Automated book detection from bookshelf images

Jack Xiang, Matthias Gabel, Sandeep Reehal, Bledine Messa Ngunte

Abstract: In this article, we explain the method we used to identify individual books from an image of a bookshelf. The techniques we used include segmentation using Hough line transforms, and optical character recognition (OCR).

## 1. Introduction

Locating and identifying in a bookshelf manually is a very monotonous process. Given some general assumptions about how books are placed in a bookshelf, we believe that this process is highly automatable, given an image of a bookshelf with sufficient resolution.

## 2. Methodology

This problem has been explored in the past. We follow a simplified version of the method proposed by Fatema et al. (2022). First we make the following assumptions about the image of our bookshelf:

1. All books are placed on flat shelves which are perfectly horizontal, and take up at least 60% of the width of the image.
2. All books are placed vertically on bookshelves.
3. The spine of each book contains at least 2 words which allow it to be identified.

We follow the general procedure below:

A black background with white rectangles

Description automatically generated

Figure 1: Methodology

### 2.1 Segmentation into rows

After obtaining the input image, we first remove background noise and use Canny edge detection to produce a binary image with the structure lines of the image.

We know that there will be long, horizontal lines corresponding to the row dividers of the bookshelf. We can thus search for these using a Hough transform:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm 1: Divide image of bookshelf into images of individual rows** | | | | |
| **1** | **get** image from input | | | |
| **2** | convert image to grayscale | | | |
| **3** | blur image using gaussian blur | | | |
| **4** | lineList = [] | | | |
| **5** | convert image to binary image with canny edge transform | | | |
| **6** | lines = houghLineTransform(image) | | | |
| **7** | **for** line **in** lines: | | | |
| **8** |  | | **if** |gradient| < 0.2 **and** length(line) ≥ 0.6 \* width(image): | |
| **9** |  | | | add line to lineList |
| **10** |  | | | |
| **11** | sort(lineList) | | | |
| **12** | **for** line **in** lineList: | | | |
| **13** |  | | **if** distance(line – previous(line)) < 0.1 \* height(image): | |
| **14** |  | | | delete line from lineList |
| **15** |  | | | |
| **16** | **for** line **in** lineList: | | | |
| **17** |  | crop(line, previous(line)) | | |

### 2.2 Finding book spine text

We use the easyOCR python module to assist us in finding text bounding boxes for the words on each book spine. As we have separated the bookshelf into individual rows, characters that share the same horizontal alignment in the image must be part of the same spine. Luckily, the easyOCR reader has a parameter where we are able to adjust the horizontal distance for which two text boxes should be merged, so we can easily implement the above method. After obtaining each text box, we store the text in a list for further processing.

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm 2: Find text bounding boxes for each book spine** | | | |
| **1** | setup easyOCR reader | | |
| **2** | set reader maximum width bounding box threshold to INF | | |
| **3** | textList = [] | | |
| **4** | row = 1 | | |
| **5** | **for** image **in** cropped\_images: | | |
| **6** |  | textboxList = readtext(image) | |
| **7** |  | **for** textbox **in** textboxList: | |
| **8** |  |  | add (row, textbox) to textList |
| **9** |  | row = row + 1 | |

### Retrieving book information

To retrieve book information, we first need to filter out irrelevant text. From testing, we found that this mostly happened with (1) Publisher group (e.g. Oxford); (2) Publisher symbols (e.g. Penguin Books symbol). The two aforementioned noise can easily be accounted for by checking the length of the text and the number of space characters:

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm 3: Retrieving book details** | | | |
| **1** | setup Google Books API | | |
| **2** | set number of results for each query to 1 | | |
| **3** | **for** (row, text) **in** textList: | | |
| **4** |  | **if** length(text) ≥ 5 **and** space\_count(text) ≥ 1: | |
| **5** |  |  | initialise new book |
| **6** |  |  | response = googlebooks\_query(text) |
| **7** |  |  | book.row = row |
| **8** |  |  | book.title = getTitle(response) |
| **9** |  |  | book.author = getAuthor(response) |
| **10** |  |  | book.ISBN = getISBN(response) |
| **11** |  | row = row + 1 | |

## 3. Results

Tests on 10 different images of bookshelves were run on a computer with i7-13620H processor and 16 GB of memory. A GPU was not used. The following results were obtained:

**Precision: 74.8%**

**Recall: 60.5%**

**Minimum runtime: 8 seconds**

**Maximum runtime: 111 seconds**

Unfortunately these results showed that while there weren’t many misidentified books, there was a significant amount of books that were not recognised at all. The primary reason is likely that the assumptions we made were largely flawed – in many cases books that were even slightly slanted were not able to be recognised; some books were also placed horizontally by design.

Furthermore, there were issues with the runtime of the program. The largest bottlenecks were cropping the images and running the OCR reader on each cropped image. With the use of a GPU, the runtimes were much faster (no longer than 10 seconds on Google Colab’s T4 GPU).

## 4. Conclusion

We successfully created a program that implements the method outlined above. Unfortunately, due to runtime and recall issues, we believe that our program still has a long way to go before it can be used functionally as a fast and reliable way to identify books.

### 4.1 Further research

We believe that the following steps can be taken to improve the functionality of our program:

1. Improved book segmentation models

There have been proposed models which use techniques such as deep learning (Zhou et al., 2022) models to segment each individual book. This would allow for books oriented at any angle to be detected.

1. Use fuzzy string matching to check if book spine and search query actually are the same

Our method of validating book spine text was fairly rudimentary, and can be further developed by comparing the top result and original text to weed out irrelevant detected text.

Acknowledgment

We would like to thank the AISC club at UC Davis and our group leader, Om, for providing invaluable guidance throughout the development of the project.

## References

1. Fatema, K., Ahmed, M.R., Arefin, M.S. (2022). Developing a System for Automatic Detection of Books. *Second International Conference on Image Processing and Capsule Networks. ICIPCN 2021.*

https://www.asisonline.org/security-management-magazine/monthly-issues/security-technology/archive/2021/december/facial-recognition-in-the-us-privacy-concerns-and-legal-developments/

1. Zhou, S., et al. (2022). Library on-shelf book segmentation and recognition based on deep visual features. *Information Processing & Management.* https://www.sciencedirect.com/science/article/pii/S0306457322002023#sec4