Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Master's Thesis Nr. 220

Systems Group, Department of Computer Science, ETH Zurich

Automated visual document parsing with document-level structure annotations

by

Fabian Bissig

Supervised by

Prof. Ce Zhang, Johannes Rausch

May 2018–December 2018

Informatik
Computer Science

## Abstract

Machine learning research relies on having high-quality labeled datasets. There are many tools for hand-labeling images. However, a tool aimed specifically at annotating documents is missing. We present a tool for manual document annotation. We support the specialties found in documents by supporting multi-page documents and hierarchical annotations. Tables with their rows and cells are a good example for typical hierarchical structures commonly found in documents. We investigate how to detect tables and their caption and cells on document pages. We wrote a table detection pipeline to automatically create a noisy labeled dataset from papers published on arXiv. We also labeled hundreds of documents by hand. We based our work on the state-of-the-art object detection framework Mask R-CNN and used weakly supervised learning. We trained a network using documents with noisy labels, that we could automatically generate from LaTeX source files. We finetuned them using a small set of hand-labeled documents to get rid of the systematic errors introduced by the noisy labeled dataset. We showed that it is possible to achieve good results in object detection with a noisy dataset, and that we can further improve detection using only a small set of hand-labeled documents. We hope that our document annotation tool will be used for creating an even larger dataset to advance further research that eventually tries to understand the complete structure of documents.

# Contents

# 1 Introduction

At the beginning of the 21st century, people and corporations around the world started to show interest in practical applications of **artificial intelligence** (AI). The development we have today was only possible thanks to advances in computer technology, access to large amounts of data (**big data**) and advanced **machine learning** techniques. Researchers work on making machines understand digital images and videos, human speech, let them compete in strategic games such as chess and Go, use them as question-answering systems (such as IBM Watson) and allow them to drive around cities as autonomous vehicles. All these technological advances have one thing in common. They require access to lots of data.

Humans have for a long time used documents for publishing information and scientific advances. Indeed, a wealth of human knowledge can be found immortalized on paper and electronic formats nowadays. There is a need for understanding them even if the content and the structure are not preserved in a machine-readable way. Especially with the long-term goal of **artificial general intelligence** (AGI) in mind, it is inevitable that machines eventually fully understand what is written down on a document, at the same level of understanding that a human exhibits.

**Information retrieval**, i.e. the art of computer-aided examination of information resources for finding complex content, has always been of interest since the emergence of modern computers. Closely related is the concept of **document retrieval**. Its goal is to find and retrieve the documents that are relevant to the users' information needs. In its easiest form, it could return as an output the names and pages of documents where a certain search term can be found. Such a system can be easily implemented using **optical character recognition** (OCR).

However, the structure of documents is complex. Documents have an outline with sections, headings and paragraphs. They contain lists, figures and tables. Independent of the general document structure, page numbers or the current section name may be found in headers and footers on every page. An exception is oftentimes the first page, which contains a title, author, etc. The document might be structured in two or more columns.

While OCR may be sufficient for simple search queries ("find all documents which contain this word"), it is vital to understand the document structure to learn more than mere presences of search terms in a document. It might be of interest to understand the table of contents or learn the structure of a table, which consists of multiple cells arranged in a specific pattern and a caption above or below the table itself.

Historically, the document structure was often learned by means of heuristics and using rule-based conversion algorithms (see Ghai et al. [6] for a review). Recent developments in neural networks have the potential to replace some of these approaches, because it is possible to leave out the writing of a wealth of complicated algorithms geared to very specific tasks. Instead of working with raw text that is the OCR's output, we shift to directly working with the visual rendering of the document. This could be more robust due to the flaws in the OCR not affecting us.

**Convolutional Neural Networks** (CNNs), a class of deep, feed-forward neural networks, are commonly applied to analyzing visual imagery. For **image classification**, raw image pixels are taken as input and scores are returned as output. In **object detection** the task is to detect instances of semantic objects of a certain class (e.g. cars or traffic signals on a street). The same approach may be used for detecting the location of elements like paragraphs, headings, figures, tables, lists, etc. in documents. **Supervised learning** from labeled training data is suitable for training a convolutional neural network. It is a commonly used type of learning algorithm in machine learning. However, large datasets of labeled data are necessary, such as the COCO dataset [13] with real-world images. The format for these labels consists of a flat list of annotations for each image specifying a category for a bounding box. Analogously, such a dataset can be created which stores the bounding boxes of all structural elements of a document.

A great source for research papers is arXiv[1]. It is a repository of e-prints in various scientific fields started in 1991. It allows users to retrieve papers as PDF but oftentimes also stores the LaTeX source. However, at the best of our knowledge there is no available dataset of annotated documents. Manual document annotation is a laborious task. It is vital for getting a ground truth dataset, though. It makes sense to optimize the time spent by human annotators such that the human mainly caters to only fixing mistakes in automatically pre-generated annotations. **Weakly supervised learning** is a technique to reduce the amount of hand-labeled documents needed.

Since Adobe published the PDF specification in 1993, PDF has grown to become the de facto standard for electronic documents, used by both governments and the private industry. It became standardized as ISO 32000 in 2008 and can be used for archiving and long-term preservation in a separate ISO-standardized version called PDF/A[2]. PDF combines the PostScript page description language that runs in an interpreter for generating layout and graphics with a font-embedding system. Since PostScript is responsible for rasterization, i.e. the conversion of elements specified in terms of lines and Bézier curves to pixels, it is important to understand the format. PostScript describes how text and other elements get rendered but not in the semantic way of a markup language like HTML. In PostScript text is represented by *text elements*. They define the characters that should be drawn at a certain position using a specified encoding of a *font resource*. This is problematic since a simple equation like $x^2$ consists of two independent text elements. Note that this does not apply to scanned documents stored in PDF files, since they are rasterized.

## 1.1 Motivation

Documents span multiple pages and are inherently structural (starting at the source format, see the LaTeX or HTML format). We compare this to real-world

[1]`https://arxiv.org/`
[2]`http://www.pdfa.org/publication/pdfa-in-a-nutshell-2-0/`

images, that contain objects in different sizes and shapes where it is not obvious on how to structure them. Therefore, labels for such a format are stored in a flat list where order does not matter and we always rely on a bounding box that gives us the position of the object in the image. If we used the same approach for documents, we were only able to tell that there is a section header at a certain location on a page but we would not be able to tell to which section a paragraph on a different page belongs.

We want to create a dataset of labeled documents for further usage in research. Since we did not find a suitable tool that takes into consideration the specific properties of documents, we built our own document annotation tool. Its interface consists of a drawing area where bounding boxes may be drawn on a page, and a tree in which elements of the entire document can be arranged in a hierarchical structure.

Tables can be found in the majority of published scientific literature. They are of very diverse nature and it is of interest to be able to parse and understand them. Tables can be seen as a "minimal example" of full document parsing, as they are also hierarchical. We evaluate how manually labeling tables with our tool improves detection precision. We use Mask R-CNN to train a neural network to detect tables, table cells and table captions on papers published on arXiv. This is possible due to the fact that the LaTeX document source (defining the table later seen in the PDF) is oftentimes available, and we can use it to derive bounding boxes for the different elements. We finetune it using the manually labeled dataset. This allows us to quickly get a large training dataset, albeit noisy. Paired with finetuning, we can "unlearn" systematic errors present in the noisy dataset and achieve good detection performance with comparatively low effort.

The thesis is structured as follows. In section 2, we survey related work in the areas of machine learning, document recognition and PDF. In section 3, we present our document annotation format and tool. The chosen research method for document recognition is explained in section 4. The experiment to detect tables in documents is described in section 5 and the evaluation of the detection performance can be found in section 6. We discuss developments for the future in section 7 and conclude our findings in section 8.

# 2 Related Work

## 2.1 Machine learning

*Object detection* is the task of identifying instances of objects of a certain class in an image. For an image as input we get a list of bounding boxes with corresponding classes as output. Object detection is often implemented using deep-learning approaches with region proposals. For this, a large amount of different regions are fed through a **Convolutional Neural Network** (CNN) to find lots of specific properties of these images, to eventually learn the corresponding class. The main challenge is to efficiently find these regions to reduce processing time. A brute-force approach is to slide windows from left to right and from up to down. To reduce the number of regions to analyze, **region-based CNNs** (R-CNNs) [8] can be used. A *selective search* (combine similar regions using greedy algorithm) is performed to find *regions of interest* (ROIs). In an R-CNN we may create thousands of ROIs that are then warped to the same size and fed into a CNN network individually.

**Fast R-CNN** [7] does not repeat feature extraction individually for each ROI. Instead a *feature extractor* (also a CNN) is used beforehand to extract features for the entire image. This reduces processing time considerably. **Faster R-CNN** [14] further improves this by replacing the region proposal method by an internal deep network. ROIs are generated considerably faster thanks to this. **Mask R-CNN** [9] extends Faster R-CNN by adding the ability to generate segmentation masks on top of the detected object instances.

## 2.2 Annotation tools

Supervised learning requires a manually labeled dataset. Such a dataset can be created by utilizing a **manual image annotation tool**. There exist a variety of such tools like Labelbox[3], LabelMe[4] and Playment[5]. However, many of them are hosted solutions and they are only meant for defining annotated regions in single images. The strictly hierarchical structure of documents, their division into multiple pages and the need for knowing the text flow (e.g. in presence of a two column layout) make them less suitable for document annotation.

## 2.3 PDF

TEX does not use a standard file format like XML or JSON but implements its own. We don't want to rely on regular expressions to find what we need in a source file, but instead need a parser that understands the format. TexSoup[6] is a Python tool which parses valid LATEX and provides a variety of functions to easily iterate and search the parse tree. This allows users to easily find if a document contains any elements of a specific type (e.g. tables) and also learn

[3]`https://www.labelbox.com/`
[4]`http://labelme.csail.mit.edu/`
[5]`https://playment.io/`
[6]`https://github.com/alvinwan/TexSoup`

their structure and contents. However, it does not return the location of this structure in the final PDF.

Listing 1: PDFMiner output example

```xml
<?xml version="1.0" encoding="utf-8" ?>
<pages>
  <page id="1" bbox="0,0,612,792" rotate="0">
    <textbox id="0" bbox="93,707,516,720">
      <textline bbox="93,707,516,720">
        <text bbox="93,707,105,720" size="12">U</text>
        <text bbox="105,707,114,720" size="12">n</text>
        <text bbox="114,707,123,720" size="12">d</text>
      </textline>
    </textbox>
  </page>
</pages>
```

Laurens [11] demonstrates a technology called SyncTEX which allows synchronization between parts of the TEX input and the position within the resulting PDF. This technology is useful for TEX users because it allows the two windows, one for entering the text and one for viewing the resulting output, to be synchronized. To achieve this, SyncTEX is able to return the page and an approximate bounding box in the resulting PDF of the portion of TEX source code currently selected. Applying this to semantic elements allows us to get bounding boxes for single table cells or lines of text.

PDFMiner[7] is a tool for parsing PDF files, which means that it understands the PostScript commands that render text elements at certain positions in the PDF. The tool can return the position of every character and uses heuristics to detect the words and lines to which they belong on every page of the document. However, any further structures such as paragraphs are problematic and cannot always be reliably detected, especially if there is not enough space between them. Listing 1 is an example of PDFMiner's output format.

## 2.4 Image processing

Wong et al. [19] investigated the segmentation and classification of digitized printed documents into regions of text and images. They applied a nonlinear, run-length smoothing algorithm (RLSA) to each row and column to find white and black areas representing blocks containing text and images.

## 2.5 Document recognition

Chanod et al. [3] present the *Legacy Document Conversion* (LegDoC) project for conversion of layout-oriented formats like PDF and HTML to semantic-oriented annotations in XML. Lecerf et al. [12] present a system for the semantic annotation of layout-oriented documents.

[7]http://www.unixuser.org/~euske/python/pdfminer/

Sarkar et al. [15] developed a framework for understanding tables of contents (TOCs). They propose a structure representation in terms of a hierarchy of entries. For an exemplary chapter "Belief Updating by Network Propagation" they would have the following XML where `<children></children>` contains all subsections.

Listing 2: Exemplary ground truth assignment for an item in a table of contents

```
<tocentry>
  <descriptor>
    Chapter 4: Belief Updating by Network Propagation
  </descriptor>
  <locator>143</locator>
  <children></children>
</tocentry>
```

As part of IBM Research Staar et al. [17] developed a Corpus Conversion Service that can ingest documents at scale and make the contained knowledge discoverable. The researchers developed a processing pipeline to ingest documents in any format and finally convert them into a structured data format. They train a model with minimal effort from a human annotator. They claim that after having annotated several dozen PDFs, the machine learning algorithm can take over. However, their implementation is proprietary and the technology will initially only be available in the IBM Cloud or for on-premise clouds.

There have long been document converters such as the commercial ABBYY FineReader[8]. It started out as an OCR application but now also does all the other steps of document recognition. It tries to faithfully keep the layout of a scanned document such that it can be output as a Microsoft Word document. However, this software is proprietary, so it is not possible to find out how exactly their software works.

[8]`https://www.abbyy.com/finereader/`

# 3 Document Annotation

## 3.1 Document Annotation Format

We design a JSON format which contains the hierarchically structured annotations of an entire document with multiple pages.

We take into consideration the following properties:

- **Hierarchical**: Elements are arranged in a tree such that every element has a parent
- **Multi-page**: Elements can appear on different pages or even span multiple pages, so the page is also stored next to the bounding box of every element
- **Table support**: Support for storing custom additional information such as the position of the cell within the table
- **HTML/LaTeX similarity**: Allowing for easy conversion of the source code to our format without loss of too much information (e.g. the hierarchical arrangement)
- **Extensible**: Making it possible to further refine elements to accompany complex layouts

In the defined format, objects are not stored in a flat structure, as it would be the case for regular image annotation (cf. COCO's data format[9]). Instead, the JSON format represents a hierarchy of nodes, similar to HTML or LaTeX documents. This has the advantage of making it possible to iterate over the children of a particular node or determine the parent of another node. An example usage is the easy retrieval of all nodes of a section, namely by including all child nodes of that section.

We also do not store annotations separately per page (i.e. storing per image). Instead, we have one hierarchical structure of nodes for the entire document. This allows us to keep the information that for instance a table spans multiple pages but should be considered as a whole. We distinguish between two kinds of nodes, namely structure and annotation nodes. Only the annotation nodes reference concrete rectangular bounding boxes on specific pages. The structure nodes exist for creating a hierarchy and storing additional properties (such as the position of a table cell within a table).

### 3.1.1 Structure/Annotation elements

The JSON file is an ordered list of objects with IDs. Since that is a flat structure, the hierarchy is recovered by referencing the parent node (with `null` representing the root node). Each object is an element that may be of the following two types:

- **Structure element**: Has a category and references its parent
- **Annotation element**: Has a bounding box, a page and references its parent

[9] `http://cocodataset.org/#format-data`

Listing 3: Structure element (root node)

```
{
  "id": 1,
  "category": "meta",
  "parent": null
}
```

Listing 3 shows an example of a structure element. It is purely structural in a sense that it does not contain any information on its position within the rendered output. The position can only be recovered by recursively following the childrens' nodes to find all annotation nodes and then combining the bounding boxes of these nodes. For example, to find the bounding box of a table, we calculate the minimum bounding rectangle of the bounding boxes of all table cells that belong to this table according to the hierarchy. It can have an optional *properties* argument to contain further attributes (e.g. the position within a table for a cell).

Listing 4: Annotation element

```
{
  "id": 121,
  "category": "box",
  "page": 0,
  "bbox": [40, 100, 400, 10],
  "parent": 12
}
```

Listing 4 shows an example of an annotation element. It references something concrete on the PDF by means of a bounding box and a page reference. Annotation nodes are leaves and thus do not have any children.

### 3.1.2 Meta/Document nodes

Documents consist of multiple pages typically with a clear reading flow (e.g. left to right from top left corner to bottom right corner in a one column layout). However, apart from this there is also auxiliary information on each page such as page numbers or the title/author on the first page. Since they don't fit into the reading flow, we opted to move these elements to a separate tree (meta) from the main tree (document). Note that this is only a convention we adopted, as both trees are merely structural elements at the root of the tree.

Figure 1 shows the general structure of the meta node. It includes elements pertaining to the entire document and are commonly found on the first page. These may include title, author, affiliations, date, abstract.

Apart from that, it also includes miscellaneous information that appears once on every page, such as a header with the section title or a footer with the current page number.

Figure 2 shows an example of the structure of the document node. It contains the sections with the paragraphs. Each paragraph consists of text lines with several

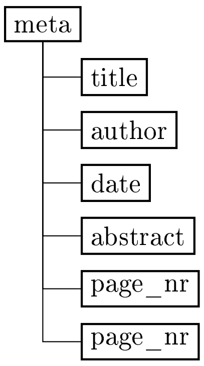


Figure 1: Example of which structure elements a meta node (that contains components outside the reading flow) could contain in our format

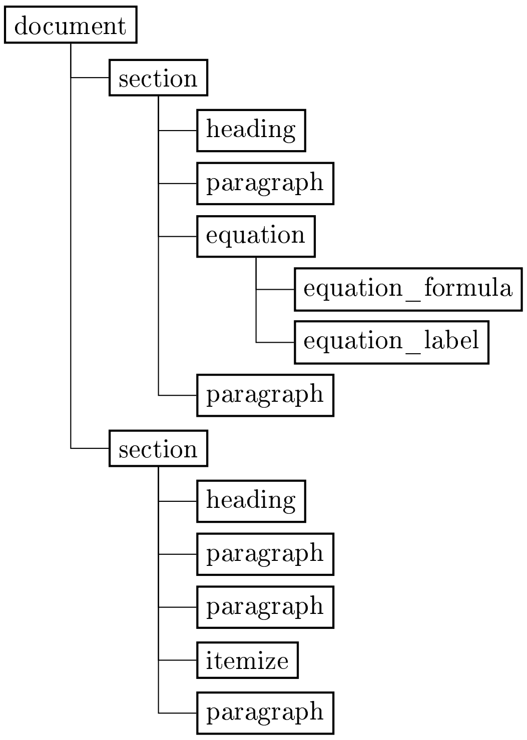


Figure 2: Example of which structure elements, hierarchically arranged according to their relationships within the document, a document node (that contains components of the reading flow) could contain in our format

words. Apart from paragraphs, there may also be equations, bulleted/numbered lists, figures or tables.

Documents follow a hierarchical structure. A general partitioning is into chapters, sections, subsections, etc. We want to keep this structure and thus subdivide the document into multiple sections with a heading each and that we arrange hierarchically.

Listing 5: HTML table example

```
<table>
  <tr>
  <td>Row 1, Column 1</td><td>Row 1, Column 2</td>
  </tr>
  <tr>
  <td>Row 2, Column 1</td><td>Row 2, Column 2</td>
  </tr>
</table>
```

Listing 6: LaTeX table example

```
\begin{table}[ht]
\begin{tabular}{ l l }
  Row 1, Column 1 & Row 1, Column 2 \\
  Row 2, Column 1 & Row 2, Column 2 \\
\end{tabular}
\caption{Example Table}
\end{table}
```

HTML (Listing 5) and LaTeX (Listing 6) implement tables in a way we will refer to as *row-hierarchical*. Both implementations aim to mimic the visual representation of the tables and thus hierarchically first designate the rows (`<tr></tr>` in HTML and number of lines separated by `\\` in LaTeX) and then the cells within each row. While we can easily retrieve the $n$th row, it is not immediately obvious how to determine a particular column.

In presence of multi-row/multi-column cells, this is further complicated. HTML uses attributes like `<td colspan="2"></td>` and `<td rowspan="2"></td>` with missing cell tags for the spanned cells, whereas LaTeX uses `\multicolumn{2}{c}{}` with missing cell tags for spanned cells and `\multirow{2}{*}{}` with present cell tags for the spanned cells.

An alternative is the implementation using a *bitmap*. We have implemented the position of cells within the table as shown in Figure 3. Table cells are direct children of the tabular structure node and have an optional properties argument which contains the position of the cell within the table (e.g. `1,2` for the first row and second column). By using a hyphen (e.g. `1,1-2`) it is possible to designate multi-column cells (e.g. a cell spanning the first two columns) and vice versa for multi-row cells.

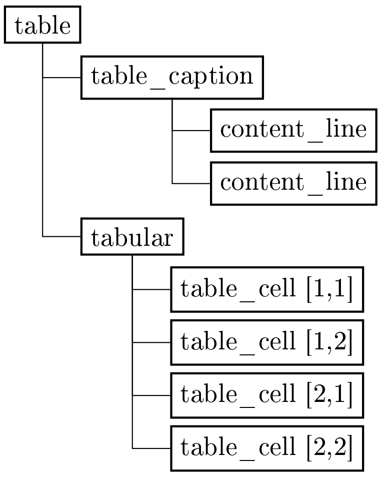


Figure 3: Exemplary structure elements of a table

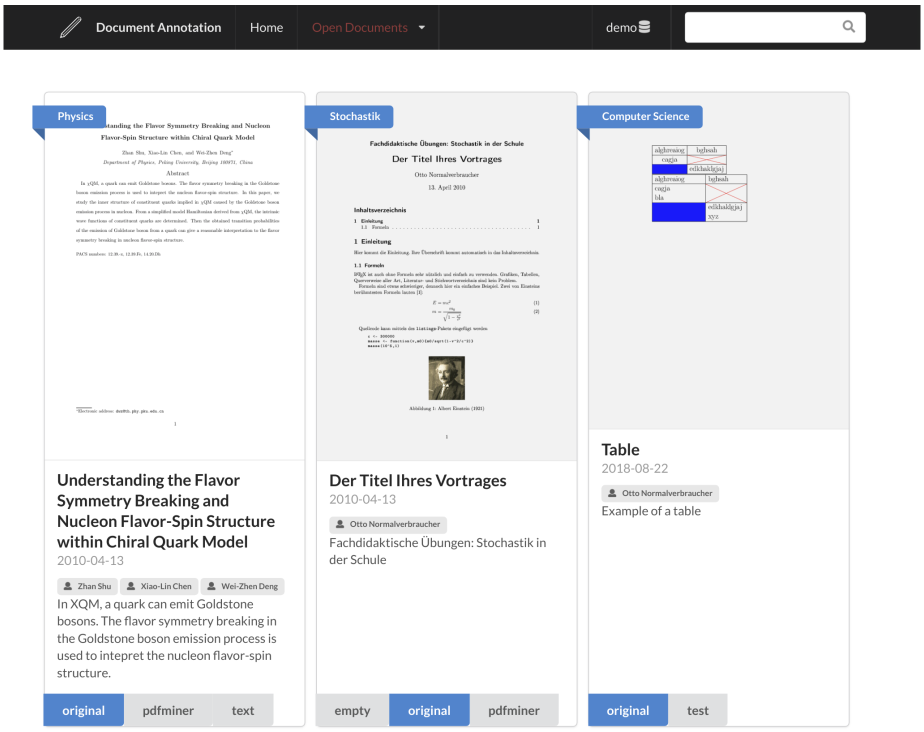
## 3.2 Document Annotation Tool

In order to create and edit annotations of the format explained in the previous subsection, we created a web application for document annotation with React and Redux. It consists of a client and server part. The server hosts multiple datasets of different document files, in the format described in the previous section. The server provides these documents to clients requesting them and also accepts changes that it persists on disk.

The client application runs in the browser and has two windows, namely the document list window and the document edit window. The list window as shown in Figure 4 offers the user an overview over all datasets and documents included. For each document, multiple annotation versions (e.g. separate PDFMiner, SyncTEX, manual annotations) may be listed.

When users select a version of a document in a dataset, they are redirected to the document edit window as shown in Figure 5. It is composed of a page view on the left, a tree view on the right, and a toolbox above. The page view shows the current page of the document with a slider making it possible to browse through the pages. All of the page's annotation boxes are depicted on the page as bounding boxes. The helpers from the toolbox can be used for changing the annotations. The tree view, on the other hand, is independent of the page currently displayed. The document has one tree for the entire document, putting the structure (with headings, paragraphs, tables, etc.) into a hierarchical outline. The two sides are synchronized such that clicking a bounding box on the left will jump to the corresponding annotation element and hovering over structure elements on the right will highlight all encompassing annotation elements.

Adding and editing annotations is an intricate interplay between the left and right side of the screen. Since annotation elements (the only ones actually getting drawn on the page with a bounding box) are always the leaves of variously nested structure nodes, adding a new element to the page means draw-



Understanding the Flavor Symmetry Breaking and Nucleon Flavor-Spin Structure within Chiral Quark Model
2010-04-13
Zhan Shu · Xiao-Lin Chen · Wei-Zhen Deng
In XQM, a quark can emit Goldstone bosons. The flavor symmetry breaking in the Goldstone boson emission process is used to intepret the nucleon flavor-spin structure.

Der Titel Ihres Vortrages
2010-04-13
Otto Normalverbraucher
Fachdidaktische Übungen: Stochastik in der Schule

Table
2018-08-22
Otto Normalverbraucher
Example of a table

Figure 4: Document list window

ing multiple bounding boxes on the page (or use pregenerated ones, e.g. using PDFMiner), selecting them, choosing the type of parent structure node (e.g. paragraph if we selected annotation nodes reflecting lines of a paragraph), and deciding where to put it in the hierarchy (e.g. as the first paragraph directly after the heading of a section).

There are various helpers which support annotation creation in the page view. The user can create new annotations by drawing a rectangle while shift-clicking or double-clicking on the drawing area. Existing annotations can be moved using a drag-and-drop operation or resized by clicking and then dragging the corner marks. The user can select multiple annotations by means of a rectangle selection. The selected annotation elements can be merged together. This is done by calculating the minimum bounding rectangle. Morever, they may be moved to an existing structure element selected in the tree view or as children of a new structure element to an existing structure element. As an example, after selecting all annotation elements that belong to a paragraph, the user can select the parent structure element (an already existing section) in the tree view and their common structure element (of type paragraph) in the toolbox and then click the add button.

Additionally, there are helpers specific to table annotation. It is possible to select all annotation elements representing table cells and move them to a new table structure element with one separate structure element for each cell. Then

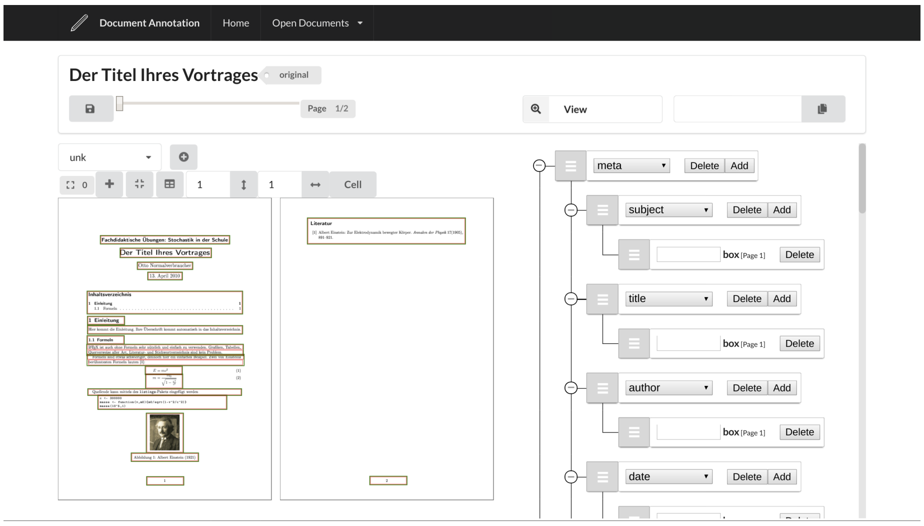


Figure 5: Document document window

multiple cells may be selected and allocated to a numeric row/column (e.g. by specifying that the selected cells belong to the first row or first column).

Keyboard shortcuts to move (I, J, K, L) or resize (W, A, S, D) selected annotations by pixel increments are present. To help with handling numerous annotation elements at once, e.g. when dealing with table cells, it is possible to use copy (CTRL + C) and paste (CTRL + V) commands. This is immensely useful for annotating tables, as only the cells of one row or column have to be created precisely, whereupon they can be copied and moved to the following rows/columns.

The hierarchy can be used for quickly seeing which annotations belong to which section. This can be achieved by hovering over a node further up in the hierarchy (e.g. a section), which then highlights all annotation elements belonging to it in the page view.

# 4 Method

We want to find the elements of a document, such as paragraphs, headings, equations, figures and tables. To solve this, we need to find the bounding boxes off all these elements and then classify them. This is an object detection task that can be solved by training a CNN to find and classify these elements. We can use existing implementations such as Mask R-CNN and adjust them to our specfiic needs.

## 4.1 Neural Networks

A **neural network** is a network of artifical neurons that receive input, change their internal state (activation function) according to the input, and produce output accordingly. Learning happens by feeding it lots of examples of input/output pairs, where over time the network adjusts to detecting these examples well. We expect the network to generalize and also work with new input/output pairs. One example is image classification, where neural networks learn to label images to be of a certain class or not. The results of this learning is a model that we should be able to use for identifying the same classes in other images.

A neural network is based on many artificial neurons that are connected with each other and transmit numbers from one side to the other in the shape of an acyclic graph. Each neuron uses a non-linear function to calculate an output based on the input. Every neuron has a certain weight associated with it that is learned gradually during training. The weight influences the “strength” that a single neuron assigns to the input signal.

Neural networks have been inspired from how brains work. In a simplified view, neurons receive signals along other synapses from dendrites. They then fire off spikes according to a specific code along one output axon to synapses which branch out. The code the neuron uses is analogous to the *activation function*.

There are different activation functions (or *non-linearities*) such as the sigmoid function in Equation 1 which squashes real numbers to a range between -1 and 1.

$$\sigma(x) = 1/(1 + e^{-x}) \tag{1}$$

There is also the tanh function that has the range between 0 and 1. The Rectified Linear Unit (ReLU) in Equation 2 basically thresholds the value at zero. Krizhevsky et al. [10] found that it can greatly accelerate the convergence compared to the tanh function.

$$f(x) = \max(0, x) \tag{2}$$

A neural network as depicted in Figure 6 consists of neurons connected in an acyclic graph. There is an input and output layer as well as one or more hidden layers in-between. Oftentimes, the layers are fully-connected, such that every

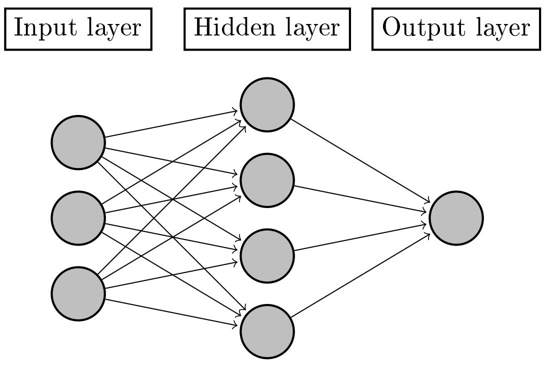


Figure 6: Example of a neural network with 3 nodes in the input layer, 4 nodes in a hidden layer and an output layer

node from the previous layer is connected with every node from the current layer. This layered architecture makes evaluation easy, it consists of consecutive matrix multiplications plus the non-linear activation function. A neural network pretty much defines a family of functions paramterized by the weights of the network.

Having more layers works better. This is not because it is more expressive, since one hidden layer is enough for every neural network to be an *universal approximator* [4], but likely because images contain hierarchical structures. This is also the case with documents, where for instance cells and the caption belong to a table. However, these larger networks also increase the risk of overfitting. This happens when the model fits the noise of the data from the training set instead of the underlying relationship as observed also in the validation and test set. With less neurons and thus less representational power, few wrong data points will be regarded as *outliers*.

However, having less layers is not a solution for the problem. Instead we opt for different *regularization* methods such as L2 regularization, L1 regularization or max norm constraints to control the overfitting. This works by penalizing peaky weight vectors. Another regularization technique is dropout [16] which complements the other methods. It works by keeping each neuron active only with some probability that can be set as a hyperparameter.

Before feeding data to the neural network there is an important preprocessing step. We use it to center the data with a mean of zero and normalize it to a scale between -1 and 1. We initialize the weights to random numbers. As an alternative, we can use the weights of another pretrained network such as those used for the COCO dataset. This is the approach that we chose.

In **image classification** the goal is to classify an image. A colored input image would typically be represented as three individual two-dimensional matrices of pixel intensities for the three color channels red, green and blue respectively. We get as output the probabilities of the image being of certain categories (e.g. 80% car, 12% street, 8% bicycle). Finally, we need to evaluate the quality of the classifier by letting it work on never before seen input images. Real-world images exhibit particular challenges for image classification like the object getting photographed from different angles or being of varying sizes. Illumination has a big impact on the numbers and the background may vary. Categories can be broad and include objects of varying types (e.g. how do we define a "car" and do we use subcategories?). Interestingly enough, since we run detection on images of analog or electronic documents (e.g. PDFs), we don't have to cope with all these challenges. While objects can be of different sizes, rotated objects are relatively rare. We have mostly black text on a white background with figures and tables interspersed.

To approach the image classification problem, we don't write an algorithm to detect a certain category, so-called *feature engineering*. This is the approach used before the use of CNNs. It would not be obvious how to write one, anyway. Instead we use a **data-driven approach** where we provide a list of hand-labeled images (i.e. a training dataset) to the application and tell it to learn the classification for these.

We define the **image classification pipeline** as follows:

- **Input**: set of images each labeled with a category (*training set*)
- **Learning**: learn to extract meaningful features from the input data that allow accurate classification by means of repeated parameter updates, using the training set to eventually learn of what category each image is (*training classifier*)
- **Evaluation**: evaluate quality of classifier by letting it predict categories on a different set of images (*valuation set*)

There are a multitude of options with different configurations for training the classifier. We call these **hyperparameters**. We tweak them to improve the performance of the classifier. However, we need to be cautious when doing so to avoid **overfitting**. To avoid this, we use a separate *test set* that is only evaluated once at the end. It is important not to use it for hyperparameter tuning as otherwise the classifier might not work well once deployed and working with real-world data because it is not generalized enough.

We define a *score function* and a *loss function*. If $\mathbf{x}_i$ denotes the input vector of raw pixels for image $i$, $\mathbf{W}$ the weight matrix (also called parameters) and $\mathbf{b}$ the bias, then $f$ in Equation 3 is the score function that outputs a vector of scores for all the categories.

$$f(\mathbf{x}_i, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x}_i + \mathbf{b} \tag{3}$$

The loss function returns a single score for the output vector of the score function to determine how good of a job it did. Let's refer to the score for the j-th class as $s_j = f(\mathbf{x}_i, \mathbf{W}, \mathbf{b})_j$. $y_i$ denotes the index of the class. Two common loss functions are the multiclass SVM loss in Equation 4 and the softmax function in Equation 5.

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta) \tag{4}$$

$$L_i = -f_{y_i} + \log \sum_j e^{f_j} \tag{5}$$

The multiclass SVM loss requires the correct class score to be higher than the scores of the other categories by the margin $\Delta$ (a hyperparameter). It is not calibrated and does not allow to easily compare scores of the different categories. The softmax function, on the other hand, returns probabilities for all categories such that they can be interpreted in comparison to the others. We use the softmax function for our task.

In **optimization** the task is to efficiently find the set of parameters $W$ that produce good predictions for the validation set while avoiding overfitting to still work good on the test set. We iteratively refine a randomly chosen set of parameters until the loss is minimized. The gradient of the loss function gives the direction in the weight space to follow to minimize the score. **Gradient descent** then refers to repeatedly evaluating the gradient and updating the parameters by the *learning rate*. It is important to choose the right learning rate. If it is too low, then it takes a long time to find the minimum of the function, whereas if it is too high there is the danger of overshooting and missing the minimum entirely. **Stochastic gradient descent** is an approximation of gradient descent optimization and works with only a subset of the entire training dataset, whose size is specified by a hyperparameter. This is the approach used in CNNs so that it is not necessary to look at the entire training set for one single parameter update. The reason why this approach works is because of correlation in the examples.

**Backpropagation** [2] is a part of learning/optimization and its goal is to adjust the weights in the network in proportion to how much they contribute to the overall error. Forward propargation is a series of nested equations. Backpropagation is an application of the *chain rule* to find the derivatives taking into consideration the variables in the nested equation. Considering implementation it is important to utilize *staged computation*. We break up the function into single parts for which the gradient can be independently derived.

## 4.2 Convolutional Neural Networks

**Convolutional Neural Networks** are Neural Networks with the same structure with learning neurons. We still map raw image pixels to class scores in the score function and finally there is a loss function. Convolutional Neural Networks are made with the assumption that they need to work well with images, which they can use to their advantage. The architecture benefits from efficient convolution implementations on modern GPUs.

A major problem of using neural networks for regular images is that the parameter sizes get unnecessarily large. This is caused by the fact that images have a size of width $\times$ height $\times$ colors (with the three colors red, green, blue). Having such a high number of parameters in the weight matrices is also wasteful and there is a real danger of overfitting. This can be explained by the colors not being relevant or the background not having much importance. Plus the pixels are arranged in a single vector, which has the disadvantage that pixels which are close together in the image are far apart in the vector. To accommodate the realities of images, we arrange the neurons in three dimensions instead. That way the neighboring values in the weight matrix actually have an influence on the pixels close together. This also applies for the three colors, namely the third dimension. We call the local region of input the *receptive field*.

Typical layers of a Convolutional Neural Network are the input layer, convolutional layers, pooling layers and the fully-connected layer. The convolutional layer is the core building block in every architecture. It calculates the dot product between its weights and a small region of image. Intuitively, it is used for learning specific filters that activate when some visual features appear such as an edge or a color spot. The Rectified Linear Unit (ReLU) layer is an elementwise activation function such as zero thresholding. The pooling layer is used for downsampling, decreasing the size of the parameters. This is also the reason why the image layer should be divisible by 2 multiple times, as multiple pooling layers are often used to continuously decrease the size of the parameters passed through the multiple layers of the network. Typical sizes include 32, 64, 96, 224, 384 and 512. We use 512 pixels for the documents. The fully-connected layer is connected to all neurons of the previous layer, as the name suggests. It can be used for calculating the class scores, one number for each of the different classes.

The spatial arrangement of the convolutional layer can be configured using hyperparameters. The depth stands for the number of filters to use, where each is likely to learn to extract different features in the image. The stride relates to the number of pixels to jump as we slide over the image. Zero-padding is used to pad the area around the image border with zeros, such that the output volume can have the same size as the input volume.

CNNs are the current state of the art in many computer vision areas such as object detection [20]. Supervised learning tasks require ground truth labels for a large set of training data. However, it is often expensive and impracticable to create such huge sets. Thus it is desirable for machine learning techniques to work with **weak supervision**. There are multiple types of weak supervision, namely *incomplete supervision* where only a small subset of all the data is labeled at all. There is *inexact supervision*, where labels are coarse-grained, and *inaccurate supervision*, where the labels are not always correct, i.e. labels are subject to random noise. We concern ourselves with inaccurate supervision due to the sometimes faulty labeling produced by SyncTEX. However, we combine this with a second step in which we perform finetuning on a small set of correctly labeled document images.

## 4.3 Object Detection

In **object detection** the goal is to identify instances of objects of a certain class in an image. This task consists of two steps where we firstly need to identify the bounding boxes of actual objects in the image and secondly identify to which class they belong. Since the second part is basically object classification, we may use a CNN for this. However, this still leaves the part of identifying regions in the image remaining.

A trivial approach to find regions is to simply use randomly chosen parts of the image or iterate over all possible (coarse-grained) rectangles and classify those. It is important to use different sizes and aspect ratios for the rectangles, since different objects can be of different sizes. Afterwards, the image patch is cut out and resized to a fixed size, then fed to a CNN to extract multiple features. A SVM classifies the image while a bounding box regressor refines the bounding box further.

We want to reduce the number of regions to classify considerably to reduce processing time. Instead of using the aforementioned brute-force approach, we use a process which suggests *regions of interest* (ROIs). In *selective search* [18], we perform hiearchical grouping of similar regions in the image based on color, texture, size and shape. **R-CNN** [8] is a method which uses regions recommended by selective search to pass to a CNN for classification and refinement of the bounding box.

**Fast R-CNN** [7] does not pass every single image patch individually through a CNN like R-CNN, but instead passes the entire image through a CNN at the beginning to extract features in the image. Combined with a region proposal method such as selective search, we can use the parts of the feature map that correspond to the image chunk under consideration. ROI pooling resizes every image chunk to a fixed size and extracts the appropriate features. Fully-connected layers return the class and the refined bounding box. Processing time is reduced significantly that way compared to R-CNN.

**Faster R-CNN** [14] is similar to Fast R-CNN but does not rely on the CPU-intensive selective search but instead uses the feature maps returned by the CNN for finding regions. A *region proposal network* determines so-called anchor boxes and ranks them using objectness scores on whether they contain an object or belong to the background. Anchor boxes are found by sliding over the image and for each position propose multiple bounding boxes of different sizes and aspect ratios, but centered on that position.

**Mask R-CNN** [9] adds image segmentation, i.e. the task of identifying which pixels within a bounding box belong to the object and which ones don't. Mask R-CNN bulids mainly on Faster R-CNN but adds an additional step where after ROI pooling two additional convolutional layers build the mask.

### 4.3.1 Mask R-CNN

We build upon a Mask R-CNN implementation found in a GitHub repository [1]. Mask R-CNN is a good choice because it is a recent, flexible framework for object instance segmentation. It is based on Faster R-CNN but only adds a small overhead due to the additional branch for predicting object masks. It received good results in the COCO suite of challenges while also being fast.

The first part of the pipeline refers to data loading and pre-processing code. Pre-configured datasets (like the COCO dataset) can be used or own code for loading a custom dataset may be provided. In the end, we need to provide a list of image files with their masks (list of boolean matrices) and the corresponding class names (as a vector of class IDs). A list of all the classes used has to be provided as well, to each of which a class ID is assigned. It is possible to inspect the images and masks at this stage to debug whether the data loading code is working.

Images are resized to a pre-configured size while preserving the aspect ratio. Non-square images are padded with zeros around the border. Since binary masks can get very large during training, they are resized to a smaller size.

To understand prediction, we take a look at the Region Proposal Network. It works by running a classifier that determines whether there is an object or not in a lot of boxes. These boxes are called the anchors, whose sizes are defined as hyperparameters beforehand. Only the anchors with a high objectness score are passed on to the next stage. The ground truth data is used to identify positive anchors with an $IoU >= 0.7$ and negative anchors with an $IoU < 0.3$. Anything in between is considered neutral and ignored. The bounding boxes are further refined by shifting and resizing to get the final detection box that covers the ground truth object completly.

In the next stage the region proposals are classified. We generate class probabilities and bounding box regressions. Boxes classified as background and boxes with low confidence are removed. In the end we get bounding boxes, assigned classes and confidence scores for all the detected objects. In the last stage, masks are generated for every instance.

### 4.3.2 Mean Average Precision

The neural network returns bounding boxes and the category (`table`, `tabular`, `table_caption`, `table_cell`) for each instance it detects. We have ground truth data which lists all the instances with the same variables that actually are in the picture. To determine whether set $A$ of proposed object pixels matches set $B$ of true object pixels, we look at *Intersection over Union* in Equation 6.

$$IoU(A, B) = \frac{A \cap B}{A \cup B} \tag{6}$$

We consider $IoU > 0.5$ a hit. This number is intentionally low to account for labeling inaccuracies in the hand-labeled ground truth data.

Precision (Equation 7) and recall (Equation 8) are defined as follows ($TP$: True Positive, $TN$: True Negative, $FP$: False Positive, $FN$: False Negative).

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

Precision measures how accurate the predicitions are, that is what proportion of the positive predictions are correct. Recall measures what amount of positives we find.

We use a method based on the one used in the VOC2007 [5] challenge for evaluating object detection. It works by calculating the precision and recall values and then calculating the average precision. *Average precision* (AP) is the average of the maximum precisions at different recall values, basically the curve underneath the precision-recall curve. We use the *interpolated average precision* in Equation 9.

$$AP = \frac{1}{|R|} \sum_{r \in R} p(r) \tag{9}$$

$R$ is the set of recall values and function $p(r)$ returns the maximum precision for that recall value. This metric works on single images or pages. We calculate the per-class document-normalized *mean average precision* (mAP). As the classes aren't easily comparable among themselves, we calculate mAP for each class separately. Because tables of the same document often look the same and we do not want to overemphasize errors of one specific type of table, we first average over all pages of a document and then take the mean of all these values.

## 5 Experiment

We want to demonstrate that the object detection framework Mask R-CNN can be used for finding hierarchically structured elements of a document.

We do not concern ourselves with full document parsing for the experiment, but concentrate on detecting the position of tables and their components. Tables are an integral part of many documents, especially scientific papers. They are a useful tool for presenting sizeable amounts of complex data. In addition to that, tables are highly hierarchical in multiple ways (a table can be captioned and consists of multiple cells arranged in columns and rows). They can be seen as a "minimal example" of full document parsing. Mastering hiearchical tables is a first step towards understanding full documents, which are hierarchical, too, after all.

We investigate ways in which to automatically find the components of tables using convolutional neural networks. This means that we try to detect the bounding boxes of every `table` (includes both the actual table and the table caption), `tabular` (table only), `table_caption`, and `table_cell`. This output could later be used to reconstruct the table according to the format.

We use the arXiv Bulk Data Access[10] to download the entirety of arXiv including processed PDF and source files. The complete set of files is available from Amazon S3 in requester pays buckets (i.e. the downloader pays for the bandwidth). We then sample 30 000 documents with source files which we then process through our table detection pipeline.

The *table detection pipeline* works by first trying to parse the LaTeX file using TexSoup and trying to produce a corresponding output file (PDF). Unfortunately, this may fail due to various reasons such as a different environment (missing packages or different versions), missing dependencies (e.g. images) or a myriad of other LaTeX specialties. TexSoup may fail to read the file if it detects syntax errors. We skip those files, in addition to ones containing rotated pages or plainly too many cells (over 1000 per page).

We use TexSoup to find all the tables of the document as well as the table caption and table cells belonging to that table. TexSoup provides easy-to-use functions for iterating and searching the parse tree. We then use SyncTeX, that needs to be fed the position inside the source file, to determine the location of the table, the table caption and all the table cells in the PDF output file. We write all this information into a JSON file which follows the format described previously.

The annotations generated in this manner are oftentimes imperfect due to SyncTeX's difficulty to figure out the location of some of the cells. We for example observe the last row of the table commonly missing in the annotations. More rarely, random cells are missing. Multiple lines of text in cells are wrongly considered to be separate cells (some authors specify multi-rows as multiple individual lines). Sometimes there is a separate cell covering multiple smaller cells or the entire table. Empty cells (without any text) are missing. When-

[10]`https://arxiv.org/help/bulk_data`

ever subscript/superscript is used (e.g. $x^2$), a separate cell is detected for the subscripted/superscripted text.

We implemented the following heuristics to try to get rid of some of these shortcomings:

- Removing very small cells covered by larger cells
- Removing cells enclosing more than one cell
- Deleting all content lines which enclose the table

However, we are not able to get rid of all the systematic issues of the table detection pipeline.

Research papers have either no tables, one table or multiple tables. We only consider documents that our table detection pipeline detects to include tables (on one or more pages). The documents are split into the datasets shown in Table 1.

| **Type** | **Labeled** | **Document count** | **Page count** |
|---|---|---|---|
| Training dataset | Auto | 9357 documents | 18721 pages |
| Finetuning dataset | Manual | 99 documents | 180 pages |
| Validation dataset | Manual | 397 documents | 750 pages |
| Test dataset | Manual | 393 documents | 748 pages |

Table 1: Datasets

We consciously keep the separation between documents with tables and pages with tables. The reason for that is that tables vary considerably in their appearance over the arXiv dataset (which began in 1991). Tables might have borders for cells, rows or columns, or none at all. Table contents might be left-aligned, right-aligned or centered. Different fonts and styles are used, etc. Nonetheless, tables in the same document look mostly alike. For evaluation, we will take this into consideration.

The majority of documents is in the training dataset, which comprises annotations automatically generated by the table annotation pipeline. The finetuning, validation and test dataset comprise annotations which were also originally generated by the table annotation pipeline. However, these annotations have been corrected by human annotation.

Since using the document annotation interface is a very time-consuming process for creating ground truth data, we opted for utilizing the interface only for creating a finetuning dataset as well as the validation and test dataset. These datasets are not hand-labeled from scratch but we use the labels as returned from the table detection pipeline as a starter. This reduces labeling time considerably, as mostly mistakes need to be corrected (such as missing table cells and captions). We do not label empty cells as they were also omitted by the table detection pipeline. If there is an interest in also finding empty cells, we recommend to write heuristics to infer them after the entire process.

## 5.1 Training

We use Mask R-CNN for training a CNN to detect four entities, namely `tables` (includes both the actual table and the table caption), `tabulars` (table only), `table_captions`, and `table_cells`. We refer to this network as the universal network. Additionally, we train individual expert networks. Every expert network is trained to detect only one of the four entities while ignoring the others.

The Mask R-CNN model uses the weights trained on the COCO dataset as a starting point. These weights are the result of training on real-world images. We use the following stages for training on the noisy labeled dataset and get what we refer to the *base network*:

1. Training network heads (heads only, 40 epochs, learning rate: 0.001)
2. Finetune layers from ResNet stage 4 and up (layers 4+, cumulative 120 epochs, learning rate: 0.001)
3. Finetune all layers (all layers, full 160 epochs, learning rate: 0.0001)

The hyperparameters used for training mostly conform to the ones used by the original Mask R-CNN for training on the COCO dataset. Only minor adjustments were made to cater to the fact that for table cell detection there are many more entities than in a usual picture of the COCO dataset. We chose 1000 instead of 200 maximum instances as tables can easily have over 200 cells.

- Initial weights file: COCO Dataset
- Image dimensions: $512 \times 512$ pixels
- Anchors: 16, 32, 64, 128, 256 pixels
- Train ROIs per image: 1000
- Detection max instances: 1000
- Steps per Epoch: 1000
- Validation Steps: 50

## 5.2 Finetuning

The CNN that results from the training is adapted to work with documents, particularly ones typeset with LaTeX with a one- or two-columnar layout. In initial tests we saw that tables are detected well but also observed shortcomings for cells and table captions. These are likely caused from systematic errors of the table detection pipeline. We therefore enhance the neural network using the finetuning dataset. We opt to only train the network heads and only use a comparatively small number of epochs.

We continue training on the parameters learned in the previous step, using our finetuning dataset, without multiple stages (we refer to this as *non-staged*) and a differing number of epochs and learning rate:

1. Training network heads (heads only)

Alternatively, we continue training on the parameters learned in the previous step, using our finetuning dataset and three different stages (we refer to this as *staged*):

1. Training network heads (heads only, 1 epoch)
2. Finetune layers from ResNet stage 4 and up (layers 4+, cumulative 3 epochs)
3. Finetune all layers (all layers, full 4 epochs, learning rate: divided by 10)

# 6 Evaluation

In the quantitative section, we first compare different hyperparameter settings resulting from the manual hyperparameter tuning process to find out which result in good performance on the validation set. We evaluate on the test set. Then we investigate how large the finetuning dataset needs to be and observe overfitting depending on the number of finetuning epochs. Finally, we want to find out whether the intermediate step of training a network using noisy labels makes a difference. In the qualitative section, we look at some results and discuss what can be improved.

| **Network** | **Epochs** | **LR** | **table** | **caption** | **cell** | **tabular** |
|---|---|---|---|---|---|---|
| (UNIV) | | | 0.94 | 0.74 | 0.77 | 0.59 |
| (UNIV) | 1 | 0.001 | 0.95 | 0.81 | 0.89 | 0.94 |
| (UNIV) | 2 | 0.0005 | 0.95 | 0.84 | 0.90 | 0.96 |
| (UNIV) | 5 | 0.001 | 0.94 | 0.84 | 0.89 | 0.95 |
| (EXP) table | | | 0.94 | | | |
| (EXP) table | 1 | 0.001 | 0.98 | | | |
| (EXP) table | 2 | 0.0005 | 0.98 | | | |
| (EXP) table | 5 | 0.001 | 0.98 | | | |
| (EXP) caption | | | | 0.87 | | |
| (EXP) caption | 1 | 0.001 | | 0.85 | | |
| (EXP) caption | 2 | 0.0005 | | 0.88 | | |
| (EXP) caption | 5 | 0.001 | | 0.86 | | |
| (EXP) cell | | | | | 0.80 | |
| (EXP) cell | 1 | 0.001 | | | 0.89 | |
| (EXP) cell | 2 | 0.0005 | | | 0.89 | |
| (EXP) cell | 5 | 0.001 | | | 0.89 | |
| (EXP) tabular | | | | | | 0.99 |
| (EXP) tabular | 1 | 0.001 | | | | 0.99 |
| (EXP) tabular | 2 | 0.0005 | | | | 0.99 |
| (EXP) tabular | 5 | 0.001 | | | | 0.99 |

Table 2: Object detection results (document-normalized mAP) for combinations of different number of epochs and learning rate (LR). The lines without hyperparameters stand for the base network that was trained using the noisy labeled dataset, whereas all other lines were further finetuned using the hand-labeled dataset. We evaluate both the universal (UNIV) and the individual expert (EXP) networks that specialize on one class each. We find that finetuning improves detection performance considerably. We observe that the number of epochs does not have a large impact on mAP except for the table caption detection.

## 6.1 Quantitative results

The resulting mAP values are displayed in result tables. We train expert networks for the different categories as well as a universal network encompassing all categories. These are displayed on the rows in the tables. The columns display per-class *document-normalized* mean average precision (mAP) results.

### 6.1.1 Hyperparameters

Table 2 shows document-normalized mAP values for the different networks (the universal network with all structure elements and four expert networks with one structure element each). The expert networks are able to match and partly surpass the performance of the universal network with all annotation categories.

Overall, we observe that finetuning increases mAP throughout. This is evident for table cells, where the hyperparameters don't have a large impact. The improvement for table captions is smaller, but here a larger number of epochs results in higher mAP values.

| **Staged** | **Epochs** | **LR** | **table** | **caption** | **cell** | **tabular** |
|---|---|---|---|---|---|---|
| No | 4 | 0.0001 | 0.945 | 0.825 | 0.894 | 0.955 |
| No | 4 | 0.0005 | 0.945 | 0.825 | 0.894 | 0.955 |
| No | 4 | 0.001 | 0.952 | 0.780 | 0.889 | 0.932 |
| No | 12 | 0.001 | 0.950 | 0.803 | 0.877 | 0.946 |
| Yes | 4 | 0.0001 | 0.945 | 0.830 | 0.894 | 0.957 |
| Yes | 4 | 0.0005 | 0.955 | 0.833 | 0.893 | 0.944 |
| Yes | 4 | 0.001 | 0.960 | 0.852 | 0.889 | 0.954 |
| Yes | 12 | 0.001 | 0.962 | 0.837 | 0.890 | 0.947 |

Table 3: Object detection results (document-normalized mAP) of the universal network for combinations of different number of epochs, learning rate (LR), and whether different stages were used. Non-staged means that only the heads of the network are trained during all 4 epochs. Staged means that we first train the heads for 1 epoch, then train ResNet stage 4 and up for 2 epochs, and finally train all layers with a tenth of the learning rate for 1 epoch. We can see that the differences are minimal.

Table 3 shows document-normalized mAP values for the universal network with different hyperparameters with four epochs. We can see that the differences are minimal. However, the staged process returns a higher mAP value for the table caption.

One surprising figure is that the table caption expert network does not show any improvement with finetuning. The initial network already seems good enough and has an even higher mAP than the universal network. Investigating the expert network we find that there are many false positives (see Figure 11d), likely caused by the fact that it simply tries to find shorter paragraphs not contained between two other paragraphs. The high detection can be explained when we look at the metric we use, namely interpolated average precision. It is an interpolated version of the curve underneath the precision-recall curve. In case the true positive was found and there are additional false positives, the recall remains 1 while only the precision values are between 0 and 1. There is only one value 1 on the X-axis and the algorithm dictates to take the maximum of the precision values, such that we end up with a mAP of 1. This is a limitation of the interpolated average precision.

An interesting observation is the low recognition of tabulars for the main network before finetuning. We remember that table is the bounding box that includes all elements related to the table (including the caption), whereas tabular is the bounding box that only includes all the cells. However, this means that if there are missing captions in the training data, table and tabular are of the same size. Because we work with region proposals and regions are classified with one type, we see that in these cases table is preferred over tabular.

### 6.1.2 Summary

| **Network** | **Steps** | **table** | **caption** | **cell** | **tabular** |
|---|---|---|---|---|---|
| (UNIV) | FT | 0.75 | 0.28 | 0.58 | 0.71 |
| (UNIV) | TR | 0.94 | 0.74 | 0.77 | 0.59 |
| (UNIV) | TR+FT | 0.95 | 0.84 | 0.9 | 0.96 |
| (EXP) table | FT | 0.89 | | | |
| (EXP) table | TR | 0.94 | | | |
| (EXP) table | TR+FT | 0.98 | | | |
| (EXP) caption | FT | | 0.54 | | |
| (EXP) caption | TR | | 0.87 | | |
| (EXP) caption | TR+FT | | 0.88 | | |
| (EXP) cell | FT | | | 0.57 | |
| (EXP) cell | TR | | | 0.8 | |
| (EXP) cell | TR+FT | | | 0.89 | |
| (EXP) tabular | FT | | | | 0.93 |
| (EXP) tabular | TR | | | | 0.99 |
| (EXP) tabular | TR+FT | | | | 0.99 |

Table 4: Object detection results (document-normalized mAP) for combinations of training with noisy labels (TR) and finetuning with 2 epochs and a learning rate of 0.0005 with hand-labeled annotations (FT). We evaluate both the universal (UNIV) and the individual expert (EXP) networks that specialize on one class each. Training with a large amount of noisy labels leads to a considerable gain compared to only finetuning with the small amount of hand-labeled annotations. We achieve the best results when combining the two and first training with the noisy labels and then finetuning with the hand-labeled annotations.

Table 4 shows document-normalized mAP values for the network that is trained with the large noisy training dataset and for the network that is only trained on the small hand-labeled finetuning dataset. The results are indicative that training on a larger dataset is clearly beneficial, as it improves mAP values considerably for all categories.

One exception is the tabular category for the universal network. This can be explained by the fact that the table and tabular category often have the same bounding box because of a lacking table caption annotation.

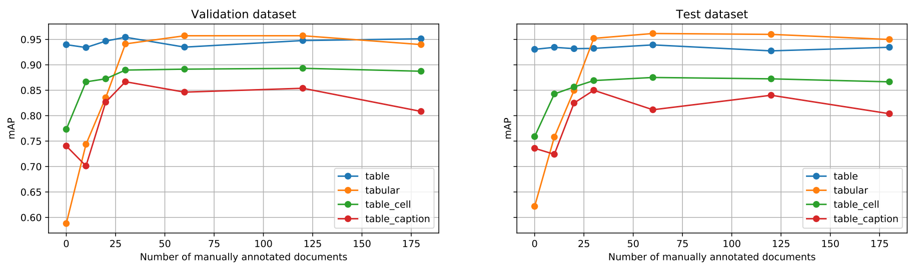


Figure 7: Object detection results (document-normalized mAP) for finetuning the base network (that was trained with the noisy labeled dataset) with differently sized hand-labeled datasets. We observe that the detection of table cells improves considerably with only 10 documents already whereas table captions need at least 30 documents for a similar improvement. These results are positive because they show that large improvements are possible with only a small hand-labeled dataset.

These results are not unexpected. Training starts with the weights of the COCO dataset, which are suited for real-world images, and were trained on a much larger dataset initially as well. The finetuning dataset is very small, though. Because the quality of the automatically labeled training dataset is still good enough, we are able to achieve a better network when using this larger dataset in an initial training phase. The reason for that is that most of the errors in the noisy dataset are of a systematic nature and we expect to be able to "unlearn" them.

### 6.1.3 Finetuning dataset size

Figure 7 shows how the size of the finetuning dataset affects detection performance. We train the heads for one epoch. We would expect the larger the size of the finetuning dataset, the better the detection. That's why only using a handful of images should have a negligible effect on mean average precision. However, as we can see in the diagram this differs depending on the category that we investigate.

Interestingly enough, table cell detection shows a large improvement bump with only 10 documents already. This can likely be explained by the fact that the small finetuning dataset is able to teach the neural network to now consistently detect the cells in the last row instead of willingly leaving them out as it learned from the training dataset. Both the table caption and tabular categories show continuous improvement with a growing finetuning dataset. The reason for that is the consistent presence of table caption annotations in the finetuning dataset with an appropriately enlarged tabular annotation. This was not the case in the training dataset where the table caption was many times missing.

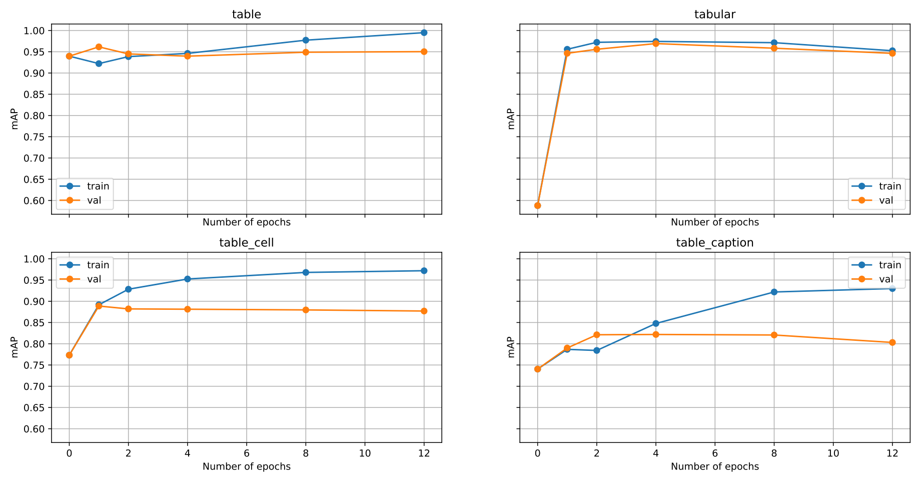


Figure 8: Object detection results (document-normalized mAP) for the different classes on the universal network during training from 1 to 12 epochs on both the training (the hand-labeled dataset that is used for finetuning) and validation dataset. These plots allow us to see after how many epochs we start overfitting. We see mAP for the training dataset to continue growing with each additional epoch, which is to be expected. mAP for the validation dataset stops growing after 2-4 epochs, which indicates that running the training for more than 4 epochs only leads to overfitting to the training dataset.

#### 6.1.4 Overfitting

An interesting observation is that 2-4 epochs of finetuning are enough to achieve a considerable improvement of mean average precision for detection of table caption, table cell and tabular annotations. We want to show that further finetuning only leads to overfitting.

Figure 8 shows four plots for the four classes with mAP results on the training and validation dataset. These plots allow us to argue about after how many epochs we are overfitting. As we can observe, mAP for the finetuning dataset continues growing during all 12 epochs, as would be expected from the fact that the neural network is getting trained with the same dataset and improves detection specifically for it. Here we can detect improvements even after 8-12 epochs. However, the validation dataset shows a different development. It stalls after the first 2-4 epochs, indicating that using any more epochs is only overfitting.

#### 6.1.5 Test dataset

Table 5 shows the same results as Table 4 but on the test dataset instead of the validation dataset. The mAP values on the test dataset are very similar to those on the validation dataset, indicating that we did not exaggerate tuning the hyperparameters for good results on the validation dataset. Looking at results

| Network | Steps | table | caption | cell | tabular |
|---|---|---|---|---|---|
| (UNIV) | FT | 0.78 | 0.28 | 0.57 | 0.73 |
| (UNIV) | TR | 0.93 | 0.74 | 0.76 | 0.62 |
| (UNIV) | TR+FT | 0.94 | 0.84 | 0.88 | 0.95 |
| (EXP) table | FT | 0.85 | | | |
| (EXP) table | TR | 0.93 | | | |
| (EXP) table | TR+FT | 0.95 | | | |
| (EXP) caption | FT | | 0.51 | | |
| (EXP) caption | TR | | 0.85 | | |
| (EXP) caption | TR+FT | | 0.86 | | |
| (EXP) cell | FT | | | 0.56 | |
| (EXP) cell | TR | | | 0.78 | |
| (EXP) cell | TR+FT | | | 0.87 | |
| (EXP) tabular | FT | | | | 0.94 |
| (EXP) tabular | TR | | | | 0.99 |
| (EXP) tabular | TR+FT | | | | 0.98 |

Table 5: Object detection results (document-normalized mAP) similar to Table 4 but on test dataset.

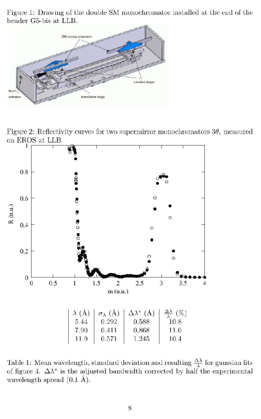
for table caption and cell, we find that results on the validation dataset are only marginally better. This suggests insignificant overfitting.

We confirmed the results of Table 2 and Table 3 for the test dataset, too, and found that mAP values are consistently similar between the two sets (results not displayed here). We observe only marginally better values for table caption and cell on the validation dataset compared to the test dataset.

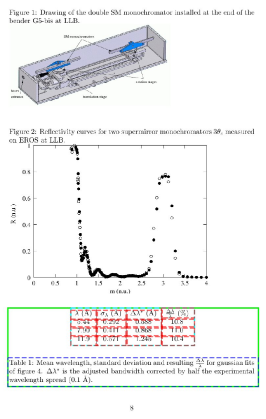
## 6.2 Qualitative results

Figure 9b shows the ground truth data for a document as displayed in Figure 9a. The colors represent the different categories, namely green for the table, light blue for the tabular, blue for the table caption, and red for the table cells.

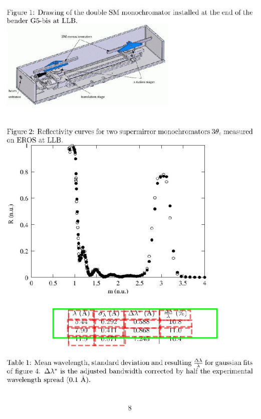
We find that in Figure 9c most cells are detected by the main network. However, two cells in the last row are missing. This is a common artifact of the training dataset. Due to how SyncTEX works, we very often do not detect the entire last row or sometimes the last 1-2 cells. This systematic error was learned in the training phase and can be observed in the majority of tables. In addition to that, the table caption is missing. The reason for that is that the caption is commonly missing in the training dataset, too. Therefore the network learns not to detect in many instances where it can quite easily be detected, such as here. We compare this to more difficult situations where there is a small piece of text above and below the table and it is hard to distinguish which one of the two is a short paragraph and which one is the table caption. The table caption is not always above or below the table as this depends on the specific document layout used. We also note that there is no tabular annotation. The reason for that is that it has around the same dimensions as the table annotation and was



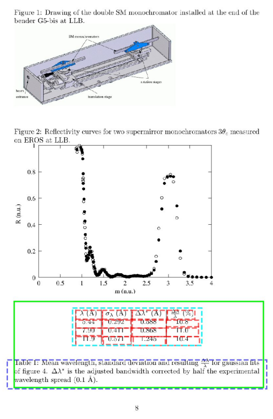
(a) Original



(b) Ground truth



(c) Universal network (TR)



(d) Universal network (TR+FT)

Figure 9: Example of detection performance on a document with the universal network trained on the noisy labeled dataset (TR) and a network trained on both the noisy labeled dataset and finetuned on the hand-labeled dataset (TR+FT). We see the shortcomings of the noisy labels, such as missing cells (red rectangles) in the last row and the missing table caption (blue rectangle). These are typical systematic errors of the table detection pipeline. The results are much better on the network also trained on the hand-labeled dataset, where everything is detected.

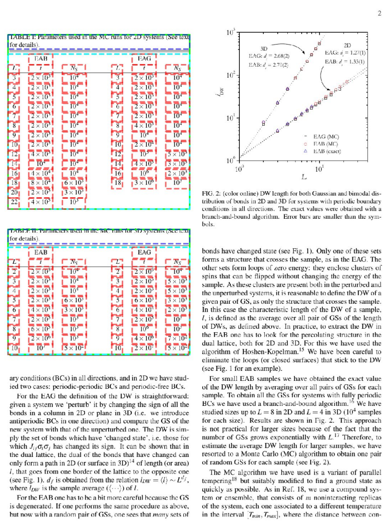

2

TABLE I: Parameters used in the MC runs for 2D systems (See text for details).

| | EAB | | | EAG | |
|---|---|---|---|---|---|
| $L$ | $t$ | $N_S$ | $L$ | $t$ | $N_S$ |
| 3 | $2\times10^3$ | $10^4$ | 3 | $2\times10^3$ | $10^4$ |
| 4 | $2\times10^3$ | $10^4$ | 4 | $2\times10^3$ | $10^4$ |
| 5 | $2\times10^3$ | $10^4$ | 5 | $2\times10^3$ | $10^4$ |
| 6 | $2\times10^3$ | $10^4$ | 6 | $2\times10^3$ | $10^4$ |
| 7 | $2\times10^3$ | $10^4$ | 7 | $2\times10^3$ | $10^4$ |
| 8 | $2\times10^3$ | $10^4$ | 8 | $4\times10^3$ | $10^4$ |
| 9 | $2\times10^3$ | $10^4$ | 9 | $10^4$ | $10^4$ |
| 10 | $2\times10^3$ | $10^4$ | 10 | $2\times10^4$ | $10^4$ |
| 12 | $4\times10^3$ | $10^4$ | 12 | $10^5$ | $5\times10^3$ |
| 14 | $10^4$ | $10^4$ | 14 | $4\times10^5$ | $3\times10^3$ |
| 16 | $4\times10^4$ | $10^4$ | 16 | $10^6$ | $2\times10^3$ |
| 18 | $8\times10^4$ | $6\times10^3$ | 18 | $3\times10^6$ | $10^3$ |
| 20 | $2\times10^5$ | $3\times10^3$ | | | |
| 22 | $4\times10^5$ | $10^3$ | | | |

TABLE II: Parameters used in the MC runs for 3D systems (See text for details).

| | EAB | | | EAG | |
|---|---|---|---|---|---|
| $L$ | $t$ | $N_S$ | $L$ | $t$ | $N_S$ |
| 2 | $2\times10^3$ | $10^4$ | 2 | $2\times10^3$ | $10^4$ |
| 3 | $2\times10^3$ | $10^4$ | 3 | $2\times10^3$ | $5\times10^3$ |
| 4 | $2\times10^3$ | $10^4$ | 4 | $2\times10^3$ | $5\times10^3$ |
| 5 | $2\times10^3$ | $6\times10^3$ | 5 | $6\times10^3$ | $3\times10^3$ |
| 6 | $4\times10^4$ | $3\times10^3$ | 6 | $4\times10^4$ | $2\times10^3$ |
| 7 | $2\times10^5$ | $10^3$ | 7 | $2\times10^5$ | $10^3$ |
| 8 | $6\times10^5$ | $10^3$ | 8 | $10^6$ | $10^3$ |
| 9 | $2\times10^6$ | $10^3$ | 9 | $4\times10^6$ | $7\times10^2$ |
| 10 | $10^7$ | $5\times10^2$ | 10 | $2\times10^7$ | $5\times10^2$ |

ary conditions (BCs) in all directions, and in 2D we have studied two cases: periodic-periodic BCs and periodic-free BCs.

For the EAG the definition of the DW is straightforward: given a system we 'perturb' it by changing the sign of all the bonds in a column in 2D or plane in 3D (i.e. we introduce antiperiodic BCs in one direction) and compare the GS of the new system with that of the unperturbed one. The DW is simply the set of bonds which have 'changed state', i.e. those for which $J_{ij}\sigma_i\sigma_j$ has changed its sign. It can be shown that in the dual lattice, the dual of the bonds that have changed can only form a path in 2D (or surface in 3D)[14] of length (or area) $l$, that goes from one border of the lattice to the opposite one (see Fig. 1). $d_f$ is obtained from the relation $l_{DW} = \langle l \rangle \sim L^{d_f}$, where $l_{DW}$ is the sample average ($\langle\cdots\rangle$) of $l$.

For the EAB one has to be a bit more careful because the GS is degenerated. If one performs the same procedure as above, but now with a random pair of GSs, one sees that *many* sets of

FIG. 2: (color online) DW length for both Gaussian and bimodal distribution of bonds in 2D and 3D for systems with periodic boundary conditions in all directions. The exact values were obtained with a branch-and-bound algorithm. Error bars are smaller than the symbols.

bonds have changed state (see Fig. 1). Only one of these sets forms a structure that crosses the sample, as in the EAG. The other sets form loops of *zero energy*: they enclose clusters of spins that can be flipped without changing the energy of the sample. As these clusters are present both in the perturbed and the unperturbed systems, it is reasonable to define the DW of a given pair of GS, as only the structure that crosses the sample. In this case the characteristic length of the DW of a sample, $l$, is defined as the average over all pair of GSs of the length of DWs, as defined above. In practice, to extract the DW in the EAB one has to look for the percolating structure in the dual lattice, both for 2D and 3D. For this we have used the algorithm of Hoshen-Kopelman.[15] We have been careful to eliminate the loops (or closed surfaces) that stick to the DW (see Fig. 1 for an example).

For small EAB samples we have obtained the exact value of the DW length by averaging over all pairs of GSs for each sample. To obtain all the GSs for systems with fully periodic BCs we have used a branch-and-bound algorithm.[16] We have studied sizes up to $L = 8$ in 2D and $L = 4$ in 3D ($10^4$ samples for each size). Results are shown in Fig. 2. This approach is not practical for larger sizes because of the fact that the number of GSs grows exponentially with $L$.[17] Therefore, to estimate the average DW length for larger samples, we have resorted to a Monte Carlo (MC) algorithm to obtain one pair of random GSs for each sample (see Fig. 2).

The MC algorithm we have used is a variant of parallel tempering[18] but suitably modified to find a ground state as quickly as possible. As in Ref. 18, we use a compound system or ensemble, that consists of *m* noninteracting replicas of the system, each one associated to a different temperature in the interval $[T_{min}, T_{max}]$, where the distance between con-

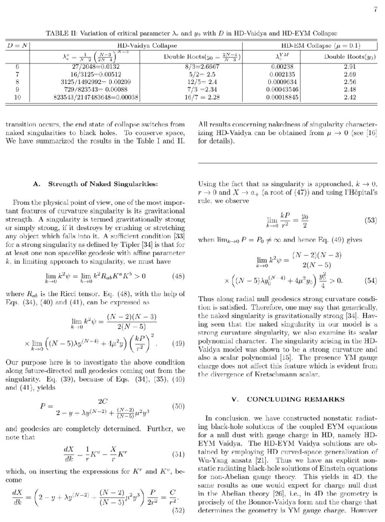
Figure 10: Ground truth for a table without visible 1-pixel border around the cells. Here the widths of the columns depend on the width of the largest cell, as it was returned by the table detection pipeline that uses heuristics to determine the cell sizes from the PDF. The annotations of columns next to each other do not nicely line up.

thus omitted, since Mask R-CNN only detects one category for each region of interest.

Figure 9d shows the improvements after finetuning. Positively, all cells are now detected, the table annotation has the right size, there actually is a tabular annotation and a table caption.

One challenge is on how to define the cell size. Figure 10 shows the ground truth data for a table without borders around the cells. SyncTEX determines that the cells next to each other do not align their bounding boxes. We have retained this definition.

Figure 11a shows a document with a two-column layout and a table on top. Figure 11b is the ground truth data and we see that the table annotation is above the table. Figure 11c shows that while the table caption was detected, a second table caption was erroneously detected, too. After finetuning in Figure 11d the table captions are incorrectly spotted below the table, even though the table and tabular annotations are correct. This is a typical situation where it would be beneficial if the neural network could benefit from the hierarchy created here. An implementation could work with multiple stages where in a first step all tables are detected and in a second step the cells and the table caption are only recognized inside the bounding box of the table. An alternative could also be a loss function which penalizes annotations outside the table.



7

TABLE II: Variation of critical parameter $\lambda_c$ and $y_0$ with $D$ in HD-Vaidya and HD-EYM Collapse

| $D = N$ | HD-Vaidya Collapse | | HD-EM Collapse ($\mu = 0.1$) | |
|---|---|---|---|---|
| | $\lambda_c^v = \frac{1}{N-2}\left(\frac{N-3}{2N-4}\right)^{N-3}$ | Double Roots($y_0 = \frac{2N-4}{N-3}$) | $\lambda_c^{YM}$ | Double Roots($y_0$) |
| 6 | 27/2048=0.0132 | 8/3=2.6667 | 0.00238 | 2.91 |
| 7 | 16/3125=0.00512 | 5/2= 2.5 | 0.002135 | 2.69 |
| 8 | 3125/1492992= 0.00209 | 12/5= 2.4 | 0.0009634 | 2.56 |
| 9 | 729/823543= 0.00088 | 7/3 =2.34 | 0.00043546 | 2.48 |
| 10 | 823543/2147483648=0.00038 | 16/7 = 2.28 | 0.00018845 | 2.42 |

transition occurs, the end state of collapse switches from naked singularities to black holes. To conserve space, We have summarized the results in the Table I and II.

A. Strength of Naked Singularities:

From the physical point of view, one of the most important features of curvature singularity is its gravitational strength. A singularity is termed gravitationally strong or simply strong, if it destroys by crushing or stretching any object which falls into it. A sufficient condition [33] for a strong singularity as defined by Tipler [34] is that for at least one non spacelike geodesic with affine parameter $k$, in limiting approach to singularity, we must have

$$\lim_{k\to 0} k^2\psi = \lim_{k\to 0} k^2 R_{ab}K^aK^b > 0 \qquad (48)$$

where $R_{ab}$ is the Ricci tensor. Eq. (48), with the help of Eqs. (34), (40) and (41), can be expressed as

$$\lim_{k\to 0} k^2\psi = \frac{(N-2)(N-3)}{2(N-5)} \times \lim_{k\to 0}\left((N-5)\lambda y^{(N-4)} + 4\mu^2 y\right)\left(\frac{kP}{r^2}\right)^2. \qquad (49)$$

Our purpose here is to investigate the above condition along future-directed null geodesics coming out from the singularity. Eq. (39), because of Eqs. (34), (35), (40) and (41), yields

$$P = \frac{2C}{2 - y + \lambda y^{(N-2)} + \frac{(N-2)}{(N-5)}\mu^2 y^3} \qquad (50)$$

and geodesics are completely determined. Further, we note that

$$\frac{dX}{dk} = \frac{1}{r}K^v - \frac{X}{r}K^r \qquad (51)$$

which, on inserting the expressions for $K^r$ and $K^v$, become

$$\frac{dX}{dk} = \left(2 - y + \lambda y^{(N-2)} + \frac{(N-2)}{(N-5)}\mu^2 y^3\right)\frac{P}{2r^2} = \frac{C}{r^2}. \qquad (52)$$

All results concerning nakedness of singularity characterizing HD-Vaidya can be obtained from $\mu \to 0$ (see [16] for details).

Using the fact that as singularity is approached, $k \to 0$, $r \to 0$ and $X \to a_+$ (a root of (47)) and using l'Hôpital's rule, we observe

$$\lim_{k\to 0} \frac{kP}{r^2} = \frac{y_0}{2} \qquad (53)$$

when $\lim_{k\to 0} P = P_0 \neq \infty$ and hence Eq. (49) gives

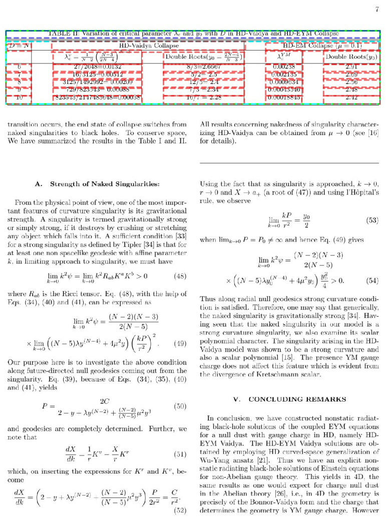
$$\lim_{k\to 0} k^2\psi = \frac{(N-2)(N-3)}{2(N-5)} \times \left((N-5)\lambda y_0^{(N-4)} + 4\mu^2 y_0\right)\frac{y_0^2}{4} > 0. \qquad (54)$$

Thus along radial null geodesics strong curvature condition is satisfied. Therefore, one may say that generically, the naked singularity is gravitationally strong [34]. Having seen that the naked singularity in our model is a strong curvature singularity, we also examine its scalar polynomial character. The singularity arising in the HD-Vaidya model was shown to be a strong curvature and also a scalar polynomial [15]. The presence YM gauge charge does not affect this feature which is evident from the divergence of Kretschmann scalar.

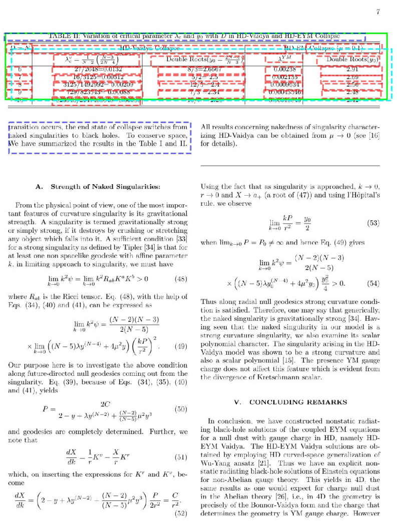
V. CONCLUDING REMARKS

In conclusion, we have constructed nonstatic radiating black-hole solutions of the coupled EYM equations for a null dust with gauge charge in HD, namely HD-EYM Vaidya. The HD-EYM Vaidya solutions are obtained by employing HD curved-space generalization of Wu-Yang ansatz [21]. Thus we have an explicit nonstatic radiating black-hole solutions of Einstein equations for non-Abelian gauge theory. This yields in 4D, the same results as one would expect for charge null dust in the Abelian theory [26], i.e., in 4D the geometry is precisely of the Bonnor-Vaidya form and the charge that determines the geometry is YM gauge charge. However

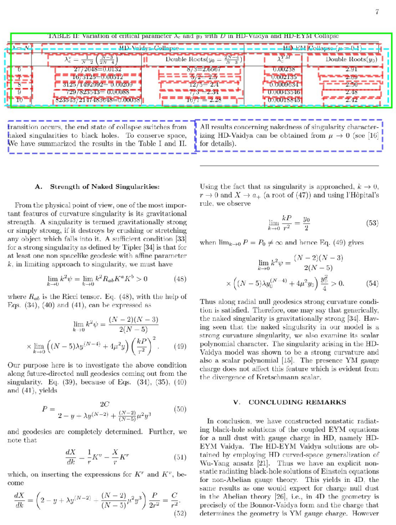
(a) Original



(b) Ground truth



(c) Universal network (TR)



(d) Universal network (TR+FT)

Figure 11: Example of detection performance on a document with the universal network trained on the noisy labeled dataset (TR) and a network trained on both the noisy labeled dataset and finetuned on the hand-labeled dataset (TR+FT). Here we see the shortcomings of the flat detection, where multiple table captions are detected, sometimes above and sometimes below the table.

# 7 Discussion

Having hand-labeled hundreds of documents, it is possible to assess the time needed and the difficulty of doing so. First of all, annotating entire tables from scratch is very time-consuming. As we rely on creating a bounding box for each cell, a bounding box for every single cell has to be drawn accordingly. The process can at least be sped up by having to annotate only one row or column and then using this as a template for the others during the manifold copy-and-paste operation. However, each copied set of cells still needs careful positioning. It might be worthwhile to examine completly different ways of defining the cells of tables, for instance using row/column separators that are placed on the borders between two rows/columns.

We have consciously refrained from additionally specifying the number of row/-column of each cell, even though that would be supported by the JSON format. The reason for that is that it is very time-consuming and error-prone having to select every row and column one after the other and assigning it a number. Instead a heuristics should be written to automatically handle that. Moreover, we only annotated cells which actually contain content and are not empty, because SyncTEX was not able to recognize them at all. It should be evaluated how well it is possible to infer empty cells. An alternative is to also label empty cells for the finetuning dataset.

It is valuable to human annotators if they only have to fix mistakes instead of having to annotate entire tables. It is therefore a good idea to have automatically generated annotations and let the human only fix mistakes in the annotations. With improved heuristics this allows annotators to mainly check the completeness and correctness of documents, only having to fix smaller mistakes.

A common issue observed was the context sensitivity of annotation transformations in the page view. Certain operations, such as adding new annotations, depend upon the user selecting the correct structure element to which it belongs, such as the corresponding table. This is further complicated by the fact that clicks on annotation elements in the page view lead to the selection in the tree view switching to this particular element. This requires selecting the correct element again, e.g. the table to which a newly created cell should be added to.

The document annotation tool has demonstrated that it is useful for creating a dataset of hand-labeled documents. Combining it with automatically labeled documents further reduces time needed for manual annotation. However, it still requires patience and full concentration from the human annotator. We deem the current interface suitable for labeling by an expert but recommend further improvements to the tool's usability for crowd labeling.

The methods used in this thesis do not fully exploit the hierarchical structure of the tables. The hierarchy is useful during annotation creation by automatically calculating the bounding box of the entire table by deriving it from the bounding boxes of the cells. This works by taking the *minimum bounding box* or *minimum bounding rectangle* which encloses the set of objects. In addition to that, the hidden layers of the neural network also implicitly help with detection of hierarchical data, albeit it is not easily possible to find out what these layers are doing exactly.

In addition to only table cells being looked for, table rows and columns could also be fed to the classifier during training. However, these annotations encompass multiple cells next to each other so it is important to ensure that the classifier can deal with such overlapping annotations. Additionally, the shape of rows and columns can be very narrow besides them being immediately next to each other. A limitation is that a region proposal is only assigned up to one category. That is why when table and tabular annotations have the same dimension, only one of the two is detected. This issue occurs for rows and columns when tables have only one of them.

The findings of this thesis can be extended to not only cover tables in documents but all other structure elements, too. Figures are similar to tables in that they also have a caption and float for example at the top of the page. Numbered/unnumbered lists have multiple list items. Headings and paragraphs also comprise hierarchical relationships, similar to how cells are part of a row and a column in a table. Paragraphs are part of a certain section or subsection that is designated a name with a heading. We welcome the research community to expand the number of elements to detect in a document to finally incorporate not just tables but everything found in a common research paper, including equations, lists, figures, etc.

Document layouts are different. There can be one or multiple columns. Paragraphs have an empty line between them or a first-line indentation instead. Figures and tables have their caption above or below them or missing completly. An approach might thus be to first classify documents according to their layout and then train separate neural networks for each different kind of layout. In the evaluation pipeline, one would first classify the entire document with all pages and then use the specific neural network for all pages.

Documents can be of different scale, resolution and rotation. Rotation could be detected using separate heuristics in the beginning and pages could be rotated before feeding them to the neural network. An alternative is to use data augmentation where pages are randomly rotated to ensure that the neural network can handle these documents routinely, too.

Newspaper layouts are also supported by our format. Unlike general documents, newspapers more often exhibit specials like multi-page figures. Those can be represented in our format by having a figure with annotation elements on multiple pages. The same is true for tables. They can continue on the next page or their cells can span multiple rows or columns. Such tables may be represented in our format since the entire annotation tree is stored per document and not per page. However, how to implement this using neural networks is up for debate.

# 8 Conclusion

We found that a tool specificially aimed at labeling documents is missing. We have thus defined a JSON format for storing document annotations and created a tool for manually viewing and editing these annotations. We catered to the specialties of documents by supporting multi-page documents and hierarchical annotations. We labeled hundreds of documents by hand using this tool and found the interface suitable for expert labeling. We have concerned ourselves with the detection of tables and their caption and cells, because tables are a good example for a typical hierarchical structure found in documents. For this task, we based our work on the state-of-the-art object detection framework Mask R-CNN and used weakly supervised learning. We trained a network using documents from arXiv, with noisy labels that we could automatically generate from LaTeX source files. We finetuned them using a small set of hand-labeled documents to get rid of the systematic errors introduced by the noisy labeled dataset. We evaluated the table detection using quantitative and qualitative methods. We showed that it is possible to achieve good results in object detection of tables, table cells and table captions with a noisy dataset and that only a small set of hand-labeled documents is needed to considerably improve detection. However, the neural network model we demonstrated did not fully exploit the hierarchical structure of the annotations, so we expect custom architectures to further improve results. Our tool helps researchers create an open dataset of labeled documents in an area that was previously mainly dominated by private companies without interest of publishing their data. Having a good and large enough dataset is crucial for any further research in this area. We consider our work, namely the document annotation tool and the experiments on table detection, as a first step towards a framework for eventually understanding the structure of entire documents.

## References

[1] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. `https://github.com/matterport/Mask_RCNN`, 2017.

[2] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18:153:1–153:43, 2017.

[3] Jean-Pierre Chanod, Boris Chidlovskii, Hervé Déjean, Olivier Fambon, Jérôme Fuselier, Thierry Jacquin, and Jean-Luc Meunier. From legacy documents to XML: A conversion framework. In Andreas Rauber, Stavros Christodoulakis, and A Min Tjoa, editors, *Research and Advanced Technology for Digital Libraries, 9th European Conference, ECDL 2005, Vienna, Austria, September 18-23, 2005, Proceedings*, volume 3652 of *Lecture Notes in Computer Science*, pages 92–103. Springer, 2005.

[4] George Cybenko. Approximation by superpositions of a sigmoidal function. *MCSS*, 5(4):455, 1992.

[5] Mark Everingham, Luc J. Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[6] Deepika Ghai and Neelu Jain. Text extraction from document images-a review. *International Journal of Computer Applications*, 84(3), 2013.

[7] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[8] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[9] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114, 2012.

[11] Jerôme Laurens. Direct and reverse synchronization with synctex. *TUGBoat*, 29:365–371, 2008.

[12] Loïc Lecerf and Boris Chidlovskii. Document annotation by active learning techniques. In *ACM Symposium on Document Engineering*, pages 125–127. ACM, 2006.

[13] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár,

and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[14] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[15] Prateek Sarkar and Eric Saund. On the reading of tables of contents. In *Document Analysis Systems*, pages 386–393. IEEE Computer Society, 2008.

[16] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[17] Peter W. J. Staar, Michele Dolfi, Christoph Auer, and Costas Bekas. Corpus conversion service: A machine learning platform to ingest documents at scale. *CoRR*, abs/1806.02284, 2018.

[18] Jasper R. R. Uijlings, Koen E. A. van de Sande, Theo Gevers, and Arnold W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

[19] Kwan Y. Wong, Richard G. Casey, and Friedrich M. Wahl. Document analysis system. *IBM Journal of Research and Development*, 26(6):647–656, 1982.

[20] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *CoRR*, abs/1807.05511, 2018.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

## Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

---

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

**Title of work** (in block letters):

Automated visual document parsing with document-level structure annotations

**Authored by** (in block letters):
*For papers written by groups the names of all authors are required.*

| **Name(s):** | **First name(s):** |
|---|---|
| Bissig | Fabian |
| | |
| | |
| | |
| | |

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

| **Place, date** | **Signature(s)** |
|---|---|
| Schwyz, 15.12.2018 | [illegible] |
| | |
| | |
| | |
| | |

*For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.*