Book Identifier Using Segmentation and Sliding Window Technique

Beth Ann Caruana
Computer Science
Indiana University - Purdue University Indianapolis
Indianapolis, United States
bcaruana@iu.edu

***Abstract*—This project attempted to take a photo of the cover of a book as input and return a website for purchase of the book. The purpose was to provide a convenient way for someone to search a book from a simple photo of the cover. This project has two main components: text detection and text scanning. Text detection was manually implemented and text scanning was provided by a library. Text detection was performed using segmentation in conjunction with the sliding window technique. Classification was done using manual data collection. Overall, the quality of the results were inconsistent and required a lot of fine tuning of variables to get sufficient results for text scanning.**

I. Introduction

Detecting text in natural images is a important technique that can enable the use of self-driving cars or aid blind people in getting around [2]. Although scanning a book cover for ease of search is far from revolutionizing anyone's life, it is a good exercise for identifying the challenges that come along with detecting text in natural images. The goal of this project is to take natural image of a book cover and isolate the text in such a way that a text scanner is able to identify important words in order to perform a search online. Many current implementations of text detection heavily rely on the use of machine learning, and few techniques are present without it. In this paper, I propose a solution to text detection that does not require the use of machine learning algorithms. This project implements the well-known sliding window technique along with my own method for classifying text in an image.

II. Literature Survey

The process of detecting and reading characters in a natural image can be performed in a multitude of ways. Most processes opt to use a Optical Character Recognition (OCR) in conjunction with a text detection program.

OCR is the process for converting an image to text [5]. This is what is commonly referred to as a "text scanner". Standard OCR's work by using a database of various font and text images in order to perform matching algorithms [5]. Methods like OCR can be very limited with regards to text styles that are not stored in the database [5]. Additionally, majority of OCR engines rely on the use of segmentation in order to distinguish characters from their background. Natural images have a wide range of image conditions such as noise, which can provide a challenge for detecting text in natural images compared to black and white documents [3]. In order to address these shortcomings, one can opt to use Intelligent Character Recognition (ICR) software which uses machine learning algorithms to mimic how humans extract text from natural imagesm[5], or utilize some software to reduce noise and isolate text in an image to help the standard OCR.

One approach to text detection includes the sliding window technique, which is performed by placing a window, ideally the size of the thing you are attempting to detect, and classify the window as an object based on some metric [6]. One of the benefits of this technique is that you can run the process for many different window and step sizes which can help detect an object regardless of its size [6]. This technique is simple to implement but can be computationally expensive for large images [7]. In "End-to-End Scene Text Recognition" by Kai Wang et al., the paper proposes a sliding window approach to text detection in conjunction with a character detecting algorithm that extracts a feature vector for a specific location and computes a score that tells the likelihood of a character being in that location [2]. This paper also opted to train their own data for text scanning. Overall, they were able to outperform standard OCR engines.

Despite the success of this group, sliding window is generally considered a poor performer in the text detection space alongside connected components methods (CCs) which detects text by using a filter for basic features such as intensity and gradient and then groups components together based on these characteristics [1]. One fairly successful solution is the Stroke Width Transform (SWT) operator which locates regions in an image where "stroke" widths in edges are uniform and removes inconsistent neighboring pixels [3]. One of the benefits of this method is the system is able to detect text regardless of font, scale, or direction [3]. Lastly one of the most popular solution consists of using Convolutional Neural Networks (CNNs) which is a type of deep learning that is used to detect text in images, similar to the ICR engines [1]. From these approaches, I decided to implement the sliding window classification method while defining my own mechanism for determining a character.

III. Libraries

The following libraries are used to run the program:

- Python3

- NumPy
- Pillow
- webbrowser
- Pytesseract

NumPy was used for pixel array manipulation. Pillow was used to open, convert, and display images. Webbrowser was used to open a new web browser based on a URL. Pytesseract was used to import the OCR scanner. Things to note about the use of Tesseract include the different Page Segmentation Modes (PSMs) which can be used to change how the OCR looks for text in an image. Baseline Tesseract with no PSM modifications assume it is receiving a page of text as an input [8]. In the case of natural images, the text will be much more variable and not necessarily formatted like a paragraph. Therefore, you can use PSMs to specify to the OCR what type of text it should be looking for. There are 14 total PSMs built into Tesseract. For this project, modes 6 (assume a single uniform block of text) and 11 (find as much text as possible in no particular order) were primarily used. The style of the book cover determined the mode used as some covers have uniform text while others do not. Mode 11 is able to pick up text on book covers without uniform text; however, due to it searching for text of any kind, it is extremely susceptible to small noise, where mode 6 is good at ignoring noise, but has trouble picking up non-uniform text.

IV. Implementation

Implementation is broken up into the following sections: segmentation, character classification, and text scanning.

A. Preprocessing

The first step is to segment the image into foreground and background. The goal of this section is to separate the text from its background. Segmentation was performed by building a histogram of the gray values in the image, smoothing the histogram with a mask, locating the minima between two maxima to find the threshold value, and finally separating the pixels based on whether they were above or below the threshold value. Normally, values above the threshold are set to 255 (white/foreground) and values below the threshold are set to 0 (black/background). For this project, these were swapped as text on book covers tend to be part of foreground, but when it comes to using OCRs, they are looking for black text on white background, not white text on a black background; therefore, the values were swapped to make the text black and the background white. Note, for some book covers the segmentation values were swapped back due to the layout of the cover. This was primarily true for covers that contained a lot of white.

B. Sliding Window Classification

After segmentation is completed, the image then goes through the sliding window process. There are three main inputs that control the quality of the sliding window: the segmented image, scale factor, and stride length. Scale factor is the fraction of the width or height (whichever is the minimum) that represents the one side of the square window. The stride length determines the distance from one window sample to the next. Scale factor and stride length can be modified to fit the specific circumstances of an image. The purpose of the sliding window is to separate text regions from non-text regions. If a window is classified as a character, those pixels in the result image will remain, and if a window does not identify a character, then those pixels will be set to 255 (white).

Fig. 1. A

C. Data Collection for Classification

Once a window has been extracted from an image, some mechanism must be used to determine whether that window contains a character. In order to classify a character, I made a word document with every letter from a to z both lowercase and upper case and print them in Arial size 58. I then cropped the the image of each letter of the alphabet into 150x150 squares. See Figure 1 for an example of the capital letter A. All the letter images where then loaded into the LetterLibrary class where the ratio of white pixels where calculated using the following formula:

$$\frac{numberofwhitepixels}{sizeoftheimage} \tag{1}$$

After running this equation for all the letters, it was determined that a character within a square box will have a ratio score between 0.86 and 0.96. I then calculated this value for each window extracted. If the ratio was between .86 and .96, the window was classified as a character. If the ratio was outside this range, then it was not classified as a character. Characters were given a score of 1 and non-character windows were given a score of 0. After the scores where calculated, if a pixel was a 1, the original image pixel was displayed and if the pixel was 0, it was set to 255. The reason for non-characters being set to 255 is because OCRs search for black contrast on a white background, so it was most optimal to ”white-out” any pixels that were not characters rather than ”black-out”.

D. OCR

After the image is finished with the sliding window, there will ideally be an image with black text on a white background. This image will then be fed into Tesseract’s OCR which will return a string of values read from the image.

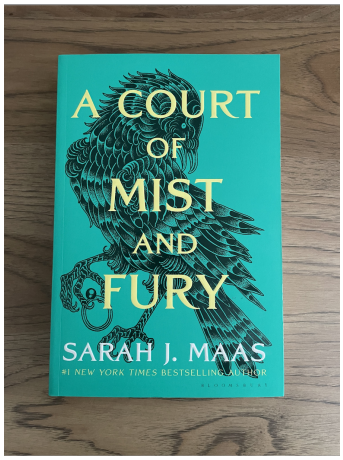

Fig. 2. Working Example: ACOMAF.jpg

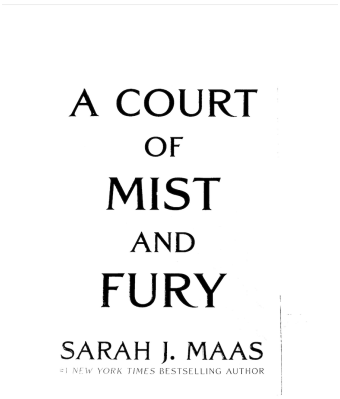

Fig. 3. Threshold: 203, Scale: .17, Stride: .03

E. Output

Lastly, the string produced by the OCR will be added to a google search URL, which once submitted, will open a new tab in the browser with the book.

V. Experimental Results

Overall, results were very mixed depending on the input. Some book covers worked well and others did not. For the results, there will be three categories that results fell into: worked perfectly, detected important words but also a lot of none sense from noise, and did not make it past segmentation.

A. Working

For the working example, the input was in Figure 2. After segmentation and sliding window, the resulting image before OCR was Figure 3. In Figure 3, you can see that the text was isolated cleanly from its background and minimal noise made it past sliding window. This image was then sent to the OCR and returned a string which was sent to google seen in Figure 4. Very few examples worked as well as this one, due to the text being very distinct from its background. This method was also tested on two different textbooks (see appendix) which were also successful. As you will see with the next example, most fiction books do not use text in a way that is easy to isolate or segment.

B. Noisy

For the second category, I will show a book cover that was able to identify the text, but some images and noise were not fully removed during sliding window. The next example is shown in Figure 5. After segmentation and sliding window we got Figure 6. A few things went differently with this compared to the first one. First, segmentation was not able to fully separate the physical book boundaries from the table, which was then picked up by the sliding window to be a character. Second, we see that the outlines of the crowns are not completely whited out. Based on how a character was classified, the edges of the crown were able to pass by as

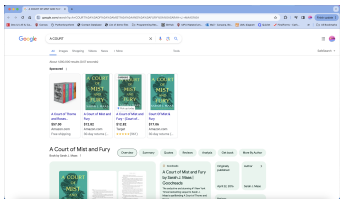

Fig. 4. Google search based on OCR, PSM=11

characters. Lastly, despite the OCR being able to pick up the important words of the title, there was a blurb at the top which mentions a different book by the author. This proved to be a challenge for the search process as the google search pulled up the authors other book instead of this one due to the ordering of words. This proved to be a challenge for multiple fiction books as these blurbs and catch-lines are very common. The results for the search are in Figure 7, which you can see are for the book mentioned in the beginning blurb.

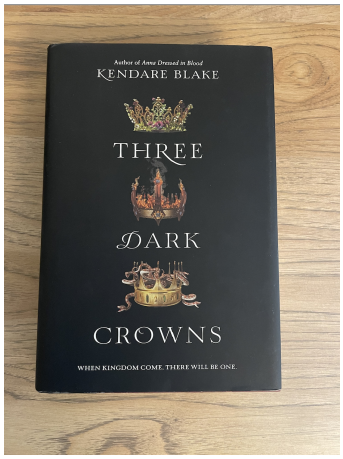

Fig. 5. Noisy Example: crown.jpg

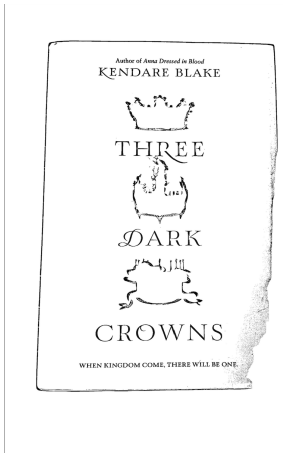

Fig. 6. Threshold: 77, Scale: .02, Stride: .03

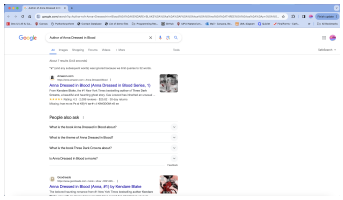

Fig. 7. Google search based on OCR, PSM=11

C. Not Working

Lastly we have the not working section. A lot of books fell into this category on the basis that they were not able to be segmented. This was usually due to the text not being distinct from the background, see Figure 8 and 9. Note the text is completely removed in the segmentation process. Additionally many book covers have text in multiple colors with different background colors or obstruct characters with designs and patterns, see Figures 10, 11, and 12. If the text was not able to be segmented from the background, sliding window was not able to save it.

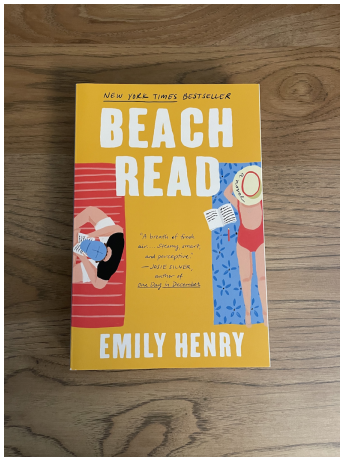

Fig. 8. Not Working Example: beach.jpg

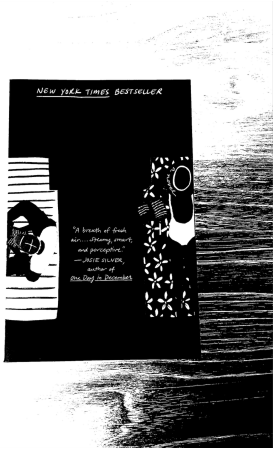

Fig. 9. Post Segmentation

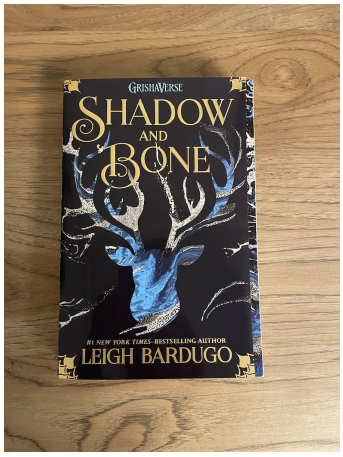

Fig. 10. Not Working Example: shadow.jpg

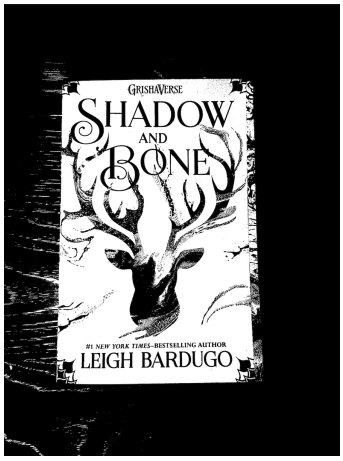

Fig. 11. Post Segmentation: Design is same gray level as text

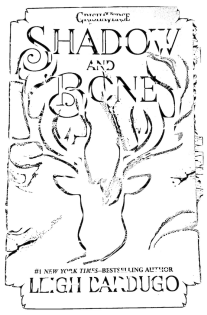

Fig. 12. Post Sliding Window: Cannot distinguish text from design

VI. Strengths and Weaknesses

Although the project worked in some scenarios, most were not successful. The successful approaches worked best for covers with strong contrast between the images on the cover and the text, specifically, this worked well for textbooks which mostly aim to be readable with minimal artwork. There were quite a few weaknesses in this project that lead to the poor results for many covers. One of them is the use of segmentation as a means to separate text from its background. In many cases, the segmentation was not able to isolate the text or removed the text all together, see Figure 9. Now knowing the variety that exists with book covers, my assumption that segmentation would be able to isolate the text was flawed. However, even if the segmentation worked well, there were many complexities with the sliding window that prevented the program from truly "automating". For instance, the scale factor and stride length for each sliding window had to be meticulously adjusted for each book cover as the needs heavily varied from cover to cover due to the large variety of text sizes. Additionally, the classification method proved to not be robust enough, as lost of small noise and borders of objects were able to pass as characters, see Figure 6. The OCR also proved to be very sensitive to noise in images, so unless the segmentation and sliding window worked perfectly, which was far from true in most cases, the OCR still might pick up a lot of random characters between words. Even when the text was distinct, there is a lot of other text on book covers outside the author and title which can heavily impact in search, see Noisy Example in Figures 5, 6, and 7. Lastly, the sliding window technique is quite slow as it has to extract and classify each individual window, which can be a lot of computation for large images if the stride length or window is small.

VII. Analysis and Discussion

This project worked well with book covers that have text that is very distinct from its background, such as textbooks. However, with many traditional fiction books, there is a lot of artwork and stylistic choices that make it difficult to fully isolate the text to be scanned. One hypothesis I formed from this project is that fiction book covers don't aim to be readable but visually appealing. I was shocked by how unreadable many of the covers were in comparison to my textbooks. I then looked into it, and found a poll that listed the most important elements for a good book cover. This list included detailed illustrations, imagery of the stories' setting, compelling color palette, and images of the protagonist [9]. Although this is just a poll, it is interesting to note that none of these mention the title or text on the book. Due to the lack of clear-distinct text in fiction book covers, my method was not very robust. Based on this, if I could start this project over, I never would have attempted to do text detection. Fiction book covers are more artwork than text; therefore, a more robust method would have been to do image matching with a database of book covers. Image matching would be able to work well with the artwork as it would have lots of unique textures to match from image to image, and this method would not require the text to be readable. Additionally, image matching would be robust to blurbs at the top of books that have catchphrases or promotion for other books.

VIII. Conclusion

Text detection in natural images is important skill in image processing that can have a lot of real world application and benefits; however, it comes with a lot of challenges. Most literature that covers text detection in natural images are being performed outside in the real world usually in reference to restaurant and street signs. One distinction is that these real world signs, despite have a large variety of text styles, tend to have large contrast between the text and the background (i.e. street signs tend to be green with white text). One constraint that was not considered for this project was the lack of contrast in text for book covers. This initial assumption was a major reason why this project did not work well in most scenarios. Overall, text detection in natural images was the incorrect approach for the given problem, and the project would have been much more robust if image matching was used instead.

References

[1] Darapaneni, N. et al., "Universal Text Scanner Solution," in *2020 IEEE 15th International Conference on Industrial and Information Systems (ICIIS)*, Rupnagar, India, 2020, pp. 431-436, doi: 10.1109/ICIIS51140.2020.9342646. Keywords: Training; Pipelines; Standards organizations; Computer architecture; Organizations; Proposals; Task analysis; Computer Vision; Sequence Models; Connectionist Text Proposals Network; Text based Images.

[2] Kai Wang, B. Babenko and S. Belongie, "End-to-end scene text recognition," 2011 International Conference on Computer Vision, Barcelona, 2011, pp. 1457-1464, doi: 10.1109/ICCV.2011.6126402. keywords: Pipelines;Training;Text recognition;Optical character recognition software;Image recognition;Object recognition;Detectors,

[3] B. Epshtein, E. Ofek and Y. Wexler, "Detecting text in natural scenes with stroke width transform," 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, USA, 2010, pp. 2963-2970, doi: 10.1109/CVPR.2010.5540041. keywords: Layout;Optical character recognition software;Pixel;Computer vision;Filter bank;Engines;Image segmentation;Colored noise;Geometry;Robustness,

[4] Xiangrong Chen and A. L. Yuille, "Detecting and reading text in natural scenes," Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004., Washington, DC, USA, 2004, pp. II-II, doi: 10.1109/CVPR.2004.1315187. keywords: Layout;Cities and towns;Optical character recognition software;Machine learning algorithms;Testing;Image databases;Software performance;Statistics;Psychology;Legged locomotion,
[5] Amazon Web Services. "OCR - Optical Character Recognition." Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/what-is/ocr/. [Accessed: April 7, 2024].
[6] M. Kropidlowski, "How to Use Sliding Windows for Object Detection with OpenCV and Python," Don't Repeat Yourself, [Online]. Available: https://dontrepeatyourself.org/post/how-to-use-sliding-windows-for-object-detection-with-opencv-and-python/. [Accessed: April 9, 2024].
[7] Supervisely. "How Sliding Window Improves Neural Network Models." Supervisely Blog, [Online]. Available: https://supervisely.com/blog/how-sliding-window-improves-neural-network-models/. [Accessed: April 8, 2024].
[8] A. Rosebrock, "Tesseract Page Segmentation Modes (PSMs) Explained: How to Improve Your OCR Accuracy," PyImageSearch, Nov. 15, 2021. [Online]. Available: https://pyimagesearch.com/2021/11/15/tesseract-page-segmentation-modes-psms-explained-how-to-improve-your-ocr-accuracy/. [Accessed: April 14, 2024].
[9] Study Finds. "Study: People Really Do Judge a Book by Its Cover, According to New Research," Study Finds, [Online]. Available: https://studyfinds.org/judge-a-book-by-its-cover/. [Accessed: April 20, 2024].

IX. Appendix

Here are some additional example images for each of the categories. First, there are two examples of textbook cover and their resulting images that produced the correct results, Figures 13-16. Next are some extra examples, Figures 17-20, for the middle case: detected important words but small noise or taglines at the beginning prevented the search results from working. Lastly we have the failed segmentation section which was unable to cleanly separate the text, Figures 21-24.

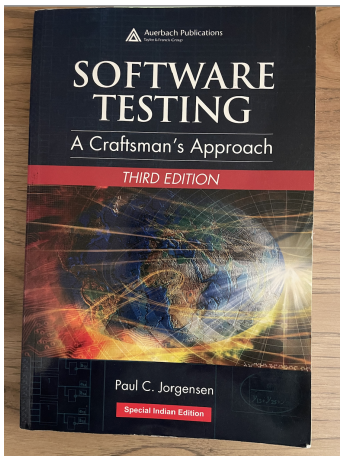

Fig. 13. software.jpg

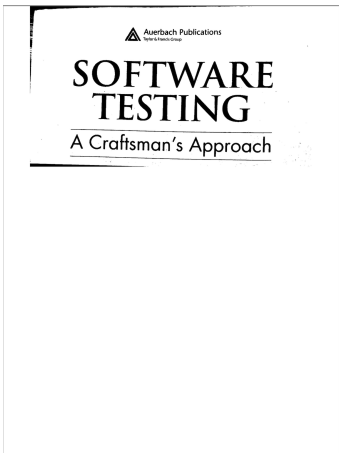

Fig. 14. Threshold: 64, Scale: .4, Stride .1

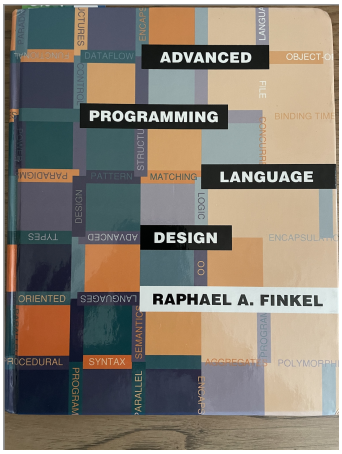

Fig. 15. programming.jpg

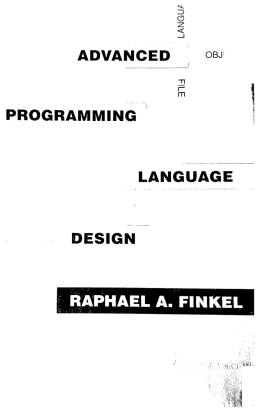

Fig. 16. Threshold: 214, Scale: .3, Stride: .1

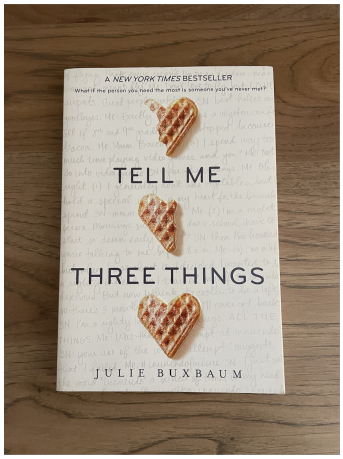

Fig. 17. three.jpg

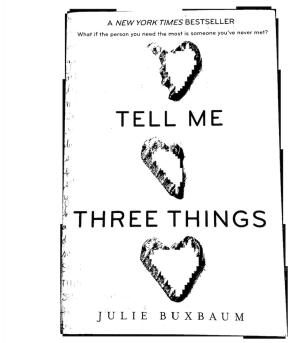

Fig. 18. Threshold: 163, Scale: .07, Stride: .1, (inverse segmentation)

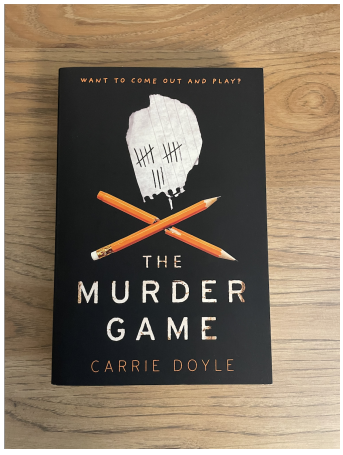

Fig. 19. murder.jpg

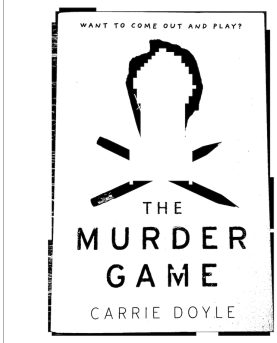

Fig. 20. Threshold: 69, Scale: .11, Stride: .1

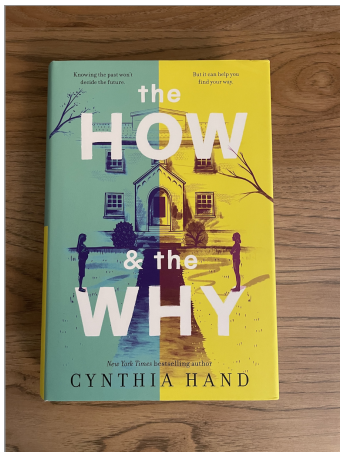

Fig. 21. how.jpg

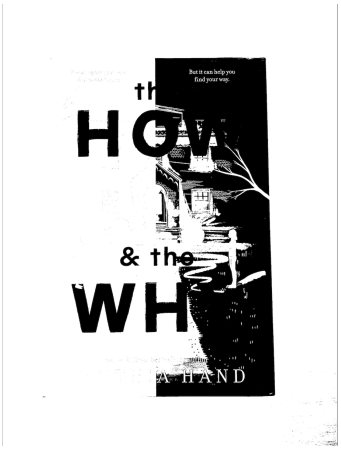

Fig. 22. Segmentation

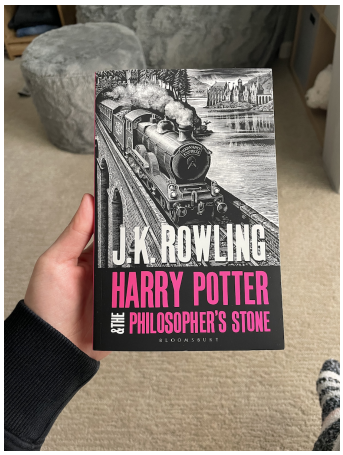

Fig. 23. potter.jpg

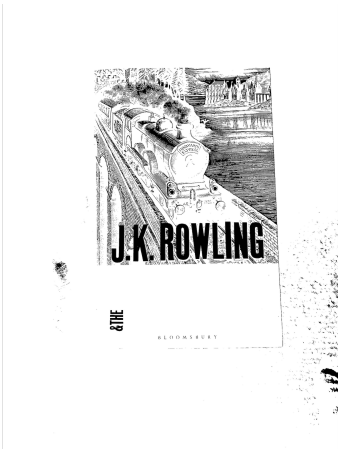

Fig. 24. Segmentation