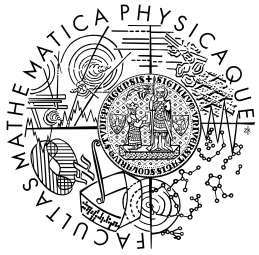


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

DOCTORAL THESIS

Martin Holeček

Information extraction and document understanding

Department of Numerical Mathematics

Supervisor of the doctoral thesis: prof. Ing. František Maršík, DrSc.
Study programme: Mathematics
Study branch: Computational mathematics

Prague 2021

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

This work is dedicated to many subjects from different points of view - my family, friends and my teachers have all created an incredible environment in which I was able to focus on the wonders of science. Ultimately the content of this work is dedicated to creating a better working environment for others.

Title: Information extraction and document understanding

Author: Martin Holeček

Department: Department of Numerical Mathematics

Supervisor: prof. Ing. František Maršík, DrSc., Institute Of Thermomechanics, Czech Academy of Sciences

Abstract: The automation of document processing is gaining recent attention due to the great potential to reduce manual work through improved methods and hardware. In this area, neural networks have been applied before – even though they have been trained only on relatively small datasets with hundreds of documents so far. To successfully explore deep learning techniques and improve the information extraction results, a dataset with more than twenty-five thousand documents (pro forma invoices, invoices and debit note documents) has been compiled, anonymized and is published as a complement of this work. In the first part of the research, we will examine the documents from the point of view of table detection, present a survey on table detection methods and ultimately rephrase the table detection as a text box labelling problem to optimize micro F_1 score of per-word classification. We will show that we can extract specific information from structurally different tables or table-like structures with one trainable model that features a comprehensive representation of a page using graph over word-boxes, positional embeddings and trainable textual features. The first part is concluded with a novel neural network model that beats multiple baselines and achieves strong, practical results on the presented dataset. Analysis of the model's performance is presented and verified, that convolutions, graph convolutions and self-attention layers can work together and exploit all the information present in a structured document. To validate the importance of the dataset and the next steps, an expert-knowledge based generative model is briefly explored. To take the fully trainable method one step further, we will ultimately design and examine various approaches to using siamese networks, concepts of similarity, one-shot learning and context/memory awareness. The results verify the hypothesis that trainable access to a similar (yet still different) page together with its already known target information improves the information extraction results. Furthermore, the experiments confirm that all proposed architecture parts (siamese networks, employing class information, query-answer attention module and skip connections to a similar page) are all required to beat the previous results. The best model inspired by a query answer architectures improves the results of the first part and achieves state-of-the-art results of 92.90 micro F_1 score. Qualitative analysis is provided to verify that the new model performs better for all target classes. Additionally, multiple structural observations about the causes of the underperformance of some architectures are revealed. All the source codes, parameters and implementation details are published together with the dataset in the hope to push the research boundaries since all the techniques used in this work are not problem-specific and can be generalized for other tasks and contexts.

Keywords: one-shot learning information extraction siamese networks similarity table detection

Contents

Part I

Introduction, overview and commentary

1. Introduction

This text presents original research on a unique task that connects the fields of natural language processing and computer vision on a dataset that allows the use of deep learning methods.

The dissertation is split into this introductory overview and three more parts, where each one of them presents a research of relevant and inspirative studies and an important step towards the final state-of-the-art results.

Parts II and IV both contain a full copy of previously published articles Holecek et al. [2019] and Holeček [2021], the intermediate part III contains previously unpublished research in the form of a technical report (that assumes prior knowledge of the first article).

The published full source codes (approximately 12 000 lines of python code) and datasets (of more than 25 000 anonymized documents) Holecek [2020, 2019] are an important complement of this research. To understand the research, it is not obligatory to go through the codes (or explore the dataset) though, but for a precise and very high level of detail, and/or reproducing the results and implementation details, the reader is kindly referred to the source codes and the accompanying comments.

Ultimately, in the appendix, there are copies of three additional past texts (Holecek [2018d,a,b]) from the author to provide some more connected information about the research, but since they do not present any significant contributions, they are omitted from the main text.

1.1 Background

The dissertation is motivated by both theoretical and practical reasons based on a unique opportunity, that have appeared recently. To fully understand the opportunity we need to take into account all the factors included since they are all equally important and have enabled the exploration:

- The development of advanced methods that were able to resume the scientific advance of artificial intelligence (from the previous so-called "AI winter" Newquist [2018]).
- Hardware improvements – driven by consumer demand for high-quality graphics – also enabled linear algebra and scientific computation parallelization in novel GPU architectures.
- The pure fact that best practice in the software engineering industry requires storing lots of historical data ultimately gave rise to access to bigger datasets.
- The rising opportunity of automatization of more and more tasks, fueled by the need for efficient work by both big companies and individuals.

The resulting steep rise of so-called "deep learning" Dargan et al. [2019] was spun up by the discovery, that methods trained on sufficiently big datasets surpass all previous expert models and even carefully tinkered features.

Apart from providing this brief background, it is impossible to provide any valuable and meaningful survey on the topic and history of machine learning and deep learning in the scope of this work - considering the field's sheer magnitude and the number of existing surveys. (Moreover, this field is taking over all other research fields where previously the crafted invariants and approximate methods dominated, such as computer vision - O' Mahony [2020])

Therefore we will resort to citing just all the relevant research in the respective parts and otherwise referencing here only Dargan et al. [2019], Newquist [2018] and (for existing industry best-practices) Burkov [2020].

Specifically - in the problem of information extraction - this work was only able to grow thanks to the access to datasets of unmatched size. The huge datasets allowed using deep learning in information extraction with the foreseen opportunity for automatization of business document processing. The task of invoice processing was, for example, previously solved only by hand (either on a per-document basis or per-template basis).

2. Overview of the articles

The research and implementation try to follow the best practices of a long explorative, research and development path, that all successful studies, researches and deep learning models have done in other fields. Document processing and information extraction Cowie and Lehnert [1996] is most interesting from a theoretical perspective since it combines both the fields of computer vision and natural language processing.

The dissertation features two published articles (Holecek et al. [2019] and Holeček [2021]) and one intermediate previously unpublished research, that helps motivate the problem.

The following chapter contains a short overview of all the following parts of the dissertation. To keep this overview brief, all in-depth explanations, exact definitions and citations are done in the parts themselves and not in this overview.

2.1 Overview of "Table understanding in structured documents"

In the first article Holecek et al. [2019] (in II) the research is carried out on a smaller subset of the whole available dataset and focuses on establishing a well defined structural approach to desired information extraction and specific table detection.

Working with just a subset of the dataset was decided for practical reasons - to have the annotations done faster and to quickly iterate in case of need. The focus on table detection methods was based on observing how humans approach the problem.

In detail - the importance of the first article is as follows. If the system would be successful in just finding any table but could not be trained to detect just one specific table type in a presented document and - at the same time - extract just some information from other tables, its practical usage would be useless. The reason is that in business documents and invoices especially, everything is structured in (sometimes even nested) tables and so older expert table detection based methods are useless for the task at hand. The older methods would either find all tables or just get confused and fail.

The metric, losses, structure of inputs and outputs and all other factors were held as close to the best practices in literature (see section 2 of II).

The experimental design was motivated by a study on how humans try to decipher business documents. Every person introduced to the task looks at the document as a whole and then tries to read it as a text document. They discover, that it can be done only in some parts of some documents and elsewhere they need to look at the local clusters of words.

These three steps translate (respectively) to global self-attention module, convolution over sequence and graph convolution.

To feed as much information as is possible to the said layers, the designed neural network was given all possible features that are present in the document - all textual information up to individual characters in words; positions of all

the words including features from geometrical algorithms; graph of neighbouring words; whole image of a page and crops of the image around each word. Moreover, features from named entity recognition are used, since business documents contain a lot of named entities and we desired the network to learn them without a need for specific tokenization techniques.

The experimental results have shown that this design was a success. Moreover, it was verified, that:

- There was no redundancy - all the modules were needed for achieving a higher score.
- The tasks of "specific whole table detection" and "extracting just a single piece of information from other parts of the document" did boost each other.
- The model was able to generalize to completely unknown layouts.
- The model was able to scale to the whole dataset ($\sim 10\times$ bigger than the initial small variant.)
- All the inputs are of importance as the score drops if each of them is omitted.

2.1.1 Overview of an intermediate technical report on generated documents based on expert knowledge

The second part III presents small research in the form of a technical report. This small research was carried out but unpublished previously due to the absence of positive results and significant findings. But since it helps to give another insight tino the problem and dataset, it is included here.

Its motivation is as follows. Given the domain knowledge of the business documents and the annotation process difficulty, a series of experiments was carried out to see if automatically created artificial documents would be of any practical use. Or if they at least could speed up the process of training the neural network.

A stochastic generator of a document was defined and implemented that would allow to create an artificial dataset just by sampling from the generator. Both a new simple convolutional architecture and the model from the first part II were trained and tested on multiple settings with the artificial dataset. The experimental results have shown, that:

- The original task cannot be easily approximated by a rule-based stochastic system. This finding has thus verified the value of a curated dataset.
- There are deeper relations in the dataset, that the model (from the first part) successfully exploits.
- The simple convolutional baseline is better on the simple simulated data, possibly because the model (from the first part) is tuned for harder problems.

No results from this experiment are of any practical use, but it does mark the next direction to not focus on generative models and instead to focus on using as much as possible from the real dataset. That direction is realized by the last part by exploring single-shot learning techniques.

2.1.2 Overview of "Learning from similarity and information extraction from structured documents"

The last part IV focuses on advanced pieces of deep learning from the general research areas of "single shot learning" and "similarity". The exploration of these highly advanced and resource-hungry methods is well motivated by the previous parts:

- All structural information from one document's page was used in the first article.
- At the same time, the model researched in the first article is shown to scale to the whole dataset.
- The second part tells us, that the dataset contains non-trivial (in the meaning of not easily modelled) relationships and hidden rules.

Therefore the central research question of the last part is: "Does there exist an architecture that uses the concept of similarity and can reduce the error even more? If so, which one?"

Since we would like to measure the contribution of the similarity concept alone over the score known from the first article, everything in the experimental design is kept as close as possible to the first article's setting, except to the table detection target. In this part, we would focus only on single word classification, not table detection (also because it would present some difficulties to annotate the tables of the whole big dataset in time).

Now the main difficulty lies in using the methods of similarity in a brute-force approach would take an incredible amount of resources in terms of computation time, complexity and memory.

A typical page contains 300 words, running a system with the complexity of $O(N^2)$ (for each word against all other words in a dataset of tens of thousands of documents) would mean running a computation for the magnitude of approximately 10^8 inputs (per each epoch).

To tackle this problem in a better way than a naive random sub-sampling, we introduce a requirement for all the presented methods to use only 1 page from a database of known pages and to process all word-boxes at once.

The one page is found by a fixed nearest neighbour search in an embedding space of all the pages (that is manually verified to indeed group visually similar pages).

Based on these specifications, 3 new different architectures featuring siamese networks and 4 more baselines related to similarity approaches are designed from scratch in a novel approach.

The model from the first part is used as a siamese network at the input part of all the new architectures. The models are all trained from scratch and no transfer learning is used.

The following architectures are designed:

- Triplet loss variants - extend the canonical triplet loss for computing the loss for all word-boxes from both the known and reference page at once, thus saving the computational time.

- Pairwise classification - this architecture tries to directly predict the outcome without trying to directly manage the embeddings in a triplet-loss induced space.
- Query answer architecture - this architecture is a different approach to similarity and tries to utilize a self-attention layer to answer a question about the input unknown page using the known page.

The experimental results of the baselines show, that:

- The documents in our dataset are different enough and so the task is not trivial
- Just a similarity search alone and a templating mechanic is not enough, even if we would fine-tune the nearest neighbour search.
- A simple linear model performs poorly, which motivates the usage of more advanced ones.

The experimental results show, that the query answer architecture performs the best and beats the results from the first part. By ablation analysis, we verify, that all the parts of the query answer architecture are important. Therefore the most probable reason for the triplet loss and pairwise classification under-performance is the absence of direct connections (in the means of information flow in neural networks) to the unknown and known page. Finally, qualitative analysis reveals, that the query answer architecture performs better over the results from the first article in all the target classes and significantly improves the scores of the previously most under-performing classes.

Ultimately the computational optimality was also reached as the whole training takes just 1 consumer-grade GPU and 4 days of training time with just one CPU process.

Part II

Table understanding in structured documents

Table understanding in structured documents

Martin Holeček*†, Antonín Hoskovec†, Petr Baudiš†, Pavel Klinger†
*Faculty of Mathematics and Physics, Charles University, Department of Numerical Mathematics
†Rossum

***Abstract*—Table detection and extraction has been studied in the context of documents like reports, where tables are clearly outlined and stand out from the document structure visually. We study this topic in a rather more challenging domain of layout-heavy business documents, particularly invoices. Invoices present the novel challenges of tables being often without outlines - either in the form of borders or surrounding text flow - with ragged columns and widely varying data content. We will also show, that we can extract specific information from structurally different tables or table-like structures with one model. We present a comprehensive representation of a page using graph over word boxes, positional embeddings, trainable textual features and rephrase the table detection as a text box labeling problem. We will work on our newly presented dataset of pro forma invoices, invoices and debit note documents using this representation and propose multiple baselines to solve this labeling problem. We then propose a novel neural network model that achieves strong, practical results on the presented dataset and analyze the model performance and effects of graph convolutions and self-attention in detail.**

***Index Terms*—table detection; neural networks; invoices; graph convolution; attention**

I. Introduction

Table detection and table extraction problems were already introduced in a competition ICDAR 2013, where the goal was to detect tables and extract cell structures from a dataset of mostly scientific documents [1]. Table can be defined as a set of content cells organized in a self-describing manner into such a structure, that groups cells into rows and columns. This was reflected in the metric defined in the competition, that scores tables based on the relations successfully extracted. Similarly, structured documents do have a self-describing structure, that often looks table-like.

We have decided to investigate the problem on business documents such as invoices, pro forma invoices and debit notes (referred for simplicity as invoices or invoice-type documents later in this text), where the aim is different. Namely - even table detection needs to be thought of in the context of document understanding, because invoices are inherently documents with textual information structured into more tables. Graphical borders and edges are sometimes present, however, they cannot be used for detection, because there is no general layout and very often there are no borders at all. Another obstacle for traditional methods is the fact, that the data can span over multiple lines of text which holds true also for the table cells.

Moreover, we require our model to 'understand' the document in a way that it could classify tables and tabular structures based on their content. In practice the goal is to detect the whole table with the so-called 'line-items' (detailed items of the total amount to pay) and, at the same time, extract only a specific information from the other tables (to find a 'field'). Simply said, not every table inside an invoice should be detected and reported as whole (see example invoice on figure 1 on page 3). Usually in commercial applications this problem is tackled using a layout system that detects the layout and extracts the table (or a field) from a position where it usually happens to be; or employing another classification module, which selects the right table from several proposals. That increases the number of modules in the architecture and requires manual layout setups, while our goal is to have a trainable system that could leverage the commonalities present in the data without ongoing human support. To verify that, we will ensure that proper generalization of models predictions is evaluated on new layouts.

II. Previous Work

The plethora of methods that have been previously used for the task is hard to summarize or compare since all the algorithms have been used/evaluated on different datasets and each have their strengths, weak spots and quirks. However, we found none of them well suited for working with structured documents (like invoices), since they in general have no fixed layout, language, captions, delimiters, fonts... For example, invoices vary in countries, companies and departments and changes in time. In order to retrieve any information from a structured document, you need to understand it.

In literature there are examples of table detection using heuristics [2], using layouts [3], regular expressions [4], or leveraging the presence of lines in tables [5], [6], [7], [8], or using clustering [9]. A great survey can be found in [10].

Tables were searched for also in HTML [11], [12], free text [13] or scientific articles with a method based on matching captions with content [14].

Machine learning methods and deep neural networks were also employed in several papers. The work [15] aims at scientific documents using fine tailored methods stacked atop each other. Reference [16] uses Fast R-CNN architecture with a novel idea of Euclidean distance feature to detect tables (which was compared to Tesseract). Reference [17] also uses (pretrained) Fast R-CNN and FCN semantic segmentation model for table extraction problem. In [8] work has been done on detection problem bottom up using the Hough transform, and extraction was solved with Markov networks and features from the cell positions. Reference [18] uses convolutions over

the number (and sizes) of spaces in a line. A deep CNN approach was being investigated in [19], which combined CNNs for detecting row and columns, dilated convolutions, CRFs and saliency maps, they have also developed a webcrawler to extend their dataset. We tried and failed to get working results using the YOLO architecture [20] with textual datasets. (We have experimented with YOLO because some works aimed at table detection do use the family of R-CNNs, Fast R-CNNs and Mask R-CNNs, that preceded the development of YOLO.)

For document understanding, a graph representation of a document was examined in [21], [22], finding similar documents and reusing their goldstandards was done in [23].

III. Methodology

We would like to define our target as creating a model for tabular or structured data understanding with relevant information detection and classification. The basic unit of information will be a word in a document's page with its placement and possibly other features such as style (see PDF format text data organization [24] for example). In this text, we will be calling them simply as *wordboxes*.

With table understanding we mean a joint task of line-item table detection and information extraction from other tables. The information to be extracted is defined by the document use-case or semantics, for line-item table it is the whole table ('table detection' task as defined in [1]), while for other structures it is just a specific infomation ('information extraction' task). No other constraints apply, i.e. the data can span over multiple lines. So the model is required to understand a type of table internally and we hope, that the two tasks will boost the learning process for each other.

With line-item table detection method, we will understand a model, that could classify each wordbox in a document as being a part of a line-item content or not (which basically identifies the table itself, because all line-items tables happen to be well separated, so no instance segmentation is needed). Same classification approach will be used for other classes representing other types of content. The classes are acquired from expert annotations and, as it turns out, we are dealing with a multilabel problem, i.e. 35 classes in total, examples being the total amount or recipient address. Also, not every document contains instances of every class.

A. Metrics and evaluation

We will observe the scores at validation and test splits, the test being composed not only of different data, but also of different layouts and invoice types, thus allowing us to observe the system scalability. The scores are:

- F_1 scores on line-item wordbox classification averaged from both positive classes and negative classes.
  At [1] a content oriented metric was defined for table detection on character level - each character being either in the table or out of the table. For us the basic unit is a wordbox, hence we will define our metric similarily to be the F_1 score of table body wordbox class classification.
- For other classes we will be looking at micro F_1 scores (only from positive classes, because the counts of positive samples are outnumbered by the negative samples - in total, the dataset contains $1.2\,\%$ positive classes).

We chose micro metric aggregation rule, because it gives higher importance to bigger documents (in the number of wordboxes) which we consider being more difficult for both human and machine.

We present our research as a novel approach, because referenced papers or commercial solutions cannot be customized to fit our aim. So we will compare only against baseline logistic regression over the model features.

B. The data and their acquisition process

The data were acquired as a result of work of annotation and review teams together with automated preprocessing and error-finding algorithms, that reported errors in nearly $3\,\%$ of the annotation labels. Classes were annotated in our annotation apps by drawing a rectangle over the area with the target text. Manual inspection has revealed, that the annotations can erroneously overlap portions of neighbouring words, so for ground truth generation we have decided to select only the wordboxes that are being overlapped by the annotation rectangle by more than $20\,\%$ of their area.

Datasets: We have a dataset with 3554 PDF invoice files consisting of 4848 pages in total. The documents are of various vendors, layouts and languages, annotated with line-item table header and table body together with other structural information. And we also have a bigger dataset of 25071 PDF files of 35880 pages with just structural information without line-items (datasets are noted as 'small' and 'big' in the results).

The documents are standard PDF files, not scanned documents or documents captured by a digital camera. This decision will not impact the robustness of our model - given a process to extract bounding boxes and text, we can use our method in a straightforward manner.

Validation split is chosen to be 1/4 the size. The validation set measures adaptation, because it could contain similar invoice types from similar vendors. So in addition, we have created another testing set of 83 documents, that have different invoice layouts and types to those in the training set to measure generalization.

Since this newly compiled dataset was never explored and made accessible before, we have published an anonymized version of the small dataset, that contains only the positions and sizes of wordboxes and annotations, no picture information and no readable text information – only a subset of some textual features. The dataset is to be found at https://github.com/rossumai/flying-rectangles

C. Our approach

We want to operate based on the principle of reflecting the structure of the data in the model's architecture, as Machine learning algorithms tend to perform better that way.

What will be the structured information at the input? The number of wordboxes per page can vary and so we have decided to perceive the input as an ordered sequence (see below).

In addition we will teach the network to not only detect line-item table in general, but also to detect a header in the table, because that could provide a meaningful information - the headers are always different from the contents.

The features of each wordbox are:

- Geometrical:
  - By geometrical algorithms we can construct a neighbourhood graph over the boxes, which can then be used by a graph CNN if we bound the number of neighbours on each edge of the box by a constant. Neighbours are generated for each wordbox (W) as follows - every other box is assigned to an edge of W, that has it in its field of view (being fair $90°$), then the closest (center to center Euclidian distance) n neighbours are chosen for that edge. For example with $n = 1$ see figure 1. The relation does not need to be symmetrical, but when higher number of closest neighbours will be used, the sets would have bigger overlap.
  - We can define a 'reading order of wordboxes'. In particular, based on the idea that if two boxes do overlap in a projection to y axis by more than a given threshold, set to $50\,\%$ in our experiments, they should be regarded to be in the same line for a human reader. This not only defines an order of the boxes in which they will be given as sequence to the network, but also assigns a line number and order-in-line number to each box. To get more information, we can run this algorithm again on a $90°$ rotated version of the document. These integers are then subject to a positional embedding. Note, that the exact ordering/reading direction (left to right and top to bottom or vice versa) should not matter in the neural network design, thus giving us the freedom to process any language.
  - Each box has 4 normalized coordinates (left, top, right, bottom) that should be presented to the network also by positional embedding.
- Textual:
  - Each word can be presented using any fixed size representation, in our case we will use tailored features common in other NLP tasks (e.g. authorship attribution [25], named entity recognition [26], and sentiment analysis [27]). The features per wordbox are the counts of all characters, the counts of first two and last two characters, length of a word, number of uppercase and lowercase letters, number of text characters and number of digits. And finally, if the word is in fact a number, then the number scaled and cropped against different scales, zeroes for other text. The reason behind these features is that in an invoice there would be a larger number of named entities, ids and numbers, which are not easily embedded.
  - Trainable word features are employed as well, using convolutional architecture over sequence of one hot encoded, deaccented, lowercase characters (only alphabet, numeric characters and special characters " ,.-+:/%?$£€#()&'", all others are discarded).
- Image features:
  - Each wordbox has its corresponding crop in the original PDF file, where it is rendered using some font settings and also background, which could be crucial to line-item table (or header) detection, if it contains lines, for example, or different background color or gradient. So the network receives a crop from the original image, offsetted outwards to be bigger than the text area to see also the surroundings.

Each presented feature can be augmented, we have decided to do a random $1\,\%$ percent perturbation on coordinates and textual features representation.

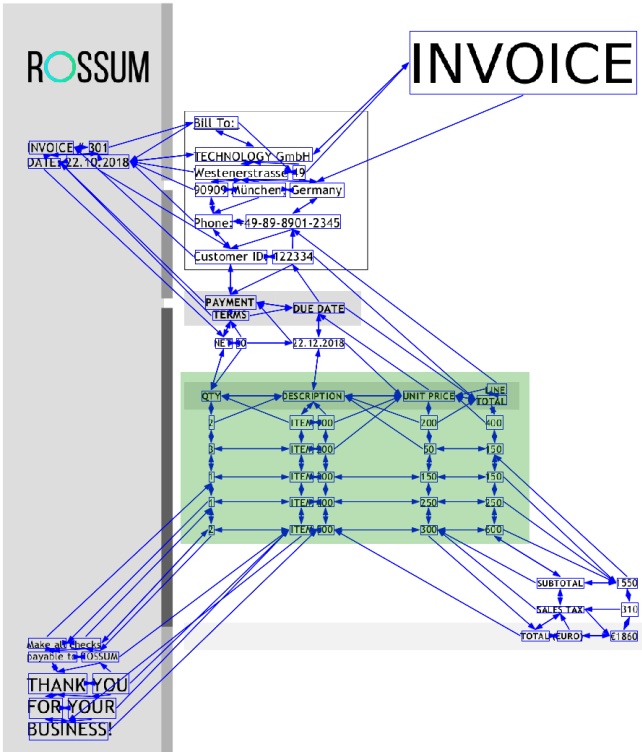


Fig. 1. Sample invoice with edges defining neighbourhood wordboxes. Only the closest neighbour is connected for each wordbox. Green area is the line-item table. Note that above it lies a smaller table of payment terms and due date, from which only some information should be extracted and the table should not be reported. This invoice is artificially created for presentation and does not represent the invoices in our dataset.

D. The architecture

As can be seen in Figure 2 on the following page and as we have stated before, the model uses 5 inputs - downsampled picture of the whole document (620×877), grayscaled; features

of the wordboxes, including their boundingbox coordinates; on-hot characters with 40 one-hot encoded characters per each wordbox; neighbour ids - integers that define the neighbouring wordboxes on each side of the wordbox; and finally integer positions of each field defined by the geometrical ordering.

The positions are embedded by positional embeddings (defined and used in [28], [29], we use embedding size equal to 4 dimensions for sin and 4 for cos, with divisor constant being 10000) and then concatenated with other field features.

The picture is reduced by classical stacked convolution and maxpooling approach and then from its inner representation, field coordinates (left, top, right, bottom) are used to get a crop of a slightly bigger area (using morphological dilation) which is then appended to the field. Finally we have decided to give the model a grasp of the image as whole - a connection to the whole image flattened and then processed to 32 float features, which are also appended to each field's features.

Before attention, dense, or graph convolution layers are used, all the input features are concatenated.

Our implementation of the graph convolution mechanism gathers features from the neighbouring wordboxes, concatenates them and feeds into a Dense layer. To note, our graph has a regularity that allows us to simplify the graph convolution - there does exist an upper bound on the number of edges for each node, so we do not need to use any general form graph convolutions as in [30], [31].

We have also employed a convolution layer over the ordering dimension (called *convolution over sequence* later in this text).

The rest of the network handles images and crops. The final output branch has an attention transformer module (from [28]) to be able to compare pairwise all the fields in hope that denser and regular areas (of texts in a table grid) can be detected better. Our attention transformer unit does not use causality, nor query masking and has 64 units and 8 heads.

Finally, the output is a multilabel problem, so sigmoidal layers are deployed together with binary crossentropy as the loss function.

The optimizer was chosen to be Adam. Model selected in each experimental run was always the one that performed best on the validation set (of the *small* dataset) in terms of loss, while the *patience* constant was 10 epochs. Batched data were padded by zeros per batch (with zero sample weights). Class weights in our multi-task classification problem were chosen based on positive class occurrences. The network has $867k$ trainable parameters in total.

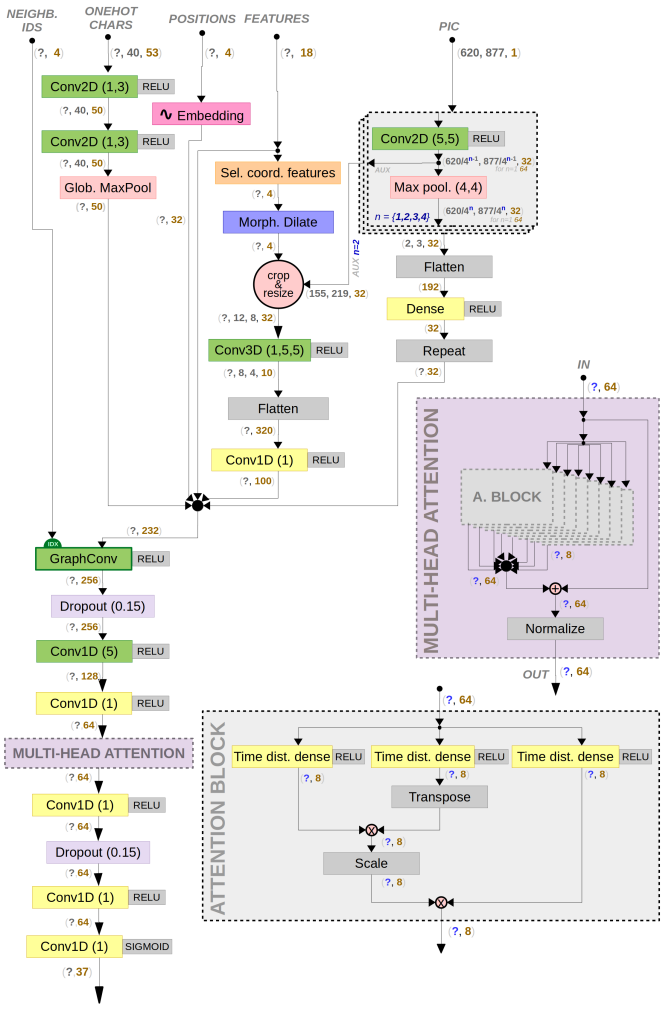


Fig. 2. The model architecture. All features are concatenated together before the self attention mechanism and final layers. The cropping of the picture, embeddings and graph convolution all happen inside the network. Note that $Conv1D(1)$ can be also called a time distributed dense layer.

IV. Experiments

The approach was tested on different data settings and different architectures. There are 4 groups of experiments:

1) Comparing logistic regression baseline against the neural network.
   To note, logistic regression baselines use all the inputs except the picture and trainable word embeddings. To inspect the importance of neighbouring boxes, we have compared the baseline without neighbours and the baseline with included information about one or more neighbours at each side (if present).
2) The importance and effect of each block of layers and each input and other parameters.
   The choice of modules to test was 'convolution with dropout after attention' to test the dropout layer, 'convolution over sequence' for the importance of input ordering and attention. Experiments dropping the graph convolution were done in variation of neighbours. Experiments on anonymized dataset fall also into this category. We have also tested the focal loss function [32], note that we do not vary final activations in our experiments here and use only sigmoidal, because they had best performance in earlier development process.
3) Specialization on a task where only line-items were

classified and specialization on a task with all but line-items.

4) Evaluating the model's adaptation performance on the big dataset (without line-items).

We will not be optimizing the number of neurons in the layers. The training was done on a single GPU and ran approximately in 23 epochs for 5 hours per experiment on small dataset.

A. Results

Table I on the next page summarizes experiments comparing the model against the logistic regression baseline, both with varying number of neighbours (more than 2 not shown, because the results were not improving with the number of neighbours). The logistic regression baselines did improve with more neighbours, but failed to generalize. We can notice the big difference between line-item table detection and other classes coming from a possible observation that sometimes the table is the biggest one. The results also do reflect the nature of a specialized structured document, which invoices indeed are - to classify all the structured information is not easy for a person not working with invoices.

On the other hand, the optimal number of neighbours for the final architecture was 1, but we can notice, that 2 neighbours do help line-item table body detections. We have designed the algorithm with more than one neighbour in mind (with a single neighbour, the relation is not symmetrical), so other positional features are possibly being exploited more efficiently.

Table II on the following page shows, that the multihead attention module helps with generalization to unseen layouts, omitting the module makes the network prioritize adaptation on already seen layouts. Also without attention, the number of training epochs was twice (27) as much as with attention (13). Focal loss, prioritizing rare classes, does help line-item header detection, but is a cause for the decrease of the nonline-item score micro metric, as rare classes contribute less.

The importance of the convolutional layer over the sequence might come from our initial guess that this would give more importance to beginnings and endings of lines of words.

Table III on the next page compares different inputs and dataset choices. Although the architecture was optimized on the small dataset, the results imply that the model has the capacity to adapt and generalize also on bigger datasets. Looking at the anonymized version of datasets, without some inputs, it can be concluded that the network can learn to detect tight areas of evenly spaced words, being the line-item table. Also even base text features help the model generalize well. Overall the score on anonymized dataset means that the positional information is passed correctly and embedded in a right way for the network.

In table IV on the following page there can be seen that the tasks of finding line-items and other structural information do boost each other, with one exception being the header detection - it does help adaptation, but when omitted, the generalization score is higher.

The architecture provided on Figure 2 on the previous page is the '**complete model**', that uses binary crossentropy, all inputs and all modules and a single neighbour at each side of each box. Its generalization performance on unseen invoice types was 93 % on detecting line-items and 66 % for other classes (87 % on similar layouts). To verify what the line-item detection scores mean in practice, we have run the prediction on the sample invoices (Figure 1 on page 3), where our algorithm correctly detected the line-item table up to 2 false positive words, which are easily filtered out (heuristic filtering results are not reported).

V. Conclusions

We have found a fully trainable method for table detection and content understanding in structured documents, that is able to detect a specific line-item table and extract only some information from other tables even in the presence of imbalanced classes and multiple layouts, languages and invoice types. Anonymized version of our dataset was published, as no similar dataset has been publicly available to date.

Trying to detect line-item headers in a single model did lead the model to underperform, with a hint to use focal loss for such task. Also, we have discovered, that attention module was important to generalization for new invoice types, while using only close neighbours did lead to better adaptation on already seen layouts.

The system's ability to correctly scale to completely new invoice types is successfully verified for the line-item table detection task at 93% and measured to be 66% on 35 'other' classes.

Future work can include line-item table extractions, architecture and hyperparameter tuning for bigger datasets, experiments with the usage of different text features or embeddings and image augmentations. It could be also measured how many annotations are needed for the 'other' classes to adapt onto new invoice types.

Acknowlegment

The work was supported by the grant SVV-2017-260455. We would also like to thank to the annotation team and the rest of the research team.

References

[1] M. Göbel, T. Hassan, E. Oro, and G. Orsi, "Icdar 2013 table competition," in *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*. IEEE, 2013, pp. 1449–1453.

[2] A. Jahan Mac and R. G. Ragel, "Locating Tables in Scanned Documents for Reconstructing and Republishing (ICIAfS14)," *arXiv e-prints*, p. arXiv:1412.7689, Dec. 2014.

[3] T. Dhiran and R. Sharma, "Table detection and extraction from image document," *International Journal of Computer & Organization Trends*, vol. 3, no. 7, pp. 275–278, 2013.

[4] S. Mandal, S. Chowdhury, A. K. Das, and B. Chanda, "A very efficient table detection system from document images." in *ICVGIP*, 2004, pp. 411–416.

[5] B. Gatos, D. Danatsas, I. Pratikakis, and S. J. Perantonis, "Automatic table detection in document images," in *International Conference on Pattern Recognition and Image Analysis*. Springer, 2005, pp. 609–618.

[6] A. Gupta, D. Tiwari, T. Khurana, and S. Das, *Table Detection and Metadata Extraction in Document Images: Proceedings of ICSICCS-2018*, 01 2019, pp. 361–372.

[7] Y. Liu, "Tableseer: automatic table extraction, search, and understanding," 2009.

TABLE I

Experiments against the baseline	Adaptation			Generalization		
	line-items		others	line-items		others
	body F_1	header F_1	micro F_1	body F_1	header F_1	micro F_1
complete model (without neighbours)	0.9666	0.9969	0.8687	0.9242	0.9876	0.6609
complete model (1 neighbour)	0.9738	**0.9967**	**0.8790**	0.9389	**0.9864**	**0.6650**
complete model (with 2 neighbours)	**0.9762**	0.9963	0.8749	**0.9408**	0.9860	0.6629
logistic regression without neighbours	0.7594	0.9477	0.0004	0.7560	0.9362	0.0000
logistic regression with 1 neighbour	0.8664	0.9663	0.1482	0.8071	0.9461	0.0327
logistic regression with 2 neighbours	0.8939	0.9724	0.2276	0.8284	0.9493	0.0525

TABLE II

Experiments with ablation	Adaptation			Generalization		
	line-items		others	line-items		others
	body F_1	header F_1	micro F_1	body F_1	header F_1	micro F_1
complete model	0.9738	0.9967	0.8790	**0.9389**	0.9864	**0.6650**
focal loss	0.9735	**0.9969**	0.8557	0.9383	**0.9878**	0.6398
no convolution over sequence	0.9670	0.9945	0.8638	0.9101	0.9800	0.6237
no attention	**0.9780**	0.9967	**0.8806**	0.9348	0.9864	0.6487
no convolution with dropout after attention	0.9646	0.9950	0.8435	0.9168	0.9807	0.6050

TABLE III

Experiments with inputs variations	dataset	Adaptation			Generalization		
		line-items		others	line-items		others
		body F_1	header F_1	micro F_1	body F_1	header F_1	micro F_1
complete model (all inputs)	**small**	**0.9738**	**0.9967**	**0.8790**	**0.9389**	**0.9864**	**0.6650**
no text embeddings	small	0.9702	0.9921	0.7772	0.9108	0.9771	0.5118
no picture, only some text features	anonym	0.9694	0.9943	0.4518	0.9185	0.9805	0.4745
no picture, no text features	anonym	0.9588	0.9848	0.6836	0.8919	0.9549	0.2152
complete model (all inputs)	big	N/A	N/A	0.8487	N/A	N/A	N/A

TABLE IV

Experiments with training target variations	dataset	Adaptation			Generalization		
		line-items		others	line-items		others
		body F_1	header F_1	micro F_1	body F_1	header F_1	micro F_1
complete model (all outputs)	**small**	**0.9738**	**0.9967**	**0.8790**	0.9389	**0.9864**	0.6650
only line-items	small	0.9027	0.9950	N/A	0.8762	0.9766	N/A
no line-item header	small	0.9736	N/A	0.8777	**0.9394**	N/A	**0.6731**
all but line-items	small	N/A	N/A	0.8632	N/A	N/A	0.6247
complete model (other than line-items targets)	big	N/A	N/A	0.8487	N/A	N/A	N/A

[8] W. Farrukh, A. Foncubierta, A.-N. Ciubotaru, G. Jaume, C. Bejas, O. Goksel, and M. Gabrani, "Interpreting data from scanned tables," 11 2017, pp. 5–6.

[9] N. Pether and T. Macdonald, "Robust pdf table locator," 2016.

[10] N. Miloševic, "A multi-layered approach to information extraction from tables in biomedical documents," 2018.

[11] A. Tengli, Y. Yang, and N. L. Ma, "Learning table extraction from examples," in *Proceedings of the 20th international conference on Computational Linguistics*. Association for Computational Linguistics, 2004, p. 987.

[12] X. Chu, Y. He, K. Chakrabarti, and K. Ganjam, "Tegra: Table extraction by global record alignment," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: ACM, 2015, pp. 1713–1728. [Online]. Available: http://doi.acm.org/10.1145/2723372.2723725

[13] H. T. Ng, C. Y. Lim, and J. L. T. Koo, "Learning to recognize tables in free text," in *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*, ser. ACL '99. Stroudsburg, PA, USA: Association for Computational Linguistics, 1999, pp. 443–450. [Online]. Available: https://doi.org/10.3115/1034678.1034746

[14] C. A. Clark and S. K. Divvala, "Looking beyond text: Extracting figures, tables and captions from computer science papers." in *AAAI Workshop: Scholarly Big Data*, 2015.

[15] M. Fan and D. S. Kim, "Detecting Table Region in PDF Documents Using Distant Supervision," *arXiv e-prints*, p. arXiv:1506.08891, Jun. 2015.

[16] A. Gilani, S. Rukh Qasim, I. Malik, and F. Shafait, "Table detection using deep learning," 09 2017.

[17] S. Schreiber, S. Agne, I. Wolf, A. Dengel, and S. Ahmed, "Deepdesrt: Deep learning for detection and structure recognition of tables in document images," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 01, Nov 2017, pp. 1162–1167.

[18] A. C. e Silva, A. Jorge, and L. Torgo, "Automatic selection of table areas in documents for information extraction," in *Progress in Artificial Intelligence*, F. M. Pires and S. Abreu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 460–465.

[19] I. Kavasidis, S. Palazzo, C. Spampinato, C. Pino, D. Giordano, D. Giuffrida, and P. Messina, "A saliency-based convolutional neural network for table and chart detection in digitized documents." *CoRR*, vol. abs/1804.06236, 2018.

[20] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[21] V. P. d'Andecy, E. Hartmann, and M. Rusinol, "Field extraction by hybrid incremental and a-priori structural templates," in *2018 13th IAPR*

International Workshop on Document Analysis Systems (DAS), April 2018, pp. 251–256.

[22] B. Coüasnon and A. Lemaitre, *Recognition of Tables and Forms*. London: Springer London, 2014, pp. 647–677. [Online]. Available: https://doi.org/10.1007/978-0-85729-859-1_20

[23] H. Hamza, Y. Belaïd, and A. Belaïd, "Case-based reasoning for invoice analysis and recognition," in *Case-Based Reasoning Research and Development*, R. O. Weber and M. M. Richter, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 404–418.

[24] pdfparser. [Online]. Available: https://github.com/rossumai/pdfparser

[25] Z. Chen, L. Huang, W. Yang, P. Meng, and H. Miao, "More than word frequencies: Authorship attribution via natural frequency zoned word distribution analysis," *CoRR*, vol. abs/1208.3001, 2012. [Online]. Available: http://arxiv.org/abs/1208.3001

[26] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[27] A. Abbasi, H. Chen, and A. Salem, "Sentiment analysis in multiple languages: Feature selection for opinion classification in web forums," *ACM Trans. Inf. Syst.*, vol. 26, no. 3, pp. 12:1–12:34, Jun. 2008. [Online]. Available: http://doi.acm.org/10.1145/1361684.1361685

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *arXiv e-prints*, p. arXiv:1706.03762, Jun. 2017.

[29] R. Liu, J. Lehman, P. Molino, F. Petroski Such, E. Frank, A. Sergeev, and J. Yosinski, "An Intriguing Failing of Convolutional Neural Networks and the CoordConv Solution," *arXiv e-prints*, p. arXiv:1807.03247, Jul. 2018.

[30] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning Convolutional Neural Networks for Graphs," *arXiv e-prints*, p. arXiv:1605.05273, May 2016.

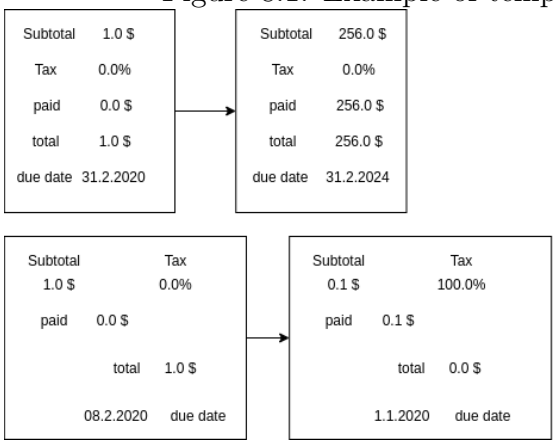
[31] A. Jean-Pierre Tixier, G. Nikolentzos, P. Meladianos, and M. Vazirgiannis, "Graph Classification with 2D Convolutional Neural Networks," *arXiv e-prints*, p. arXiv:1708.02218, Jul. 2017.

[32] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," *arXiv e-prints*, p. arXiv:1708.02002, Aug. 2017.

Part III

Expert knowledge and artificial document generation - a technical report

Figure 8.1: Example of templates in business documents.



To present more comprehensive research, it is important to try to use all the domain knowledge about the task and dataset that we have gathered during years of working with the dataset and exploring it.

This technical report is meant as an accompanying text to the experimental code, that is present here Holecek [2019].

The chosen approach greatly complements our existing research by shifting the point of view as much as it could be done. Therefore the chosen way to approach the problem is to use a generative approach (instead of classification) based on domain knowledge assumptions (as contrary to it being fully trainable).

The added value we get from this exploration is the investigation of the value of the annotation process. As stated before, to build a training set, each document was reviewed by two different expert annotators to mitigate the error (and push it under 3 %). A simplification of the process or a verification of the workload should be of research value.

Note that there are plenty of manual online resources that would help with creating any (even fake) business documents - being it invoice, receipt or anything; and they all would offer lots of templates. These resources and tools are of limited use to us since we would have zero control over the parameters of layouts, which we would desire to be stochastic.

8.1 The insight from a domain knowledge

The extracted information always consists of one or more word-boxes together with their respective classes while uninteresting (unlabeled) data are ignored (not extracted).

By reviewing a large sample of documents, we gathered the insight, that the information we want to extract is usually located:

- Near the explaining text that implicitly or explicitly defines the class
- In the same column that defines its class
- In the same usual place on a page (e.g. page numbers)

As an example of these rules - column headers can tell us that the texts in one column are amounts to be paid and the texts in another column are vendor postal addresses.

Generally speaking, every target text comes with several data items ("explaining texts") that collectively define the meaning (class) of the data we want to identify and extract.

Since the data (word-boxes) we want to extract and classify are sparsely located in the document, we add a final observation:

- We can omit most of the texts we usually see in the documents (if they are not the important explaining texts).

All these observations were verified manually and also through:

- statistical queries and
- feature and input importance methods on various working models

The former was done simply by erasing the supposed "explaining" texts and observing the influence on model predictions.

We kindly ask the reader to consider the observations more of a manual exploration, since this statement covers just the results of the analysis, but more details will not be shared here due to privacy, data protection and proprietary know-how policies.

8.1.1 The construction of the simulated documents

Based on these assumptions, how would the simulated documents be constructed? Each word-box has a specific size and is located at a specific location on the page. As texts are usually presented to the neural networks as embeddings, the simulated word-boxes will also be assigned a sampled vector from a given distribution.

As a next step, we need to define a rule for the observations mentioned above. Let's define a new term "concept" that we will use to describe the random generator. In this context, a concept is a set of target word-boxes of a known class that are to be extracted and classified together with other explaining non-target word-boxes.

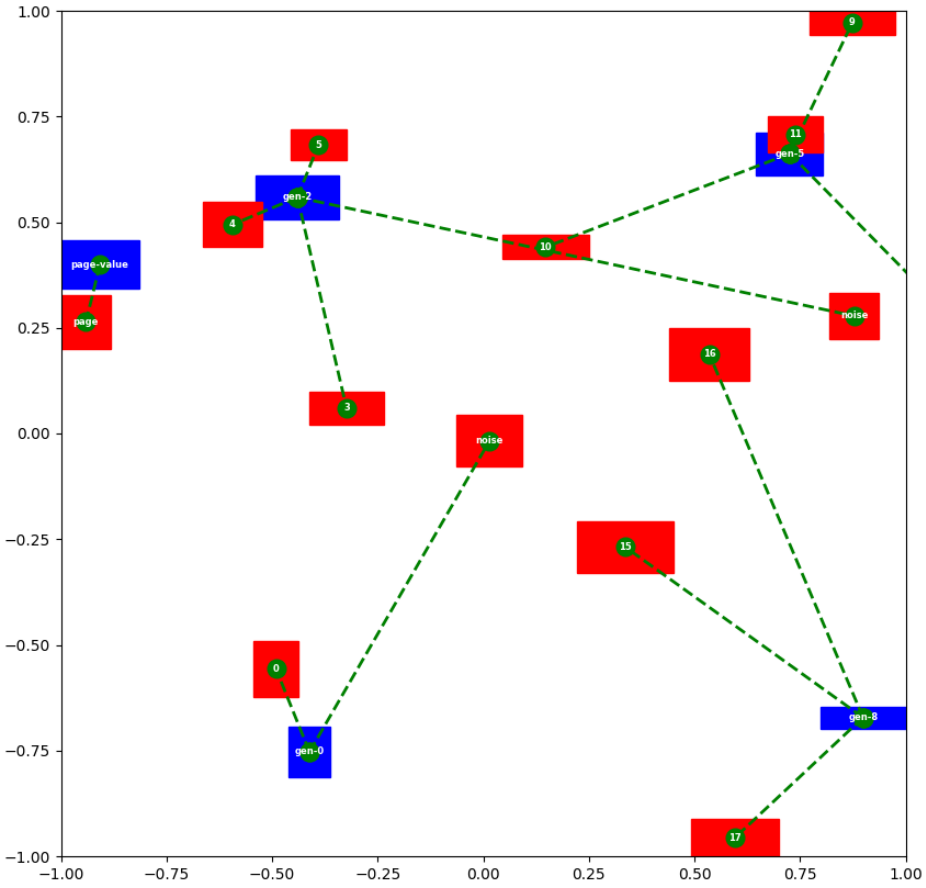
Every property of every word-box in a concept is also a random variable and can come from any given probability distribution.

A page of our artificial document is then constructed by sampling a reasonable amount of concepts.

Ultimately we will also add some "noise" word-boxes that are not important for the extraction or for assessing the meaning of the extracted texts. But adding noise helps neural networks regularize to real-world data since the real-world data do contain lots of the noise from our experience. The noise word-boxes may also appear with some probability in a random number of cases.

In the sample simulated document page (8.2), the blue word-boxes contain the target data that we would want to extract and classify. Red word-boxes represent the explaining neighbours and they define the classes of the blue word-boxes to which they are connected by dashed lines.

Figure 8.2: An example of a generated “document”.



8.2 Experiments and results

At this point, we have a parameterizable generator that represents the important information in business documents to our best knowledge and we can set up this generator and sample from it.

The ultimate goal is to use this generator to enhance the training of our neural network from the previous part. To explore and verify everything in depth, we will now take small steps in terms of model and generator complexity towards this target.

A note on how to read this section This section briefly summarizes all the experimental steps with their respective results. For in-depth explanations, details and constants, please see the accompanying code Holecek [2019] (greyed in the text).

8.2.1 First initial experiment - small architecture and fixed positions/distances

In the first experiment (see concepts_test.py) we have featured only a small convolutional architecture and let the generator generate only problems, where the class to be extracted depends on fixed position or distance only. This has been designed only as a trivial problem and the simple network proved to be sufficient.

8.2.2 Second batch of experiments - realistic generator and model with cheating information

In the second experiment, we moved to a more realistic generator configured based on our experience and visual inspection. We let the network explicitly know which boxes should be grouped (which is making the problem simpler, since no such information is present in the real world) and tuned the network's size to predict the target values with more than 0.9 F_1 score (see
run_experiments.py: fixed_known_borders, fixed_known_borders_bigger).

8.2.3 Third batch of experiments - realistic generator, shuffling and non-interesting boxes

In the third step, we added some random shuffling of the order of boxes inside each group and made the network predict also non-interesting boxes
(fixed_known_borders_all_boxes_noshuffle ,
fixed_known_borders_bigger_all_boxes_noshuffle ,
fixed_known_borders_all_boxes_shuffle), the score stayed similar at 0.88 F_1.

8.2.4 Final experiments - realistic generator and deep extraction model versus a simple one

In this step, no (cheating) group information is present to the network and all the information is passed in a text reading order on a page - exactly as in the

first article II. All the other features are passed in the same way too (including neighbours for each box).

Now by comparing two models - the model from the previous article and the simple baseline from the previous batch of experiments (articlemodel, baseline_rendered), the results showed (even after simplifying the generated problems and tuning the class weights) the simple baseline performing better than the article model (0.86 vs 0.67 F_1).

After more systematic data inspection, we have tuned our generators (realistic_experiment_articlemodel_local) to produce a greater amount of local dependencies, which made the article model perform better - 0.81 F_1 score, which still did not beat the baseline.

8.3 The conclusion

We have created an experimental environment, that allows for a quick experiment setup. We tried to emulate invoice documents based on human-gathered assumptions on the data (observation about the data being close to 8.1).

The reasoning about the dependent "explaining" boxes is quite trivial for some cases, as the example text "amount total: 100$" shows.

For the experiments, we have created an artificial set of information extraction problems of gradually increasing difficulty. These artificial problems were designed to be similar to the original one and simplified to be easily generated. The simple convolutional baseline was able to solve them up to 0.86 micro F_1 score.

We have also observed, that all the model's variants, that are stronger on the original dataset failed and scored below the simple baseline.

Therefore what are the key takeaways from these experiments? The created problem of simulated documents is indeed harder for the original model and so the model's bias is most probably the focus on the hidden dependencies, that were not captured by our initial assumptions.

Therefore the models that are stronger on business documents seem better in exploiting the structures, similarities and bonds concerning the word-boxes, that are not easily visible for the human eye.

On the other hand, our experiments proved, that a well-curated and annotated training dataset is invaluable and cannot be replaced by a simpler approximation. And if a method for generating invoices would be desired, it cannot be as simple as in these experiments here and would need to feature a more complex generative model to capture all the dependencies.

Ultimately, the direction for improving the results (as from II) would be to help the models exploit the relations unseen to the human eye (as is done in the next part IV) instead of relying on expert knowledge with generative models.

Part IV

Learning from similarity and information extraction from structured documents

International Journal on Document Analysis and Recognition (IJDAR)
https://doi.org/10.1007/s10032-021-00375-3

SPECIAL ISSUE PAPER

Learning from similarity and information extraction from structured documents

Martin Holeček[1]

Received: 18 October 2020 / Revised: 17 May 2021 / Accepted: 27 May 2021

Abstract
The automation of document processing has recently gained attention owing to its great potential to reduce manual work. Any improvement in information extraction systems or reduction in their error rates aids companies working with business documents because lowering reliance on cost-heavy and error-prone human work significantly improves the revenue. Neural networks have been applied to this area before, but they have been trained only on relatively small datasets with hundreds of documents so far. To successfully explore deep learning techniques and improve information extraction, we compiled a dataset with more than 25,000 documents. We expand on our previous work in which we proved that convolutions, graph convolutions, and self-attention can work together and exploit all the information within a structured document. Taking the fully trainable method one step further, we now design and examine various approaches to using Siamese networks, concepts of similarity, one-shot learning, and context/memory awareness. The aim is to improve micro F_1 of per-word classification in the huge real-world document dataset. The results verify that trainable access to a similar (yet still different) page, together with its already known target information, improves the information extraction. The experiments confirm that all proposed architecture parts (Siamese networks, employing class information, query-answer attention module and skip connections to a similar page) are all required to beat the previous results. The best model yields an 8.25% gain in the F_1 score over the previous state-of-the-art results. Qualitative analysis verifies that the new model performs better for all target classes. Additionally, multiple structural observations about the causes of the underperformance of some architectures are revealed, since all the techniques used in this work are not problem-specific and can be generalized for other tasks and contexts.

Keywords One-shot learning · Information extraction · Siamese networks · Similarity · Attention

1 Introduction

The challenge of information extraction is not a new problem. The task has been defined as the transformation of an array of texts into information that can be more readily understood and analyzed. It isolates relevant pieces of text, derives information from them, and then compiles the targeted information into a coherent whole [9].

The explicit category of business documents varies. Existing works on information extraction [25,32,46,49] define them as either "visually rich documents" or "structured" or "semi-structured" documents. In this work, we use the term "structured documents" since the structure of the documents is clear and understandable to a person working in the relevant field, even though the specific structure varies. Moreover, the documents are machine-readable in terms of individual words and pictures (including their positions) on a page, although not "understandable" for a machine with respect to extracting important information.

Classifying all of the information in the financial/accounting industry is important for the "users" of the documents. For example, they may need to know the payment details, amount to be paid, and the issuer information from a collection of documents. In this setting, the input is a document's page, and the ultimate goal is to identify and output all the words and entities from that page and to classify them by category.

We aim to improve information extraction from business documents and to generally contribute to the field of auto-

This work was supported by the Grant SVV-2020-260583.

Martin Holeček
martin.holecek.ai@gmail.com

[1] Department of Numerical Mathematics, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Published online: 11 June 2021

mated document processing. Our proposed approach yields a higher success metric compared with previous work and reduces the manual work involved in data entry and/or annotation in the financial/accounting industry.

We focus on the text of business documents including invoices, pro forma invoices, and debit notes, among others. In particular, we target the information that helps in automating various business processes, such as payment on invoices. The typical user of our method would be a medium-sized or larger company that is spending significant time on document processing owing to its size. Although details are scarce in referenced and peer-reviewed works because companies tend to keep their spending information secret, approximations from unofficial (nonscientific) sources (e.g., [34] and [50]) enable estimating company savings. As a hypothetical example, a typical medium-sized company can have approximately 25,000 invoices per month and an improvement of even 1% roughly translates to a savings of more than $500 monthly, which scales with the company size. This potential saving has motivated the topic of information extraction specially in business documents.

1.1 Details and overview

Figure reffig:Example presents one example of an input invoice and output extraction. The documents are clearly not easily understandable inputs. In contrast, an example of trivial inputs would be found in an XML document that has the desired target classes incorporated in a machine-readable way.

With this study, we aim to expand our previous work ([21], referenced as "previous"), in which we showed that neural networks can succeed in the task of extracting important information and even identifying whole, highly specific tables.

The current research question focuses on a "similarity"-based mechanism with various model implementations, and whether they can improve on the existing solution [21]. We hypothesize that a model incorporating similarity techniques will significantly improve the results compared with the existing solution. Moreover, since the presented mechanism is theoretically applicable beyond the scope of document processing, it can be utilized more broadly, whenever it makes sense to include similar known data and apply the query-answer technique.

Ultimately, we present a model along with its source code [22] that outperforms the previous state-of-the-art model. An anonymized version of the dataset is included as an open-source resource, and it represents a notable contribution since its size exceeds that of any other similar dataset to date.

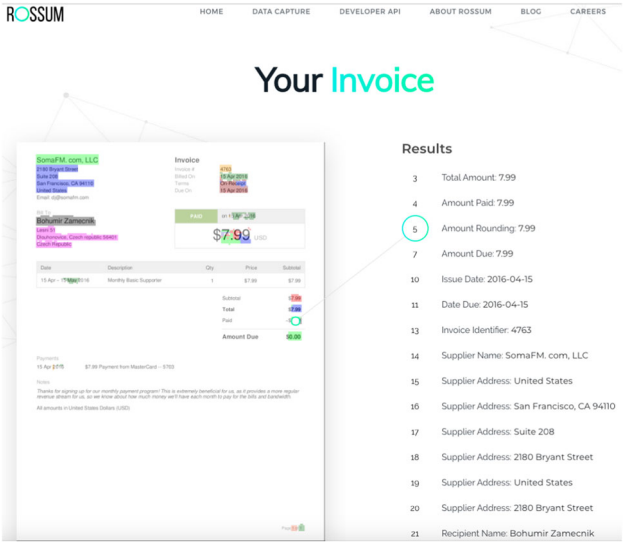


Fig. 1 An example of an invoice and an extraction system together with its output. This example also illustrates why invoices are called "structured documents." When the various pieces of information in the document are visually grouped together, it usually signals that it belongs together. There is a heading "Invoice," under which segments of information about the invoice are written next to their explanations. Some rectangular areas do not have these explanations, and to determine what rectangular area indicates about the sender and supplier, it is necessary to look for a small "Bill To": heading. These rules apply only to this specific example, and other invoices are notably different. (Online image source [47])

2 Related works

In this subsection, we focus on previous works and approaches in the field of information extraction. The content is heavily based on the text from [21].

A plethora of methods have historically been used for general information extraction. A complete review of these methods, much less comparisons between them, is beyond the scope of the current paper. In general, however, these methods were developed for and evaluated on fundamentally different datasets.

Furthermore, we determined that none of these previous methods is well-suited for working with structured documents (e.g., invoices) because such documents generally do not have any fixed layout, language, caption set, delimiters, fonts, and so forth. For example, invoices vary between countries, companies, and departments, and they change over time. In order to retrieve any information from a structured document, it must first be understood. Our criterion for a reference method is that no human-controlled preprocessing such as template specification or layout fixing is required; we aim for a fully automated and general solution. Therefore, no historical method can serve as a baseline.

Nevertheless, a significant number of recent works do successfully use a graph representation of a document

[8,11,26,32,33,46] and use graph neural networks. Also, a key idea close to the one-shot principle [24,53] in information extraction is used and examined, for example, in [20] and [12]. Both works use notions of finding similar documents and reusing their gold standards (such as already annotated target classes). The latter [12] applies the principle in the form of template matching without the need for any learnable parameters.

2.1 Broader inspiration

More broadly, our approach draws on several research streams pertaining to deep network architectures.

2.1.1 One-shot learning and similarity

A design concept that aims to improve how models contend with new data without retraining of the network is presented in [24,53].

Typically, a classification model is trained to recognize a specific set of classes. In one-shot learning, it is usually possible to correctly identify classes by comparing them with already known data. In contrast to traditional multi-class classification, one-shot learning allows attaining better scores, even with surprisingly low numbers of samples [14]. Sometimes it can even work for classes that are not present in the training set [53].

This concept can help in areas ranging from computer vision variants—from omniglot challenge [28] (also strokes similarity [27]) to object detection [55], finding similar images [57], face detection [51], autonomous vision [19], and speech [13]—and also in the NLP (Natural Language Processing) area [10,29,58].

Among the methods that enable one-shot learning, the most fundamental one utilizes the concept of similarity. Similarity relies on two types of data—"unknown" and "known." The target values of the known data are already recognized by the method and/or the model. To classify any unknown input, the usual practice is to assign it to the same class as the most similar known input.

Technically speaking, the architecture (typically) contains a "Siamese" part. In particular, both inputs (unknown and known) are passed to the same network architecture with tied weights. We draw inspiration from this basic principle, and we leave more advanced methods of one-shot learning (e.g., GANs [35]) for further research.

For performance reasons, the model is usually not asked to compare new inputs to every other known input—only to a subset. Therefore, a prior pruning technique needs to be incorporated, for example, in the form of a nearest-neighbor search in embedding space, as in [17]. Another option would be to incorporate a memory concept [6] (even in the form of neural Turing machines [48]).

The loss used for similarity learning is called triplet loss because it is applied on a triplet of classes (R reference, P positive, N negative) for each data point:

$$L(R, P, N) = \max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0)$$

where α is a margin between positive and negative classes, and f is the model function mapping inputs to embedding space (with euclidean norm).

Generally speaking, one-shot learning can be classified as a meta-learning technique. For more on meta-learning, we suggest a recent study, like [44] (or just a compiled bibliography online at [36]). Taking the concept one step further yields a concept called "zero-shot learning" [15,37,43].

2.1.2 Other sources of inspiration

Several other sources of inspiration are also meaningfully close to one-shot learning. Since we ask "what labels are similar in the new data," we need to consider a "query-answer" approach. Recently, the attention principle (i.e., transformer architecture) successfully helped to pave the way to a state-of-the-art performance in language models [45]. It is not uncommon to use attention in one-shot approaches [54] and also in settings related to query answer [16,40,56].

The task of similarity can also be approached as pairwise classification, or even dissimilarity [30].

3 Methodology overview

As we previously argued, every incremental improvement matters. In the current work, we focus on improving the metrics (established by our previous work) by selecting relevant techniques from the field of deep learning. A classical heuristic way to generally improve a target metric is to provide more relevant information to the network. Previous implementations have featured various well-performing techniques that have used the information present in a single invoice, and here we focus on techniques related to similarity.

Since the idea of providing more information is fundamental even for simpler templating techniques [12], we need to stress that, due to the nature of our dataset (available as an anonymized version at [22]), our problem cannot be solved by using templates.

It is important to clarify here the differences between other works and our stream of research (meaning this work and the previous [21]).

The most important difference comes from the dataset that is at our disposal. The dataset explored here is far larger than the datasets used elsewhere and allows for exploring deeper

models as opposed to only using graph neural networks. Indeed, in our previous paper, we proved that graph neural networks work in synergy with additional convolution-over-sequence layers and even global self-attention. Moreover, the dataset quality allowed us to discover (in our previous work) that information extraction and line-item table detection targets do in fact boost each other.

As the current research is focused on deeper models, we will not be using any of the other works as baselines and the commonly used graph neural networks will be incorporated only as one layer amidst many, with no special focus.

We will explore models that could benefit from access to a known similar document's page. We hope that the model can exploit similarities between documents, even if they do not have similar templates.

In addition, we want to explore the added effect of similarity while keeping everything as close to the previous setting as possible to make sure no other effect intervenes.

The following description mirrors that provided in previous work (in sections 3.3 and 3.4 of [21]). Note that the previous work did not use any means of "similarity" or "nearest pages," which are introduced in the current work.

3.1 Definition of concepts

The main unit of our scope is every individual word on every individual page of each document. Note that other works (e.g., [32]) use the notion of "text segments" instead of "words." For this work, we define a "word" as a text segment that is separated from the rest of the text by (at least) one white space, and we do not consider any other text segmentation. In general, our approach can also be called "word classification" approach as written in [42], a work where an end-to-end architecture with a concept of memory is explored.

3.1.1 Inputs and outputs

Conceptually, the whole page of a document is considered to be the input to the whole system. Specifically, the inputs are the document's rendered image, the words on the page, and the bounding boxes of the words. As PDF files are considered, any possible library for reading PDF files can be used for reading the files and getting the inputs. Note that by using any standardized OCR technique, the method could also theoretically be applied to scanned images. (Measuring the effect of OCR errors on the extraction quality is not done here.)

These inputs then undergo feature engineering, as described in 3.2.1, and become inputs for a neural network model.

Each word, together with its positional information, constitutes a "word-box" that is to be classified into zero, one, or more target classes as the output. We are contending with a multi-label problem with 35 possible classes in total. The classes include the "total amount," tax information, banking information, issuer, and recipient information, among others. (The full set is defined in the code [22].) To obtain a ground truth, the classes were manually annotated by expert human annotators. Interestingly, they had a roughly 3% error rate, which was eliminated by a second annotation round.

3.1.2 The dataset and the metric

Overall, we have a dataset with 25, 071 documents as PDF files totaling 35, 880 pages. The documents are from various vendors and have differing layouts and languages. We split the documents into training, validation, and test sets at random (80%/10%/10%).

The validation set is used for model selection and early stopping. The metric used is computed first by calculating all the F_1 scores of all the classes. It is then aggregated by micro-metric principle (more can be found for example in [39]) over all the word-boxes, over all the pages. We then observe and report the scores of the testing set.

The metric choice is inspired by the work [18] in which a content-oriented metric was defined on a character level. In our setting, the smallest unit is a word-box. The choice of the F_1 score is based on the observation that the counts of positive samples are outnumbered by the negative samples. In total, the dataset contains 1.2% positive classes.

3.2 Shared architecture parts

In the current work, we refer to the architecture from our previous work as a "simple data extraction model." It serves as one of the baselines here. The architecture of the current model is the same as in the previous work, with the exception of a minor manual parameter tuning. A notable part of the current model, called the "basic building block," is used in all the new models (defined in Sect. 3.4). Both the simple data extraction model and the basic building block are depicted in Fig. 3.

Since the goal of the overall task and the whole basic building block architecture are shared across all models, by describing the "simple data extraction model," we also describe all the shared and inherited parts—notably the input and output requirements. We use the full geometrical, visual, and textual information as the input, and the model outputs a multi-class classification for each word-box.

3.2.1 Detailed feature engineering of the inputs

We operate based on the principle of reflecting the structure of the data in the model's architecture, as machine learning algorithms tend to perform better with this approach.

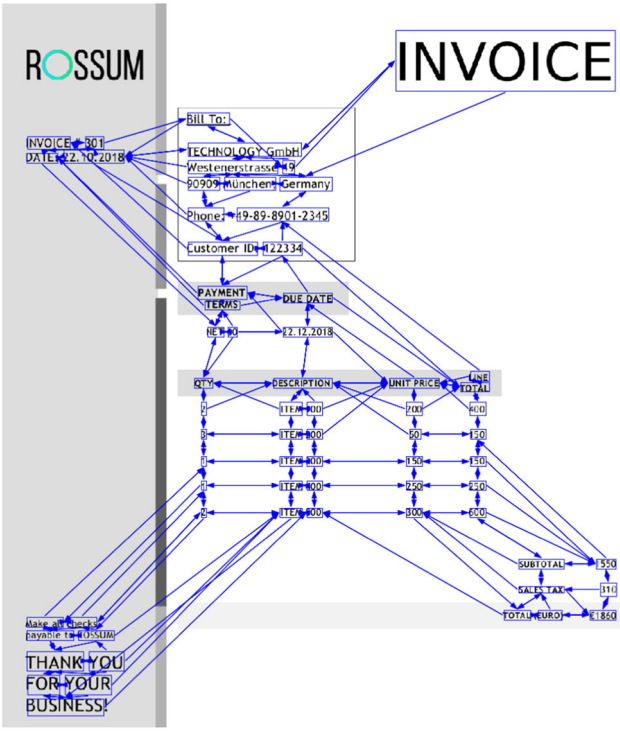


Fig. 2 A sample invoice with edges defining neighborhood word-boxes. Only one closest neighbor is connected to each word-box's edge. The resulting graph is directional, and so each word-box has four outbound arrows—one ($n = 1$) for each side of the box—but the number of inbound arrows is not bounded (this invoice was created for presentation and does not represent the invoices in the dataset)

The structured information at the input is an ordered sequence of all the word-boxes present on a page. This number can vary by page.

Each word-box has the following features:

- Geometrical:
  - Using a geometrical algorithm, we can construct a directed neighborhood graph over the boxes, which can then be used by a graph CNN (see 3.2.2). Neighbors are generated for each word-box (W) by formally assigning every other box to an edge of W that has it in its field of view (being the same 90°). Then, the closest (center-to-center Euclidean distance) n neighbors are chosen for each side of the box. Our previous results indicated that the optimal number is $n = 1$, and so this number is used in the experiments here. For an example of the constructed directed graph, see Fig. 2. Internally, the graph is saved and passed only as integer indexes denoting the position of each neighbor in a global sequence of word-boxes.
  - We can define a "reading order of word-boxes." In particular, based on the idea that if two boxes overlap in a projection to the y axis by more than a given threshold (set to 50% in the experiments), they should be regarded as being in the same line from the perspective of a human reader. This not only defines the sequence in which the boxes will be given to the network, but it also assigns a line number and order-in-line number to each box. To get more information, this algorithm can be run again on a 90° rotated version of the document. Note that the exact ordering/reading direction (left to right and top to bottom or vice versa) does not matter in the neural network design, thus giving us the freedom to process any language.
  - Each box has four normalized coordinates (left, top, right, bottom) that should be presented to the network.
- Textual:
  - Each word can be presented using any fixed-size representation. Here, we use tailored features common in other NLP tasks (e.g., authorship attribution [7], named entity recognition [38] and sentiment analysis [2]). The features per word-box are the counts of all characters, the counts of the first two and last two characters, length of a word, number of uppercase and lowercase letters, number of text characters, and number of digits. Finally, another feature is engineered to determine whether the word is a number or amount. This feature is produced by scaling and min/maxing the amount by different ranges. (If the word is not a number, this feature is set to zero.) We chose all these features because invoices usually include a large number of entities, IDs, and numbers that the network needs to be able to use.
  - Trainable word features are employed as well, using convolutional architecture over a sequence of one-hot encoded, de-accented, and lowercase characters (only alphanumeric characters and special characters " ,.−+:/%?$£ #()&'"; all others are discarded). We expect these trainable features to learn the representations of common words that are not named entities.
- Images:
  - Each word-box has its corresponding crop in the original PDF file, where the word is rendered using particular font settings and also has a background. This could be crucial to detect a header or heading, for example, if it contains lines or a different background color or gradient. So for each word-box, the network receives a crop from the original image, offset outwards to be bigger than the text area so the surroundings can also be detected.

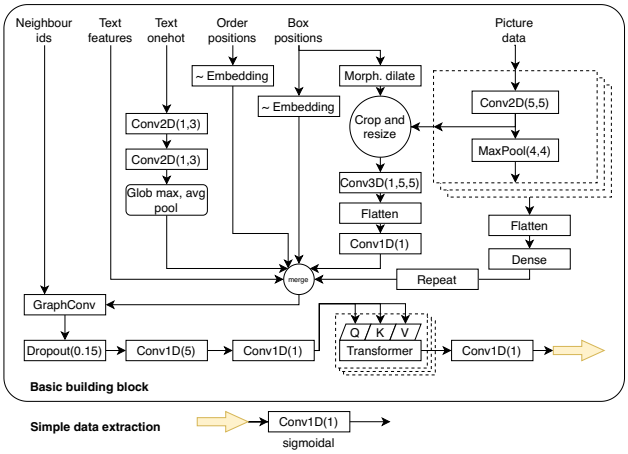


Fig. 3 Simple data extraction model. Formally, the whole model consists of two parts: a basic building block and a final classification layer. The basic building block will be used (as a Siamese network) in other models. By removing the final classification layer, we hope to get the best feature representation for each word-box

Each presented feature can be augmented, and we present a random 1% perturbation on coordinates and textual features to regularize the problem and help with generalization.

3.2.2 Simple data extraction model details

To summarize the document's features described in the previous section, we now explain how they are processed by the model (as Fig. 3 shows). In total, we have five inputs that the neural networks will use:

- Down-sampled picture of the whole document (620 × 877), gray-scaled
- Features of all word-boxes (as defined in the previous section), including their coordinates
- Text as first 40 one-hot encoded characters per each word-box
- Neighbor ids, which are look-up indexes that define the neighboring word-box on each side of the word-box (only one closest neighbor per side is used)
- The integer positions of each word-box defined by the geometrical ordering

In the simple data extraction model, the positions are embedded by positional embeddings (as defined in [31,52]. An embedding size equal to four dimensions for sin and cos, with a divisor constant of 10, 000, is used. The embedded positions are then concatenated with other word-box features.

The image input is reduced by a classical stacked convolution and max-pooling approach. The word-box coordinates (left, top, right, bottom) are not only used as a feature, but also to crop the inner representation of the picture input (see "morphological dilation" in Fig. 3). Finally, we give the model the ability to grasp the image as a whole and supply a connection to the said inner representation, which is flattened and then processed to 32 float features.

Before attention, dense, or graph convolution layers are used, all the features are simply concatenated. To supplement this description, equations and network definitions are given in[22].

As shown in our previous work, all three means of assessing relations between word-boxes are used:

- Graph convolution (also denoted as "GCN") over the geometrical neighbors of word-boxes is employed to exploit any form of local context. (Details are provided at the end of this section in graph convolution mechanism details.)
- A *convolution over sequence* layer is a dense layer (or equivalently a 1D convolution layer) applied over the word-boxes ordered by the reading order and allows the network to follow any natural text flow. Implementation-wise, all the word-boxes are ordered in the second dimension at the input (all dimensions being [batch, ordering, feature space]).
- The attention transformer module (from [52]) allows the network to relate word-boxes across the page. Our attention transformer unit does not use causality or query masking.

After these layers are applied, the basic building block definition ends with each word-box embedded in a feature space of a specified dimension, which is 640 unless stated otherwise. The following layer, for the simple data extraction model, is a sigmoidal layer with binary cross-entropy as the loss function. This is a standard setting, since the output of this model is meant to solve a multi-class multi-label problem.

To note implementation detail, batched data fed to the model are padded by zeros per batch (with zero sample weights). Class weights in the multi-task classification problem were chosen (and manually tuned) based on positive class occurrences.

Graph convolution mechanism details The word-box graph (with word-boxes as nodes and neighborhood relation as edges, as depicted in Fig. 2) has a regularity that allows simplifying the graph convolution. First, a small upper bound exists on the number of edges for each node, and second, we do not use any edge classification or specific edge features, in contrast to other works (e.g., [46]). Therefore, we use a simpler implementation than the general form graph convolutions (as in [23,41]).

More specifically, the implementation uses the generic simplicity present in convolutions at the cost of an additional input. Even a classical convolutional layer over regular picture data can be represented by two basic operations. First, a

gather operation (using tf.gather_nd function from [1]) prepares the data to a regular array (matrix of size number of data points multiplied by the number of data points in one convolutional operation). The second operation is a time-distributed dense layer (equivalently called Conv1D) that simulates the weights of such convolution.

The gather operation needs additional input for each point (pixel or graph node) that specifies the integer indexes of its neighbors (and the node itself). These integer indexes are constructed exactly as stated in 3.2.1.

3.2.3 Differences from the previous setting

Just as we have noted the differences from existing research in 2, it is also important to note some detailed differences from our previous work.

The novelty of this work with regard to the previous setting Our previous work [21] did not use any nearest-neighbor search or any models that used the notion of similarity or allowed more than one input page at once. In short, our previous work simply laid the fundamental principles of the data, task, and metric and introduced the basic building block (with ablation analysis). Everything that follows this point is new.

Details changed from the previous setting Unlike the previous setting, here we do not classify the line-item tabular structures, but only extract (above mentioned) information from the page. In doing so, we demonstrate that the model, despite being optimized on line-item table detection, is versatile. Hence, we make only minor tweaks in the model's architecture (results of the modifications depicted in Fig. 3).

Previously, two datasets were used for training and validation—"small" (published) and "big" (previously unpublished). The models were tuned on the small dataset, and the big dataset was only used in two experiments to validate that the model scales. In this work, we use the same big dataset. Its previous validation set is split into a new validation set and a new test set to make the test set larger and properly address generalization.

Multiple baselines are employed to prove that the new test set contains documents with layers that are sufficiently different. (The previous work's test set was small and manually selected.)

3.2.4 Differences from one-shot learning

As stated in the introduction, we want to boost the model's performance for existing target classes by giving the network access to known data (documents) in ways similar to one-shot learning. The main difference is that we are utilizing experiments and architectures that include a fixed selection of classes and/or class information (from the nearest page). Clarifying this detail is important because usually in one-shot learning, no classes are explicitly present in the model—the aim is to generalize to those classes. Our aim, by contrast, is to generalize the model to different and unseen documents with different layouts (instead of classes) that still feature those word-box classes.

3.3 The learning framework

The easiest step to boost predictions of an unknown page is to add one more page that is similar and includes word-box classes (annotation) that are already known to the system. That annotation information can then be used in a trained model.

Overall the method works as follows:

- The system needs to keep a notion of already known documents in a reasonably sized set. We call them "known" because their classes/annotations should be ready to use.
- When a "new" or "unknown" page is presented to the system, it searches for the page that is most similar to the "known" pages (given any reasonable algorithm).
- The model is allowed to use all the information from both pages (and "learn from similarity") to make the prediction.

The system can then even present the predictions to a human for verification and then add the page to the existing database of known pages. However, we do not explore the database size effects here.

Before making predictions, the incorporated model should be trained on pairs of pages to simulate this behavior.

In this process, there are multiple points to be examined, but we posit that the most interesting research question is the following:

Holding all other factors fixed (meaning the train/test/validation split, evaluation metrics, data format, and method for searching for a similar page), what approach and what neural network architecture are able to raise the test score the most?

We argue that this is the right question to ask since all other factors usually have a known effect on the result if best practices are followed. As an example, we note that bigger datasets typically yield better scores; the presence of more "nearest neighbors" typically has a boosting effect similar to ensembling, and so on.

Further, from a practical point of view, only two pages can fit into a single GPU memory with all the features described before.

As already stated earlier, we draw inspiration from the one-shot learning framework. For predicting an unknown page, we define a way to search for one "nearest" known page and allow the model access to its annotations as known target classes. Note that not all explored models use the nearest known page. In addition to the simple data extraction model,

we consider some baselines that do not require the nearest page to verify the assumptions.

3.3.1 Nearest-neighbor definition

For one-shot learning to work on a new and unknown page (sometimes denoted as the "reference"), the system always needs to have a known (also denoted as "similar" or "nearest") document with known annotations at its disposal. Since focusing on that task properly is beyond the scope of the current paper, we have used the nearest-neighbor search in the space of the page's embeddings to select only one closest page of a different invoice document.

The embeddings were created through a process similar to a standard one, as described in [5]. We used a different model (older and proprietary) that was trained to extract information from a page. To change the classification model into an embedding model, we removed its latest layer and added a simple pooling layer. This modified the model to output 4850 float features based only on image input. These features were then assigned to each page as its embedding.

We then manually verified that the system would group similar, or at least partially similar, pages near each other in the embedded space.

These embeddings were held fixed during training and inference and computed only once in advance.

3.3.2 Constraints of nearest-neighbor search

We want the trained model to behave as close to the real world as possible, so the nearest page search process needs to be constrained. Each document's page can select the nearest annotated page only from the previous documents in a given order. As in a real service, we can only see the received and processed documents.

In addition, we want the method to be robust, so before each epoch, the order of all pages is shuffled and only the previous pages (in the given order) from a different document are allowed to be selected.

This holds for all sets (training, validation, and test) separately. To verify the consistency of this strategy, some experiments are tweaked by the following variations:

- Allowed to additionally use the training set as a data source for the "nearest annotated" input. We expect the performance to rise.
- Made "blind" by selecting a random document's page as the nearest known input. We expect the performance to fall.

3.3.3 Baselines

To challenge our approach from all possible viewpoints, we consider multiple baselines:

1. To use only the simple data extraction model (Sect. 3.2 and Fig. 3) without any access to the nearest known page.
2. "Copypaste" baseline. This model will only take the target classes from the nearest page's word-boxes and overlay them on the new page's word-boxes (where possible). We expect a low score since the documents in the dataset are different, and this operation will not copy anything from any nearest page's word-box that does not intersect with a new page's word-box. This approach uses no trainable weights and is the simplest example of a templated approach that does not have hard-coded classes.
3. "Oracle" baseline. This model will always correctly predict all classes that are present in the nearest page. We use this model to measure the quality of the nearest-page embeddings to gain additional insight into the dataset's properties. The metric used for this model is not F_1, but a percentage of all word-boxes that can be classified correctly. The score is expected to be only moderately good, as the embeddings are created in a rather unsupervised manner (regarding their usage). We want to explore a different influence than that already explored by existing works aimed at finding the best helping pages [12]. Ultimately, we want to present a model that can work even if the quality of the embeddings is just moderate.
4. Fully linear model with access to concatenated features from both new and known pages. This model does not feature picture data.

The choice of baselines (and ablations later in experiments) helps to verify and demonstrate multiple claims:

- The newly proposed models can beat the previous results (which is achieved if the simple data extraction model is beaten).
- The documents are different enough.
- A similarity search alone is not enough, even if the embeddings have better-than-moderate quality with regard to the similarity.
- To justify the complexity of models presented in the following section, 3.4.

All baselines and all models presented in the current work will have the same desired output—they will provide the multi-class classification for each word-box.

3.4 Model architectures

We have described the basic information extraction block that aims to output the best trained latent features for each word-box. All the model architectures incorporate this block used as a Siamese network for the inputs of both unknown and known pages. Each architecture is trained as a whole, no pre-training or transfer learning takes place, and every model is always implemented as a single computation graph in Tensorflow.

We explore multiple different architectural designs for predicting the targets (at their outputs) by using the closest nearest page from already annotated documents.

1. "Triplet Loss architecture"—using Siamese networks "canonically" with triplet loss.
2. "Pairwise classification"—using a trainable classifier pairwise over all combinations of word-box features from reference and nearest page.
3. "Query-answer architecture" (or "QA" for short)—using the attention transformer as an answering machine to a question of "which word-box class is the most similar."

The copypaste baseline represents a reasonable basic counterpart for triplet loss and pairwise classification. The fully linear model represents the simplest counterpart for the query-answer approach, which also has all the classes hard-coded.

There is a slight distinction between the first two architectures and the third. In QA architecture the class is a direct prediction of the network for each word-box. In triplet loss and pairwise classification, the models predict (for each unknown word-box) all the similarities to all the word-boxes from the known page. All the similarity values then collectively determine the target class for the word-box.

Since the embeddings used to search for the nearest page are not ideal, the models may not be able to predict some classes. To assess these methods fairly, we scale the metrics used to measure the success by the performance of the corresponding oracle baseline (defined in refsubsec:Baselines). Or put differently, we do not count errors that the model cannot predict correctly owing to some classes being absent from the nearest page. This reflects our aim to explore the effects of the models that can operate with the nearest page.

In reality, if these (triplet loss and pairwise classification) methods prove to be the most efficient, the hyperparameters, such as the quality of the embeddings (or the number of the nearest pages), would need to be addressed to overcome the score of the previous results. A perfect performance of the scaled metric means that the extraction is only as good as the oracle baseline.

In the experimental result Sect. 4, we include a test of triplet loss and pairwise classification models that makes them predict a different and possibly easier target. Instead of "do these two word-boxes have the same target class," the easier testing target is "do these two word-boxes have the same length of text inside." This test is meant to show that the method is well-grounded and usable for any other reasonable target definition.

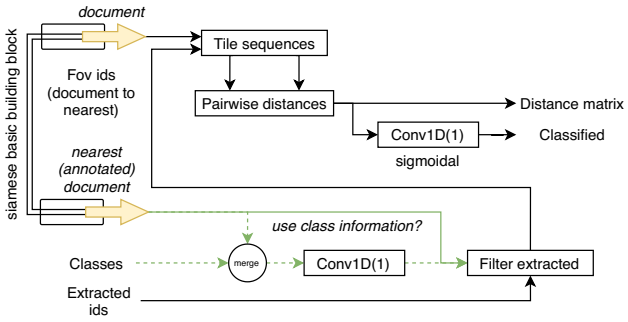


Fig. 4 The triplet loss architecture. The Siamese part at the input is represented by the arrows (they denote the basic building block from Fig. 3), and it processes the document and its nearest similar counterpart. The network construct pairs of (extracted features of) all the word-boxes from the document and the nearest one and computes the distances. The nearest page's word-boxes are filtered to feature only word-boxes with any positive class. If we want to add class information from the nearest page, the green dashed version is used (colour figure online)

In comparison, QA architecture has the classes hard-coded in the design, which means it can predict a class not present in the nearest page. Therefore, no metric scaling is necessary in the evaluation of the QA model.

3.4.1 Triplet loss architecture

Since our data point is a word-box, strictly adhering to the use of triplets of word-boxes for triplet loss would mean executing the model for each word-box pair once. To avoid impairing the performance (as there can be as many as 300 word-boxes per page) and/or losing in-page dependencies, the proposed architecture (see Fig. 4) features a mechanism of tiling and filtering to pass all combinations of word-boxes at once.

The filtering mechanism filters out all but the annotated word-boxes from the nearest page. It eliminates most of the unused information and, in doing so, saves memory and computation time. The tiling mechanism takes two sequences—first, the sequence of reference page word-boxes, and second, the sequence of nearest page filtered word-boxes. It subsequently produces a bipartite matrix. The model is then able to compute pairwise distances between the same and different classes. These distances are then used for triplet loss computation (see mathematical definition in the section below).

Additionally, we can include a single classification layer to be automatically calibrated on the distances, which adds

a binary cross-entropy term to the loss. The loss is averaged over all the word-boxes to account for the fact that no word-box is ever alone on a page.

We rely on the (manually verified) fact that during training each page has more than one class annotated. Consequently, there are always positive and negative samples present, as there should be in the triplet loss.

There are three possible modifications to explore:

- Adding annotated class information to the nearest page's features.
- Using a "loss-less triplet loss," which is a loss similar to the triplet loss but without the min–max functions (see definition below).
- Modifying the distance and/or loss computations by means of constants or by using cosine similarity instead of Euclidean space.

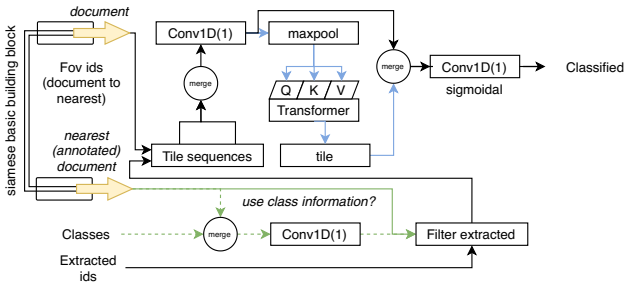


Fig. 5 Pairwise classification architecture with an optional refinement module. The same principle as in the triplet loss is used; that is, the Siamese part at the input is represented by the arrows (they denote the basic building block in Fig. 3). The network constructs pairs of (extracted features of) all the word-boxes from the document and filtered word-boxes from the nearest document and then performs a classification layer over the paired features. To experiment with the model complexity, an optional global refinement module (denoted with blue arrows) could be employed (colour figure online)

3.4.2 Triplet-loss inspired losses

The purpose of this model is to use the triplet loss in the most straightforward manner in our setting. The only mathematically interesting description to be given here is the triplet loss and "loss-less triplet loss" defined over word-boxes since all trainable layers in this model (and binary cross-entropy loss) are defined in referenced works.

In traditional triplet loss, positive, negative, and reference samples are necessary. Since we need to account for a whole page full of word-boxes, we must compute all combinations at once.

We denote the quantity truth_similar(i, j) to indicate if the word-boxes i, j (i-th being from the unknown page, j-th being in the nearest page) share the same ground truth class (1.0 = yes, 0.0 otherwise). Next we define pred_dist(i, j) as the predicted distances between the feature spaces of the word-boxes by the model. Then, we can calculate two loss variants ("triplet_like" and "loss-less") inspired by triplet loss as follows:

$$\begin{aligned}
\text{pos_dist}_{i,j} &= \text{truth_similar}(i, j) \cdot \text{pred_dist}(i, j) \\
\text{neg_dist}_{i,j} &= (1.0 - \text{truth_similar}(i, j)) \cdot \text{pred_dist}(i, j) \\
\text{triplet_like} &= \max_{i,j}(0, \alpha + \max(\text{pos_dist}_{i,j}) \\
&\quad + \min_{i,j}(-\text{neg_dist}_{i,j})) \\
\text{lossless} &= \sum_{i,j} \text{pos_dist}_{i,j} - \sum_{i,j} \text{neg_dist}_{i,j}
\end{aligned}$$

(The equations are present in a form to be most similar to the source code, i.e., not simplified.) The quantities pos_dist and neg_dist are just helper variables to reveal the similarity with the original triplet loss, and α is a parameter of the same meaning as in the original triplet loss. The two new losses represent two different approaches used in the reduction from a matrix to a single number. We can either take the largest positive and negative values and use them in the triplet loss equation, or we can sum all the positive and negative terms. The real difference is how the gradients are propagated; variants with min/max always propagate fewer gradients than the former per gradient-update step in the training phase. All the losses can be used at once with a simple summation.

The name "loss-less" comes from the idea described in [3]. To our knowledge, it does not occur in any other scientific work beyond this online article.

Finally, we present different options for the loss terms. Since we focus on different architectures and not on hyperparameters, we omit from this description the specific constants used to sum the terms. In the experiment Sect. 4, we present the best results that we were able to achieve by manual hyperparameter tuning. The results of the tuning and various options are clearly defined in the accompanying code [22] together with all the specific hyperparameters.

3.4.3 Pairwise classification

Pairwise classification architecture (see Fig. 5) uses the same tiling and filtering mechanism as described in 3.4.1. But instead of being projected into a specific feature space to compute distances, the data points are simply "classified" by using a traditional approach of sigmoidal activation function and binary cross-entropy loss.

As in our previous model, we have the option of adding annotated class information to the nearest page's features. We have also explored various sample weight options and an optional "global refinement" section. The optional refinement pools information from each word-box uses a global transformer and propagates the information back to each reference word-box.

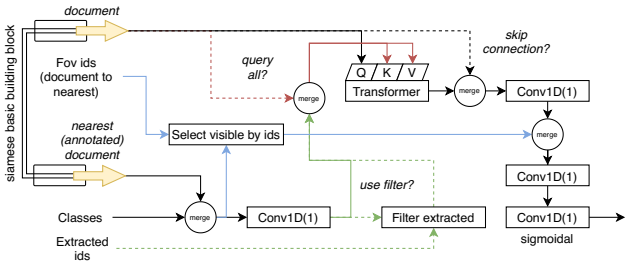


Fig. 6 The QA architecture. The centerpiece of this model is the transformer module, which allows each word-box to be paired with matching information (in the means of attention) from either only the nearest document (red arrow only) or both the nearest document and the reference document (red dashed arrow used). The class information is merged with the features of the nearest document's word-boxes and can be directly passed to the transformer (green arrow) or filtered to contain only positively annotated word-boxes (green dashed arrow). The part of the model with blue arrows allows the "field of view" information flow (colour figure online)

3.4.4 Query-answer architecture

In the heart of the QA architecture (see Fig. 6) lies the fact that the transformer module with three inputs can be used as a query-answer machine.

More variants could be explored here:

- "Query all": Does it help if the transformer can query not only the nearest page's word-boxes, but also those of the new page itself?
- "Skip connection": Would a skip connection to the base information extraction block improve the performance?
- "Filter": Should it filter only annotated word-boxes from the nearest page (as in the two previous approaches)?
- "Field of view": Would adding a field of view information flow from the new page's word-boxes to the nearest page make a difference?

Technically a field of view is realized by providing indexes, in which word-boxes would be close to each other by geometrically projecting each word-box from the reference page to the annotated page and selecting a fixed number of Euclidean-closest word-boxes. The limits for the distances were chosen based on average distances between word-boxes of the same class on different documents. The loss used for this model is classical binary cross-entropy.

The main idea of this architecture is a query-answer mechanism and so it can be applied in any different setting with Siamese networks.

4 Experiments and results

In this section, we present the results for each group of experiments. An Adam optimizer was used together with an early stopping parameter of 20 epochs (to maximally 200 epochs). The average time was 40 min per epoch on a single GPU. The baseline needed only 10 min per epoch (since it did not need any "nearest" page mechanism). The model selected in each experimental run was always the one that performed the best on the validation set in terms of loss.

The basic building blocks present in every architecture were usually set to produce feature space of dimensionality 640 (unless noted otherwise in the tables as "feature space n").

Additionally, experiments on the anonymized dataset were performed on the best architecture and the baseline model. The anonymized dataset does not include picture information, and each character in any textual information is replaced by the letter "a" (e.g., a word such as "amount" would be replaced with "aaaaaa"). Moreover, some features in some documents are randomly adjusted in various ways to prevent mapping the anonymized documents to reality.

Some experiments with architecture variations are included to show how the model's variance affects the score—for that reason, we have slightly varied the number of the transformer layers—("1x attention layer" marks single layer, "2x attention layer" marks two consecutive layers being used), as that is the single most complex layer present.

Table 1 Simple data extraction model experimental results

Previous state of the art, re-tuned (and possible notable tweaks, see Sect. 4)	Test micro F_1 score
2x attention layer, feature space 640	0.6220
1x attention layer, feature space 640	0.8081
1x attention layer, feature space 64	**0.8465**
1x attention layer, f. space 64, fully anonymized	0.6128
1x attention layer, f. space 64, only text features	0.7505

The bold number indicates the best achievable results of our previous work alone, therefore it is the score we aim to beat in this article

4.1 Baseline results

We report some variations of architecture parameters for the simple data extraction model (introduced in Sect. 3.2) in Table 1. The goal is to show how sensitive the basic model is to various changes and to tune the baseline for extracting the classes.

The results could be interpreted as the model reaching its maximal reasonable complexity at one transformer layer and smaller feature space. As we will see, this does not apply to the Siamese settings as the gradients propagate differently when parts of the architecture have tied weights.

Table 2 Copypaste baseline results

Experiments architecture (and possible notable tweaks, see Sect. 4)	Test micro F_1 score
Nearest page by embeddings and from validation set (standard)	0.0582
Nearest page search from validation and train set	0.0599
Nearest page set to random	0.0552

Table 3 Oracle results. The metric "Hits" denotes the percentage of word-boxes that have their corresponding class in the nearest page

Oracle setting	Hits (%)
Nearest page by embeddings and from validation set (standard)	59.52
Nearest page search from validation and train set	60.43
Nearest page set to random	60.84

To beat our previous state-of-the-art results, we need to improve the F_1 score to exceed 0.8465, which is the best score for the simple data extraction model.

4.1.1 Copypaste baselines

Table 2 shows the fairly low score of those simple baselines. Such a low score illustrates the complexity of the task and variability in the dataset. Simply put, it is not enough to just overlay a different similar known page on the unknown page because the dataset does not contain completely identical layouts.

We can also see that an important consistency principle holds for the nearest neighbors:

- Selecting a random page decreases the score.
- Using a bigger search space for the nearest page increases the score.

4.1.2 Oracle baseline

Table 3 displays the "moderate quality" of the embeddings. Specifically, only roughly 60% of word-boxes have their counterpart (class-wise) found in the nearest page.

When the nearest-neighbor search is replaced with a completely random pick, an interesting property of the dataset emerges in that the number of word-boxes that have a similar class on the random page increases a little. This is because the distribution of class presence in the pages is skewed, which is explained by vendors usually wanting to incorporate more information into their business documents.

4.1.3 Linear baseline

The linear model has attained 0.3085 test micro F_1 score. Its performance justifies the progress from the basic copypaste model toward trainable architectures with similarity.

Table 4 Experimental results of triplet loss architectures

Experiments architecture (and possible notable tweaks, see Sect. 4)	Test micro F_1 score
1x attention layer, loss-less variant	0.0619
2x attention layer, loss-less variant	0.0909
1x attention layer	0.1409
2x attention layer	0.1464

Table 5 Experimental results of pairwise architectures

Experiments architecture (and possible notable tweaks, see Sect. 4)	Test micro F_1 score
2x attention layer + refine section	0.2080
2x attention layer	0.2658
1x attention layer	0.2605

But since it does not beat the previous baseline results, we find that the similarity principle alone does not help, and thus, the design of more complicated models is justified.

4.2 Results of architectures with similarity

In this section, we consider all the designed architectures that compete with the baselines.

The results for triplet loss architecture are presented in Table 4, and the results for pairwise classification are in Table 5.

Both pure triplet loss approaches and pairwise classification performed better than simple copypaste, but still worse than linear architecture. We suggest two possible reasons for this outcome:

- The existence and great prevalence of unclassified (uninteresting) data in the documents.

Table 6 Experimental results of QA architecture

Experiments architecture (and possible notable tweaks, see Sect. 3.4.4)	Test micro F_1 score
All QA improvements in place	0.9290
Fully anonymized dataset	0.7078
Only text features	0.8726
Nearest page set to random	0.8555
Without field of view	0.8957
Without query all	0.7997
Without skip connection	0.9002
Without filtering	0.8788

This reason is supported by the fact that all methods with hard-coded class information (including simple linear baseline) scored better. Unfortunately, this phenomenon could be specific to the dataset. We could not replicate the suboptimal results by modeling this situation in an existing and otherwise successful task (omniglot challenge) by adding non-classifiable types and by increasing the percentage of negative pairs.

– Missing connections to the unknown page.
Table 6 shows how the score drops in QA architecture when we switch to the variant "without query all." We conclude that even the best architecture needs a meaningful information flow from the reference page itself and not only from the nearest page. That information flow is missing in triplet loss and pairwise classification.

To gain more insight, we tested the architectures on a different target value, which was defined as "does the text in these word-boxes have the same length." In this setting, the architectures achieved a significantly higher score of 0.7886. This supports our theory that the unclassified data (see above) was responsible for the underperformance of triplet loss and pairwise classification, since all data in the document were useful for the text lengths target.

4.2.1 Query answer

The query-answer architecture scored the best, with a micro F_1 score of 0.9290 with all the proposed architectural variants employed at once. In Table 6, we present an ablation study, showing that each of the components (field of view, query all, skip connection, filter, nearest search as defined in 3.4.4) related to QA architecture is clearly needed, as the score drops if any single one is turned off.

Compared with the previous model Table 1, an improvement of 0.0825 in the F_1 score is achieved. Also, the experiment on the anonymized dataset and the dataset with only text features shows that the architecture is versatile enough to not fail the task and to show similar improvement in the score on the anonymized dataset (by 0.0950). It also verifies that all the visual, geometric, and textual features are important for good quality results.

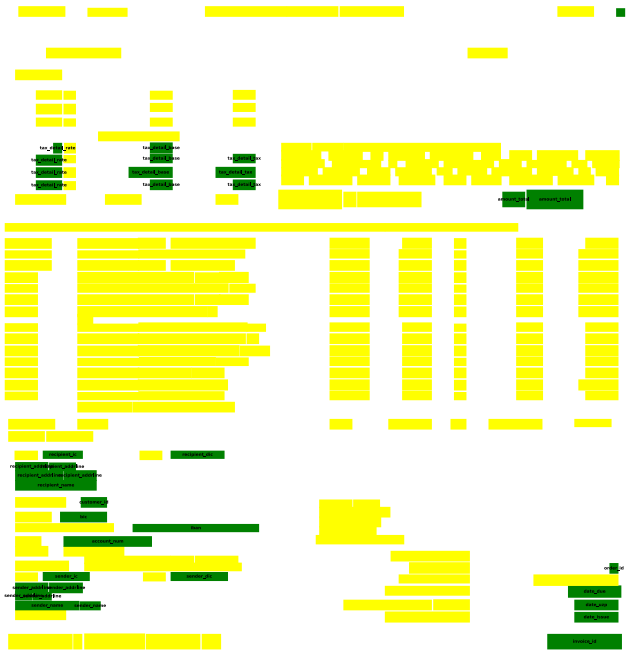


Fig. 7 Best classification result of the QA model—only true positives and true negatives can be seen (green = true positive; yellow = true negative) (colour figure online)

4.2.2 Qualitative comparison

We conclude with more qualitative analysis, specifically, a comparison of the best QA model and the simple data extraction model.

To start, we select pages from a random subset of the test set and present example visualizations in Figs. 7, 8 and 9 to illustrate a manual inspection of prediction visualizations. They show the best prediction from the query-answer model (Fig. 7), the worst prediction from the query-answer model (Fig. 8), and finally the worst prediction of the simple data extraction model (Fig. 9). A successfully classified word-box is a true positive, while successfully classified unimportant text is a true negative. Misclassifications of true positives ("miss") and true negatives ("extra") are also indicated.

Both the simple data extraction model and the QA model have examples of pages that look like results in Fig. 7 and are 100% perfectly extracted (or classified). However, the results vary in the worst cases, which is why examples from both models are presented in Figs. 8 and 9.

Motivated by this difference, we can look at which classes both models extract best and worst. Those scores are presented in Table 7.

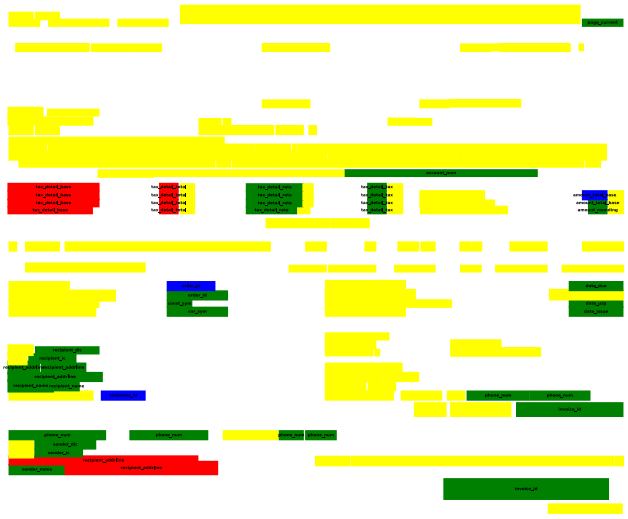


Fig. 8 Worst result of the QA model. Each blue and red area denotes a mistake (blue = misclassified as true negative; red = misclassified as true positive) (colour figure online)

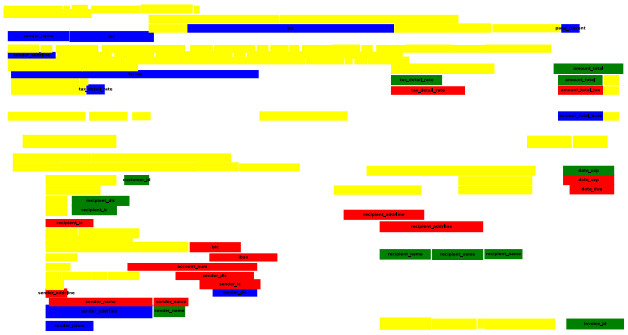


Fig. 9 Worst result of the simple data extraction model. Note the minimal count of true positive areas and the dominance of errors (green = true positive; yellow = true negative; blue = misclassified as true negative; red = misclassified as true positive) (colour figure online)

This detailed inspection shows that both models excel at classes that usually appear together (but not in any fixed layout or order) in business documents. Those classes are all the recipient information (DIC, IC, Spec symbol) and the sender information. Moreover, recipient information is usually required information on an invoice, and thus, it is the most frequent class and the network easily excels at detecting it.

Interestingly, page numbering could be seen as an easy class to classify, but the previous model actually classified it with a very low score. The score jumps to a very high value when we switch to the QA model though. One possible reason for this improvement is that the page number usually appears alone somewhere near an edge of the page. Thus, its nearest word-boxes are random and might cause confusion for the GCN module and for convolution over the sequence as well. When a similar page is presented to the model, the score jumps higher possibly because the nearest page might have page numbering in a similar position.

Table 7 Best and worst classes performance of QA model and simple data extraction model

Best- and worst-performing fields (and their scores)	Simple—test micro F_1 score	QA—test micro F_1 score
Worst classes of simple data extraction model		
Page current	0.30	0.90
Page total	0.35	0.88
Terms	0.62	0.78
Best classes of simple data extraction model		
Recipient DIC	0.94	0.96
Recipient IC	0.94	0.97
Spec symbol	0.94	0.96
Worst classes of QA		
Order ID	0.65	0.75
Terms	0.62	0.78
Customer ID	0.75	0.83
Best classes of QA		
Sender IC	0.93	0.96
Spec symbol	0.94	0.96
Recipient IC	0.94	0.97

The QA model, as a possible improvement from our previous results, holds an important property we desire. In particular, we have verified that the score for all classes has increased uniformly by at least 0.02 points (median gain being 0.04), even for the previously best-performing classes. This property is important to verify, since the QA architecture incorporates the simple data extraction model, and we expect it to "fall back" to it when the nearest page does not provide enough information. If this fallback does not happen, some gradients would not be propagated correctly.

The improvement of some fields by only roughly 2% may be seen as a small improvement. But in reality (as stated in the introduction), the 2% improvement translates into less time and effort for companies processing more than 500 invoices per month. This reduced time and effort translates to more than $1000 of savings per month as well as a reduction in the company's carbon footprint.

5 Conclusions

Multiple baselines were provided and evaluated to gain more knowledge about the data and establish the need for bigger and more complicated models.

We have designed multiple ways to incorporate similarity and memory—in terms of access to existing data—into the existing data extraction model, and we studied the gains of

various models in a fixed setting. The successful gain was an 8.25% increase in the F_1 score compared with our previous results by using "query-answer" inspired architecture. By the referenced heuristics, this improvement translates roughly into $4000 dollars savings of manual work per month for a medium-sized company.

We have verified that all possible parts of the architecture are needed in the training and prediction of the QA model to achieve the highest score. Moreover, the improvement holds even in the case of the anonymized dataset.

In a qualitative analysis of the results, we showed that the score improvement is meaningful across all the classes. Furthermore, the solution was shown to significantly boost the previously most-problematic classes.

For the other models that underperform (as triplet loss and pairwise classification), we have identified the possible cause as most words on the page not belonging to any class, and we supported the hypothesis with an additional experiment.

Further work could incorporate a way to create some artificial classes and measure the supposed increase in score for the triplet and pairwise classification models.

From a quantitative point of view, an opportunity exists to explore and improve extraction scores by tuning all the possible parameters of the system, namely the number of the nearest pages used and the quality of the page embeddings. The page embeddings can possibly be jointly trained with the word-box classifier.

Qualitatively, we identify some possible new research questions:

- What is the effect of the size of the datasets? By exploring the effect of the size of the training dataset and/or the search space for the nearest pages, we could ask if (and when) the model needs to be retrained and what a sample of a difficult-to-extract document looks like.
- How to improve the means of generalization? Currently, the method generalizes to unseen documents. In theory, we could desire a method to generalize to new classes of words because in the current approach, the model needs to be retrained if a new class is desired to be detected and extracted.

In practice, our solution has a particular strength that transforms these two points from potential hurdles to interesting research questions. The model can fit into just one consumer-grade GPU and trains from scratch within 4 days at most using only one CPU process. Compared with recent state-of-the-art NLP methods that take ample resources to train (such as [4]), our model can be retrained and/or fine-tuned for any particular use-case quickly and effectively (even more with transfer learning techniques). Thus, any assortment of problems are solved by the industrial standards [5].

As a part of this work, the dataset and source codes are published in [22]. This resource should enable wider research of deep learning models for information extraction because it was previously impossible for researchers to collect a dataset of this size and quality.

Acknowledgements The Rossum.ai team deserves thanks for providing the data and background that enabled the development and growth of this work.

Author Contributions The principal author is responsible for the study concept and design, execution, coding, and research. The rest of the Rossum team is responsible for data acquisition, annotation, and storage and for the creation of a working product and environment that enabled a scientific study of this scope.

Funding This work was supported by the Grant SVV-2020-260583. Partial financial support was received from Rossum and Charles University.

Data availability An anonymized version of the dataset is publicly available at [22], together with all the codes. The improvement on previous results can be reproduced using the anonymized data without disclosing any sensitive information.

Code availability The source codes are publicly available from a GitHub repository [22].

Declarations

Conflict of interest The author (Martin Holeček) has received financial support from Rossum and from Charles University, where he is currently pursuing a PhD. The author has an employment and/or contractual relationship with Rossum, Medicalc, and AMP Solar Group.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: large-scale machine learning on heterogeneous systems. Software available from tensorflow.org (2015). https://www.tensorflow.org/
2. Abbasi, A., Chen, H., Salem, A.: Sentiment analysis in multiple languages: feature selection for opinion classification in web forums. ACM Trans. Inf. Syst **26**(3), 1–34 (2008). https://doi.org/10.1145/1361684.1361685
3. Arsenault, M.O.: Lossless triplet loss (2018). https://towardsdatascience.com/lossless-triplet-loss-7e932f990b24
4. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020)
5. Burkov, A.: Machine Learning Engineering. True Positive Incorporated (2020)
6. Cai, Q., Pan, Y., Yao, T., Yan, C., Mei, T.: Memory matching networks for one-shot image recognition. In: Proceedings of the IEEE

Conference on Computer Vision and Pattern Recognition, pp 4080–4088 (2018)
7. Chen, Z., Huang, L., Yang, W., Meng, P., Miao, H.: More than word frequencies: authorship attribution via natural frequency zoned word distribution analysis. CoRR (2012). arXiv:1208.3001
8. Coüasnon, B., Lemaitre, A.: Recognition of Tables and Forms, pp. 647–677. Springer, London (2014). https://doi.org/10.1007/978-0-85729-859-1_20
9. Cowie, J., Lehnert, W.: Information extraction. Commun. ACM **39**, 80–91 (1996)
10. Dalvi, B.B., Cohen, W.W., Callan, J.: Websets: extracting sets of entities from the web using unsupervised information extraction. In: Proceedings of the Fifth ACM International Conference on Web Search and Data Mining, pp. 243–252 (2012)
11. d'Andecy, V.P., Hartmann, E., Rusinol, M.: Field extraction by hybrid incremental and a-priori structural templates. In: 2018 13th IAPR International Workshop on Document Analysis Systems (DAS), pp. 251–256 (2018). https://doi.org/10.1109/DAS.2018.29
12. Dhakal, P., Munikar, M., Dahal, B.: One-shot template matching for automatic document data capture. In: 2019 Artificial Intelligence for Transforming Business and Society (AITB), vol. 1, pp. 1–6 (2019). https://doi.org/10.1109/AITB48515.2019.8947440
13. Eloff, R., Engelbrecht, H.A., Kamper, H.: Multimodal one-shot learning of speech and images. In: ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 8623–8627. IEEE (2019)
14. Fei-Fei, L., Fergus, R., Perona, P.: One-shot learning of object categories. IEEE Trans. Pattern Anal. Mach. Intell. **28**(4), 594–611 (2006)
15. Felix, R., Sasdelli, M., Reid, I., Carneiro, G.: Multi-modal ensemble classification for generalized zero shot learning. arXiv preprint arXiv:1901.04623 (2019)
16. Galassi, A., Lippi, M., Torroni, P.: Attention in natural language processing. Computation and Language (2020)
17. Ghosh, S.K., Valveny, E.: R-phoc: segmentation-free word spotting using CNN. In: 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), vol. 1, pp. 801–806. IEEE (2017)
18. Göbel, M., Hassan, T., Oro, E., Orsi, G.: ICDAR 2013 table competition. In: 2013 12th International Conference on Document Analysis and Recognition (ICDAR), pp. 1449–1453. IEEE (2013)
19. Grigorescu, S.M.: Generative one-shot learning (GOL): s semi-parametric approach to one-shot learning in autonomous vision. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 7127–7134. IEEE (2018)
20. Hamza, H., Belaïd, Y., Belaïd, A.: Case-based reasoning for invoice analysis and recognition. In: Weber, R.O., Richter, M.M. (eds.) Case-Based Reasoning Research and Development, pp. 404–418. Springer, Berlin (2007)
21. Holecek, M., Hoskovec, A., Baudis, P., Klinger, P.: Table understanding in structured documents. In: 2019 International Conference on Document Analysis and Recognition Workshops (ICDARW), vol. 5, pp. 158–164 (2019). https://doi.org/10.1109/ICDARW.2019.40098
22. Implementation details for this work, source codes and curated anonymized dataset to reproduce results. https://github.com/Darthholi/similarity-models
23. Jean-Pierre Tixier, A., Nikolentzos, G., Meladianos, P., Vazirgiannis, M.: Graph Classification with 2D Convolutional Neural Networks. arXiv e-prints arXiv:1708.02218 (2017)
24. Koch, G., Zemel, R., Salakhutdinov, R.: Siamese neural networks for one-shot image recognition. In: ICML Deep Learning Workshop, vol. 2. Lille (2015)
25. Kosala, R., Van den Bussche, J., Bruynooghe, M., Blockeel, H.: Information extraction in structured documents using tree automata induction. In: Elomaa, T., Mannila, H., Toivonen, H. (eds.) Principles of Data Mining and Knowledge Discovery, pp. 299–311. Springer, Berlin (2002)
26. Krieger, F., Drews, P., Funk, B., Wobbe, T.: Information extraction from invoices: a graph neural network approach for datasets with high layout variety (2021)
27. Lake, B., Salakhutdinov, R., Gross, J., Tenenbaum, J.: One shot learning of simple visual concepts. In: Proceedings of the Annual Meeting of the Cognitive Science Society, vol. 33 (2011)
28. Lake, B.M., Salakhutdinov, R., Tenenbaum, J.B.: The omniglot challenge: a 3-year progress report. Curr. Opin. Behav. Sci. **29**, 97–104 (2019)
29. Lampinen, A.K., McClelland, J.L.: One-shot and few-shot learning of word embeddings. CoRR (2017). arXiv:1710.10280
30. Lin, Z., Davis, L.S.: Learning pairwise dissimilarity profiles for appearance recognition in visual surveillance. In: Advances in Visual Computing, pp. 23–34. Springer, Berlin (2008)
31. Liu, R., Lehman, J., Molino, P., Petroski Such, F., Frank, E., Sergeev, A., Yosinski, J.: An Intriguing Failing of Convolutional Neural Networks and the CoordConv Solution. arXiv e-prints arXiv:1807.03247 (2018)
32. Liu, X., Gao, F., Zhang, Q., Zhao, H.: Graph convolution for multimodal information extraction from visually rich documents. arXiv preprint arXiv:1903.11279 (2019)
33. Lohani, D., Abdel, B., Belaïd, Y.: An Invoice Reading System using a Graph Convolutional Network. In: International Workshop on Robust Reading. PERTH, Australia (2018). https://hal.inria.fr/hal-01960846
34. Manual typing is expensive: The tco of invoice data capture (part 2). https://rossum.ai/blog/manual-typing-is-expensive-the-tco-of-invoice-data-capture-part-2/
35. Mehrotra, A., Dukkipati, A.: Generative adversarial residual pairwise networks for one shot learning. arXiv preprint arXiv:1703.08033 (2017)
36. Meta learning papers. https://github.com/floodsung/Meta-Learning-Papers
37. Mishra, A., Krishna Reddy, S., Mittal, A., Murthy, H.A.: A generative model for zero shot learning using conditional variational autoencoders. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 2188–2196 (2018)
38. Nadeau, D., Sekine, S.: A survey of named entity recognition and classification. Lingvisticae Investigationes **30**(1), 3–26 (2007)
39. Narasimhan, H., Pan, W., Kar, P., Protopapas, P., Ramaswamy, H.G.: Optimizing the multiclass f-measure via biconcave programming. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 1101–1106. IEEE (2016)
40. Nie, Y.P., Han, Y., Huang, J.M., Jiao, B., Li, A.P.: Attention-based encoder–decoder model for answer selection in question answering. Front. Inf. Technol. Electron. Eng. **18**(4), 535–544 (2017)
41. Niepert, M., Ahmed, M., Kutzkov, K.: Learning Convolutional Neural Networks for Graphs. arXiv e-prints arXiv:1605.05273 (2016)
42. Palm, R., Laws, F., Winther, O.: Attend, copy, parse end-to-end information extraction from documents. In: 2019 International Conference on Document Analysis and Recognition (ICDAR), pp. 329–336 (2019)
43. Paul, A., Krishnan, N.C., Munjal, P.: Semantically aligned bias reducing zero shot learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7056–7065 (2019)
44. Peng, H.: A comprehensive overview and survey of recent advances in meta-learning. arXiv:2004.11149 (2020)
45. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners. OpenAI Blog **1**(8), 9 (2019)

46. Riba, P., Dutta, A., Goldmann, L., Fornes, A., Ramos, O., Llados, J.: Table detection in invoice documents by graph neural networks, pp. 122–127 (2019). https://doi.org/10.1109/ICDAR.2019.00028
47. Rossum's blogpost "extracting invoices using ai" at medium.com. https://medium.com/@bzamecnik/extracting-invoices-using-ai-in-a-few-lines-of-code-96e412df7a7a
48. Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., Lillicrap, T.: One-shot learning with memory-augmented neural networks. arXiv preprint arXiv:1605.06065 (2016)
49. Smith, D., Lopez, M.: Information extraction for semi-structured documents. In: Proceedings of the Workshop on Management of Semistructured Data (1997)
50. Tenhunen, M., Penttinen, E.: Assessing the carbon footprint of paper vs. electronic invoicing (2010). https://aisel.aisnet.org/acis2010/95
51. Thakurdesai, N., Raut, N., Tripathi, A.: Face recognition using one-shot learning. Int. J. Comput. Appl. **182**, 35–39 (2018). https://doi.org/10.5120/ijca2018918032
52. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is All You Need. arXiv e-prints arXiv:1706.03762 (2017)
53. Vinyals, O., Blundell, C., Lillicrap, T.P., Kavukcuoglu, K., Wierstra, D.: Matching networks for one shot learning. CoRR (2016). arXiv:1606.04080
54. Wang, P., Liu, L., Shen, C., Huang, Z., van den Hengel, A., Tao Shen, H.: Multi-attention network for one shot learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2721–2729 (2017)
55. Xu, L., Wang, Y., Li, X., Pan, M.: Recognition of handwritten Chinese characters based on concept learning. IEEE Access **7**, 102039–102053 (2019)
56. Yang, Z., He, X., Gao, J., Deng, L., Smola, A.: Stacked attention networks for image question answering. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 21–29 (2016)
57. Yim, J., Kim, J., Shin, D.: One-shot item search with multimodal data. arXiv:1811.10969 (2018)
58. Yin, W.: Meta-learning for few-shot natural language processing: a survey. arXiv:2007.09604 (2020)

Part V

Conclusion

A method that lies in the hearth of information extraction for automatization of business documents processing was developed by taking well motivated and scientifically-grounded small steps. During the journey, it was shown that no other previously published method and/or dataset is suitable for the task and that all the parts and bits of the model, the data and the training and validation process are highly important for the state-of-the-art success in the task.

The main high-level concepts that have worked for reaching the goal were:

- Emulating the problem-solving approach of humans using neural networks.
- Giving the trainable model full access to all features and modalities of the data and dataset.
- Grouping relevant sub-tasks.

An anonymized dataset and full source codes were published to allow the exact reproducibility of the results and to assist in solving newly created open questions.

The side goal of computational efficiency was successfully met, as the system and the experiments allow for running on consumer-grade hardware without the need for transfer learning – as opposed to other recent state-of-the-art NLP and computer vision models.

Bibliography

Andriy Burkov. *Machine Learning Engineering*. True Positive Incorporated, 2020. 1.1

J. Cowie and W. Lehnert. Information extraction. *Commun. ACM*, 39:80–91, 1996. 2

Shaveta Dargan, Munish Kumar, Maruthi Rohit Ayyagari, and Gulshan Kumar. A survey of deep learning and its applications: A new paradigm to machine learning. *Archives of Computational Methods in Engineering*, pages 1–22, 2019. 1.1

Kyle Genova, Forrester Cole, Aaron Maschinot, Aaron Sarna, Daniel Vlasic, and William T. Freeman. Unsupervised training for 3d morphable model regression. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. A.3

Max Göbel, Tamir Hassan, Linda Oro, and Giorgio Orsi. Icdar 2013 table competition. In *Proc. of the 12th Intl Conf. on Document Analysis and Recognition (ICDAR)*, pages 1449–1453, 2013. doi: 10.1109/ICDAR.2013.292. URL `http://dx.doi.org/10.1109/ICDAR.2013.292`. Competition Report. A.1, A.2

M. Holecek, A. Hoskovec, P. Baudis, and P. Klinger. Table understanding in structured documents. In *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*, volume 5, pages 158–164, Sep. 2019. doi: 10.1109/ICDARW.2019.40098. 1, 2, 2.1, V

Martin Holecek. Rossum.ai blogpost, 2018a. URL `https://rossum.ai/blog/inside-line-items-our-progress-and-evaluation-techniques/`. 1, A

Martin Holecek. Rossum.ai blogpost, 2018b. URL `https://rossum.ai/blog/tensorflow-for-cropping-and-rendering/`. 1, A

Martin Holecek. https://colab.research.google.com/drive/1bn-pbzqsl5uhvng5ltazok2ibgjl6kws, 2018c. URL `https://colab.research.google.com/drive/1Bn-pbZQsl5uhvnG5LTAZoK2iBGJL6kWS`. A.3

Martin Holecek. Rossum.ai blogpost, 2018d. URL `https://rossum.ai/blog/update-line-item-extraction-invoices/`. 1, A

Martin Holecek. Codes for simple invoice generator, 2019. URL `https://github.com/Darthholi/DocumentConcepts`. 1, III, 8.2

Martin Holecek. Implementation details for this work, source codes and curated anonymized dataset to reproduce results, 2020. URL `https://github.com/Darthholi/similarity-models`. 1

Martin Holeček. Learning from similarity and information extraction from structured documents. *International Journal on Document Analysis and Recognition (IJDAR)*, pages 1–17, 2021. 1, 2, V

Raymond Kosala, Jan Van den Bussche, Maurice Bruynooghe, and Hendrik Blockeel. Information extraction in structured documents using tree automata induction. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Principles of Data Mining and Knowledge Discovery*, pages 299–311, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45681-0. A.1

Todd J. Kosloff. Fast image filters for depth-of-field postprocessing. 2010. A.3

Todd Jerome Kosloff, Justin Hensley, and Brian A. Barsky. Fast filter spreading and its applications. Technical Report UCB/EECS-2009-54, EECS Department, University of California, Berkeley, Apr 2009. URL `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-54.html`. A.3

Andrew Long. Data science performance metrics for everyone, 2018. URL `https://towardsdatascience.com/data-science-performance-metrics-for-everyone-4d68f4859eef`. A.2

Akmal Jahan Mac and Roshan G Ragel. Locating tables in scanned documents for reconstructing and republishing (iciafs14), 2014. A.1

HP Newquist. *The Brain Makers, Second Edition.* The Relayer Group, New York, NY, 2018. 1.1

Hwee Tou Ng, Chung Yong Lim, and Jessica Li Teng Koo. Learning to recognize tables in free text. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*, ACL '99, pages 443–450, Stroudsburg, PA, USA, 1999. Association for Computational Linguistics. ISBN 1-55860-609-3. doi: 10.3115/1034678.1034746. URL `https://doi.org/10.3115/1034678.1034746`. A.1

N O' Mahony. Advances in computer vision. *Advances in Intelligent Systems and Computing*, 2020. ISSN 2194-5365. doi: 10.1007/978-3-030-17795-9. URL `http://dx.doi.org/10.1007/978-3-030-17795-9`. 1.1

Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018. A.1

Bohumir Zamecnik. On rendering with complex camera models. 2012. A.3

List of publications

- Table understanding in structured documents (Holecek et al. [2019])
- Learning from similarity and information extraction from structured documents (Holeček [2021])

A. Attachments - related, co-authored and non-peer reviewed texts

The following sections contains copies of blog-posts (Holecek [2018d,a,b]) - texts that I have authored or co-authored while working on topics related to information extraction, table detection and/or table extraction and are included here to provide the full context of the research, to inform about other approaches (that did not ultimately yield results and so were omitted from the main text) and ultimately as another source of interesting and related information.

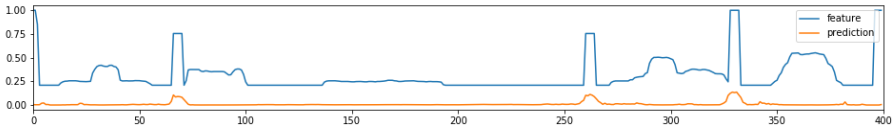
A.1 Update on Rossum's line item extraction from invoices

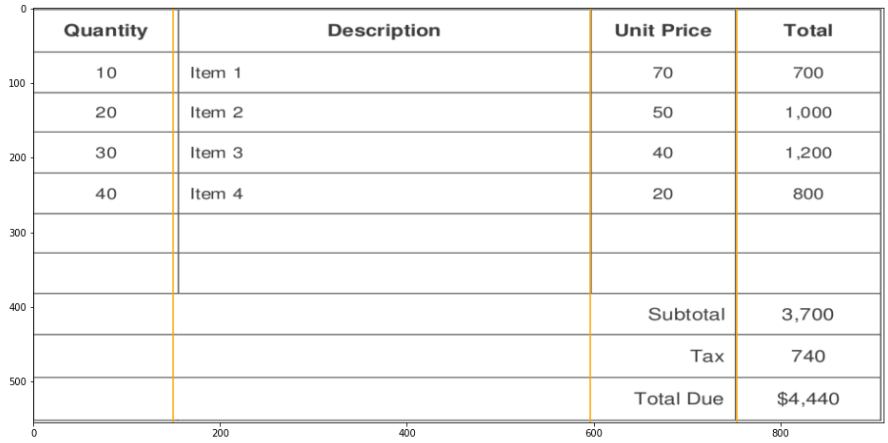
At Rossum, we have been hard at work researching line item extraction from invoices. It is a daunting task, but we are not afraid. We know you have been waiting patiently to hear from us, so we have put together a brief update of what has been going on in research, as well as some conclusions we have made from the results thus far. There is still more to learn – but we now know we are on the right path! Usually, for automatic processing, line item extraction from invoices is needed. For starters, it could be connected to a personal database of expenses, serve as an automatic ending step for storage systems, or even simplify the hard work of auditors needing to pair orders and invoices. In the scientific world, this problem translates to table detection and table extraction or understanding. During a competition called ICDAR 2013 Göbel et al. [2013], that compared submissions with commercial products, originated table extractions from HTML Kosala et al. [2002] or pure texts Ng et al. [1999]. The goal was to extract a table from a scientific, or any other type of article. Usually, the first algorithms used only heuristics Mac and Ragel [2014], getting to an 80% success rate only on specific types of tables (eg. tables with lines). From another point of view, there was recently a big boom of object detection algorithms (for example, for AI driven automatic cars), one of the most successful being YOLO Redmon and Farhadi [2018]. How did these algorithms fare on our data set, where an invoice is, from a perspective, a schematic layout of tabular structures? We were not happy with the results of heuristics and even with YOLO, at first. Even on simple tables, the imagined success was not achieved. So we need to dig into more experiments, trying to teach the neural network to paint where the table resides and connect rows and columns (aiming to extract simple tables at the beginning). From first glance, it seems we are on the right track – the algorithm can detect whole tables and even identify individual rows as shown below:

In parallel, we also work on hybrid methods (a combination of the older methods mentioned above and artificial intelligence). Here are some examples of column search, based on custom image-processing features and learned 1D dilated segmentation.

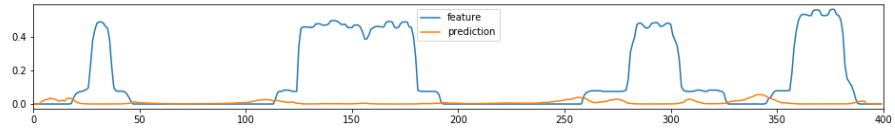
For a simple insight, these images show how we want to decide on the positions of columns from the image features shown in blue (some of them can be thought of as a position histogram of all ink used in the image). The orange line is a

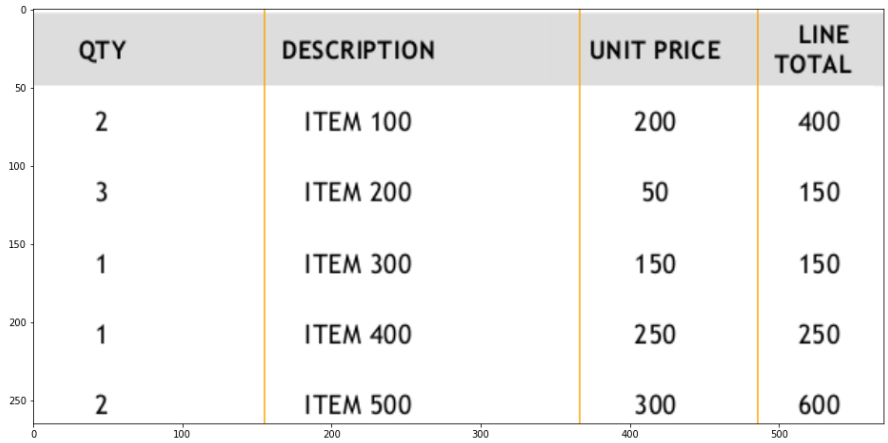
learned decision, from which, if we select its peaks, we get the nice column splits as shown on the images of tables.



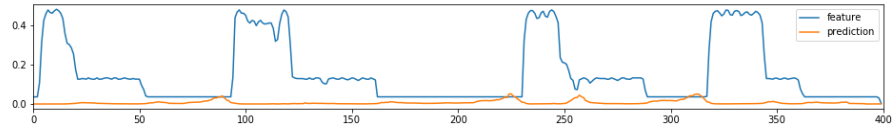


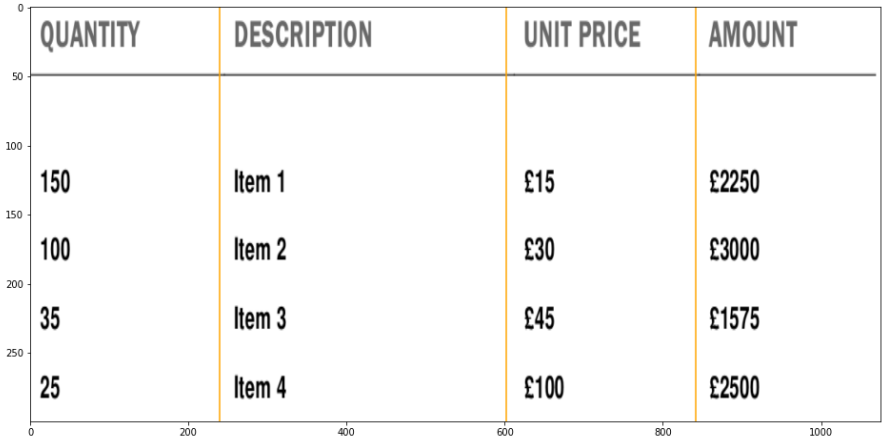
Quantity	Description	Unit Price	Total
10	Item 1	70	700
20	Item 2	50	1,000
30	Item 3	40	1,200
40	Item 4	20	800
		Subtotal	3,700
		Tax	740
		Total Due	$4,440





QTY	DESCRIPTION	UNIT PRICE	LINE TOTAL
2	ITEM 100	200	400
3	ITEM 200	50	150
1	ITEM 300	150	150
1	ITEM 400	250	250
2	ITEM 500	300	600



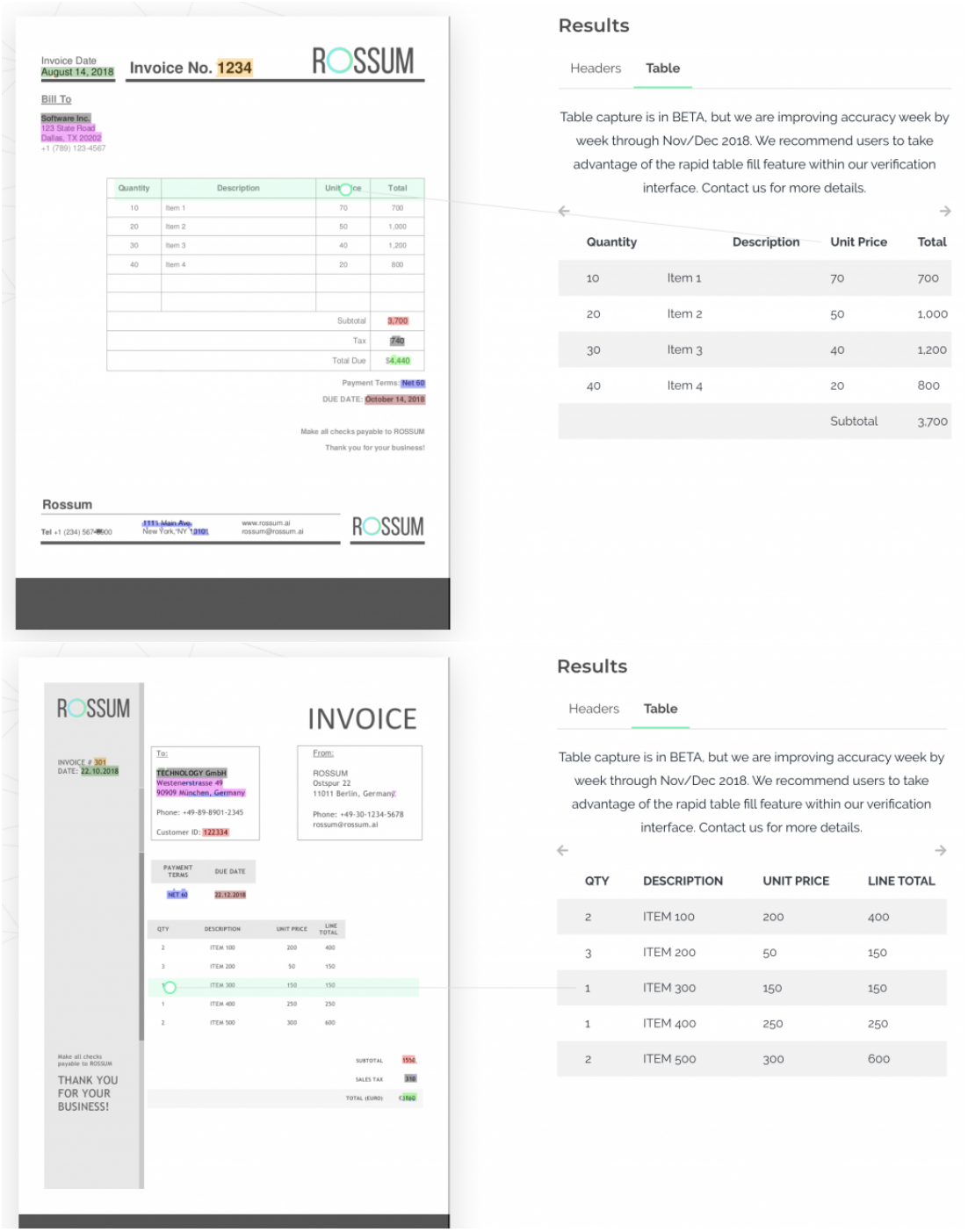


Universal table extraction from invoices without templates is still an unsolved problem across all data capture software. But from the moment we gave it as the main research focus here at Rossum, we have already seen huge progress towards the goal, some of which we wanted to share above. Right now, we are fitting the last missing pieces of the end-to-end extraction pipeline, as well as experimenting with ways to make our neural networks even more accurate. While our current focus is perfectly extracting information from simpler tables, for longer term work, we have our sights set on complex cases like overlapping columns or nested tables.

A.2 Inside line items: Our progress and evaluation techniques

The gist of the story, short & sweet, is that users of the Rossum verification interface can now enjoy dramatic quality improvement in the semi-automatic line item capture we offer, and fully automatic table extraction is now available in our Data Extraction API as an experimental feature.

To follow up on our previous line items blog post, let us show you a few examples. It simply works for many layouts already.



One number to describe our precision Current industry and quality standards demand evaluated measurements on the quality of service – the days are gone when 'it works sometimes' is better than having no service at all. Usually, in the data science field, performance of an algorithm is evaluated using common standard metrics Long [2018] that can be computed when we know the algorithms' output (predictions) and the truth value (gold standard or annotations). A quick example would be to count the number of mistakes (in the text, referred later as 'bads') and the number of correct outputs (referred to as 'goods').

When we look at two tables, it might not be clear what exactly to count. So, what does the literature have to say? Usually, our problem is referred to as 'table detection' and 'table extraction', where first we need to find the table and then we need to extract the data from it. In the literature, there was a metric based

on connections in the table, that was first used on ICDAR 2013 Göbel et al. [2013] to evaluate heuristic table detection and extraction from scientific texts. For table detection, they used a simple comparison of similarities of rectangles, called 'intersection over union', where a 'good' detection is one with an IoU score above 0.7, arguing that we do not need to draw the rectangle around the table in an overly precise manner.

The ICDAR metric for table extraction is based on a simple idea – that tables tell us about column-wise and row-wise relations where we can count how well the relations are fulfilled. The metric is defined as follows: for each cell in the table, let's take the first non-empty (a cell that has a meaningful text) right-hand neighbor and first bottom neighbor; record the two cells' distance, texts and direction, thus forming a record of four items. For the predicted and ground truth table, we get two lists of records, which we can match against each other and count common records ('goods'), missed records and extra records. From these integer values, the data science field commonly calculates metrics as precision, recall and f1 score – so far so good.

Human-like numbers We have implemented the metric, used it a bit, and found out that it is not the easiest to explain and also does not strictly follow how a human would judge the table prediction against the gold standard. So, we came out with our own metric that is trying to mimic how humans would look at it. If I had two tables and had to say how similar they are, the first problem I would face is that the tables can be of different sizes. To reduce our problem, I would cut out rows and columns, so that the tables would be the same (smaller) size. Which columns and rows should we cut? I would look for those that have the craziest content, looking like nothing from the second table (from implementation point of view, the keyword is edit distance). We would have two tables of the same size and then I could look at each cell and decide if it matches or not.

The numbers (good and bad), together with the number of cells cut from both tables (meaning a miss or an extra, based on which table we cut it from), can give nicely defined metrics. The algorithm for cutting out rows or columns does not need to be overly clever, because as the user, I would not spend a lot of time looking at all combinations. We want to capture the feeling in the metrics. Validation is done, when we look at the table and it's computed score and say 'that looks like 60%, nodding our heads in wise agreement. The nice property here is that if the prediction is perfect, we will get a perfect score regardless of the used algorithm for deciding which rows/columns to cut (and similarly – no table extraction would yield 0%).

An example: Let's take this table:

ORDERED	QUANTITY *BK. ORD.	CODE	DESCRIPTION	UNIT PRICE	TAX	PRODUCT TOTAL
5		1234	CIRCUIT BRKRS/SWITCHES OR PART	10.00		50.00

Lets assume, it gets extracted as:

```
ORDERED SRKAITEY    CODE        DESCRIPTION                    UNIT PRICETAX PRODUCT TOTAL
5                   1234        CIRCUIT BRKRS/SWITCHES OR PART 10.0050.00
```

Then the number of rows (2) is the same in both tables, but the number of

columns differ (the algorithm merged some columns), so we would cut out the worse columns from the original (specifically 3: quantity, tax, product total) and we would end with statistics like this:

- good: 5, CODE, 1234, DESCRIPTION, CIRCUIT BR. . . ,
- bad: ORDERED SRKAITEY, UNIT PRICETAX PRODUCT TOTAL, 10.0050.00

and adding the number of cells we did cut out:

- miss: 6 (= 3 cols, each with 2 rows)

Final Note We get the score for each table in each document, and in our development process we are averaging these numbers (f1 scores) into one final score. This process is called 'macro metrics', as opposed to 'micro metrics', which would first sum all the goods, bads, misses and extras and then compute precision, recall, and f1 score from it.

We wanted to share our methodology of evaluation in a way that could inspire others (not only data scientists), open a discussion and provide some links to the literature that has inspired us. Keep your eyes peeled for more updates in the near future as we continue our quest to solve line item extraction from invoices and feel free to reach out and comment on the methodology (or implement it as you wish).

A.3 Tensorflow for cropping and rendering

In tensorflow, there are many usable features for tasks besides just for learning. One worth mentioning is the function for rendering framed rects for given boundingboxes that can be used for fast evaluation of predictions of an image. But what if we want to get fancy? We have recently faced a task, to crop given areas from an image, process them into features and render them back onto the image. This could be done in keras or tensorflow so that it can be fast and embedded in a model. Or even possibly – overlay an image, not with framed boundingboxes, but alphablend with filled rectangles. To note, we want a feature vector rendering, not a full graphics renderer with tensorflow, which already exists (at least in the form of this example Genova et al. [2018]).

Here we will be introducing the concepts with the code, but we have also published the code Holecek [2018c] that should be able to run in ipython notebook.

1) Cropping There is a nice function tf.image.crop_and_resize in tensorflow, that does the cropping almost exactly how we want it. It needs a bit of care because the input types need an index to the batch of images provided. Our original intent was to have bboxes saved per batch.

Below is the full code, along with a simple test, and an example on how to use the function as a keras layer (we are using it that way).

```
import itertools
import tensorflow as tf
import numpy as np
from keras.layers import Lambda
```

```
def tf_crop_and_resize_batches_of_arrays(image_input, boxes_input, crop_size
    =(10, 10)):
    """
    The dimension of boxes is [batch, maxnumperbatch, 4]  # some can be just a
        bogus-zero-fill
    The dimension of the output should be [batch, maxnumperbatch, crop_0, crop_1
        , 1] crops.
    """

    bboxes_per_batch = tf.shape(boxes_input)[1]
    batch_size = tf.shape(boxes_input)[0]  # should be the same as image_input.
        shape[0]

    # the goal is to create a [batch, maxnumperbatch] field of values,
    #  which are the same across batch and equal to the batch_id
    # and then to reshape it in the same way as we do reshape the boxes_input to
         just tell tf about
    #  each bboxes batch (and image).
    index_to_batch = tf.tile(tf.expand_dims(tf.range(batch_size), -1), (1,
        bboxes_per_batch))

    # now both get reshaped as tf wants it:
    boxes_processed = tf.reshape(boxes_input, (-1, 4))
    box_ind_processed = tf.reshape(index_to_batch, (-1,))


    # the method wants boxes = [num_boxes, 4], box_ind = [num_boxes] to index
        into the batch
    # the method returns [num_boxes, crop_height, crop_width, depth]

    tf_produced_crops = tf.image.crop_and_resize(
        image_input,
        boxes_processed,
        box_ind_processed,
        crop_size,
        method='bilinear',
        extrapolation_value=0,
        name=None
    )
    new_shape = tf.concat([tf.stack([batch_size, bboxes_per_batch]), tf.shape(
        tf_produced_crops)[1:]], axis=0)
    crops_resized_to_original = tf.reshape(tf_produced_crops,
                                           new_shape)
    return crops_resized_to_original


def keras_crop_and_resize_batches_of_arrays(image_input, boxes_input, crop_size
    =(10, 10)):
    """
    A helper function for tf_crop_and_resize_batches_of_arrays,
     assuming, that the crop_size would be a constant and not a tensorflow
         operation.
    """

    def f_crop(packed):
        image, boxes = packed
        return tf_crop_and_resize_batches_of_arrays(image, boxes, crop_size)

    return Lambda(f_crop)([image_input, boxes_input])


def test_crops():
    # the intended usage:

    crop_size = (10, 10)

    image_input = np.ones((2, 200, 200, 1))
    boxes_input = np.array([[[0.0, 0.0, 0.5, 0.5], [0.0, 0.0, 0.5, 0.5], [0.0,
        0.0, 1.0, 1.0]],
                            [[0.0, 0.0, 0.5, 0.5], [0.0, 0.0, 0.5, 0.5],
                                [0.0, 0.0, 1.0, 1.0]]])
```

```
    # the dimension of boxes is [batch, maxnumperbatch, 4]  # some can be just a
        bogus-zero-fill

    with tf.Session() as sess:
        image_input_ph = tf.placeholder(tf.float32, [None, None, None, None])
        boxes_input_ph = tf.placeholder(tf.float32, [None, None, 4])
        crop_result = sess.run(tf_crop_and_resize_batches_of_arrays(
            image_input_ph, boxes_input_ph, crop_size=crop_size),
                               feed_dict={image_input_ph: image_input,
                                   boxes_input_ph: boxes_input})
        assert np.all(crop_result == 1.0), "when cropping image full of ones, we
            should get all ones too!"
        assert crop_result.shape == (boxes_input.shape[0], boxes_input.shape[1],
                                     crop_size[0], crop_size[1], image_input.
                                         shape[-1])
```

2) Rendering rectangles In numpy, we would simply slice a tensor and assign the values. But in tensorflow, assignment is not a valid operation on tensors.

So what options do we have?

modify variables use python operation over the numpy code and make it a tensorflow op use custom kernels (which can be the fastest way, but will not be covered here) we can process each bbox separately and then sum it over the rendering plane we can process all bboxes at each pixel and ask if they belong there we can use sparse_to_dense operation, or scatter_nd which should be faster.

How would this work? We could produce lists of points that are inside of the rectangles, but that is not easy to make parallel because each bbox would have a different number of pixels inside.

Fortunately, we are also fans of computer graphics, so we know about spreading filters (more in our beloved colleague's thesis Zamecnik [2012] and in Kosloff et al. [2009], Kosloff [2010]) that uses, for nearly the same goal, a cumulative sum operation which is available in tensorflow. We will explain more in a moment.

Let's imagine a 1d case with a 1d image, where we would need to fill intervals. We cannot tell tensorflow directly to start at a specified left corner and then start filling until the specified right corner is found. But, if we could mark all left corners with +1 and all right corners with -1 and then use cumulative sum (from the left), then at each pixel, we would know how many intervals cover it. We can actually mark the edges/corners by using scatter_nd. Even more, we can also parallel the gathering of data for this function, because the number of edges or corners is, unlike the area, the same.

How can we do it for 2d (or higher dimensions)? Step by step, dimension after dimension – if we first generate the data for top and bottom edges, then we can go with cumulative sum from top to bottom and we would have the whole box filled. We only need to remember that the second phase of the process needs pluses at the top and minuses at the bottom so that it would start at the top and end at the bottom, as the asciiart picture shows:

```
..........     ..........     ..........
....+..-...     ....++++...     ....++++...
.......... ->  .......... ->  ....++++...
....-..+...     ....----...     ....++++...
..........     ..........     ..........
original       after cumsums:
              (left to right) (top to bottom)
```

And to produce the pluses and minuses like we want, we will iterate over the corners in python using itertools.product and to alter the signs, so that the right

+/- is produced. We will count the number of ones in the corner and index and assign pluses to evens and minus to odds. Notice, that this design choice will make the function produce boxes in any dimension. It is fixed before the tensorflow graph is generated (at compile time) which should be sufficient. A test function is provided in a simple form for only 2d. The whole code is presented below together with a python operation and a test.

```
def render_bboxes_pyfunc_2d(elems, target_shape):
    """
    2d only numpy + tf.py_func replacement for render_nd_bboxes_tf_spreading.
    For testing purposes.
    """
    # target_shape = [dimx, dimy,....]

    def py_render_boxes_2d(x_boxes_data, out_shape):
        # x will be a numpy array with the contents of the placeholder below
        if len(x_boxes_data.shape) <= 2:
            result = np.zeros(list(out_shape) + [x_boxes_data.shape[-1] - 2 *
                2], dtype=np.float32)
            for box in x_boxes_data:
                result[box[0]:box[2], box[1]: box[3], :] += box[4:]
        else:  # also batch dimension is provided
            result = np.zeros([x_boxes_data.shape[0]] + list(out_shape) + [
                x_boxes_data.shape[-1] - 2 * 2], dtype=np.float32)
            for i, batch in enumerate(x_boxes_data):
                for box in batch:
                    result[i, box[0]:box[2], box[1]: box[3], :] += box[4:]
        return result

    return tf.py_func(py_render_boxes_2d, [elems, target_shape], tf.float32)


def render_nd_bboxes_tf_spreading(elems, target_shape, ndim=2):
    """
    elems: tensor of size [..., n_boxes, 2*ndim + val_dim], where in the last
        dimension,
     there are packed edge coordinates and values (of val_dim) to be filled in
         the specified box.
    target_shape: list/tuple of ndim entries.
    returns: rendered image of size [elems(...), target_shape..., val_dim]
    ('elems(...)' usually means batch_size)
    """
    assert_shape_ndim = tf.Assert(tf.equal(tf.size(target_shape), ndim), [
        target_shape])
    assert_nonempty_data = tf.Assert(tf.greater(tf.shape(elems)[-1], 2*ndim), [
        elems])

    with tf.control_dependencies([assert_shape_ndim, assert_nonempty_data]):
        '''
        In 3d there must be another wall of minuses. looking like that:

        -   +
        .....
        +   -

        so when indexing [0,1] to ltrb... pluses are when there is even number
            of 0s in corner index, minuses when odd.
        '''
        el_ndim = len(elems.shape)
        # we do not access this property in tensorflow runtime, but in 'compile
            time', because, well, number of dimensions
        # should be known before

        assert el_ndim >= 2 and el_ndim <= 3, "elements should be in the form of
             [batch, n, coordinates] or [n, coordinates]"
        if el_ndim == 3:  # we use batch_size dimension also!
            bboxes_per_batch = tf.shape(elems)[1]
            batch_size = tf.shape(elems)[0]  # should be the same as image_input
                .shape[0]
            index_to_batch = tf.tile(tf.expand_dims(tf.range(batch_size), -1),
                (1, bboxes_per_batch))
```

```
            index_to_batch = tf.reshape(index_to_batch, (-1, 1))
        else:
            index_to_batch = None

        val_vector_size = tf.shape(elems)[-1] - 2 * ndim

        corner_ids = list(itertools.product([0, 1], repeat=ndim))
        corners_lists = []
        corners_values = []
        for corner in corner_ids:
            plus = sum(corner) {86385
                ef8c424def43c570938a9943967855f43526a201d16486274d8a74d2e91} 2
                == 0
            id_from_corner = [i + ndim * c for i, c in
                              enumerate(corner)]  # indexes a corner into [left,
                                   top, right, bottom] notation
            corner_coord = tf.gather(elems[..., 0: 2 * ndim], id_from_corner,
                axis=-1)
            corner_value = elems[..., 2 * ndim:] * (1 if plus else -1)  # last
                dimension is == val_vector_size
            if index_to_batch is not None:
                # if the operation is called in batches, remember to rehape it
                    all into one long list for scatter_nd
                # and add (concatenate) the batch ids
                corner_coord = tf.concat([index_to_batch, tf.reshape(
                    corner_coord, (-1, 2))], axis=-1)
                corner_value = tf.reshape(corner_value, (-1, val_vector_size))
            corners_lists.append(corner_coord)
            corners_values.append(corner_value)

        indices = tf.concat(corners_lists, axis=0)
        updates = tf.concat(corners_values, axis=0)
        shape = tf.concat([tf.shape(elems)[:-2], target_shape, [val_vector_size
            ]], axis=0)

        dense_orig = tf.scatter_nd(
            indices,
            updates,
            shape=shape,
        )

        dense = dense_orig
        for dim in range(ndim):
            # we want to start from the axis before the last one. The last one
                is the value dimension, and
            # the first dimensions might be the batched dimensions
            dense = tf.cumsum(dense, axis=-2-dim, exclusive=False, reverse=False
                , name=None)

        return dense

def test_render_bboxes_2d():
    # test without batch_size dimension
    elems = [[0, 1, 3, 10, 1], [1, 13, 4, 17, 1], [0, 1, 3, 10, 1]] + [[8, 9,
        15, 15, 1]]*1000
    target_shape = [20, 20]

    with tf.Session() as sess:

        elems_ph = tf.placeholder(tf.int32, [None, None])
        shape_ph = tf.placeholder(tf.int32, [None])

        start_np = timer()
        np_result = sess.run(render_bboxes_pyfunc_2d(elems_ph, shape_ph),
                             feed_dict={elems_ph: elems, shape_ph:target_shape})
        end_np = timer()

        elems_ph2 = tf.placeholder(tf.int32, [None, None])
        shape_ph2 = tf.placeholder(tf.int32, [None])
        start_tf = timer()
        tf_result = sess.run(render_nd_bboxes_tf_spreading(elems_ph2, shape_ph2,
            ndim=2),
                             feed_dict={elems_ph2: elems, shape_ph2:
```

```
                                    target_shape})  # or a list of things.
        end_tf = timer()

        assert np.all(np.equal(np_result, tf_result))
        print((end_np-start_np, end_tf-start_tf))


def test_render_bboxes_batch_2d():
    # test with the batch_size dimension (being 2)
    elems = [[[0, 1, 3, 10, 1], [1, 13, 4, 17, 1], [0, 1, 3, 10, 1]],
             [[0, 2, 3, 10, 1], [1, 14, 4, 17, 1], [0, 2, 3, 10, 1]]]
    target_shape = [20, 20]

    with tf.Session() as sess:

        elems_ph = tf.placeholder(tf.int32, [None, None, None])
        shape_ph = tf.placeholder(tf.int32, [None])

        start_np = timer()
        np_result = sess.run(render_bboxes_pyfunc_2d(elems_ph, shape_ph),
                             feed_dict={elems_ph: elems, shape_ph:target_shape})
        end_np = timer()

        elems_ph2 = tf.placeholder(tf.int32, [None, None, None])
        shape_ph2 = tf.placeholder(tf.int32, [None])
        start_tf = timer()
        tf_result = sess.run(render_nd_bboxes_tf_spreading(elems_ph2, shape_ph2,
             ndim=2),
                             feed_dict={elems_ph2: elems, shape_ph2:
                                 target_shape})  # or a list of things.
        end_tf = timer()

        assert tf_result.ndim == 4, "bboxes should be able to be rendered into
            batches of images"
        assert tf_result.shape[-1] == len(elems[0][0]) - 2*2
        assert np_result.shape[0] == tf_result.shape[0] == len(elems), "we have
            provided a different number of batches"
        assert np.all(np.equal(np_result, tf_result))
        print((end_np-start_np, end_tf-start_tf))
```

Conclusion and visualizations Note, that we have created two functions that need differently scaled inputs (one needs floats, the second needs integers) because we were preserving the format set by the original tensorflow functions. And for the final visualization – here is the code that builds the computational graph, loads a picture, displays crops, processes the crops (as a toy case, it multiplies the original image with the mask) and outputs a picture:

```
from matplotlib import pyplot as plt
from PIL import Image
import requests


def test_with_lena():
    pic = np.asarray(Image.open(requests.get("https://upload.wikimedia.org/
        wikipedia/en/7/7d/Lenna_{86385
        ef8c424def43c570938a9943967855f43526a201d16486274d8a74d2e91}28test_image
        {86385ef8c424def43c570938a9943967855f43526a201d16486274d8a74d2e91}29.png
        ", stream=True).raw))

    boxes_input = np.array([[[100.0 / pic.shape[0], 100.0 / pic.shape[1], 200.0
        / pic.shape[0], 200.0 / pic.shape[1]]]])

    crop_size = (800, 800)

    with tf.Session() as sess:
        image_input_ph = tf.placeholder(tf.float32, [None, None, None, None])
        boxes_input_ph = tf.placeholder(tf.float32, [None, None, 4])
        boxes_with_ones = tf.to_int32(
```

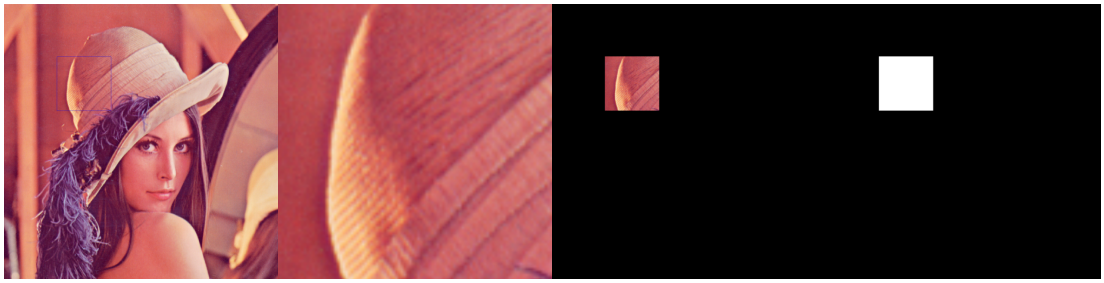
```
            tf.concat([tf.multiply(boxes_input_ph, tf.to_float(tf.concat([tf.
                shape(image_input_ph)[1:3], tf.shape(image_input_ph)[1:3]], axis
                =-1))),
                       tf.ones(tf.concat([tf.shape(boxes_input_ph)[0:2], [1]],
                           axis=-1))],
                      axis=-1))
        shape_ph = tf.shape(image_input_ph)[1:3]
        crop_result, render_result, only_box = sess.run(
            [tf_crop_and_resize_batches_of_arrays(image_input_ph, boxes_input_ph
                , crop_size=crop_size),
             #tf.tile(tf.to_float(render_nd_bboxes_tf_spreading(boxes_with_ones,
                  shape_ph, ndim=2)), [1, 1, 1, 3])
             tf.multiply(tf.to_float(render_nd_bboxes_tf_spreading(
                 boxes_with_ones, shape_ph, ndim=2)),
             image_input_ph),
             tf.tile(tf.to_float(render_nd_bboxes_tf_spreading(boxes_with_ones,
                 shape_ph, ndim=2)), [1, 1, 1, 3])
             ],
            feed_dict={image_input_ph: np.expand_dims(pic, 0), boxes_input_ph:
                boxes_input})


        fig = plt.figure(figsize=(24, 32))
        plt.imshow(np.asarray(pic).astype(dtype='B'))
        plt.tick_params(left='off', bottom='off', labelleft='off', labelbottom='
            off')
        l, t, r, b = [100, 100, 200, 200]
        plt.gca().add_patch(plt.Rectangle((l, b), r - l, t - b, color='b', fill=
            False))
        plt.savefig('pic.png')

        fig = plt.figure(figsize=(24, 32))
        plt.imshow(np.asarray(crop_result[0, 0, :, :, :]).astype(dtype='B'))
        plt.tick_params(left='off', bottom='off', labelleft='off', labelbottom='
            off')
        plt.savefig('crop.png')

        fig = plt.figure(figsize=(24, 32))
        plt.imshow(np.asarray(render_result[0, :, :, :]).astype(dtype='B'))
        plt.tick_params(left='off', bottom='off', labelleft='off', labelbottom='
            off')
        plt.savefig('render.png')

        fig = plt.figure(figsize=(24, 32))
        plt.imshow(np.asarray(only_box[0, :, :, :]*255).astype(dtype='B'))
        plt.tick_params(left='off', bottom='off', labelleft='off', labelbottom='
            off')
        plt.savefig('only.png')
```



B. Datasets and source codes

As a tiny tribute to the environment, please find the complementary source codes and datasets in the following open public repositories instead of on an attached medium:

- https://github.com/Darthholi/similarity-models
- https://www.kaggle.com/martholi/anonymized-invoices
- https://github.com/Darthholi/DocumentConcepts