POLITECNICO DI TORINO
SPRINT REPLY S.R.L.

Master's Degree in DATA SCIENCE AND ENGINEERING

Master's Degree Thesis

Analysis of semi-structured data based on Named Entity Recognition and Computer Vision techniques

Supervisors

Prof. Luca CAGLIERO

Dott. Matteo SARTORI

Candidate

Federico Lorenzo PES

July 2023

Summary

Extracting data from invoices is crucial for many reasons. First of all, accurate financial records provide information about expenses, vendor payments, and the overall financial health of the organization. Furthermore, verifying invoice details such as the vendor name, invoice number, and payment amounts ensures timely and accurate payments, avoiding late fees. Moreover, invoice data extraction plays a vital role in ensuring compliance with financial regulations and internal auditing requirements. These were only three of the reasons why it is important to extract data from invoices in an efficient and accurate way.

The task of extracting information from invoices is highly recurrent, for this reason, it is optimal to be automated. The main challenge with this task is the fact that for each issuer the text layout of the invoice may vary. We refer to this type of data as semi-structured, which does not follow a tabular schema but does exhibit a certain level of organization. Hence, while rule-based techniques may provide excellent results for a certain layout, due to the variability of the layouts, they need to be manually adapted to a specific case.

Recently Graph Neural Networks, thanks to their flexibility, have been applied to different fields of research including NLP. The base idea of this type of Neural Network is to build a graph from the dataset, defining nodes and edges. GNNs can exploit many different types of features to create a graph. For example, nodes can represent words, sentences or whole documents. While edges can be represented by any type of relationship, for example dependency trees, references or to encode proximity.

Furthermore, different tasks can be applied to these networks like node classification, edge prediction or graph classification. Subsequently to the creation of the dataset, these models rely on a pipeline that can be divided into message passing, aggregation and update. For which each node can share information with the linked nodes. This information passing is used to

update the embedding for each node exploiting the messages from all the neighbours. Each layer of the model represents a message passing level, if we have only a layer for the model, each node will be updated with information obtained only from the direct neighbours, while more layers can help to capture details from further nodes.

This concept allows us to think at each document as a graph, where every word represents a node, and the edges can be formed with the closest nodes in the document.

The focus of this project is to develop a pipeline to extract entities from semi-structured data. The solution involves both Computer Vision and NLP which are linked by Graph Neural Networks.

The dataset is composed of 1400 invoice images, from the public dataset rvl_cdip from the HuggingFace library. Subsequently, these documents were labelled manually using LabelImg, a tool for image labelling for object detection tasks, and DocTR, a tool for optical character recognition.

The labels taken into account for the analysis are:

- Company_name: a string that can be extracted from the logo or full corporate name.
- Company_address: a string containing the full address of the company releasing the invoice.
- Invoice_date: a string containing the date on which the invoice was released. The format of the date can change.
- Invoice_number: an alphanumerical string that identifies the invoice for the company.
- total_due: a string containing the total that needs to pay, it could also include the currency used.

The pipeline can be divided into the following steps:

Text and bounding boxes extraction with the OCR. Each document is passed through the DocTR library to extract the text along with the coordinates that refer to the bounding box for each word. Each document was previously labelled using LabelImg, a graphical image annotation tool. Subsequently, the labels and the bounding boxes are aligned to create the basis for our dataset.

Graph creation exploiting the bounding boxes. The next step is to model each document as a graph. The bounding boxes provide information about the position of each word in the document. Hence, it is to create a graph with "proximity" as the main criterion to choose the edges for each node. In the base implementation, each node is linked to the five closest words.

Extract for each token the embedding from the full text. Since each node in a graph needs a representation, word embeddings are exploited as a basis to create this node representation. Using pretrained BERT as a basis we reconstruct the "full text" to extract the token embedding. Also, some words may be represented by more than one token, so they aggregated in order to have only a single representation for a node

Solve Node Classification task by exploiting GNNs. Lastly, we train a GNN architecture to solve a Node Classification task. The base architecture will be a Graph Convolutional network composed of 2 layers of convolution stacked on top of each other.

There are many aspects to consider while working on this task. For this reason, on top of this pipeline, I propose different experiments and ablation studies, that can be divided into 3 sections.

Graph creation. Firstly, we must consider how to create the graph, the main idea is to link a node only to the nearest node in each direction (if any) with unweighted edges. It is interesting to consider different solutions. We will consider changing the number of edges for each node, considering the K nearest neighbours for each node. Furthermore, providing weights to the edges based on distance or similarity between words could prove meaningful to improve the performance of the task.

Embedding creation. Regarding the embedding creation, the proposal is to exploit BERT main encoder to provide the embeddings for each node. It could also be interesting to see how enriching this representation with other types of embeddings/ features impacts the performance of the model. For example, it is possible to add morphological embedding (CharBERT) Furthermore, encode the coordinates of the bounding box or those of the centre of a word, with respect to the page.

Choice of the Graph Neural network. Lastly, about the choice of the graph neural network, the experiments can be about different types of main architecture and the number of layers, and training. There are different types of GNNs, like Graph Convolutional Networks or Graph Attention Networks. The number of layers modifies how the final representation of a node is impacted by the other nodes in the graph. There are also self-supervised tasks for GNNs that we could study to see how they impact the performance of the whole task.

The final results are obtained by exploiting BERT to obtain the word-level embeddings, concatenated to spatial features regarding the position in the image. Subsequently, the graph is created by linking each node to the closest five neighbours. Furthermore, the distance is used to obtain a weighting score for the edges. Finally, the node classification task is performed by a Graph Neural Network is composed of two layers of GATv2Conv stacked on top of each other.

This pipeline obtained an average f1-score equal to 0.81. This score is calculated by applying the mean over the f1-scores obtained for the key elements.

Table of Contents

List of Tables IX

List of Figures XI

1 Introduction 1
- 1.1 Problem Statement . . . 1
- 1.2 Solution . . . 3
- 1.3 Applications . . . 4
- 1.4 Contributions . . . 5

2 Related Works 7
- 2.1 Rule-based . . . 8
- 2.2 Token Classification . . . 8
- 2.3 Grid Based . . . 9
- 2.4 Graph Based . . . 9
- 2.5 Baseline . . . 11

3 Theory fundamentals 12
- 3.1 Optical character recognition . . . 12
  - 3.1.1 docTR . . . 13
- 3.2 Word-level Embeddings . . . 14
  - 3.2.1 Fast Text . . . 15
  - 3.2.2 Contextual embeddings . . . 16
  - 3.2.3 Attention mechanism . . . 17
  - 3.2.4 Transformers . . . 17
  - 3.2.5 BERT . . . 19
  - 3.2.6 CharBert . . . 21
- 3.3 Graphs . . . 24

3.4 Graph Neural Networks 26
3.4.1 Graph Convolutional Networks 27
3.4.2 Graph Attention Networks 28
3.4.3 GraphSAGE 30

4 Pipeline 32
4.1 General Pipeline 32
4.2 Dataset 33
4.2.1 DocTR 36
4.3 Node embedding 38
4.3.1 Spatial features 39
4.3.2 Word Embeddings 39
4.4 Graph Creation 41
4.4.1 Data object 42
4.5 Model 43
4.5.1 Architectures 43
4.5.2 Training Settings 45
4.5.3 Metrics 46

5 Results 48
5.1 Analysis of the results 48
5.1.1 Expectations 48
5.1.2 Results 49
5.2 Ablation study 51
5.2.1 Embedding model 51
5.2.2 Edge weigths 53
5.2.3 Number of Message Passing Layer 54
5.2.4 Number of edges 55

6 Conclusions 57
6.1 Conclusion 57
6.2 Future Works 58

Bibliography 59

List of Tables

3.1 Summary of the parameters for models BERT Base and BERT Large. 19

4.1 Statistic on how the dataset is split following a 80-10-10 split 38
4.2 Number of lines, words and how they are divided into each class. This statistic is presented for train, test and validation splits 38
4.3 Summary of the parameters of the models. 44
4.4 Summary of the parameters used for the training step. . . . 45
4.5 Summary of the weights used to calculate the Cross-Entropy Loss. 46

5.1 Results obtained with the GCNConv-based architecture. . . 49
5.2 Results obtained with the GATv2Conv-based architecture. . 50
5.3 Results obtained with the SAGEConv-based architecture and mean_pooling aggregation method. 50
5.4 Results obtained with the SAGEConv-based architecture and max_pooling aggregation method. 51
5.5 Results obtained with the GATv2Conv-based architecture and BERT to provide the textual embeddings. 52
5.6 Results obtained with the GATv2Conv-based architecture and FastText to provide the textual embeddings. 52
5.7 Results obtained with the GATv2Conv-based architecture and CharBERT to provide the textual embeddings. 53
5.8 Results obtained with the GATv2Conv-based architecture and using edge weights. 54
5.9 Results obtained with the GATv2Conv-based architecture and not using edge weights. 54
5.10 Results obtained with the 3-layer GATv2Conv architecture. . 55

5.11 Results obtained with the GATv2Conv-based architecture and creating the graph with 5 neighbours for each node. 56

List of Figures

1.1 Example of invoice taken from the RVL-CDIP dataset. . . . 6

2.1 Image from [8] by *Lohani et al.*. 10

3.1 Example of distributional hypothesis. 14
3.2 How CBOW and Skip-gram architecture work. 15
3.3 Example of the breakdown of the word "Invoice" into a bag of n-grams. 16
3.4 Example of homonyms. 17
3.5 Architecture of the Transformer model [12]. 18
3.6 Example of Masked Language Model MLM. 21
3.7 Example of Next Sentence Prediction. 21
3.8 Architecture of the CharBert model [13]. 22
3.9 Graphical representation of the pre-training task *Noisy Language Modeling* [13]. 24
3.10 Example of the protein interaction network. 25
3.11 Examples of sampling and aggregation step from “Inductive Representation Learning on Large Graphs” [20]. 30

4.1 Graphical representation of the pipeline divided into steps. . 32
4.2 Example of Object Annotations for an item labelled as *total_due* 36
4.3 The general schema of the Message Passing Layer. 44
4.4 How the output of the GATv2Conv Layer is handled. 45

Chapter 1

Introduction

1.1 Problem Statement

Daily, we encounter semi-structured documents. Semi-structured data is a type of data that does not follow the traditional schema of relational databases, and even though they do not adhere strictly to a predefined structure, they are provided with headers or metadata to provide a certain level of structure. The most common are invoices.

These documents are related to the financial aspect of our life and need particular attention. Extracting data from invoices is crucial for many reasons. First of all, accurate financial records provide information about expenses, vendor payments, and the overall financial health of the organization. Furthermore, verifying invoice details such as the vendor name, invoice number, and payment amounts ensures timely and accurate payments, avoiding late fees. Moreover, invoice data extraction plays a vital role in ensuring compliance with financial regulations and internal auditing requirements.

These were only three of the reasons why it is important to extract data from invoices. Many companies solved this task manually, but nowadays this process of elaborating on them is being automated more and more. The main reasons why automating this task is important are:

- Time and cost savings: by reducing manual effort, which is time-consuming and prone to errors, companies can save significant time and associated labour costs.
- Scalability: as businesses grow, the volume of invoices they handle increases. Scalability enables businesses to handle larger volumes of

invoices without the need to hire additional resources.

- Data insights and decision making: automated systems can aggregate and analyze this data, enabling businesses to make informed decisions, identify cost-saving opportunities, and optimize their financial strategies.

The main task to solve is Information Extraction (IE). The high quantity of information is encoded in different ways and structured with different layouts depending on the company that produces the invoice. This information can be located through context and position on the page.
The IE step for this purpose is defined as a Named Entity Task (NER). In Natural Language Processing (NLP) the latter is defined as the task of locating and classifying the tokens that represent a Named Entity in the text.

When working with a dataset of invoices with a homogeneous structure, the task is usually solved by applying rule-based techniques built *ad hoc* for a specific layout. They usually rely on regular expression or approximate string research. Nonetheless, they have negative aspects:

- Lack of flexibility: rule-based techniques exploit a precise structure to extract data. This makes them less robust in handling diverse invoice structures and invoices which present noise.
- Difficulty in handling unstructured data: rule-based techniques are efficient when facing structured data, with a key-value structure. Invoices are usually semi-structured data and do not always provide this type of structure.
- Maintenance and scalability: the lack of flexibility also reflects on this aspect. Invoice layouts evolve and change over time. For this reason, companies need to face the cost of maintaining these rules to adapt them to new layouts.
- Lack of adaptability to new data sources: when working with invoices from new vendors or industries, these techniques may need customization because they are difficult to re-adapt.

In case of high heterogeneity between a collection of documents, it is possible to face all these problems. Hence, not only robust and flexible techniques are needed, but also a way to model this type of document. For

this purpose, the pipeline proposed will exploit deep learning solutions such as word embeddings and transformer-based models, which are known to be more flexible, and may provide more accurate results.

The semi-structured nature of the invoices may be a problem for these types of models, since the context may not be enough to extract information reliably, but also the relative position of the word on a page may be important. In order to handle non-sequential data, it can be modelled as a graph. A graph is a structure that is composed of a set of vertices (also known as nodes) connected by edges.

This type of data can be elaborated by **Graph Neural Networks**. Thanks to their flexibility, and the ability to capture relationships, dependencies and patterns in the graph, these models are recognized as state-of-the-art for Information Extraction over semi-structured data.

Graph Neural Networks enable solving a series of tasks, the main ones are:

- Graph Classification;
- Link Prediction;
- Node Classification.

In this context, where a node represents a textbox, the node classification task can be thought of as a token classification task. For this reason, the NLP pipeline will be composed of two main steps:

- Text encoding: exploiting Word Embeddings.
- Node classification: which will be solved by Graph Neural Networks.

Moreover, there is the possibility that not all the documents taken into account are natively digital. A large quantity of data is scanned, which can bring problems due to corruption or the quality of the image. To tackle this problem, the first step is to extract the text from these images in a reliable way. The pipeline proposed will exploit Optical Character Recognition (OCR) to convert scanned documents into machine-readable text.

1.2 Solution

As anticipated, the solution for this problem is an end-to-end pipeline for Information Extraction from scanned invoices. First of all, the document is

scanned through an OCR tool, the one selected for this pipeline is **DocTr** which is a system that utilizes advanced machine learning techniques, including deep neural networks, to accurately recognize and extract text from various types of documents.

Afterwards, each text box goes through an embedding step, according to the type of embedding model a pre-processing step is deployed.

Subsequently, the document is modelled as a graph where each box of text represents a node. The criterion selected to create edges is proximity. In fact, each node forms an edge with the closest ones, also self-loops are included.

Lastly, the whole document is a data entry that is passed to a Graph Neural Network, which is trained to solve the token classification class over an invoice.

1.3 Applications

The result of this study is an end-to-end pipeline for Information Extraction over scanned invoices. As previously mentioned, this solution finds use in various applications:

- Streamlined Invoice Processing: the pipeline makes it possible to extract invoice data quickly and automatically, which eliminates the need for human data entry and shortens the processing time for invoices. This simplification facilitates quicker invoice approval and payment procedures while increasing productivity and reducing errors.

- Financial Management and Reporting: data from extracted invoices can be utilized for tracking expenses, examining spending trends, and producing financial reports as part of financial management. The pipeline makes it easier to collect accurate and timely data, improving financial visibility and decision-making.

- Compliance and Auditing: the process helps to ensure compliance with regulatory standards and streamlines auditing procedures by automatically collecting data from invoices. Extracted data can be used for internal or external audits, account reconciliation, and accuracy checks.

- Data Consolidation and Insights: by integrating the retrieved invoice data with other business systems and databases, it is possible to enable

data consolidation and produce insightful data.

- Fraud Detection and Risk Management: by collecting invoice data for anomaly detection, spotting odd trends, or reporting suspect transactions, the pipeline can help with fraud detection efforts. This improves risk management and protects from fraud.
- Workflow Automation and Efficiency: the automated extraction technique used by the pipeline minimizes human labour requirements, enables more efficient processes, and boosts overall operational effectiveness. It gets rid of tedious duties and frees up resources for more worthwhile endeavours.

1.4 Contributions

The main contributions brought by this work are:

- The development of a pipeline to extract entities from heterogeneous semi-structured data is the main goal of this research. The method uses Graph Neural Networks to connect Computer Vision with Natural Language Processing. First, DocTR receives the scanned papers to extract the words and bounding boxes. Subsequently, the document is modelled as a graph using the bounding boxes and relative distance among tokens. Then, word-level embeddings are utilized to get each word's representation. The Graph Neural Network is then given the node embeddings and the graph structure to categorize each token.
- A well-rounded analysis of the different methods for the three main steps: word-level embedding, graph creation and node classification.
- A public dataset [1] composed of 1400 labelled invoices with both text and computer vision features. The dataset provides for each document the following entities: *Company Name*, *Company Address*, *Invoice Date*, *Invoice Number* and *Total Due* as shown in Figure 1.1.

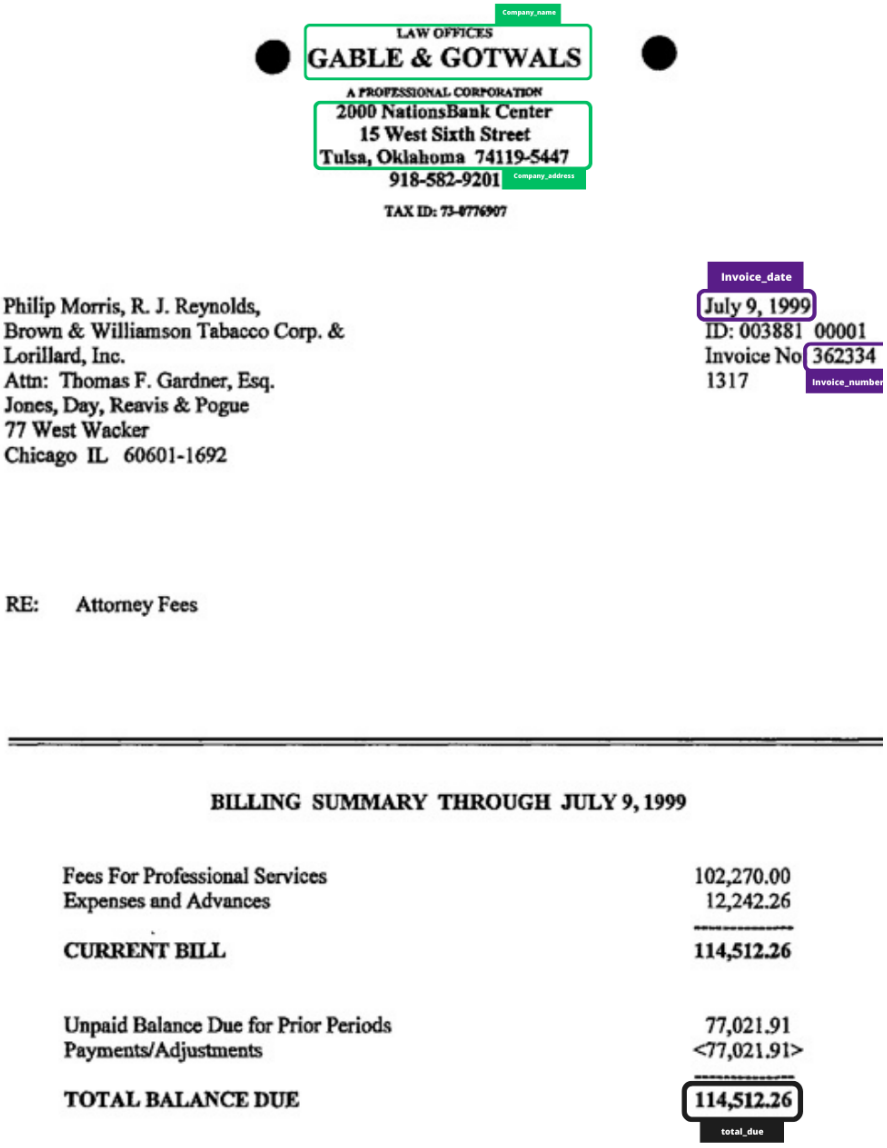

Figure 1.1: Example of invoice taken from the RVL-CDIP dataset. The invoice also presents the labeling for the classes *company_name*, *company_address*, *invoice_number*, *invoice_address*, and *total_due*.

Chapter 2

Related Works

The extraction of structured information from unstructured or semi-structured data sources is a crucial problem in natural language processing and data analysis. Throughout the years, several ideas and methodologies have been offered to address this difficulty.

The goal of this chapter is to provide a thorough exploration of each topic, highlighting central ideas, approaches, and achievements in the subject. By examining these works, the objective is to obtain a better understanding of the strengths, limits, and application of each strategy in multiple contexts and domains.

This type of analysis will take into account the following technologies:

- Rule-based: the main advantages are interpretability, simplicity, and effectiveness. These models thrive with pre-defined and homogeneous layouts. The disadvantages are related to their limits of generalization on unknown patterns, requiring constant maintenance.
- Token classification techniques: these models are able to correctly extract the dependencies of the words by exploiting their context. On the other hand, these models are not able to elaborate multi-dimensional data, requiring them to be sequential.
- Grid-based: these models are capable to elaborate 2-dimensional documents and perform best with structured or grid-like documents. On the contrary, they are not able to generalise well documents with an unstructured layout.
- Graph-based: the main advantages are that they are able to handle

graph-structured data, and they can incorporate both local and global information making them able to solve various tasks (e.g. node classification and graph classification). On the other hand, they are dependent on the graph structure.

2.1 Rule-based

Rule-based and label-based methods, which rely on predefined rules and layouts [2] to extract information, were among the first techniques for Information Extraction. These approaches often leverage handcrafted rules or regular expressions to identify and capture specific data elements.

The main advantages of rule-based methods include their interpretability, simplicity, and effectiveness in scenarios with well-defined extraction patterns.

However, the challenges of maintaining and updating rules as the data and extraction requirements are one of the main concerns of this technique. Furthermore, these solutions are not capable of generalizing on unknown layouts or patterns, requiring human intervention.

Machine learning and deep learning techniques try to solve these limitations.

2.2 Token Classification

For Token Classification tasks, sequence models have been widely applied in many domains. Models like Bi-LSTM CRF [3] and Trasnformer-based models [4] have been largely recognized with one-dimensional data as state-of-the-art for NER tasks.

The capacity of the BiLSTM-CRF architecture to capture both local and global contextual information is what makes it so significant. While the CRF component models global dependencies by considering label transitions, the BiLSTM component models local context by considering nearby tokens. Combining these factors enables the model to take into account a wider context and produce predictions that are more accurate and consistent. The Transformer-based models utilize self-attention mechanisms to capture contextual information from both preceding and subsequent tokens, enabling effective token-level predictions.

But, when working with semi-structured data and invoices in particular, reducing the data to only one dimension without considering the layout of

the document could be a limitation to the solution.

For this reason, many solutions try to solve this problem by taking into account the structure of the document, pointing toward a two-dimensional analysis of the data.

2.3 Grid Based

Grid-based solutions are frequently employed for information extraction from semi-structured data, especially when dealing with texts or data sources having tabular or grid-like architecture. These strategies seek to extract structured information by utilizing the grid format's layout and structuring of data. These techniques apply a grid over the document, diving it into regions. Subsequently, these regions are used to embed the spatial features that would be used along with the features extracted from the text.

CharGrid [5] and *BERTGrid* [6] are approaches that rely on this concept. They both text and spatial features to apply NLP downstream tasks over semi-structured documents. The textual features can be more character-oriented [5] or exploring contextual embeddings [6] by transforming the regions into a 1-dimensional representation to obtain BERT embeddings. These features along with the spatial features extracted from the regions, are used to solve downstream tasks. The Information Extraction can be modelled as semantic segmentation or bounding box regression.

The result obtained by [6] over the token classification task is a mean accuracy, over six different classes, of 65.48%.

2.4 Graph Based

Graph-based solutions thrive when dealing with data that does not adhere to a traditional grid layout, such as unstructured text or related entities. Graphs are more adaptable in depicting complicated interactions and dependencies among data elements, allowing for more precise and thorough analysis.

These techniques also exploit a three-step stage approach. The first step is to extract textual features to create a node embedding. Secondly, the graph creation step is applied, which usually exploits spatial features to create edges between nodes. Finally, an architecture is structured to solve Information Extraction over the semi-structured data.

Graph-based techniques generally follow this schema and focus on customizing the single steps.

Each node could be represented by text segments [7] encoded by sequential models like a Bi-LSTM module, or they can be represented by single words [8] which embeddings are provided by *BPEmb* [9], which is a collection of pre-trained BPE [10] arrays. The node embedding could be represented only using textual features [7] or concatenated to spatial features [8] or other attributes that could help describe the node, for example, boolean features.

The graph could represent edges using a set of features [7] (relative distances, ratios of the sizes of the bounding boxes) or the graph could be unweighted [8]. The edges could be selected by simply exploiting the Euclidean distance [7] or with a specific algorithm [8]. For example, for each node calculates the relative distances on the horizontal and vertical plane. Subsequently, the nodes with the lowest relative distance are chosen to form an edge. In this way, each node has a maximum of 4 edges, each one representing one of the four main directions as shown in Figure 2.1.

Finally, the architecture could be composed of Graph Convolutional layers stacked on top of each other [8] or could include a more structured architecture by involving other components [7]. *Liu et al.* in provide an architecture that comprises graph convolutions to extract node embeddings, which are passed to the architecture's final module, the Bi-LSTM CRF model.

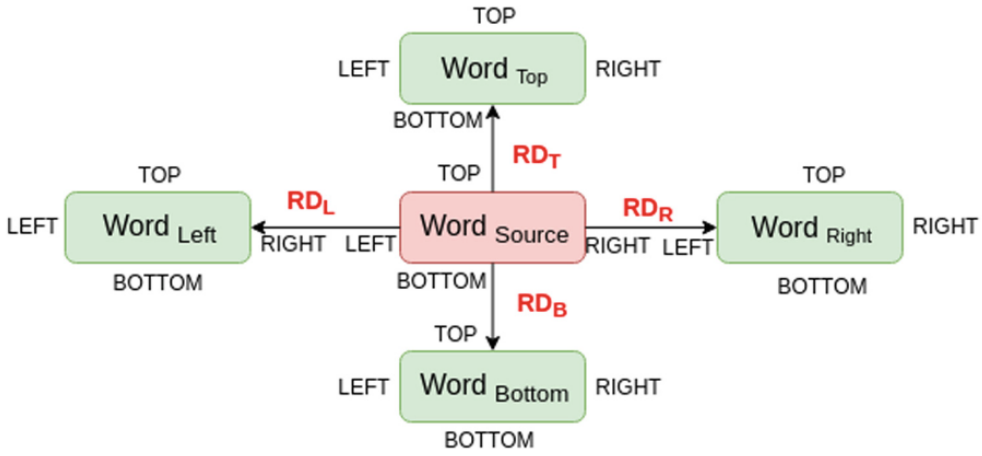

Figure 2.1: Image from [8] by *Lohani et al.*, which represents the formation of edges for the $Word_{source}$ with four nearest neighbours in the main directions

2.5 Baseline

The graph-based solutions, thanks to their flexibility and general ability to model variable structures, present promising results on Information Extraction over semi-structured data.

But, even if the aforementioned works try to tackle a similar task to the one presented in this thesis, it is not possible to provide a clear comparison. In fact, the data used for this case were private [8] or provided a small number of layouts, having homogeneous datasets [7]. The purpose of this research is to analyze the performances of an end-to-end pipeline over a heterogeneous dataset.

The study of this thesis is made over a dataset handcrafted for this task. The main reason for this choice is the unavailability of public datasets that not only provide the text extracted by an invoice but also provide the bounding box of each textbox and also the corresponding label. Hence, it would not be possible to apply a clear baseline for this work.

Chapter 3

Theory fundamentals

The purpose of this chapter is to provide the reader with the theory to understand each element of the pipeline and the choice made for each step.

The chapter will go through these three main topics:

- Optical Character Recognition;
- Word Level Embeddings;
- Graph Neural Networks.

3.1 Optical character recognition

Optical character recognition is known as OCR. This technology makes it possible to transform printed or scanned text into machine-readable text. OCR is a technology that enables computers to read text from documents, photographs, and other sources that contain text and recognize it.

When it comes to digitizing and extracting data from physical documents, such as scanned paper records, invoices, receipts, or even handwritten text, OCR technology is essential. OCR makes it possible to automate data processing, text analysis, and information retrieval by identifying and converting the text in these documents into a digital format. Nowadays the most performing OCR solutions are based on machine learning techniques.

Machine Learning-based OCR: to learn the patterns and properties of characters and text, these systems make use of algorithms and models that have been trained on enormous datasets. In this method, a model is trained using a labelled dataset of images and the associated ground truth text.

Based on the visual patterns it detects during training, the model gains the ability to identify and categorize characters. OCR which is based on machine learning has the potential to be more adaptable and versatile to various fonts, languages, and document layouts. Modern machine learning-based OCR systems frequently employ deep learning techniques like convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

As previously mentioned, there are usually two main steps that OCR solutions follow to solve their task:

- Text Detection: OCR software recognizes and pinpoints text-containing regions in images. To isolate the text for recognition, it divides the material into discrete character or word sections.
- Text Recognition: it is a procedure where the machine analyzes the text sections that have been divided up into an image to locate and recognize the individual characters inside those regions.

3.1.1 docTR

The OCR selected for this pipeline is **docTR**. It is an open-source OCR by Mindee. It provides state-of-the-art performance on public document datasets. Like most of the OCR tools, it provides a 2-stage classification. Furthermore, it provides a set of pre-trained models for both text detection and text recognition steps. In this work, the chosen models are:

- **db_resnet50**: it involves a DBNet as described in "Real-time Scene Text Detection with Differentiable Binarization", using a ResNet-50 backbone. It works by integrating Fully connected Convolutional Networks, in this case, ResNet-50, local feature extraction, and differentiable binarization.
- **crnn_vgg16_bn**: CRNN with a VGG-16 backbone as described in "An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition".

The OCR predictor extracts text for each page. Furthermore, it is able to hierarchically dive the page into blocks, lines, and single words. This type of configuration allows customization during the word embedding step, especially for the contextual embedding models. Moreover, the output provides many details for each word, like the bounding box and the confidence

of classification. The bounding box follows the same format for PASCAL VOC, so the coordinates as x_min, x_max, y_min, and y_max.

3.2 Word-level Embeddings

The development made in the field of NLP relies on the evolution of the tools to represent the text as vectors. Dense vectors are acknowledged as the main fashion to represent text. The reason for their popularity is the ability to encode both semantic and syntactic similarity between words, by mapping the words into a latent space.

The key idea behind these embedding models is the Distributional Hypothesis, which fundamentally states that words that occur in the same contexts tend to have similar meanings. An example can be found in Figure 3.1.

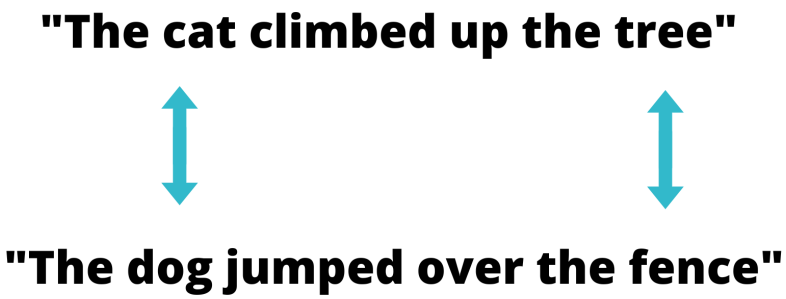

Figure 3.1: Example of distributional hypothesis.

These dense vectors are usually high-dimensional vector, in which a single dimension does not provide a specific concept or do not correspond to a specific textual unit. This makes these vectors less interpretable but at the same time, the features are generated automatically.

The most common way to train these models is firstly to collect a rich collection of documents in order to create a vocabulary to provide an encoding for each word. Then, a sliding window is exploited to build training examples. The sliding window passes along a large collection of documents to find which word composes the context for each word.

Subsequently, the models are trained to solve tasks to learn context, the main frameworks are:

- Skip-gram: predicts the surrounding words given the target word.
- Continuous bag of words: predicts the target word given the surrounding words.

Both architectures are shown in Figure 3.2.

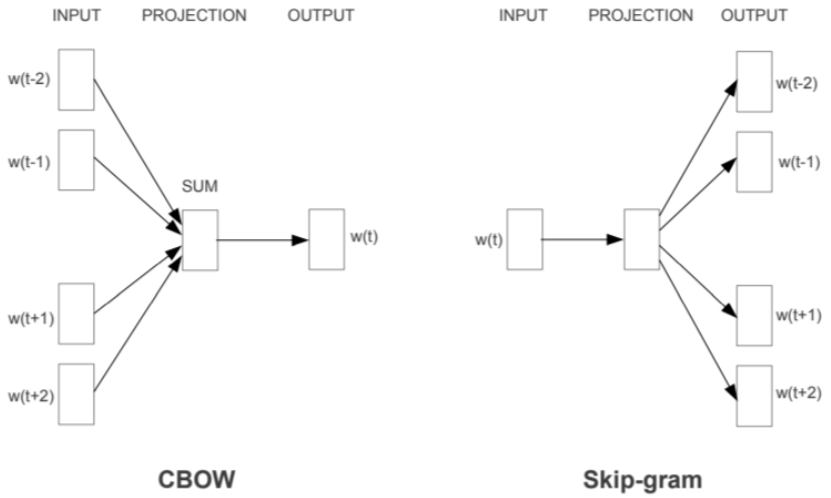

Figure 3.2: The image shows how the CBOW and Skip-gram architecture work.

3.2.1 Fast Text

One problem with the base approach for word embeddings is the possibility of encountering out-of-vocabulary words, which is likely to happen with morphologically rich languages. Furthermore, the morphology of a word can help to provide more meaningful representations. Hence, the Fast Text *Bojanowski et al.* [11] model introduces a new approach that represents each word as a bag of character n-grams.

This solution is based on the skip-gram model, in which each word corresponds to a bag of character n-gram as shown in Figure 3.3. Furthermore, special symbols $<$ and $>$ are added to distinguish between prefixes and suffixes. Also, the word itself is included in its set of n-grams.

Given a dictionary of n-grams of size **G**, The scoring function for the

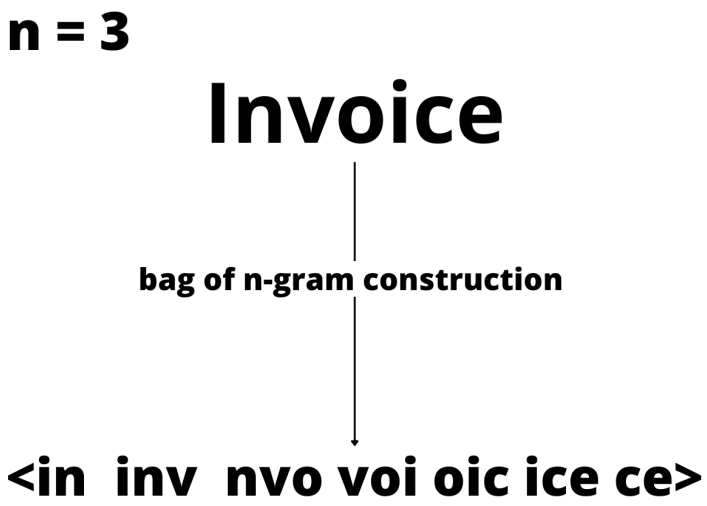

Figure 3.3: Example of the breakdown of the word "Invoice" into a bag of n-grams.

skip-gram model is modified as:

$$s(w, c) = \sum_{g \in G_w} z_g^T v_c \tag{3.1}$$

where s is the scoring function, z_g is the vector representation for the n-gram g and v_c is the vector representation for the context word c.

In the end, each word passed to the model is encoded into a 300-dimensional continuous feature vector.

3.2.2 Contextual embeddings

A limitation of word embeddings is that they are not able to exploit the context in which the word is used. In fact, the same word in different contexts can have different meanings, as shown in Figure 3.4. To overcome this limit, contextual embeddings are introduced in the landscape of NLP. Contextual embeddings can be obtained from a deep learning model, which is trained on a large text corpus, usually as a bidirectional language model. The word vectors are represented by the hidden state of such model. By doing so we are able to extract the representation for a word in the context in which is used.

These models are usually structured as an encoder-decoder mechanism:

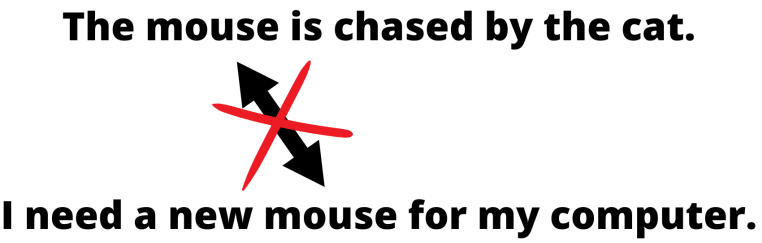

Figure 3.4: Example of homonyms. Here the meaning of the word *mouse* depends on its context.

- Encoder: reads a variable-length sequence and maps it into a fixed-size vector.
- Decoder: takes the fixed-size vector and generates a variable-length sequence.

3.2.3 Attention mechanism

This mechanism is the fundamental step for transformer-based models. The attention function receives the vectors query Q, key K, and value V to provide an output. The output is a weighted sum of the values, where the weight is a function of the query and the key. This function can be thought of as a similarity function. The application taken into account in the paper [12] uses a Scaled Dot-Product.

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3.2}$$

where the normalization factord_k is the dimension of the vector K.
In the paper [12] the application also involves multiple attention functions by mapping the vectors Q, K, and V into n = number of heads times, with learned linear projections. The resulting outputs are then concatenated and projected again into the final values.

3.2.4 Transformers

The base model that benefits from this structure and overcomes the limit of sequential computation, is the Transformer by *Vaswani et al.* [12]. In fact,

this model does not exploit recurrence to analyze sequences but it is based on the Attention mechanism, which allows more parallelization.

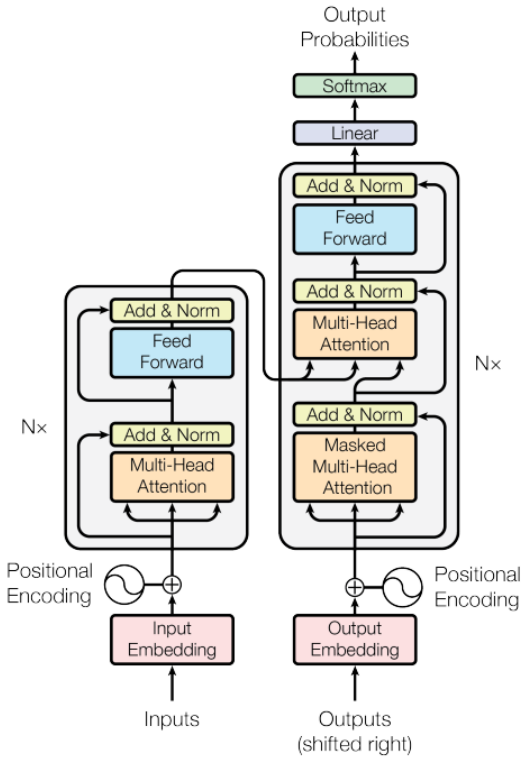

Figure 3.5: Architecture of the Transformer model [12], composed of their characteristic Encoder and Decoder modules, which both implement Multi-Head Attention.

It is an auto-regressive model which at each step takes its output and adds it to the previously generated tokens. The architecture, which is shown in Figure 3.5 is composed of:

- Encoder: it is composed of N=6 identical layers, each one divided into a Multi-head attention Layer and a feed-forward Neural Network. Residual connections are employed in both the sub-layer of attention.
- Decoder: it is composed of N=6 identical layers, each one divided into a first Multi-head attention Layer, a second Multi-head attention layer that elaborates on the output of the encoder stack, and a feed-forward Neural Network. Residual connections are employed in both the sub-layer of attention. A mask mechanism is exploited to not take into

account the tokens that have not been yet generated from the model.

In order to inject information about the position of each token, a positional encoding has been introduced. This encoding is a finite representation of the location of a unit in the sequence. These representations have the same dimensions d of the input embedding, in order to be added. This is done by means of sine and cosine functions:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{3.3}$$

$$PE_{(pos,2i+i)} = cos(pos/10000^{2i/d_{model}}) \tag{3.4}$$

where pos is the position of the unit in the sequence, and i is the index in the array.

The advantage of using these functions is that it should be easier for the model to learn the relative position. In fact, PE_{pos+1} can be thought as a linear transformation of PE_{pos}.

3.2.5 BERT

BERT, which stands for "Bidirectional Encoder Representations from Transformers", is a transformer-based language model introduced by *Devlin et al.* in [4]. The base architecture of this model is composed of 12 layers of transformers stacked on top of each other, with a hidden dimension of 768 each, and with a number of self-attention heads each equal to 12, for a total of 100 million parameters, as shown in Table 3.1.

	BERT Base	BERT Large
Layers	12	24
Hidden dimension	768	1024
Attention heads	12	16
Number of parameters	110M	340M

Table 3.1: Summary of the parameters for models BERT Base and BERT Large.

This model's purpose is mainly to provide deep bidirectional representation conditioned by the both right and left context at all layers. This is done

by pre-training the model on a large unlabeled text corpus to solve self-supervised tasks.

The model has high flexibility, in fact by finetuning the model it is possible to reach high performances on various tasks. This is due to its capability of creating representations that embed highly semantic features.

To handle the huge corpus this model employs the encoding technique known as Byte-Pair Encoding (BPE) by *R. Sennrich et al.* [10] or one of its variants. BPE is a data compression technique used for sub-word tokenization. It is frequently used in natural language processing activities like language modelling and machine translation. In accordance with the statistical characteristics of the input text, BPE divides words into sub-word units.

The model accepts a sequence of tokens as input, this sequence can be composed of one or two sentences. Each unit in the sequences is transformed using a 30000 tokens vocabulary. BERT is also able to handle out-of-vocabulary tokens by using sub-tokens. The first token in the sequence is always the special token [CLS] which in the end would provide a representation for the whole sequence, it is mainly used for classification tasks on the whole sentence.

If the sequence is composed of two different sentences, a special token [SEP] is used to separate the two sentences. Furthermore, a segment embedding is added to the input embedding to differentiate even more the two sentences.

As previously seen in the transformers model, also BERT exploits the same Positional Encoding which is added to the input embedding.

The pre-training of BERT is done simultaneously on two tasks:

- Masked Language Model MLM: during the training, 15% of the tokens are chosen, then 80% of the chosen tokens are replaced with the [MASK] token, 10% of the chosen tokens are replaced with a random token, and the remaining 10% are left without changes. In the end, the model will predict the masked tokens. Example in Figure 3.6.

- Next Sentence Prediction NSP: the purpose of this task is to train the model to understand the relationship among sentences. In order to do so, when building the training set 50% of the pair of sentences are chosen to be consecutive, while the other 50% is chosen randomly. Example in Figure 3.7.

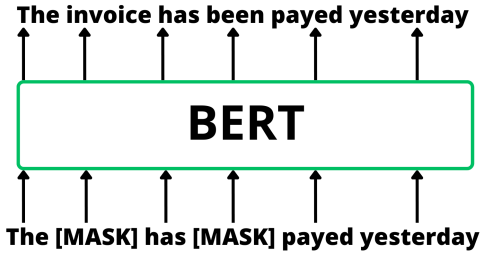

Figure 3.6: Example of Masked Language Model MLM.

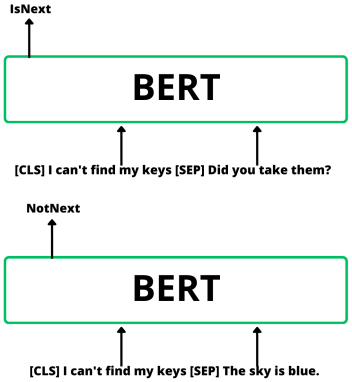

Figure 3.7: Example of Next Sentence Prediction.

3.2.6 CharBert

When data taken into account for the analysis are also numbers like prices or dates, the morphological structure of each word can be beneficial for the purpose. For this reason, the usage of character embedding along with contextual embeddings could enrich word representations.

CharBert by *Wentao Ma et al.* in [13] is a character-aware version of BERT. To improve word representations and the model's performance on tasks involving uncommon words, spelling variations, and morphologically complex languages, it integrates character-level information. A scheme of the architecture is presented in Figure 3.8.

In fact, while most pre-trained language models exploit PBE [10], it still presents two main problems:

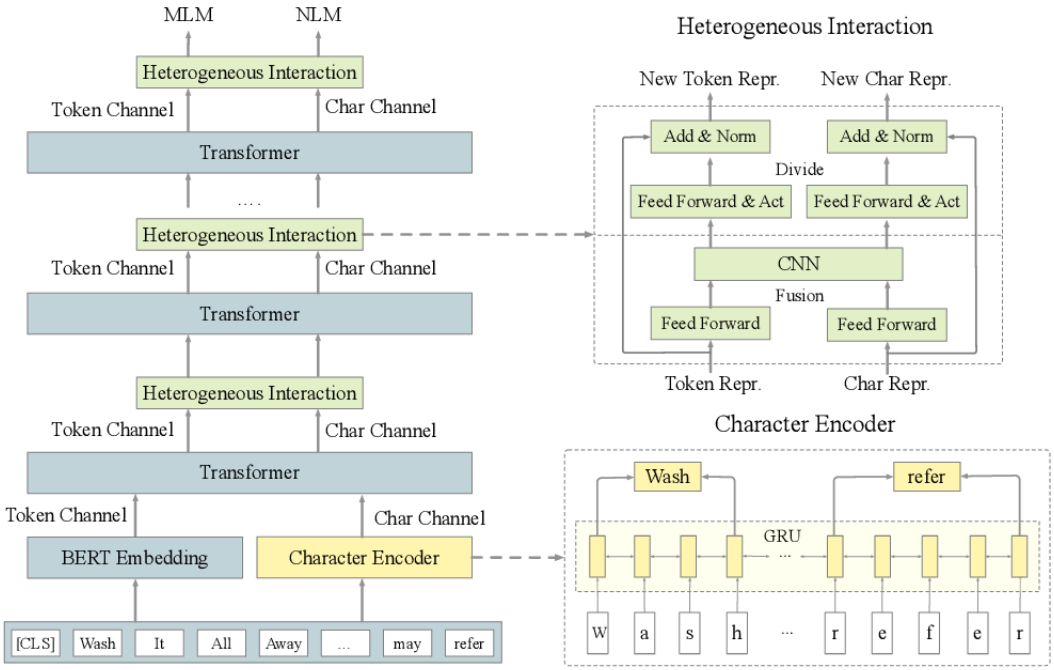

Figure 3.8: Architecture of the CharBert model [13], with focus on the *Character Encoder* and *Heterogeneous Interaction* modules.

- incomplete modelling: it is possible that the subword representations do not include both the representation of the complete word and the fine-grained character information;
- fragile representation: minor errors can significantly alter the BPE tokens, producing incomplete or erroneous representations.

These two problems may cause a lack of robustness. The main idea to solve this problem is to join the original pre-trained models' character representations and the subwords information together.

Model Architecture

Even if the main architecture is composed of transformers layers following the structure of Pre-trained Language Models like BERT, the innovation brought by this model is made by two other modules:

Character Encoder With the input sentences represented as character sequences, we must create token-level embeddings. In order to do so, the

token sequences are turned into characters and then incorporated into fixed-size vectors. Each character is embedded thanks to a character embedding matrix W_c:

$$e_j^i = c_j^i * W_c \tag{3.5}$$

where c_j^i is the character j of the token i and e_j^i is its respective embedding input vector, to extract the embedding.

Then, the sequence is passed into a bidirectional GRU model (**Cho et al.** [14]):

$$h_j^i(x) = BiGRU(e_j^i) \tag{3.6}$$

While the whole sequence is passed, the purpose of this model is to extract token-level embeddings. This is obtained by concatenating the hidden layers representing the first and the last character of the token:

$$h_i(x) = [h_1^i(x); h_{n_i}^i(x)] \tag{3.7}$$

Here, n_i represents the number of characters composing the token i.

In the end, the character contextual embeddings are passed to the Transformer layer along with the BERT embeddings.

Heterogeneous Interaction After the BERT embedding and the Char Encoder embedding go through each transformer layer, they are fused and separated by the heterogeneous interaction module. Here, the procedure can be divided into two steps:

- Fusion step: the two embeddings $t_i(x)$ and $h_i(x)$ for the i^{th} token are passed to two different fully-connected layers. Then the two representations are concatenated $w_i(x) = [t_i(x); h_i(x)]$ and passed to a Convolutional Neural Network.

  $$m_{j,t} = tahn(W_j * w_{t:t+s_j-1} + b_j) \tag{3.8}$$

  Where W and b are a weight matrix and bias, s_j is the window size of the j filter, $w_{t:t+s_j-1}$ represents the concatenation of the embeddings w in the window size s_j and m is the fusion and output of this step.

- Division step: this step exploits residual connections to retain both $t_i(x)$ and $h_i(x)$. In fact, after passing m through to different GELU layers,

the resulting $m_i^t(x)$ and $h_i^t(x)$ are summed with their respective residual connections:

$$T_i(x) = t_i(x) + m_i^t(x); H_i(x) = h_i(x) + m_i^h(x) \tag{3.9}$$

In the end, $T_i(x)$ and $H_i(x)$ go through a Layer Normalization step and are passed as input to the next Transformer layer.

Pre-training task

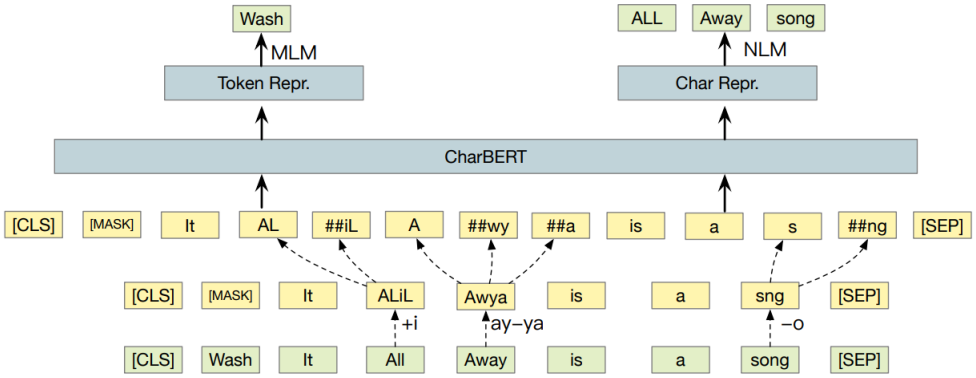

Figure 3.9: Graphical representation of the pre-training task *Noisy Language Modeling* [13].

This two-channel architecture enables the possibility of two different pre-training tasks. While the Token Channel solves the MLM task, the Character channel solves a new pre-training task called Noisy Language Modeling (NLM). This task was made *ad hoc* for this architecture.

In the beginning, the task modifies the word's original character order by deleting, adding, and switching internal characters. The objective is to reconstruct the whole word.

3.3 Graphs

The study of graphs, which are mathematical structures used to express relationships between objects, is a topic covered in the discipline of mathematics known as graph theory. A graph is made up of vertices, usually referred to as nodes, and edges, which connect pairs of vertices.

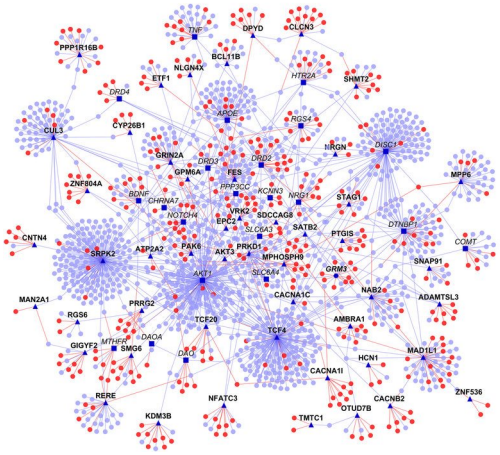

Figure 3.10: Example of the protein interaction network, in particular, it is a network view of the schizophrenia interactome shown as a graph.

Vertices represent units or entities, while edges depict the links or relationships between these items. A broad variety of scenarios from everyday life can be modelled using graphs, including social networks, transportation networks, computer networks, biological networks, and more.

In graph theory, the basic elements are the following:

- Vertices: these are the distinct components or entities that make up a graph. Vertices can stand in for individuals, communities, websites, proteins, etc.

- Edges: these are the links or interconnections that exist between vertices in pairs. Edges come in two different kinds: directed (have a specified direction) and undirected (have no direction at all).

- Degree: a vertex's degree is determined by how many edges are connected to it. The degree in an undirected graph represents how many neighbours a vertex has. In a directed graph, there is a distinction between in-going edges and out-going edges. From these two notions derive the concepts of in-degree and out-degree.

- Vertex sequences joined by edges are referred to as paths. In a graph, connectivity refers to the ability to connect any two vertex points through a network of edges.

These are only a few of the concepts that come from graph theory, this provides only an overview of the capabilities of graph theory.

As a matter of fact, everything that can be described as a set of items and connections can be structured as a graph (as shown in Figure 3.10). For this reason, there has been a need for Neural Networks capable of handling this type of data, like Graph Neural Networks.

3.4 Graph Neural Networks

Graph Neural Networks (GNN) were indirectly first introduced by *Gori et al.* [15]. In this research, a methodology for using neural networks to process structured data represented as graphs was provided, and the groundwork for GNNs was laid.

Later on, the GNNs were explicitly introduced by *Scarselli et al.* [16]. In this paper, are presented: the main architecture, the fundamental step, and the learning process behind these Neural Networks stressing how they are able to manage graphs with a range of sizes and structures.

In fact, they learn to encode the graph structure and node features to predict various tasks such as:

- Graph Classification: it refers to the process of giving a whole graph a label or category. The objective of this assignment is to classify or categorize each graph in the input collection of graphs based on its structural characteristics, qualities, or relationships. For instance, the assignment could entail categorizing each graph into a separate chemical class given a set of molecular graphs that represent chemical compounds.

- Link Prediction: its goal is to foretell missing or upcoming links between nodes in a graph. Examining the current graph structure and estimating the probability of a connection or link between two nodes that are not currently connected are required for the task. When only a partial graph is given and it is necessary to infer or fill in the missing connections, link prediction is very pertinent. It has uses in knowledge graph completion, recommender systems, social network analysis, and other areas.

- Node Classification: it is also known as node labelling or vertex classification, and it entails predicting the labels or categories of individual nodes in a network. Each node is labelled depending on its properties,

neighbourhood information, or other graph-related features. Node categorization is useful in a variety of applications, including recommendation systems, social network analysis, and fraud detection.

In order to solve these tasks, these models try to learn representations of nodes and edges in a graph that capture their structural and semantic properties. They operate in an iterative fashion, with a message-passing scheme. In fact, at each step nodes share information with their neighbors and update their representation accordingly.

For this reason, the main elements of the GNNs can be divided into:

- Node embedding function: this function maps each node in the graph to a high-dimensional vector. It depends on the domain of the analysis.
- Message-passing function: the purpose of this function is to aggregate the representation of the neighbours for a certain node and use them to update its representation, this is done for every node in the graph. This function is usually associated with a Graph Convolutional Operation. The function can be a neural network layer and depends on the chosen architecture. If the graph has edge features, they are usually involved as a weight to update the node embeddings in this step.

Depending on the objective of the task, there might be a readout function aggregates the node embeddings to produce a representation of the entire graph. This might be a simple aggregation function or the result of an encoder architecture for example.

3.4.1 Graph Convolutional Networks

Graph Convolutional Networks (GCNs) are a category of GNNs introduced by *Thomas N. Kipf et al.* [17].

The basic units for this network are the Graph Convolutional Layers. They are a generalization of the common convolutional layers usually exploited in the image analysis domain. They are designed to operate on the graph structure. With respect to past implementations, this model has a single weight matrix for each layer, and the adjacency matrix is properly normalized to allow for a range of node degrees.

For this reason, they take as input the graph, which is represented as a matrix X, where each row corresponds to a node in the graph and the columns are the dimension of the representation h, and the list of edges or

an adjacency matrix A, that is needed for the message-passing step. This step and the graph convolutional layer mathematically are defined as:

$$h_i^{(l+1)} = \sigma(\sum_{j \in \mathcal{N}(i)} \frac{1}{c_{ij}} W^{(l)} h_j^{(l)}) \tag{3.10}$$

where $h_i^{(l)}$ denotes the hidden representation of node i in the l-th layer, $W^{(l)}$ is the trainable weight matrix for layer l, it is important to specify that this weight matrix is shared among all the nodes in the graph.σ is the activation function, $\mathcal{N}(i)$ represents the neighbours nodes for node i, and c_{ij} is a normalization constant defined as:

$$c_{ij} = \sqrt{d_i d_j} \tag{3.11}$$

where d_i and d_j are the degrees of nodes i and j, respectively.

These architectures which efficiently reuse their local filters with learnable parameters, perform better with data where the underlying representation of the data is grid-like. When the kind of data does not respect this feature, there is a need for more complex architectures.

3.4.2 Graph Attention Networks

Graph Attention Networks (GAT), which were introduced by *Veličković et al.* [18], try to overcome this problem by introducing the concept of self-attention to graphs.

This type of GNN is based on the idea of assigning attention weights to the neighbouring nodes in the graph. In fact, they have a behavior similar to GCNs, as they take the same type of input, and produce similar output. The main change is in the aggregation function used in the message-passing step.

While, GCNs, fail to give distinct nodes in the same neighbourhood varying degrees of importance, GATs use a self-attention mechanism that allows each node to attend to its neighbours and selectively weigh the importance of each neighbour's features.

This is done by computing a weighted sum of neighbouring node features using attention coefficients α, which are learned for each node, neighbouring node pair i, j and layer l.

$$h_i^{l+1} = \sigma\left(\sum_{j \in \mathcal{N}i} \alpha i j^l W^l h_j\right) \tag{3.12}$$

The attention coefficients e are computed by first concatenating the feature vectors of the node and its neighbours and passing them through a feed-forward neural network that outputs a scalar value.

$$e_{ij}^{l} = \text{LeakyReLU}\left(\vec{a}^{T}[W^{l}h_{i}||W^{l}h_{j}]\right) \tag{3.13}$$

These scalar values are then passed through a softmax function to obtain the final attention coefficients, which sum to one across all neighbours.

$$\alpha_{ij}^{l} = \text{softmax}j(eij^{l}) = \frac{\exp(e_{ij}^{l})}{\sum_{k\in\mathcal{N}i}\exp(eik^{l})} \tag{3.14}$$

where $\mathcal{N}_i$ is the set of neighbours of node i.

The advantage of GATs is the ability to attend to only the most relevant neighbours for each node. They have proven to be an effective and efficient approach for learning node embeddings on graph data, and have achieved state-of-the-art performance on several node classification benchmarks.

Furthermore, the usage of attention weights also provides benefits in terms of the interpretability of the model.

Static Attention Problem Studies over GATs *Brody et al.*[19] revealed an underlying problem with this architecture, that is the Static Attention Problem.

In fact, based on their initial characteristics, the nearby nodes are given fixed attention weights via the attention mechanism. This indicates that the attention weights do not adjust or alter as a result of learning. As a result, nodes with comparable starting attributes get the same attention, which makes it difficult to distinguish between and pay attention to nodes depending on their dynamic relationships within the graph.

The research suggests a Dynamic Graph Attention (DGA) extension to GATs to alleviate the static attention issue. To help GATs capture more complex and adaptive attention patterns, DGA adds a second learnable attention weight matrix that enables attention weights to be adjusted dynamically during training.

This results in the introduction of a novel neural network architecture whose basic element is the GATv2 Layer. This is obtained by applying a after the non-linearity activation:

$$e_{ij}^{l} = a^{T}\text{LeakyReLU}\left([W^{l}h_{i}||W^{l}h_{j}]\right) \tag{3.15}$$

Though the change is simple, it is effective in providing an improvement in performance and robustness to noise.

3.4.3 GraphSAGE

Hamilton et al. presented GraphSAGE [20] (Graph Sample and Aggregated), a graph-based learning system that uses the concepts of sampling and aggregation to learn node representations in large-scale networks. An example is provided in Figure 3.11.

Firstly a sampling step is applied, which given a graph, samples a fixed number of nodes from each node's local neighbourhood. With respect to the previous approaches, there is a difference that not all the nodes belonging to the neighbourhood of a node i are used to calculate the embedding of that node.

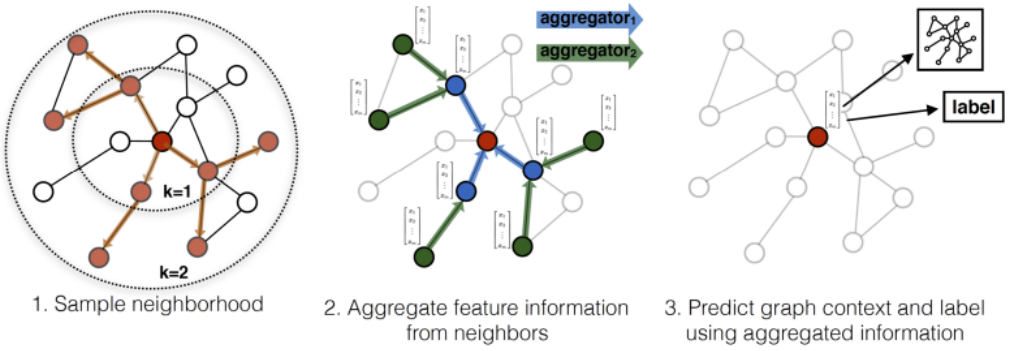

Figure 3.11: Examples of sampling and aggregation step from "Inductive Representation Learning on Large Graphs" [20]. Here k represents the number of layers of the model, which is linked to the depth of the neighbourhood, for example, $k = 1$ means that only direct neighbours are considered

After sampling the nearby nodes, GraphSAGE combines their attributes to provide a summary representation. This stage of aggregation tries to gather the neighbourhood's joint information. Mean pooling and maximum pooling are two examples of aggregate functions.

Mean pooling:

$$\mathbf{h}_v^{(l+1)} = \sigma\left(\mathbf{W}^{(l)} \cdot \text{CONCAT}\left(\mathbf{h}_v^{(l)}, \text{MEAN}\left(\{\mathbf{h}_u^{(l)} : u \in \mathcal{N}(v) \cup \{v\}\}\right)\right)\right) \tag{3.16}$$

Max pooling:

$$\mathbf{h}_v^{(l+1)} = \sigma\left(\mathbf{W}^{(l)} \cdot \text{CONCAT}\left(\mathbf{h}_v^{(l)}, \text{MAX}\left(\{\mathbf{h}_u^{(l)} : u \in \mathcal{N}(v) \cup \{v\}\}\right)\right)\right) \tag{3.17}$$

where, $\mathbf{h}_v^{(l)}$ is the representation at hidden layer l of node v , $\mathbf{W}^{(l)}$ is a the learned weight matrix of layer l and u is a node that belongs to $\mathcal{N}(v)$ which is the neighbourhood of v.

On each layer, the information is gained from the representations of the direct neighbours. Also, in this case, changing the number of layers can help to provide information levels from different depths.

Though the application of GraphSAGE takes into account the sampling step, in this work the implementation of the SAGEConv layer from *pytorch geometric* will be applied. This architecture does not imply a sampling step but provides the aggregation function discussed previously.

Chapter 4

Pipeline

4.1 General Pipeline

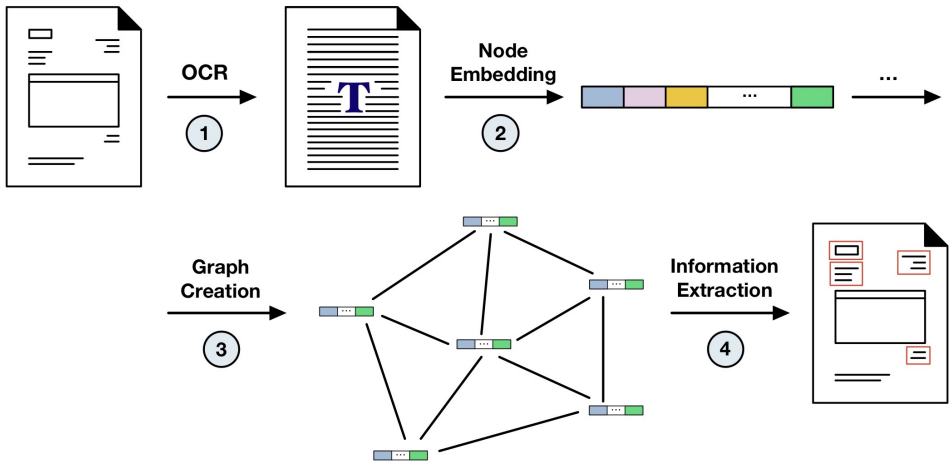

Figure 4.1: Graphical representation of the pipeline divided into steps. Step 1: The OCR tool docTR extracts text along with bounding boxes. Step 2: The text, along with the spatial features are used to create the node embeddings. Step 3: The graph is created using the Euclidean distance among nodes to select the edges. Step 4: Finally the graph is passed as a data point to the Graph Neural Network to extract the entities.

The end-to-end pipeline, shown in Figure 4.1, developed in this work is able to extract pieces of information from scanned invoices. The whole

pipeline can be divided into the following steps:

- **Optical Character Recognition**: the document is scanned using an OCR tool; the one chosen for this workflow is docTR, a system that accurately recognizes and extracts text along with the bounding boxes for each textbox.
- **Node Embedding**: each text box undergoes an extraction step to obtain spatial features and word-level embeddings to represent the node. In order to extract the latter, this work will exploit FastText, BERT and CharBERT.
- **Graph Creation**: each text box in the document is a node in a graph that reflects the document as a whole. The parameter chosen to define edges is the Euclidean distance. In actuality, each node forms an edge with the neighbours it is closest to.
- **Information Extraction**: the entire document is entered as data and sent to the selected Grap Neural Network model, which has been trained to classify tokens in an invoice. The models taken into account for the analysis will include the following convolutional layers: GCNConv, GATv2Conv and SAGEConv.

However, a dataset labelling step is executed before creating the whole pipeline. In this step 1400 invoices are selected and subsequently labelled by using LabelImg. For each document, 5 entities have been selected and labelled by drawing the corresponding bounding boxes.

In this chapter, the implementation of each step of this pipeline will be analyzed in deep, presenting also the different possibilities for each part.

4.2 Dataset

The dataset obtained for the purpose of the analysis is a part of the RVL-CDIP Dataset, created by *Adam W. Harley et al.* [21]. RVL-CDIP is a subset of IIT-CDIP, which came from the Legacy Tobacco Document Library.

This dataset is composed of 400,000 grayscale images. The images are scaled so that the dimension does not exceed 1000 pixels. The original purpose of the dataset was to be used for Image Classification tasks. The dataset is divided into 16 classes, having 25,000 samples for each class. That labels are:

- 0: Letter
- 1: Form
- 2: Email
- 3: Handwritten
- 4: Advertisement
- 5: Scientific Report
- 6: Scientific Publication
- 7: Specification
- 8: File Folder
- 9: News Article
- 10: Budget
- 11: Invoice
- 12: Presentation
- 13: Questionnaire
- 14: Resume
- 15: Memo

The dataset is also divided into train, test and validation having respectively 320,000, 40,000 and 40,000 samples for each split.

The goal of our task is to extract key information from the text in the images and in particular from the invoices. For this reason, the dataset needed some manual processing to be exploited.

Firstly, through a Python script, the class *Invoice* has been filtered in order to avoid considering the other classes in the analysis.

Subsequently, 1200 images have been selected. the reason why it was needed to select filter images is that many of them presented too much noise. In order to proceed successfully with the task, the OCR tool should be able to extract text from the image. For this reason, the quality should be enough to proceed with the task. The OCR tool chosen for this project is DocTR.

LabelImg is a graphical image annotation tool. This tool is written in Python and uses Qt for its graphical interface. LabelImg has been widely used to label datasets for object detection, image segmentation, and object recognition tasks. For this reason, it presents a large list of default classes, but it also allows customization.

Hence each image is passed to LabelImg and we define the following labels for the correct item in the document:

- Company_name: it is the name of the company releasing the invoice. It is a string that can be extracted from the logo or full corporate name.
- Company_address: a string containing the full address of the company releasing the invoice. In some invoices, there are more addresses relative to the same company. These are usually, the address where to send the payment and the legal address of the company.
- Invoice_date: a string containing the date on which the invoice was released. The format of the date can change.
- Invoice_number: an alphanumerical string that identifies the invoice for the company.
- Total_due: a string containing the total that needs to pay, it could also include the currency used.

The *Company_name* and *Company_address* of the company receiving the invoices are not labelled, as only a few companies of this type are present in the dataset. This could also result in a bias towards them.

LabelImg supports different formats: YOLO, CreateML, and the PASCAL VOC format which was the one chosen for the analysis. Each image in the collection has structured annotations provided by the format, which comprises XML files. This file is called the Annotation file and it contains data for only one image. The following are the main components of the PASCAL VOC labelling format present in the Annotation file:

- Image Information: the image's filename, width, height, and depth (the number of colour channels) are all included in the information at the beginning of the annotation file.
- Object Annotations: the annotation file contains a single XML element for each annotated object in the image. The object element includes details about the item. Figure 4.2 proposes an example.

- Class Label: the category or kind of the object is represented by the class label. It is usually displayed as a text string.
- Bounding Box: it indicates the object's spatial range in the image. The x and y coordinates of the top-left corner and the x and y coordinates of the bottom-right corner make up its four defined coordinates. These coordinates identify the tightly contained rectangular area around the object. Also, the coordinates are defined over the size of the image and will need scaling.

```
<object>
    <name>total_due</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
        <xmin>697</xmin>
        <ymin>927</ymin>
        <xmax>755</xmax>
        <ymax>951</ymax>
    </bndbox>
</object>
<object>
```

Figure 4.2: Example of Object Annotations for an item labelled as *total_due*.

4.2.1 DocTR

In order the work on the dataset, the first step is to extract the text from images. To do so an OCR is exploited which is DocTR. As previously mentioned, it is a Python library that allows exploiting pre-trained models for End-to-End OCR. This task is solved in two steps: text detection (localizing words), then text recognition (identifying all characters in the word). The pre-trained model chosen was the default models provided by DocTr which are the 'db_resnet50', for the text detection task and the 'crnn_vgg16_bn' text and character recognition.

Pre-processing Each image is passed through the OCR to extract the text and its respective bounding box.

One preprocessing step before aligning the text with the labels is to eliminate noise from the dataset. As the images in the dataset are scanned, DocTR could have trouble distinguishing noise from the real text. Hence, the models provide a confidence level for each text box extracted. In order to, remove noise and also keep as much text as possible, a threshold level of 60% has been chosen as a filter. Furthermore, text boxes containing only special characters are removed, as most of them were the result of image noise.

Label Alignment As previously mentioned, DocTR hierarchically decomposes the image into pages, blocks, lines and words. Each word has its own text-box and, for this reason, the text box of the label could contain different words. In order to provide a label for the text boxes, the centre of each bounding box is calculated. Subsequently, the coordinates of the centre are passed to a step, that checks if the values are inside the bounding box of an object of the image in analysis. If the text box does not belong to any object of the document, it is labelled as *O* as Outside the Entity.

Output structuring The hierarchical structure provided by DocTR into blocks and lines may be useful for the construction of contextual embeddings, for this reason, each page is composed of a list of blocks, where each block is a list of dictionaries.

A single dictionary serves to contain the information for the text box, and it will have the following keys:

- bounding_box: the coordinates of the text-box as s x_min, x_max, y_min, and y_max.
- value: the string of text representing the word.
- label: the string that shows if the text corresponds to an object.

The images are then collected into a list and formatted into a JSON file. Before this whole pipeline, the whole dataset is split into train, test and validation, as presented in Table 4.1.

The split follows an 80-10-10 percentage, in order to maximize the number of samples in the training set, having a limited number of images to train the model.

	Train	Test	Validation
#Images	1120	140	140

Table 4.1: Statistic on how the dataset is split following a 80-10-10 split

	Train	Test	Validation
Words	95905	12185	12188
Lines	37828	4864	4711
Company_name	4661	593	619
Company_address	8103	1072	1042
Invoice_date	2249	266	279
Invoice_number	948	125	116
total_due	1164	143	148
Outside the entity	78780	9986	9984

Table 4.2: Number of lines, words and how they are divided into each class. This statistic is presented for train, test and validation splits

Table 4.2 presents the number of text boxes, lines, and class distribution over the invoices.

Some considerations need to be made about the dataset. First of all, lines can be composed of many words but also one word can be considered as a line if it is clearly separated from other blocks of text.

Secondly, as previously mentioned more than one word can correspond to a label, but also, some objects may repeat in the invoice. For example, *Company_name* or *Company_address* are the most common to show this behaviour.

Lastly, the fact that the dataset is unbalanced is the most evident element on the table. As expected, most of the text present in the invoices is not an entity. It is important to note that the single items and their respective values are not considered entities in this work.

4.3 Node embedding

Node Embedding is one of the most important steps in the pipeline. The idea is to collect both information from the pure text and also from images to create node embeddings. Hence, not only semantic or syntactic features

but also spatial attributes.

4.3.1 Spatial features

The extraction of spatial features can be easily linked to the features of the bounding box. In fact, each piece of text comes with its coordinates in the image. These coordinates not only provide the location of the text box in the image but they also can be used to calculate the area that the text box occupies in space.

All of these additional features can be obtained through linear transformations of the bounding box coordinates. For this reason, the starting spatial features will be:

- x_min
- x_max
- y_min
- y_max

They are all floating point numbers, which are concatenated into a 4-dimensional vector.

4.3.2 Word Embeddings

While spatial features could help the model to understand relationships in the image, the most important information comes from the text. In order to use text as an input feature, it has to be transformed into a fixed-size vector. As aforementioned, Word Embeddings are a popular solution to this task The advantage brought by them is the ability to encode both syntactic and semantic features of the text by mapping it into a latent space. In the end, the text is represented as a continuous dense vector.

FastText FastText is a largely known library for word representation. Not only it encodes semantic properties of text, but also morphological information by working with subwords. On top of that, it is also able to handle out-of-vocabulary words or rare words.

Moreover, is a highly efficient and scalable solution. Using methods like hierarchical softmax and negative sampling, it can handle big text datasets

and train embeddings quickly. This qualifies it for use in contexts with limited resources and industrial-scale applications.

Furthermore, it also provides pre-trained word embedding for different languages. For this reason, in this work, the pre-trained model *cc.en.300* from the FastText library will be chosen to create the embeddings. It is a pre-trained model for the English language. It is trained on the *Common Crawl* dataset.

This dataset comprises online pages from many sources that span a wide range of subjects and areas. It comprises textual material, HTML coding, visual elements, and more website resources. The often updated dataset enables academics, developers, and data scientists to examine and gather important insights from the web.

The *cc.en.300* produces a single continuous dense vector made of 300 features for each word. This makes it easy to obtain a single representation of a word to use as Node embedding.

BERT and CharBERT BERT-based models work in a different way. The first unit of these architectures is the Tokenizer. In order to handle out-of-vocabulary words, this module functions at the sub-word level by dividing words into smaller units called tokens or sub-words. This division is made upon a predefined vocabulary, mapping each token to an ID. The symbol "##" serves to recognize that the token is part of a word. Furthermore, special tokens are used to perform different tasks. The [CLS] task serves for classification tasks over a whole sentence instead of the single tokens. While the [SEP] tokens serve for defining the end of a sentence and the start of another (if it is present).

BERT models exploit these tokens along with positional embeddings and provide fixes sized vector representation of 768 dimensions for each token.

In order to provide a Node embedding for this work there are different options:

- Token as Node: each token can be considered as a node itself. This comes with some concept issues. Two tokens coming from the same word will have the same spatial features, it is not efficient to separate the boxes according to sub-words. Furthermore, while generating the graph the majority of the nodes will have as adjacent nodes mostly tokens coming from the same word.

- Aggregation functions: the node embeddings represent complete words.

In order to do so, there is the need to apply aggregation functions over the different token vectors. Different examples of these functions are: mean, sum, and max pooling.

- CLS token as Node embedding: this special token is used for sentence classification tasks and, for this reason, serves as sentence embedding. The idea would be to pass a single word to the BERT-based model and exploit the CLS token's final representation as token embedding. By doing this, even if the word is divided into different tokens, it will be treated as a sentence and it would be embedded into the CLS token. The main problem with this solution is the inability to extract information from the context provided by different words.

The main choice for this research will be using Aggregation Functions, on top of token embedding vectors. This type of solution, even if simple, it is the one with fewer cons. In fact, applying aggregation functions like mean on top of Word Embeddings has been one of the most used techniques to obtain Sentence Embeddings, for example considering Sent2Vec.

4.4 Graph Creation

The last step of Data preparation before obtaining the input for the Graph Neural Network is the creation of the Adjacency Matrix, which is fundamental for the Message Passing and Node update steps.

Adjacency Matrix The rows and columns of the adjacency matrix are equivalent to the graph's vertices. Each element in the matrix represents the presence or absence of an edge between the two nodes. If the edges are not weighted, usually if there is a connection between the two nodes, the value of the entry would be usually 1 and 0 otherwise. For weighted graphs, the values for the entries will be the weights assigned to each edge.

Furthermore, if the graph is undirected, the adjacency matrix will be symmetric. This means that by taking two nodes i and j if the value of the entry in the position (i, j) will be equal to the one in (j, i).

The most evident relationship that can be exploited to create edges is *proximity*. The whole document is represented by the words and their

position (bounding box), it is natural to exploit the spatial position that can be extracted in an unsupervised fashion as the foundation for the graph.

The idea is to calculate the center of whole the bounding boxes. Then the Euclidean distance among the centre coordinates is calculated for all the nodes in the invoice. Subsequently, the distances between one node and all the others are stored in a vector. These vectors are then sorted in crescent order. Afterwards, for each node, the k closest nodes are chosen to form an edge, and since this is an undirected graph also the symmetric edge is added to the matrix. In the end, also the values of the distances are stored.

Both k and also the values of the distance will be used in the experiments section. The degree of each node influences the way it is updated by each layer of Graph Convolution. In a similar way, the values of the distances used as edge weight can be used to influence the "influence" that adjacent nodes to a certain node have on the update of the latter.

The base setup for our analysis includes a $k = 10$ and the edge weight obtained with the inverse of the distance and then normalized to be included in the range [0,1]. It is important to note that not all Graph Convolutional Layers are able to elaborate edge weight or edge attributes (if multi-dimensional).

In the end, each document is represented by an instance of the *Data* class from *Pytorch Geometric*. The latter is a PyTorch-based toolkit that makes it simple to create and train GNNs for a variety of applications using structured data.

4.4.1 Data object

A data item that describes a homogeneous graph, which is a type of graph where all nodes and edges belong to the same domain and are of the same type. The data object can store properties at the node, link, and graph levels. Data generally attempts to behave like a typical Python dictionary. It allows for storing different parameters:

- x: a matrix storing the features for all nodes in the graphs.
- y: it can store both graph-level and node-level labels.
- edge_index: it represents the adjacency matrix, but uses COO format. The latter is used usually for sparse matrices, the format transforms the adjacency matrix into columns and rows arrays.

- edge_attr: it can store eventual features for each edge. The features can be torch Tensors.

4.5 Model

4.5.1 Architectures

The architecture will exploit both textual and spatial embeddings to create node embeddings, which will be passed to a Graph Neural Network to solve the Node Classification Task.

The spatial embeddings will be passed to a Linear layer that will upscale the 4-dimensional vector to a 256 features array. This array is concatenated to the 768-dimensional word-level embedding provided by BERT. For a total *input_size* of 1024.

The last step is building the Graph Neural Network architecture. For this purpose, the main units composing this architecture will be Graph Convolutional Operators. Even if Pytorch Geometric provides also standard models, in this work the models will be built from the Convolutional Blocks.

Pytorch Geometric provides a series of customizable Convolutional Layers from many research papers, all the layers can be taken from *torch_geometric* library:

- GCNConv: it is the Graph Convolutional layer from [17].
- GATv2Conv: it is the implementation of the attentive convolutional layer from [19].
- SAGEConv: it is the GraphSAGE operator from [20]. Both mean pooling and max pooling will be explored.

The main structure of the model will be equal for all the models which will be divided into Message Passing Layer as shown in Figure 4.3.

This layer will be composed of the chosen Graph Convolutional Layer, the ReLU activation function, and a Dropout layer providing a regularization function to limit overfitting.

In the model which exploits GATv2Conv, the Message Passing Layer has the addiction of Linear layers after each Dropout layer as it is represented in Figure 4.4. This system is needed because the attentive layers are provided with attention heads, and each one of them provides its output. The layer

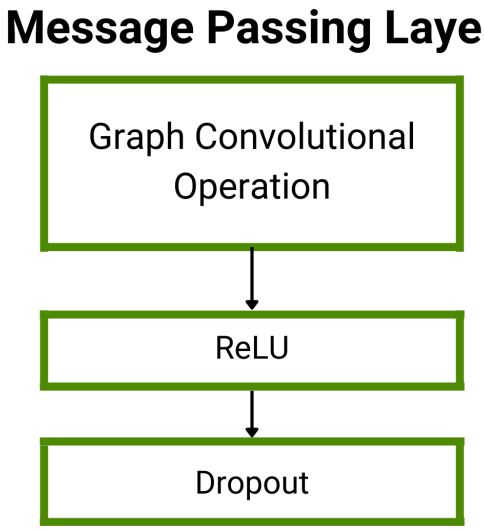

Figure 4.3: The general schema of the Message Passing Layer with Graph Convolutional Layer, ReLU activation Layer, and Dropout.

can average the output or concatenate it to obtain a single representation. The second solution, combined with linear layers, allows the model to learn how to aggregate the output of the heads.

All the models will have as a base model two Message Passing Layers stacked on top of each other and a final Linear layer that will bring the output size to the actual tagset size. The values for all the layers are shown in Table 4.3.

	GCNConv	GATv2Conv	SAGEConv
Linear layer out_channel	256	256	256
Layer 1 input_channel	input_size	input_size	input_size
Layer 1 out_channel	1024	512	1024
Layer 2 out_channel	512	256	512
Layer 1 heads	-	12	-
Layer 2 heads	-	7	-

Table 4.3: Summary of the parameters of the models.

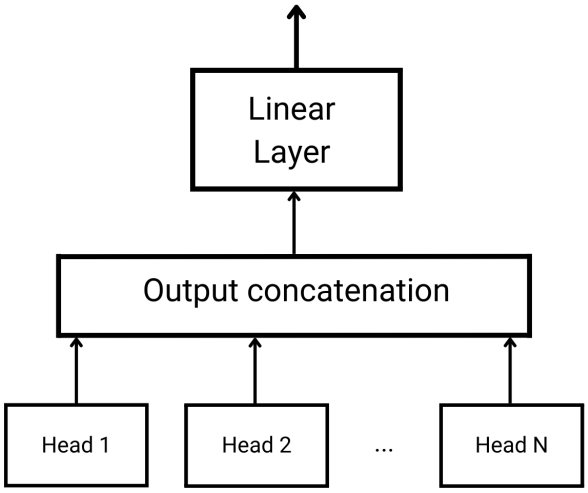

Figure 4.4: How the output of the GATv2Conv Layer is handled. If the layer has N attention heads, their output is concatenated into $N{*}output\_channels$ array. This vector is passed to a Linear Layer which has an output size equal to output_channels dimension.

4.5.2 Training Settings

The whole training process for this work is done on Google Colab with an NVIDIA T4 GPU. The model is trained for 300 epochs with a batch size of 32.The optimizer chosen is *Adam*, implemented by the *pytorch* library, with the following parameters: learning rate $lr = 10^{-3}$, $\epsilon = 10^{-8}$ and $weight\_decay = 0.01$. The parameters are summarized in Table 4.4.

Epochs	300
Batch size	32
Adam Learning rate	10^{-8}
Adam epsilon	10^{-3}
Adam weight decay	10^{-3}

Table 4.4: Summary of the parameters used for the training step.

To calculate the loss during the whole training process, Cross Entropy has

been selected. It is a popular loss function in machine learning, particularly for classification problems. It quantifies the discrepancy between the actual distribution of class labels and the projected probability distribution.

$$L = -\sum_{i} y_i \log(p_i) \tag{4.1}$$

Where L is the loss, y_i is the true label for class i, and p_i is the probability of the prediction for class i.

Furthermore, the *pytorch* implementation for this loss is able to ignore certain classes during the calculations. For this reason, the PAD label used to pad the number of nodes and features in the graph will be ignored. Moreover, there is the possibility of handling an unbalanced dataset by giving a weight to each class. The equation for weighted Cross Entropy Loss is:

$$L = -\sum_{i} w_i \cdot y_i \log(p_i) \tag{4.2}$$

Where w_i is the weight for class i. These weights are passed as a 1-dimensional tensor used as a parameter. The chosen weights are listed in the Table 4.5.

Non_Entity	0.05
Company_name	1
Company_address	1
Invoice_date	1
Invoice_number	1
total_due	1

Table 4.5: Summary of the weights used to calculate the Cross-Entropy Loss.

4.5.3 Metrics

The metrics used in this work are calculated by exploiting the *sklearn.metrics*. The following three metrics are calculated: Precision (P) Table 4.3, Recall (R) Table 4.4 and F1 score (F1) Table 4.5. Furthermore, the F1 score is the criterion to choose which model to save during training. All the metrics and their components are explained below in detail. Given a class i belong to a

set of classes C, and a class j such that $j \epsilon C : \{j \neq i\}$, we can calculate the following metrics for i :

- True Positive(TP): the number of entities i that are correctly classified as i.
- False Positive(FP): the number of j classified as i.
- True Negative(TN): the number of j non entities classified as j.
- False Negative(FN): the number of entities i that are classified as j.

Then, to evaluate the model, the following three metrics are calculated: Precision (P) in Eq. 4.3, Recall (R) in Eq. 4.4, and F1 score (F1) in Eq. 4.5.

$$P = \frac{TP}{TP + FP} \tag{4.3}$$

$$R = \frac{TP}{TP + FN} \tag{4.4}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4.5}$$

The reason why F1 is used as the criterion to save the best model during training, with respect to other metrics, is because this dataset and in general this type of data are unbalanced. There is a high presence of non-entity nodes.

In these situations, accuracy alone can be deceiving since a model that consistently predicts the majority class would have high accuracy but may struggle with the minority class. The F1 score offers a balanced evaluation measure that takes into account the performance of both classes, providing a better assessment of the model's overall performance.

The F1 score finds a compromise between accuracy and recall, merging both measures into one number that represents the model's capacity to simultaneously detect pertinent occurrences (recall) and reduce false positives (precision).

Chapter 5

Results

5.1 Analysis of the results

This section is dedicated to the discussion of the results obtained with the aforementioned pipeline.

The experiments were all done over Google Colap exploiting an NVIDIA T4 GPU. This GPU is equipped with 16 GB of high-speed GDDR6 memory and 2,560 CUDA cores. For every test, *Precision*, *Recall* and *F1-score* are calculated and the latter is used as the criterion to save the best model.

These experiments along with the Ablation Study, want to analyze the performances of the difference architecture on Information Extraction over this type of heterogeneous semi-structured data. Furthermore, these works want to uncover which components and configurations can really generalize this type of data.

For this reason, the results will be provided for the chosen Graph Neural Networks:

- GCNConv;
- GATv2Conv;
- SAGEConv, testing both mean_pooling and max_pooling.

5.1.1 Expectations

Due to the heterogeneous nature of the data, which changes and does not provide a grid-like structure, the expectations are that the GCNConv-based

architecture will perform poorly with respect to the others. This is because GCNConv works on the assumption that neighbouring nodes in the graph have equivalent relevance and can give relevant information. Instead, the mean pooling aggregation in SAGEConv and the attention mechanism in GATv2Conv allow them to consider a wider variety of neighbourhood information, leading to more expressive representations. Furthermore, SAGEConv and GATv2Conv are better suited for graphs with changing neighbourhood sizes or node degrees.

Comparing SAGEConv and GATv2Conv, it is important to mention that the latter is able to process edge features, that can be used along with the attention mechanism to handle graphs with varying neighbourhood sizes and heterogeneous node degrees. Thanks to this flexibility, the GATv2Conv-based architecture is expected to provide more accurate results.

5.1.2 Results

GCNConv	**Precision**	**Recall**	**F1-Score**
Non-Entity	0.99	0.89	0.94
Company_name	0.74	0.85	0.79
Company_address	0.64	0.96	0.77
Invoice_number	0.59	0.75	0.66
Invoice_date	0.78	0.87	0.82
total_due	0.36	0.93	0.52
macro_average	0.68	0.88	**0.75**

Table 5.1: Results obtained with the GCNConv-based architecture.

As expected the GCNConv-based model is outperformed by the others, as shown in Table 5.1. This is probably to the lack of flexibility with respect to the other architectures.

The results show that the GATv2Conv-based model is able to predict in a more precise way the entities in the invoice, obtaining a macro average **F1-score** of 0.86, as shown in Table 5.2. This is probably thanks to its ability to provide attention weights to each adjacent node and elaborate edge weights.

Nonetheless, looking at Table 5.4, similar results are obtained from the SAGEConv-based model with mean-pooling aggregation, which obtained 0.85

GATv2Conv	Precision	Recall	F1-Score
Non-Entity	0.98	0.96	0.97
Company_name	0.90	0.87	0.88
Company_address	0.83	0.95	0.89
Invoice_number	0.85	0.75	0.80
Invoice_date	0.87	0.89	0.88
total_due	0.62	0.84	0.72
macro_average	0.84	0.88	**0.86**

Table 5.2: Results obtained with the GATv2Conv-based architecture.

SAGEConv	Precision	Recall	F1-Score
Non-Entity	0.98	0.95	0.97
Company_name	0.84	0.91	0.87
Company_address	0.85	0.92	0.88
Invoice_number	0.81	0.79	0.80
Invoice_date	0.83	0.93	0.88
total_due	0.54	0.92	0.68
macro_average	0.81	0.90	**0.85**

Table 5.3: Results obtained with the SAGEConv-based architecture and mean_pooling aggregation method.

macro-average **F1-score**, as visible from Table 5.3. The fact that this model does not leverage edge weights, raises the question about how important it is to employ this feature for the prediction.

Furthermore, from the tables, it is observable a difficulty of the models to classify accurately the *Invoice_number* and the *total_due* classes.

First of all, these are the two least represented classes in the dataset. This probably causes the models to be biased towards the others.

Moreover, the invoices in the dataset are all from 20^{th} century. Hence, they tend to be a little outdated and less structured than nowadays invoices, not always providing key-value pairs.

Additionally, there were cases where the invoice presented only one item and the *total_due* was not presented in a different voice. Consequently, the model could wrongly classify some item prices as *total_due* classes. This

SAGEConv	Precision	Recall	F1-Score
Non-Entity	0.98	0.95	0.97
Company_name	0.77	0.90	0.83
Company_address	0.86	0.93	0.89
Invoice_number	0.86	0.79	0.82
Invoice_date	0.83	0.94	0.88
total_due	0.52	0.93	0.66
macro_average	0.80	0.91	**0.84**

Table 5.4: Results obtained with the SAGEConv-based architecture and max_pooling aggregation method.

can be seen from the **Precision** score which is significantly lower than the **Recall**. The disparity between the two scores highlights the presence of a high number of False Positives for the class *total_due*.

5.2 Ablation study

This section is dedicated to the ablation study on the whole solution. Hence, it will be analysed how the performances change by modifying elements not only in the architecture of the Graph Neural Network but also by working on how the nodes embedding and the graph itself are created.

The following experiments will be performed:

- Changing embedding model from BERT to CharBERT and FastText.
- Not applying edge weight over the graph and the GATv2Conv-based model.
- Analysing the performance with a 1-layer GNN and a 3-layer GNN.
- Lowering the number of edges for each node.

5.2.1 Embedding model

The two models chosen for this analysis are CharBERT and FastText. CharBERT follows an implementation for the input system similar to BERT.

For this reason, the input will follow the same steps. Firstly rearrange the textbox into a line then, tokenize the different words and pass them to CharBERT. Subsequently, the output embeddings are groped by word and aggregated into only a vector for each word with mean pooling.

Instead, FastText can elaborate out-of-vocabulary words without providing an embedding for each subword. Hence, each word from a textbox is passed to the model to obtain its representation.

BERT	**Precision**	**Recall**	**F1-Score**
Non-Entity	0.98	0.96	0.97
Company_name	0.90	0.87	0.88
Company_address	0.83	0.95	0.89
Invoice_number	0.85	0.75	0.80
Invoice_date	0.87	0.89	0.88
total_due	0.62	0.84	0.72
macro_average	0.84	0.88	**0.86**

Table 5.5: Results obtained with the GATv2Conv-based architecture and BERT to provide the textual embeddings.

FastText	**Precision**	**Recall**	**F1-Score**
NE	0.99	0.89	0.94
Company_name	0.71	0.88	0.79
Company_address	0.72	0.94	0.81
Invoice_number	0.44	0.85	0.58
Invoice_date	0.73	0.95	0.83
total_due	0.31	0.94	0.46
macro_average	0.65	0.91	**0.73**

Table 5.6: Results obtained with the GATv2Conv-based architecture and FastText to provide the textual embeddings.

By comparing the different results in Tables 5.5 and 5.6, it is possible to affirm that the model exploiting contextual embeddings obtained better results for all the classes.

The contextual embeddings in general provide better semantic representations and also these representations are dynamic. Based on the particular

CharBERT	Precision	Recall	F1-Score
NE	0.98	0.96	0.97
Company_name	0.84	0.86	0.85
Company_address	0.87	0.91	0.89
Invoice_number	0.79	0.85	0.82
Invoice_date	0.84	0.86	0.85
total_due	0.44	0.75	0.55
macro_average	0.79	0.87	**0.82**

Table 5.7: Results obtained with the GATv2Conv-based architecture and CharBERT to provide the textual embeddings.

context of the sentence or text being examined, embeddings are formed dynamically.

From the results, it is evident that this feature is also relevant in the invoice domain, where the lines composing the document are not mostly sentences or descriptions, but key-value pairs.

Analysing the results in table 5.7, it appears that, in this case, CharBERT not only does not provide an improvement concerning BERT but it also underperforms. The answer to this behaviour could be found in the *Robustness Analysis* provided by *Ma et al.* in [13] shows that CharBERT has a higher Sensitivity than BERT model.

The dataset is composed of scanned invoices which may present noise because they also present handwritten notes or the quality is poor.

5.2.2 Edge weigths

Analyzing the results provided by the GATv2Conv-based architecture and the ones provided by the SAGEConv-based ones. A question about the importance of edge weights has been raised.

The edge weights are used by the GATv2Conv layer as a score different from the attention mechanism. As a matter of fact, from table 5.9 it is possible to see a decrease in performance in all the classes, with respect to results shown in Table 5.8. This provides clear evidence of the importance of edge weights in this task.

edge weights	Precision	Recall	F1-Score
Non-Entity	0.98	0.96	0.97
Company_name	0.90	0.87	0.88
Company_address	0.83	0.95	0.89
Invoice_number	0.85	0.75	0.80
Invoice_date	0.87	0.89	0.88
total_due	0.62	0.84	0.72
macro_average	0.84	0.88	**0.86**

Table 5.8: Results obtained with the GATv2Conv-based architecture and using edge weights.

No edge_weight	Precision	Recall	F1-Score
NE	0.98	0.93	0.95
Company_name	0.72	0.89	0.79
Company_address	0.77	0.92	0.84
Invoice_number	0.69	0.66	0.68
Invoice_date	0.72	0.85	0.78
total_due	0.53	0.83	0.65
macro_average	0.74	0.85	**0.78**

Table 5.9: Results obtained with the GATv2Conv-based architecture and not using edge weights.

5.2.3 Number of Message Passing Layer

The number of layers in the architecture represents also the length of the path of message passing. For example, if there is only one layer, the node embedding will be influenced only by direct neighbours, while an increasing number of layers enables the influence of further neighbours.

This possibility can be beneficial in terms of the context provided to each node, allowing the model to capture more complex and high-level relationships in the graph.

However, it also increases the complexity in terms of memory and computations. Furthermore, especially when the training data is scarce or noisy, the model becomes more prone to overfitting. Moreover, additional layers might not significantly increase performance and might even make the learnt representations more noisy or unstable.

As it is visible from the results in Table 5.10 the increase in layers, in this case, does not provide an improvement. Probably, due to the aspects aforementioned.

3-layer	Precision	Recall	F1-Score
NE	0.98	0.95	0.96
Company_name	0.79	0.89	0.84
Company_address	0.87	0.91	0.89
Invoice_number	0.76	0.74	0.75
Invoice_date	0.85	0.82	0.84
total_due	0.41	0.83	0.55
macro_average	0.78	0.86	**0.80**

Table 5.10: Results obtained with the 3-layer GATv2Conv architecture.

5.2.4 Number of edges

Changing the number of nodes selected to form an edge is another way to impact the way the node embedding is conditioned by its neighbourhood. As a matter of fact, the challenges are similar to changing the number of layers in the architecture.

A node can collect data from a greater number of nearby nodes by adding additional edges to it. A richer and more complete representation of the local graph structure is made possible by the enhanced connectedness, which may be able to capture more complex dependencies and relationships. Still, with many connections, the node may become overly dependent on some surrounding nodes, which would result in poor generalization of unknown data. Additionally, the neighbourhood data of the node may become more noisy and redundant with more edges.

Also, reducing the number of edges makes a node's local neighbourhood information easier to understand. By doing so, the node can concentrate on its most useful neighbours and reduce noise and unimportant connections. However, the model's capacity to generalize may be limited if the fewer edges do not accurately describe the graph's underlying structure. Furthermore, a node can only access a certain level of local connectivity when the number of edges is reduced. Important relationships and dependencies that were present in the graph may be lost as a result of this. The model's capacity to

identify fine-grained patterns and complex interactions may be constrained by the limited local information. In the end, for this type of task lowering the number of edges provides worse results, as shown in Table 5.11.

5-node neighbourhood	**Precision**	**Recall**	**F1-Score**
NE	0.98	0.96	0.97
Company_name	0.82	0.86	0.84
Company_address	0.86	0.92	0.89
Invoice_number	0.84	0.82	0.83
Invoice_date	0.88	0.85	0.87
total_due	0.44	0.85	0.58
macro_average	0.80	0.88	**0.83**

Table 5.11: Results obtained with the GATv2Conv-based architecture and creating the graph with 5 neighbours for each node.

Chapter 6

Conclusions

6.1 Conclusion

In the end, this works provides an end-to-end tool for Information Extraction over scanned invoices.

This is done by a multi-step pipeline that extracts text and positional features from the scanned invoices. Secondly, text features are embedded into continuous vectors by BERT, and along with the spatial features are used to generate the node embedding. Subsequently, the graph is generated, by selecting using the Euclidean distance as a criterion for the creation of a node's neighbourhood. Finally, the document is passed into a Graph Neural Network to extract the entities in analysis.

This work examines the results of this task, analyzing a dataset of invoices with a variety of compositions, while also giving a general review of the various methods for the three key processes of graph formation, token embedding, and node classification.

While the performances of this solution are not sufficient to completely deploy the pipeline for a full-automated tool, this works provides a well-rounded research and a good starting point for the performances of Information Extraction over a dataset of heterogeneous invoices.

Furthermore, this thesis also provides a handcrafted public dataset [1] composed of 1400 invoices, with 5 entities labelled. The hope is that this could represent the start of more development in this field of research.

6.2 Future Works

Due to limitations, caused by the need for the creation of a dataset, some solutions have been left unexplored.

One possibility is to provide BERT with an ulterior step of pre-training by exploiting the invoices. This solution creates a BERT model specialized over a certain domain. As it has been already proved by *Syed et al.* in [22], the technique of adapting BERT to a certain domain can bring improvement to the task by creating richer and more contextualized embeddings.

Alternatively, self-supervised techniques have been frequently applied to Graph Neural Networks. Two examples of these techniques are:

- Graph Reconstruction: in this assignment, a GNN is trained to decipher partial or corrupted graph versions and recreate the original graph structure. Throughout the reconstruction process, the model gains knowledge of the underlying connection patterns and dependencies in the graph.

- Graph Context Prediction: the GNN gains the ability to predict a node's surroundings or context. The model is conditioned to distinguish between the real neighbourhood and randomly generated or manufactured neighbourhoods. As a result, the GNN is motivated to identify important local patterns and dependencies.

These solutions can be applied to the architectures already presented in this analysis. It could be interesting to see if applying a pre-training step to the architecture could help them to perform better or even decrease the number of invoices needed to train the model.

Additionally, any expansions brought to the dataset could be useful. Probably one of the most important improvements could be the addition of more entities, for example regarding the price of each item or their description. This could limit both the presence of False Positive for the class *total_due* and also would help to balance the dataset.

Bibliography

[1] *Invoice Dataset*. URL: https://github.com/FedericoPes/InformationExtraction-over-Invoices (cit. on pp. 5, 57).

[2] Daniel Esser, Daniel Schuster, Klemens Muthmann, Michael Berger, and Alexander Schill. «Automatic indexing of scanned documents: a layout-based approach». In: *Document Recognition and Retrieval XIX*. Ed. by Christian Viard-Gaudin and Richard Zanibbi. Vol. 8297. International Society for Optics and Photonics. SPIE, 2012, 82970H. DOI: 10.1117/12.908542. URL: https://doi.org/10.1117/12.908542 (cit. on p. 8).

[3] Zhiheng Huang, Wei Xu, and Kai Yu. «End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF». In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics. 2015, pp. 647–657 (cit. on p. 8).

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805 (cit. on pp. 8, 19).

[5] Anoop R. Katti, Christian Reisswig, Cordula Guder, Sebastian Brarda, Steffen Bickel, Johannes Höhne, and Jean Baptiste Faddoul. «Chargrid: Towards Understanding 2D Documents». In: *CoRR* abs/1809.08799 (2018). arXiv: 1809.08799. URL: http://arxiv.org/abs/1809.08799 (cit. on p. 9).

[6] Timo I. Denk and Christian Reisswig. «BERTgrid: Contextualized Embedding for 2D Document Representation and Understanding».

In: *CoRR* abs/1909.04948 (2019). arXiv: 1909.04948. URL: http://arxiv.org/abs/1909.04948 (cit. on p. 9).

[7] Xiaojing Liu, Feiyu Gao, Qiong Zhang, and Huasha Zhao. «Graph Convolution for Multimodal Information Extraction from Visually Rich Documents». In: *CoRR* abs/1903.11279 (2019). arXiv: 1903.11279. URL: http://arxiv.org/abs/1903.11279 (cit. on pp. 10, 11).

[8] Devashish Lohani, Abdel Belaïd, and Yolande Belaïd. «An Invoice Reading System Using a Graph Convolutional Network». In: *ACCV Workshops*. 2018 (cit. on pp. 10, 11).

[9] Benjamin Heinzerling and Michael Strube. «BPEmb: tokenization-free pre-trained subword embeddings in 275 languages». In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Association for Computational Linguistics. 2019, pp. 328–334 (cit. on p. 10).

[10] Rico Sennrich, Barry Haddow, and Alexandra Birch. «Neural Machine Translation of Rare Words with Subword Units». In: *CoRR* abs/1508.07909 (2015). arXiv: 1508.07909. URL: http://arxiv.org/abs/1508.07909 (cit. on pp. 10, 20, 21).

[11] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. «Enriching Word Vectors with Subword Information». In: *CoRR* abs/1607.04606 (2016). arXiv: 1607.04606. URL: http://arxiv.org/abs/1607.04606 (cit. on p. 15).

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. «Attention Is All You Need». In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762 (cit. on pp. 17, 18).

[13] Wentao Ma, Yiming Cui, Chenglei Si, Ting Liu, Shijin Wang, and Guoping Hu. «CharBERT: Character-aware Pre-trained Language Model». In: *CoRR* abs/2011.01513 (2020). arXiv: 2011.01513. URL: https://arxiv.org/abs/2011.01513 (cit. on pp. 21, 22, 24, 53).

[14] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. «Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation». In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078 (cit. on p. 23).

[15] Marco Gori, Gabriele Monfardini, and Franco Scarselli. «Neural network for graphs: A contextual hierarchical approach». In: *IEEE Transactions on Neural Networks* 16.6 (2005), pp. 1452–1465 (cit. on p. 26).

[16] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. «Graph neural networks». In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80 (cit. on p. 26).

[17] Thomas N. Kipf and Max Welling. «Semi-Supervised Classification with Graph Convolutional Networks». In: *CoRR* abs/1609.02907 (2016). arXiv: 1609.02907. URL: http://arxiv.org/abs/1609.02907 (cit. on pp. 27, 43).

[18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML] (cit. on p. 28).

[19] Shaked Brody, Uri Alon, and Eran Yahav. *How Attentive are Graph Attention Networks?* 2022. arXiv: 2105.14491 [cs.LG] (cit. on pp. 29, 43).

[20] William L. Hamilton, Rex Ying, and Jure Leskovec. «Inductive Representation Learning on Large Graphs». In: *CoRR* abs/1706.02216 (2017). arXiv: 1706.02216. URL: http://arxiv.org/abs/1706.02216 (cit. on pp. 30, 43).

[21] Adam W Harley, Alex Ufkes, and Konstantinos G Derpanis. «Evaluation of Deep Convolutional Nets for Document Image Classification and Retrieval». In: *International Conference on Document Analysis and Recognition (ICDAR)*. 2015 (cit. on p. 33).

[22] Muzamil Hussain Syed and Sun-Tae Chung. «MenuNER: Domain-Adapted BERT Based NER Approach for a Domain with Limited Dataset and Its Application to Food Menu Domain». In: *Applied Sciences* (2021) (cit. on p. 58).