https://doi.org/10.2352/issn.2168-3204.2019.1.0.17

Creating artificial ground-truth data for document image page segmentation

Oliver Paetzel and Hauke Bluhm, both intranda GmbH (Göttingen, Germany)

Abstract

We propose a framework that can be used to create artificial ground-truth data for document images. The resulting data can then be used to train machine-learning systems to perform page segmentation tasks. The main focus of this system is on images of historical documents. The framework creates document images with headlines of differing sizes, multiple column layouts, pictures and decorative elements. To improve the resemblance with historical document images, a set of backgrounds is created manually by extracting background textures from real historical documents. The fading and curling typical of old manuscripts are also simulated.

Experiments with a neural network – trained on data generated using the proposed framework and applied to real-world images – show promising results with robust segmentation of text and non-text image areas.

Evaluating Tesseract 4

OCR is a common procedure when creating large-scale book digitization workflows in the open-source workflow management system Goobi. The OCR process is usually fully automated, i.e. without human involvement. In the past, this step was performed using commercial software that we had to purchase. Unfortunately, troubleshooting was often problematic when errors occurred.

Figure 1. *Cut-out from an uncropped book document image with a large black frame around it..*

When the first beta release of Tesseract 4 became available, we evaluated it against the commercial solution. For simple layouts with just one block of text, we found that Tesseract delivered comparable and sometimes better results. However, Tesseract was less effective when it came to preprocessing document images with more complex layouts. The preprocessing steps that caused problems were binarization, page segmentation and, in some cases, line segmentation.

Preprocessing document images

Tesseract's preprocessing pipeline consists of two steps. The first of these is to binarize the image using Otsu's method [1]. The second is to extract text lines, trying to ignore ornaments and images (a hybrid task that also involves a certain amount of page segmentation). Both parts of the pipeline have proven to be error-prone using the materials that we deal with in our day-to-day work. The binarization process does not handle document images with large black borders very well. The reason is that Otsu's method uses the histogram of the gray-scale image to determine a fixed threshold for the whole image. The black border skews the histogram. As a result, the threshold selected using Otsu's method is too conservative, and faint characters are erased.

Figure 2. *Example of an advertisement with an ornamental frame in which Tesseract detects characters.*

You can see an example in Figure 1. The text line extraction process recognizes possible characters in ornamental frames (see an example in Figure 2) and has problems with skewed text lines and lines that are printed with little or no margin (see Figure 3).

To make Tesseract more effective, we replace the whole preprocessing pipeline with our own implementation. This can be done by using the Tesseract API and only letting it recognize text in previously computed bounding boxes that represent the text lines in the image. Our preprocessing pipeline consists of the usual steps but differs in the order of those steps. We first segment the image into text blocks, images and ornaments. Next, the text lines are extracted from the text blocks, and, as the last step, the text lines are

binarized. As convolutional neural networks have proven to be efficient in image classification [2], [3], object detection [4]–[6] and document image page segmentation [7]–[9], we decided to use this approach for our page segmentation step.

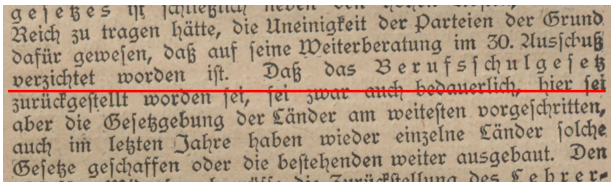


Figure 3. *Example of skewed lines which Tesseract's text line extraction algorithm handles badly. Note the horizontal red line.*

The problem with training data

Although Oliveira et al. [7] and Schreiber et al. [8] have shown that convolutional neural networks are effective in segmenting document images, they did so using relatively small training and evaluation sets with a total of 150 images [10]. These are unlikely to be sufficiently representative of all the data that is digitized in libraries today. As we are unable to create this kind of data manually, we need a framework to automatically generate document images with ground-truth annotations. To mitigate the problem of an insufficiently diverse training set, the framework needs to offer a very wide range of possible layouts and also has to emulate the typical features of historical documents, e.g. fading characters, curling paper and vertically connected text lines.

The document image generator

The proposed application generates completely random document images using two static resources. The background textures and the pictures used in the layout come from a pool that has to be manually created. We also need to configure maximum values for the number of columns and the number of pictures per page.

The actual process of generating document images is divided into three phases: the decision phase, the layout phase and the render phase. The decision phase randomly determines some general layout specifications:

- Render an ornamental frame?
- Render separators between columns?
- Render a top margin for running titles?
- Render a bottom margin for footnotes?
- Render a page number?
- If a page number is rendered: page number position
- Number of columns
- Distance between columns
- Vertical text line distances
- Font size for body text line
- Font size for headings
- Text line skew angle
- Number of pictures to be rendered.

When these decisions have been made, the layout phase begins. Some of these steps are only conditionally executed, depending on the decisions made in the first phase. Also, some decisions (e.g. picture size) have to be made in this phase as the bounds for the picture size are only available after the columns have been fitted. The steps in the layout phase are:

- Reserve areas for the ornamental frame
- Reserve areas for the margins
- Fit column areas into the print space left
- Add pictures into the columns.

Figure 4. *Example of a generated document image.*

After the layout phase, all the ground-truth data has been generated for the ornaments, pictures and text blocks. The last step is the rendering phase, where pictures, ornaments and text are added and filters applied to better emulate historical documents. The rendering phase starts with a fully transparent canvas and involves performing the following steps:

- Render the text on the blank canvas
- Skew the text lines
- Make some black text pixels more transparent
- Select random pictures and add them in the correct position
- Add the background texture.

Figure 4 shows a document image generated using this method.

Evaluation

For our method to be successful, we needed to show that a neural network trained using only artificial data generalizes well enough to work on real-world data as well.

We therefore created a total of three datasets. Two of these were created using the software described above and solely contained artificial data: the training set consisted of 1,000 artificial document images, the validation set 100. The third set was the real-world set. The real-world set was made up of 32 document images collected from German and Austrian libraries. For the real-world set, the ground-truth data was generated manually.

For all three sets, there were two kinds of ground-truth data: bounding boxes for object detection and pixel-level annotations for semantic segmentation. The classes of bounding box were: text block, ornamental frame, and image. The ground-truth data for the segmentation task looked a little different. Here, every pixel in the image was assigned its own label. The labels we used were background, image, frame, and baseline. The baseline label was intended for use in textline extraction. A baseline is the line that most characters touch with the bottom of their bounding box, and some (e.g. "p" and "g") cut to the bottom. The baselines in our ground-truth data were annotated with a 10-pixel-wide line.

Experiments with object detection networks

The first experiments were performed with the Single-Shot MultiBox Detector (SSD) and the bounding-box annotations. SSD was used with vgg16 300x300 trained on ImageNet as the base network and a learning rate of 0.001. The training consisted of 150 epochs using the training dataset. The results of the best model can be seen in tables 1 and 2.

Class	mAP validation	mAP real-world
Text block	0.99	0.58
Ornamental frame	1.00	1.00
Image	0.99	1.00
Average	0.99	0.86

Table 1: *SSD results with IoU threshold=0.5*

While the results look good for an IoU threshold of 0.5, the results with a higher threshold of 0.8 highlight the problem: The bounding boxes did not fit tightly around the text blocks and ornamental frames. An example of this can also be seen in Figure 5. The network was able to detect the general position of text blocks and ornamental frames but could not provide exact bounding boxes. This was not a problem in terms of object detection, but for our specific purpose we needed a clean segmentation of the page.

Class	mAP validation	mAP real-world
Text block	0.78	0.00
Ornamental frame	1.00	0.00
Image	0.64	1.00
Average	0.81	0.33

Table 2: *SSD results with IoU threshold=0.8*

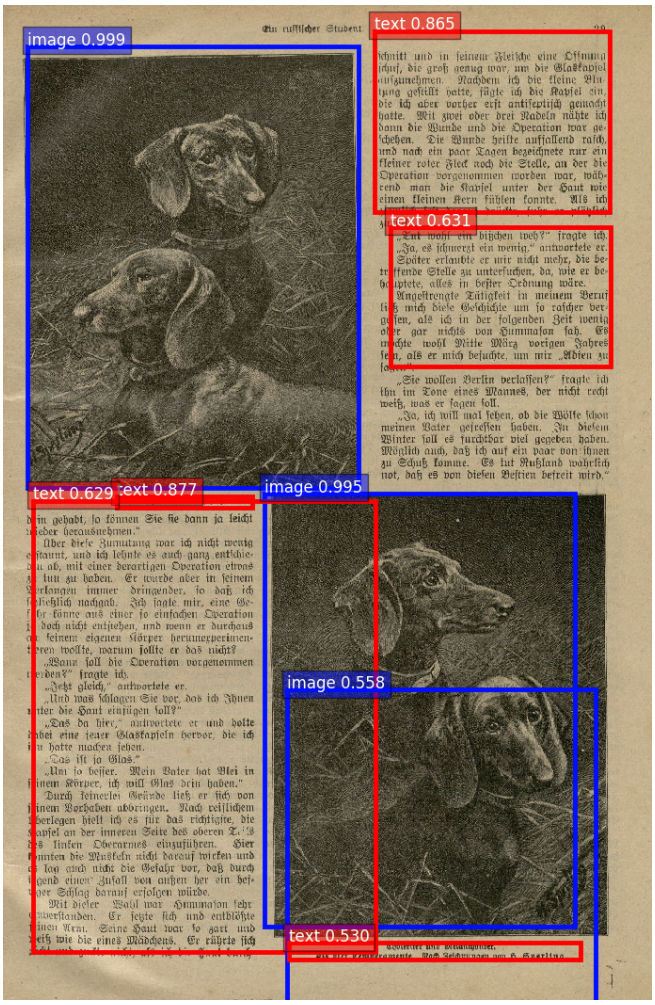


Figure 5. *SSD results for a real-world image*

Experiments with semantic segmentation

For semantic segmentation, we used the dhSegment framework provided by Oliveira et al. [7]. The training images were the same, but the ground-truth data was provided on a pixel level, so each pixel had its own label.

We configured dhSegment to train for 50 epochs, with a learning rate of 5e-5. The results for the validation and real-world datasets can be seen in table 3. Although these results don't seem very promising, a look at the source images with the labels overlaid explains the numbers. An example of a full document image with overlaid predictions is shown in Figure 9.

	validation	real-world
pixAcc	0.78	0.58
mIoU	0.27	0.19
IoU background	0.65	0.54
IoU images	0.29	0.21
IoU text	0.02	0.005
IoU frame	0.13	0.02

Table 3: *Semantic segmentation results*

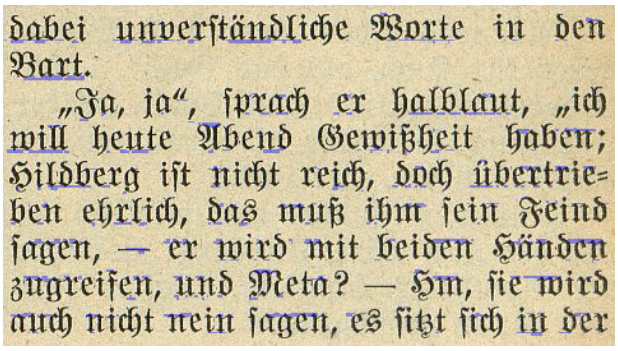
One recurring error that led to poor accuracy values was that page background pixels were misclassified as *image* pixels. This error had two underlying causes: The first had to do with the way we handled images that were rendered to the page. To imitate copper engravings, we binarized a fraction of the pictures before adding them to the document image.

Figure 6. *Example of a binarized picture that led to skewed ground-truth data*

When rendering them on the background, we set every white pixel in the rendered image to be fully opaque. With pictures where there was a light blue sky and the horizon was roughly in the middle of the image, this resulted in 50% fully opaque pixels that were annotated as *image* in the ground-truth data (see Figure 6 for an example).

The second cause for the background misclassification was that the documents we generated for the training run all had very little padding, so there were no examples during training in which larger padding was part of the background.

Baseline extraction was not as successful as we had hoped. The network labeled only parts of the baseline as such and also some pixels at the top of the words' bounding boxes. The issue with parts at the top of lines labeled as baseline might have been due to insufficient line spacing and an excessively large annotation with 10-pixel thickness for the baseline. However, we were pleased to note that text blocks were not misclassified as images or ornamental frames in any of our real-world images. Every text block was labeled as background with the baseline annotation scheme that can be seen in Figure 7.


dabei unverſtändliche Worte in den Bart.

„Ja, ja“, ſprach er halblaut, „ich will heute Abend Gewißheit haben; Hildberg iſt nicht reich, doch übertrieben ehrlich, das muß ihm ſein Feind ſagen, — er wird mit beiden Händen zugreifen, und Meta? — Hm, ſie wird auch nicht nein ſagen, es ſitzt ſich in der

Figure 7. *Example of baseline extraction*

The poor accuracy values obtained for frames using the real-world data appeared to be a result of false positives. In fact, they were not really false positives, but were just not labeled by us, as they were not frames per se, but horizontal rules. These were often annotated as ornamental frames by the neural network. See Figure 8 for an example.

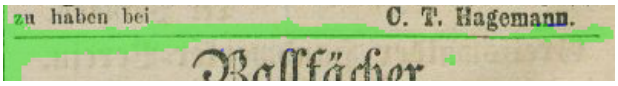


Figure 8. *Horizontal rule that resulted in false positive “frame” pixels*

Conclusions and future work

Our experiments showed that the SSD object detection neural network is not particularly effective for document image segmentation. However, the pixel-based classification approach delivered promising results. The accuracy numbers were not that good, but a look at the actual images reveals that many of the issues were either a problem with the training data or with evaluation data that was annotated without sufficient care. Based on the results that the network yields right now, it would be possible to extract non-textual elements from a document image. The annotation of pictures would still be a problem, as we have false positive areas where we would expect the classification to be *background pixel*. We will attempt to fix this with a higher padding for the whole print space and a more careful choice of images that will be binarized before they are added to the page.

Another problem that remains to be solved is that of baseline extraction performance. This would be the biggest improvement for Tesseract, as the current implementation has problems with skewed lines. Looking forward, our main focus will be on improving that part of the system. One possible approach here could be to change the thickness of the ground-truth lines, i.e. making them a little smaller. Another option, instead of annotating the baseline, would be to mark the whole character and connect all the characters in a line so that the network could extract one connected component per line.

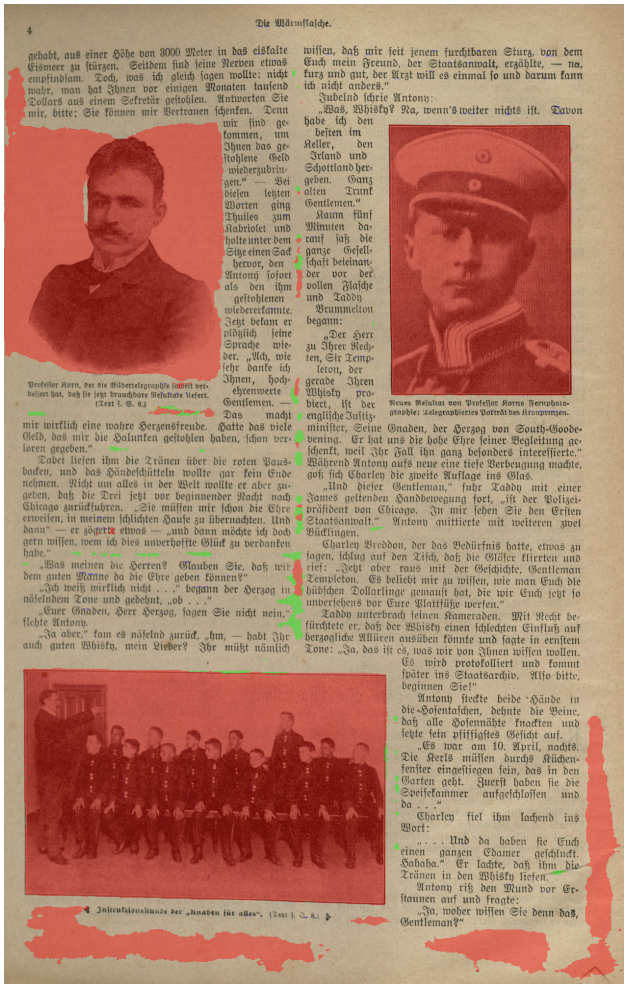


Figure 9. *Example of a real-world document image annotated with the pixel-based annotation approach. Red areas were classified as image, green as ornamental frame, and blue as baseline.*

References

[1] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms", IEEE Transactions on Systems, Man, and Cybernetics, vol. 9, no. 1, pp. 62–66, Jan. 1979.

[2] A. G. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", arXiv:1704.04861 [cs], Apr. 2017.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", arXiv:1512.03385 [cs], Dec. 2015.

[4] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement", arXiv:1804.02767 [cs], Apr. 2018.

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

[6] W. Liu et al., "SSD: Single Shot MultiBox Detector", Computer Vision – ECCV 2016, vol. 9905, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 21–37.

[7] S. A. Oliveira, B. Seguin, and F. Kaplan, "dhSegment: A generic deep-learning approach for document segmentation", arXiv:1804.10371 [cs], Apr. 2018.

[8] S. Schreiber, S. Agne, I. Wolf, A. Dengel, and S. Ahmed, "DeepDeSRT: Deep Learning for Detection and Structure Recognition of Tables in Document Images", 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), 2017, vol. 01, pp. 1162–1167.

[9] M. Alberti, M. Seuret, V. Pondenkandath, R. Ingold, and M. Liwicki, "Historical Document Image Segmentation with LDA-Initialized Deep Neural Networks", arXiv:1710.07363 [cs], pp. 95–100, 2017.

[10] F. Simistira, M. Seuret, N. Eichenberger, A. Garz, M. Liwicki, and R. Ingold, "DIVA-HisDB: A Precisely Annotated Large Dataset of Challenging Medieval Manuscripts", 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), Shenzhen, China, 2016, pp. 471–476.

Author biography

Oliver Paetzel studied applied computer science in Göttingen (Germany), specializing in digital humanities. He joined the German software company intranda GmbH as a software developer in 2012 and concentrates on machine-learning technology and workflow automation. As product manager, he is also responsible for the development of the open-source workflow management tool Goobi.

Hauke Bluhm studied physics in Göttingen (Germany), with a focus on complex systems. In 2018, he joined the German company intranda GmbH as a software developer. His work includes OCR system training and workflow automation with Goobi

.