Einollah: Layout detection, a road into OCR's Utopia

Vahid Rezanezhad[1], Clemens Neudecker[1], Mike Gerber[1], Kai Labusch[1], Robin Schaefer[1], Konstantin Barier[1] [1]

[1] Staatsbibliotheck Berlin (SBB)

December 15, 2020

Abstract

Many works have accomplished to segment layout of documents in order to do a better Optical character recognition (OCR). Many features in documents can disturb OCR like Drop capitals, curved lines, scan noises out of print space, skewed lines and so on. So, here we have tried to do a perfect layout segmentation to improve OCR results with end-to-end pipeline named EYNOLLAH. Beside layout detection which is implemented by pixel-wise segmentation, we have also used some heuristic methods to classify marginals and to find order of reading for text regions. This tool have used multiple models for variety of tasks where those models are trained with Ground Truth we have in SBB. In this paper we will explain how the models are trained and how the tool in general works.

Keywords— Pixel-wise segmentation, Artificial intelligence, Layout detection, Deep learning, Neural networks

1 Introduction

In the last years, many institutes have the idea of digitalisation in order to have data online and to get the data fast. Digitalization also will helps us to provide a faster searching engine. To do so it is required to transform scanned documents into a digitalized form, namely to read scanned documents by machine. This can be done in two stages, first layout detection and then optical character recognition.

In historic documents we need to recognize for example illustrations, text region and etc. Extracting text regions in documents would enable us to do OCR with machine as well. Recently great improvements have been done in semantic segmentation for images. However this improvements were not enough in the case of historic documents in order to do a clean and great layout detection. We here tried to apply different kind of image classification and semantic segmentation to get a state of art layout and textline detector. By the way for this purpose only artificial intelligence methods were not enough and we had to use also some heuristic methods to boost layout detection.

As already mentioned in EYNOLLAH we are benefited from machine learning for semantic segmentation. Recently many people have tried to use Convolutional neural networks (CNN) models to do layout detection in order to feed optical character recognition (OCR) models. This is now possible with image segmentation which uses Fully Convolutional models (FCN) , Unet , Resnet-Unet or other models. All those models actually build from two parts, encoder and decoder part.

In SBB as Qurator team we first prepared a tool by using pixel-wise segmentation for layout and textline detection and it was working quit good for our data resources in SBB. Our data in SBB have good quality and scale and therefore we did not face big problems with layout detection. But as soon as we received other documents from other source where the data quality were not good or scale was so big we realized serious problems in the tool (**?**). Beside this our previous tool was trained to detect only main regions like background, text regions, images and separators. And the reading order of text regions were not considered. This pushed us to change our strategy and it was this that only a pixel-wise segmentation could not be enough. We realized that for some elements we

need heuristic methods. One of main elements which is detected by heuristic method was marginals. The other thing we add to our pipeline in EYNOLLAH was scale classifier. It is crucial to know the scale of documents in order to do a clean layout detection. So, with all experience gained from previous work and learning from other works like dhsegment tool (**?**), at the end of day we were able to produce a tool which output state of art result in the case of layout detection and textline detection.

Analyzing and testing thousands of documents give us this knowledge that elements of layout are playing a big role in reading order of text regions. We discovered that separators and headers are two main factor in reading order. So using those elements from layout detection we now had this ability to find reading order too. So, EYNOLLAH alongside a layout, textline detection, enhancing, deskewing and scaling can also say sth about reading order. However, in the case of wrong reading order, other methods can be used for finding it. One of the methods we have thought about it and we have "how to" knowledge in SBB to do it is to detect reading order semantically. It means after OCR, using text mining we can arrange text regions based on their semantics.

2 Model

In this work for all pixel-wise segmentation models and even for enhancing we have used a resnet50-unet model. In the case of scale classifier resnet50 encoder with a two dense connected layer at the top is used. For encoder part (resnet50) we applied pretrained weights (weights url). The encoder part, Resnet-50 does map input resolution into low resolution in return it increase features of input. On other side encoder part is mapping from low resolution back into higher original resolution of input image.

The encoder part, Resnet-50, takes advantage of pretrianed weights which are trained with image classification task (imagenet).

Our used Resnet50-unet model has 38.15 M parameters where only parameters of decoder part

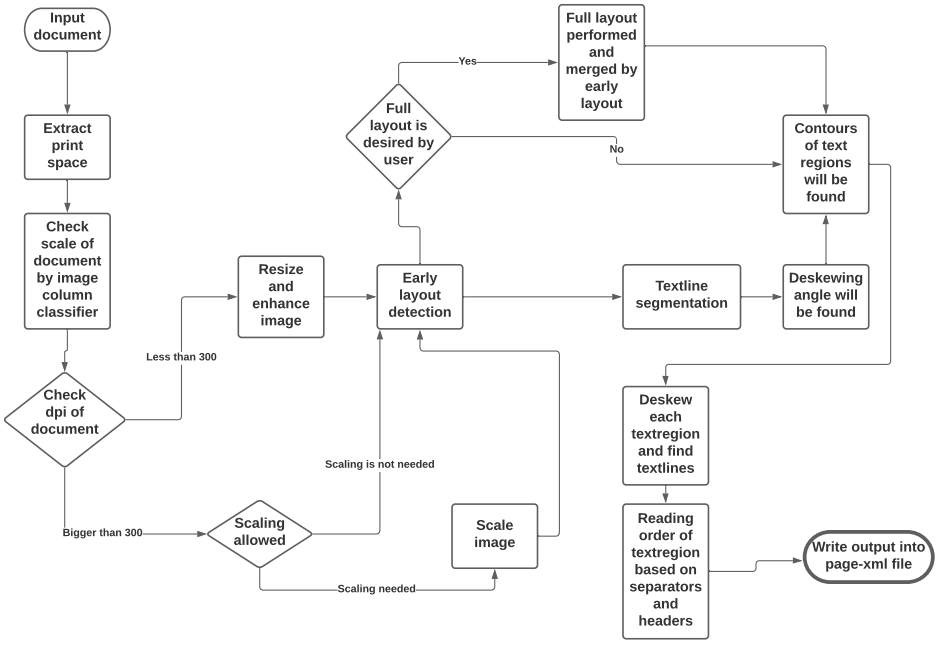

Figure 1: Algorithm flowchart

are trained fully (It means 14.71 M parameters are trained fully). The scale classifier model includes 25.6 M parameters where again only parameters of dense layers are fully trained (2.16 M parameters).

3 Algorithm stages

- Check dpi and scale of Document.
- if dpi is less than 300 and scale is lower than desired scale, document will be scaled and enhanced. During this project we found out in order to do a great layout detection we need to first scale the document to scale of our interest. When we are talking about a great layout detection it does not mean that only a higher IOU metric, however we mean that ability to isolate each textblock from others. We can get documents from different sources with different scales and quality, so scale document to the desired scale of your trained model (we

have trained our models with specific scale of documents) would lead to detect text regions separately and moreover this let the model to recognize headers much easier and better. Since, this is only the scale that differentiate headers from other types of text regions.

- extract print space in document.
- implementing first stage layout detection which its goal is to detect main regions (text regions, images, separators, background) by machine and marginals with heuristic method.
- if full layout is interest of user, this tool would go through document in order to detect headers and drop capitals too.
- textline detection will be done.
- for each text region , slope of deskewing will be calculated
- by applying heuristic method textlines in each text region will be found after deskewing.
- result will be written in a Page-xml data.

4 Scaling

As already mentioned in order to do an optimized layout detection we need to scale input into scale that we have trained our models. In our GT in SBB, our document had somehow same scale. One column documents had width between 1000 till 2000 pixels, for two columns document it was 2500 pixels in width and for three, four and five columns documents it was respectively 3000, 4000 and 5000 pixels. And finally for documents with six columns or more than it , the width was more than 6000 pixel. So, in order to get best performance (separated and isolated text regions) we need to feed model with mentioned scale. To do so we needed to first recognize number of columns in a document. For this purpose we trained an document classifier which returns the number of columns in document. We classified all documents in SBB into six classes which includes documents with

one, two, three, four, five columns and last class was documents with six columns and more. This classification can include obviously more classes but since we have not enough newspapers with more than six columns, therefore we decided to make newspapers with six columns and more that it one class. This is worthy to mention again and highlight the role of scaling in a clean and sensitive layout detection. So, it is recommended to boost GT for scaling purpose.

5 Enhancing

This tool is able to work with all document from any sources with verity of qualities.And one of the key models which we trained in SBB was image enhancing model. If a document with low quality and low scale is fed into our layout model it was so difficult to detect background (or better say the space between text regions) precisely. So, after scaling we applied enhancing model to increase the quality of input.

To train an enhancing model which is again image to image model, we needed documents with low quality and GT would be corresponding image with higher quality. Since we did not have such a GT in SBB and the quality of documents in SBB was good we decided with scaling down and scaling up again to original size, provide low quality versions of our documents. With this method we had already a great data to train our enhancing model.

6 Print space extraction

An early problem that can occur with layout detection and correspondingly affect OCR is out of print space black offsets which are caused mostly by document scanning.Again DNN models help us to extract print space by segmenting document by pixel-wise segmentation method. For print space extraction it is necessary to feed model with full document or better to say that model should not be trained in patches. Dhsegmnet project (**?**) has provided a big GT for print space extraction, however we in SBB also annotated 800 documents more in order to boost our trained model. We

Figure 2: Print space extraction for few examples is shown

have shown few examples of print space extraction in **??**.

7 Layout detection

7.1 Early layout detection: Only main regions

As already mentioned, since object detection does not meet the best performance for OCR purposes (this is checked in SBB and we have seen that object detection for text region detection does not work well for l-form text regions and in the case of documents with multi-columns), so we decided to first train models (pixel-wise segmentation) which can see the documents in patches and tried to detect text regions (or more importantly background) as separate and isolated possible. This

early and important and sensible model only detect main and mother regions in a document like background, text regions, separators and images. Thanks to sensibility of this early layout, we are able to detect marginals in a good manner with heuristic method. This is a key decision and step which help us to detect regions perfectly however this step cost longer time in the case of full layout detection.

7.2 Full layout detection

One of source of problems in a state of art OCR is of course problems occurred because of drop capitals in a document. It was a motivation for our group in SBB to care about drop capitals in layout detection too. Beside drop capitals we also have another important text type which can play key role in reading order detection. As it will be discussed in following sections, headers alongside separators are two main factor in reading order in a document. But this is not the only reason we are interested in headers. For search engines, headers are great source of information. In full layout detection we care at this time more about these two classes. After full layout is performed we merge it with early layout detection.

8 textline detection

Textline detection is a binary image segmentation which help us to detect where the textlines are located. For textline segmentation we had a great GT but the problem was all documents contain only one column. As it is imaginable, even with augmentation (scaling) we still had problem with documents with dense textlines. We could come over this problem with manually scaling the document and some parameterization in our prediction tool in order to prepare GT for bigger variety of documents including more columns in the document. Afterward we trained our model again and this time result was as it was expected.

Thanks to augmentation, our model is capable of segmenting vertical lines too. However as it is

Figure 3: Full layout detection

clear in our results , textline segmentation includes artifacts for example for separators or images. But this artifacts does not affect our result so much since we care only of textlines in text regions. Textline segmentation result has a great role in deskewing and consequently detecting textlines bounding boxes. This amazing role will be explained in the next section.

9 deskewing

Skewed documents can strongly affect textline detection, in order to resolve this of course a robust deskewing is needed. We in SBB applied many different ways and algorithms to do deskewing. At the end we realized that textline segmentation result can help us in this way. The idea was that if we rotate a skewed document for a specific angle, deskewing angle, mean of variance of sum of textline segmentation in direction X should be maximum. This metric helped us to deskew documents properly. You can see this metric for some given documents in **??**.

Figure 4: Pixel-wise textline segmentation

For historic documents we face those that are skewed locally and in a different manner from each other. So, "eynollah" does not only deskew the document but also apply it on each text region. This helped us to detect textline rectangle for documents with heterogeneous skewed regions.

10 Curved lines textline detection

11 reading order

see figures ????

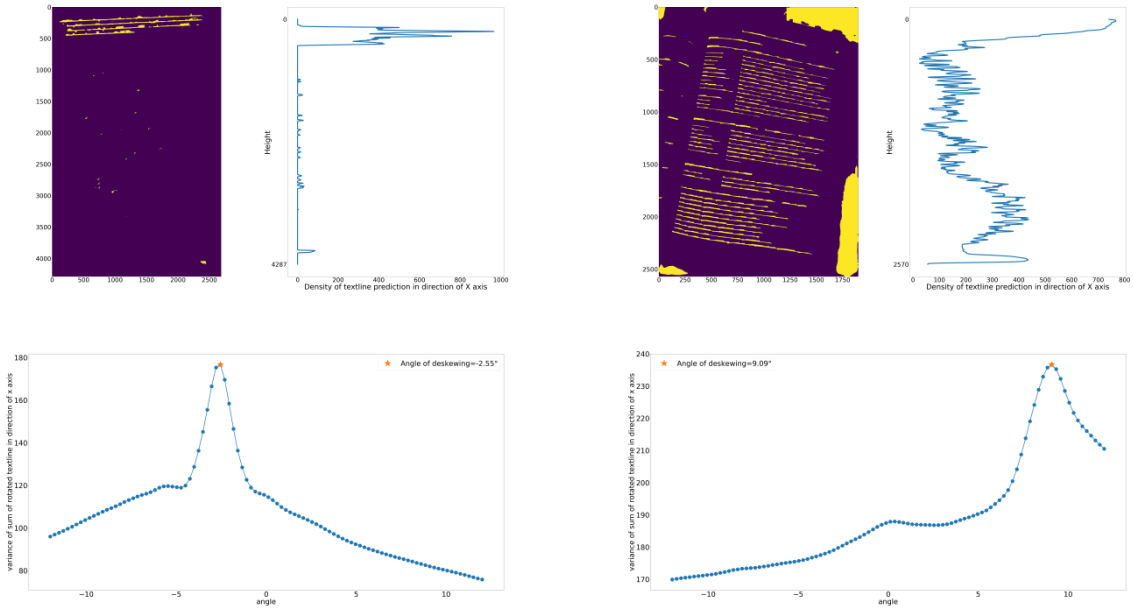

Figure 5: Deskewing procedure

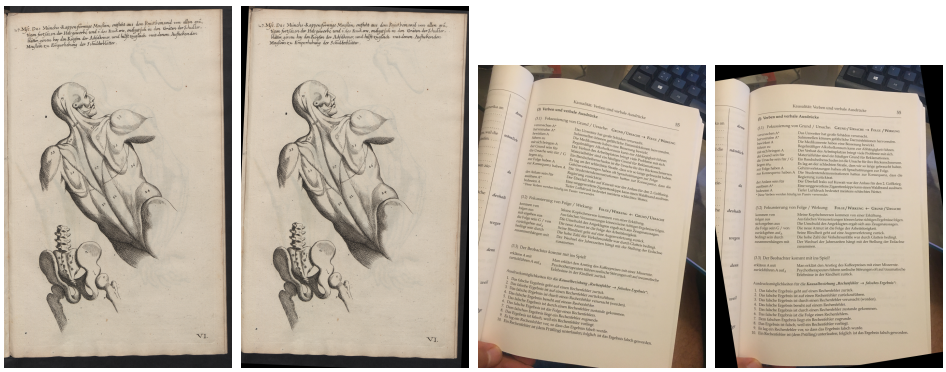

Figure 6: Deskewed documents

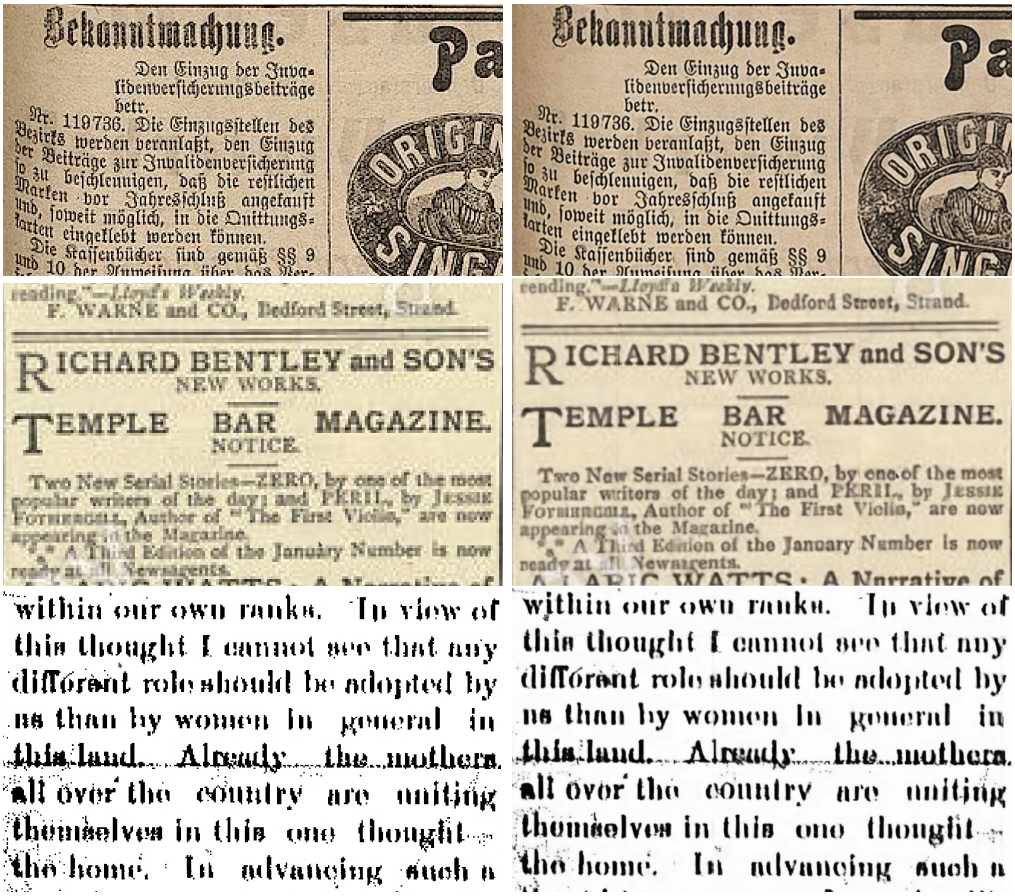

Figure 7: Enhancing. Left figures show org low quality images and on the right side enhanced images can be seen

Figure 8: curved lines detection

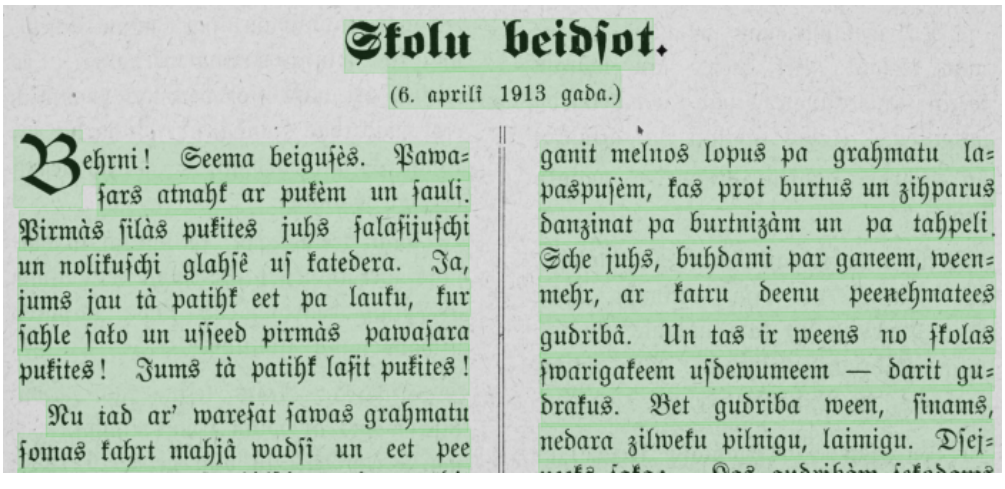

Figure 9: Drop capital integration in textlines

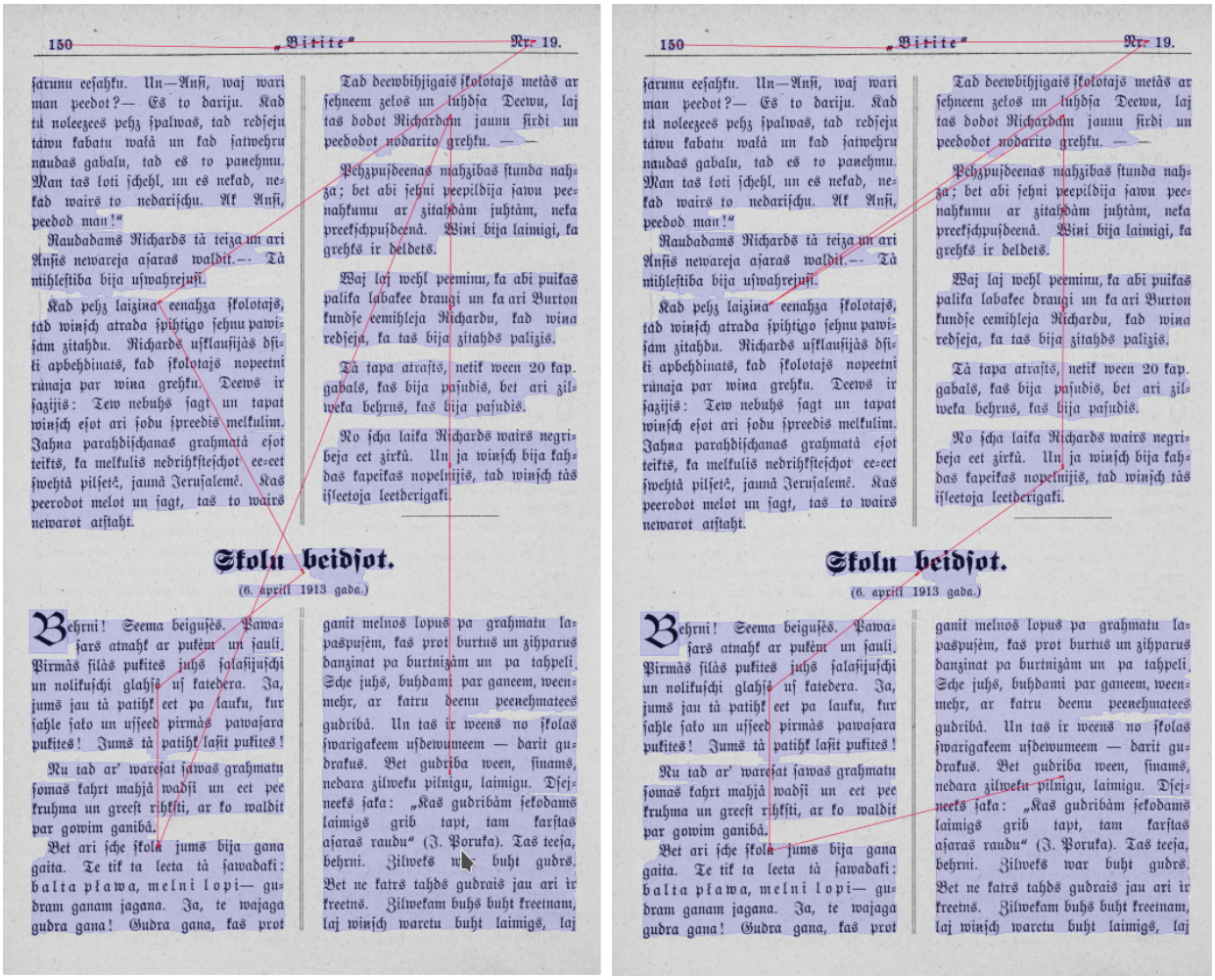

Figure 10: Headers role in reading order

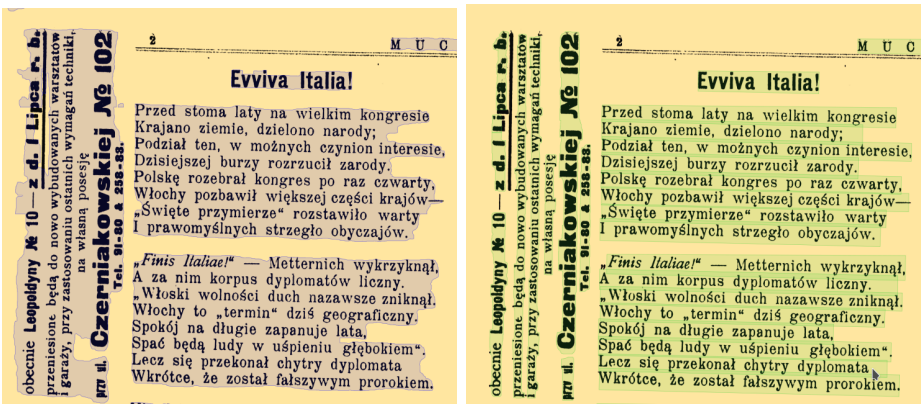

Figure 11: Vertical textlines

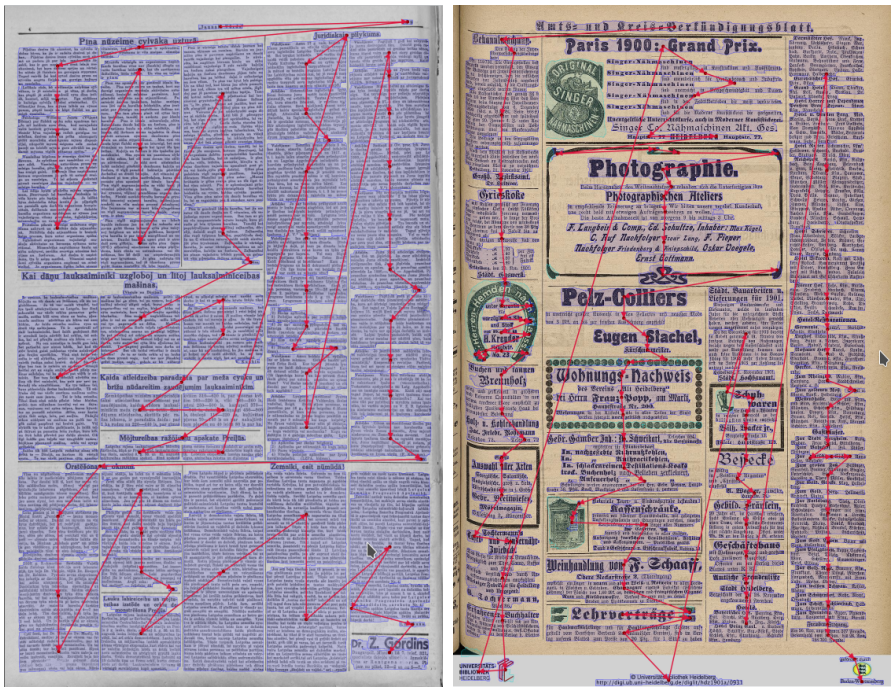

Figure 12: Reading Order