meunier, J.-L. Versatile Layout Understanding via Conjugate Graph. In: *2019 International Conference on Document Analysis and Recognition (ICDAR)* [online]. 2019, p. 287–294 [cit. 2021-12-10]. DOI: 10.1109/ICDAR.2019.00054. Available at: https://ieeexplore.ieee.org/document/8978117.

[34] Prieto, J. R. and Vidal, E. Improved Graph Methods for Table Layout Understanding. In: Josep, L., Daniel, L. and Seiichi,,, U., ed. *Document Analysis and Recognition – ICDAR 2021* [online]. Cham: Springer International Publishing, 2021, p. 507–522 [cit. 2021-10-16]. ISBN 978-3-030-86331-9. Available at: https://rdcu.be/czBYq.

[35] Qasim, S. R., Kieseler, J., Iiyama, Y. and Pierini, M. Learning representations of irregular particle-detector geometry with distance-weighted graph networks. *The European Physical Journal C* [online]. Jul 2019, vol. 79, no. 7, [cit. 2021-12-22]. DOI: 10.1140/epjc/s10052-019-7113-9. ISSN 1434-6052. Available at: https://arxiv.org/abs/1902.07987.

[36] Qasim, S. R., Mahmood, H. and Shafait, F. Rethinking Table Recognition using Graph Neural Networks. [online]. 2019, [cit. 2021-12-09]. Available at: https://arxiv.org/pdf/1905.13391.pdf.

[37] Tomita, E., Tanaka, A. and Takahashi, H. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science* [online]. 2006, vol. 363, no. 1, p. 28–42, [cit. 2021-12-09]. DOI: https://doi.org/10.1016/j.tcs.2006.06.015. ISSN 0304-3975. Available at: https://www.sciencedirect.com/science/article/pii/S0304397506003586.

[38] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L. et al. Attention Is All You Need. [online]. 2017, [cit. 2021-12-30]. Available at: https://arxiv.org/abs/1706.03762.

[39] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P. et al. Graph Attention Networks. [online]. 2018, [cit. 2021-12-14]. Available at: https://arxiv.org/abs/1710.10903.

[40] Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M. et al. Dynamic Graph CNN for Learning on Point Clouds. [online]. 2019, [cit. 2021-12-22]. Available at: https://arxiv.org/abs/1801.07829.

[41] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L. et al. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining* [online]. New York, NY, USA: Association for Computing Machinery, 2018, p. 974–983 [cit. 2021-12-10]. KDD '18. DOI: 10.1145/3219819.3219890. ISBN 9781450355520. Available at: https://doi.org/10.1145/3219819.3219890.

[42] Zheng, J., Ramasinghe, S. and Lucey, S. Rethinking Positional Encoding. [online]. 2021, [cit. 2021-12-30]. Available at: https://arxiv.org/abs/2107.02561.

[43] Zhong, X., ShafieiBavani, E. and Yepes, A. J. Image-based table recognition: data, model, and evaluation. [online]. 2020, [cit. 2021-12-09]. Available at: https://arxiv.org/pdf/1911.10683.pdf.

[44] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C. et al. Graph Neural Networks: A Review of Methods and Applications. [online]. 2021, [cit. 2021-12-12]. Available at: https://arxiv.org/abs/1812.08434.