Line Segmentation in a Historical Document

Computer Vision

Assignment 1

Varun Edachali (2022101029)

January 2025

1 Notes

We aim to segment each line of text in the historical document below.

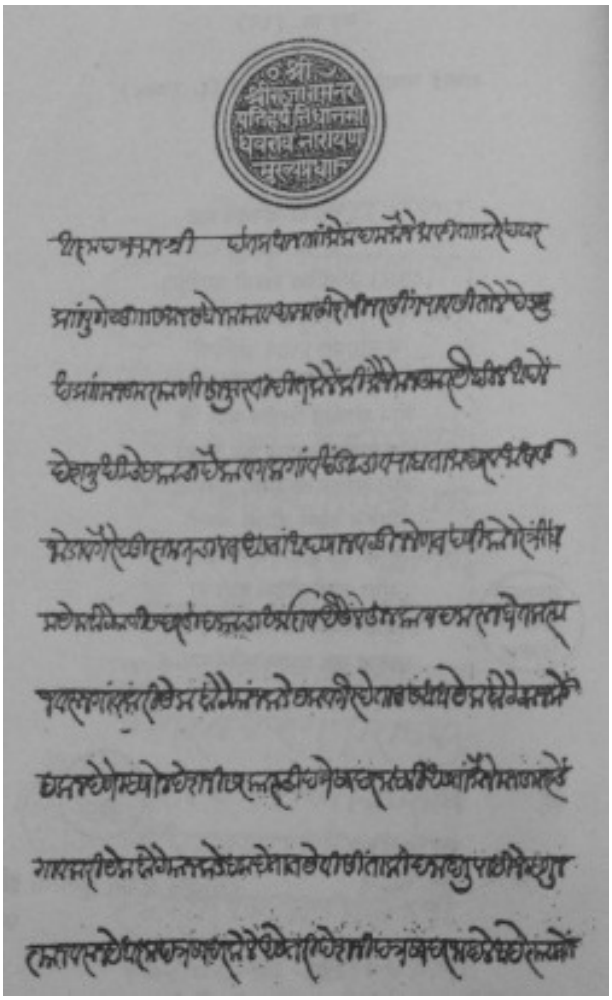

Figure 1: the initial loaded image

1.1 Data Loading and Exploration

We implement a function to load the document image and provide basic information about it.

property	value
shape	(411, 251, 3)
dtype	uint8
min intensity	28
max intensity	190
mean intensity	150.76

The low contrast explains why the image seems dark.

In addition, we calculate the histograms of the image (manually and using opencv):

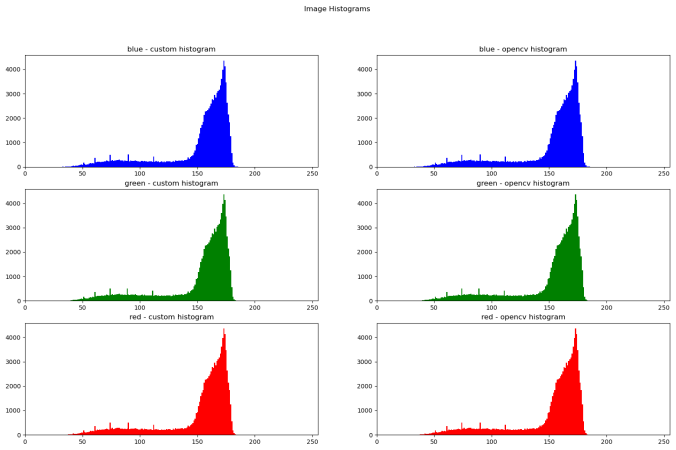

Figure 2: histograms of the given image

The fact that the values of RGB seem similar suggest this is intended to be a grayscale image, but we load it as RGB by default.

1.2 Image pre-processing for text-line detection

The pre-processing takes place in multiple phases, operating on the top and bottom halves separately.

- **grayscale** convert the image into a single channel (just for formality).
- **bottom half**
  - **blur** adding some gaussian blur to help reduce minor noise and peppering.
  - **thresholded** otsu's thresholding to separate the text from the page itself.

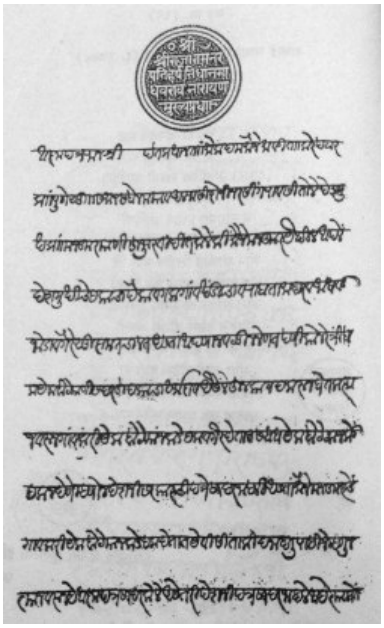

Figure 3: fully gray image

- **top half** (this will take a while)
  - **thresholded** to separate the darker seal from the background. Simple Otsu's worked here.
  - **opening** to highlight the seal (which will help later). A larger kernel of siz (5, 5) helps to completely remove the text and make the seal black, allowing for accurate contour detection in a later step.
  - **invert** to make the text black and the background white
  - **colour the part outside the seal white** by using the result of the morph open (the contour detected on this image is the entire top part apart from the seal).
  - **blur** use gaussian blurring to remove some noise within the seal. Qualitatively, a $(5, 5)$ kernel gave good results, but $(3, 3)$ also worked, suggesting very minor noise removal.
  - **threshold** adaptive thresholding to separate the text from the rest. Simple `Otsu`'s fails to find the correct demarcation. Qualitatively, gaussian thresholding with the parameters in the documentation shown gave good results.
  - **mask** away parts outside the seal to remove the boundary circle and maintain only the text.

  This yields our final pre-processed image.

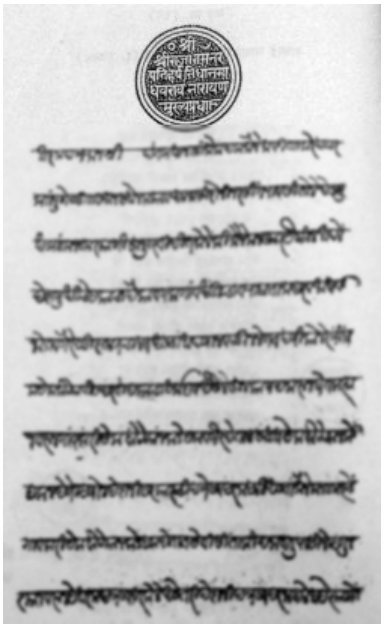

Figure 4: with gaussian blurring

1.3 Rectangular Bounding Box Detection

We simply draw contours on this pre-processed image and draw rectangular bounding boxes around them in order to get the bounding box per continuous set of text.

The reason this was implemented on a per continuous set of text basis and not a per-line basis (by applying morphological operations to get rid of the thin lines, as in the first line of the document) was to provide more accurate polygonal fitting.

We then merge rectangles that have a close value of y (particularly, by sorting the rectangles by the y-coordinate of the top left point, then merging two rectangles if the top left corner is above the vertical mid-point of the immediately previous rectangle) in order to obtain them on a line-wise basis.

NOTE the separate images denoting each line is also stored during the run.

This algorithm heavily depends on good pre-processing, as all it really does is simple contour detection beyond that. The pre-processing, in turn, can be heavily affected by a number of small changes. The processing method for the bottom half is quite general and may work for most clean high contrast documents. The top half, however, assumes light text within a darker circular seal in particular and may fail on other cases. Even with these assumptions, the processing may lose very fine details or lines because of morphological operations, so we require some thicker text on a smooth background to ensure maximal characters survive the pre-processing.

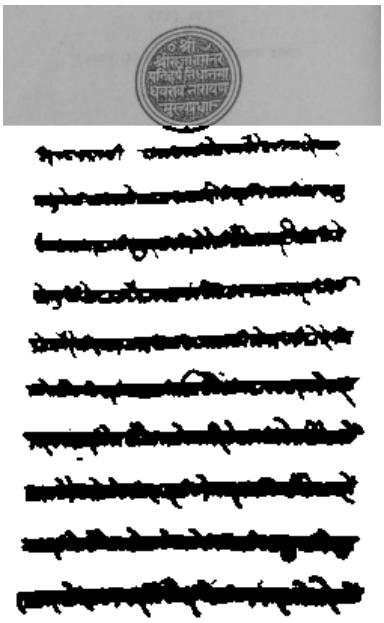

Figure 5: with thresholding

1.4 Segmenting Lines within the Seal

The same method as above is applied on the seal as well. The complications with the pre-processing was discussed before. The primary reason for this complication is because the text is lighter with dark, peppery noise surrounding it. The low contrast and minute details make it difficult to prevent the loss of detail while performing operations, leading to the necessity of separate processing (for me).

1.5 Tighter Polygonal bound on the Text Boundary

We simply draw the detected contours and get a highly accurate bound in both the seal and the document itself, as below.

The polygonal bound detects well separated text as distinct (such as in the first line, and within the seal) while the line-wise segmentation may not. This is an advantage as it prevents the capture of redundant information that line-wise segmentation does not.

Using contour drawing on the results seems to provide good results, capturing most intricacies of the text at hand, but it loses out on a few thin lines that escape its grasp, perhaps because they are connected very feebly (very few or no pixels) to the main body of text. An idea to capture these would be to dilate the text, if the image is higher contrast and the text is thin.

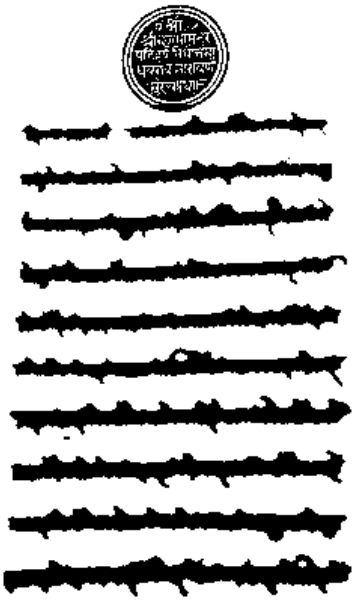

Figure 6: with thresholding

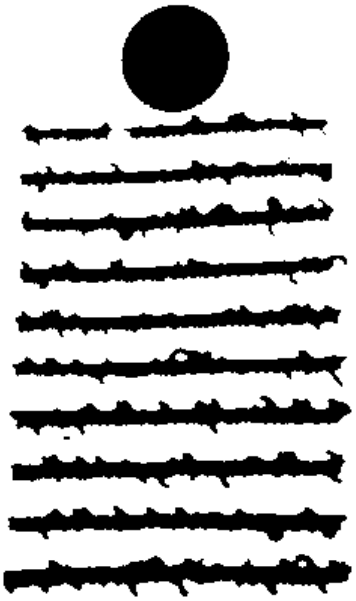

Figure 7: with opening

Figure 8: with inversion

Figure 9: with seal filled

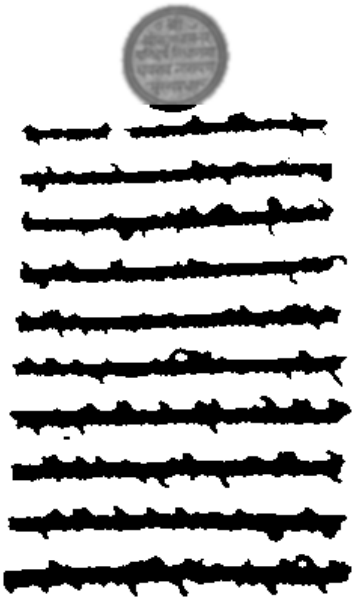

Figure 10: with blurring

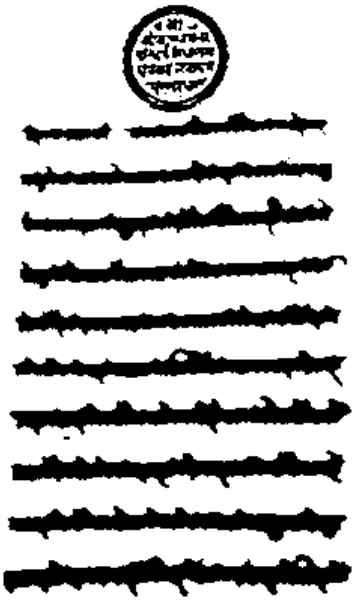

Figure 11: with adaptive thresholding

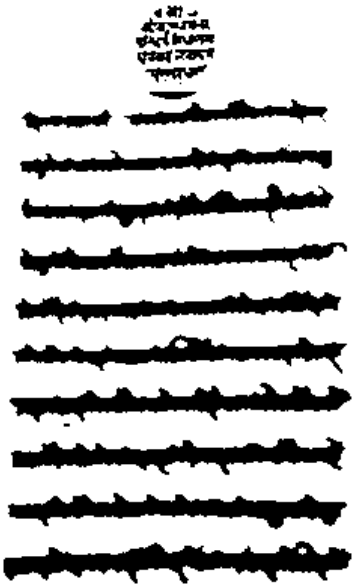

Figure 12: without the seal circle

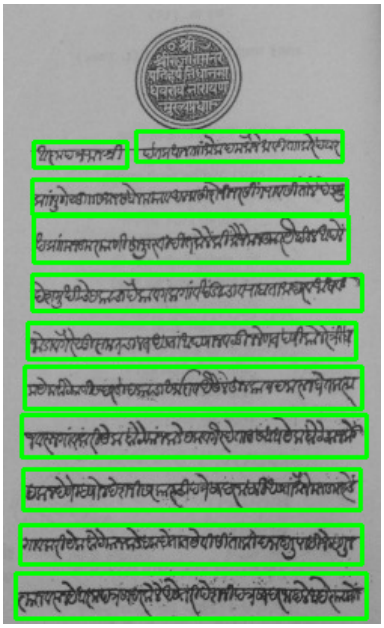

Figure 13: text wise bounding boxes on the document

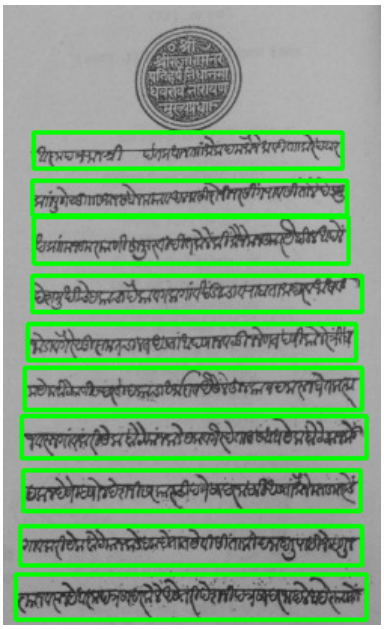

Figure 14: line wise bounding boxes on the document

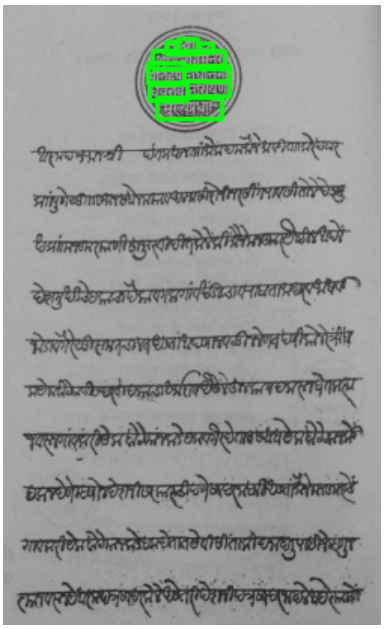

Figure 15: text wise bounding boxes on the seal

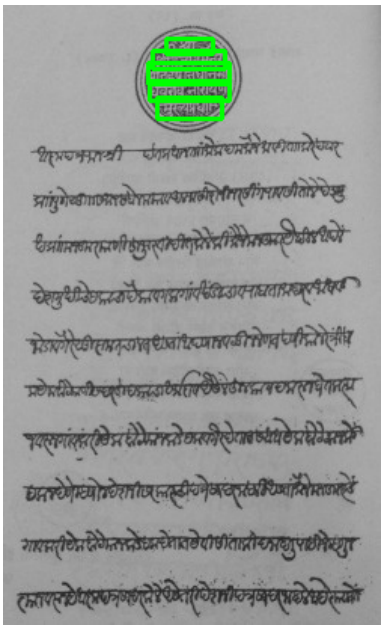

Figure 16: line wise bounding boxes on the seal

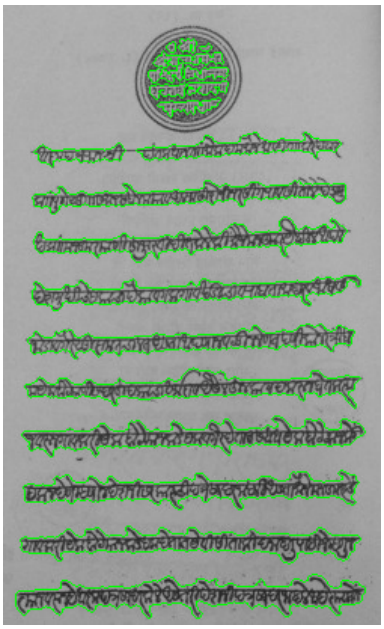

Figure 17: polygonal bounds on image