Optical Character Recognition for Braille

CSCE 580 Project Report

Matthew Duffell-Hoffman

Introduction

Optical Character Recognition (OCR) has been an area of great interest for many years. The ability to convert hard copy printed or even hand written documents into an electronic format which can be searched, translated, converted between file types, or made accessible to screen readers is incredibly useful for businesses, schools, and individuals. OCR is available through commercial applications, open source projects, and is even commonly used as an introduction problem for people exploring machine learning. This project extends the concept of OCR, which has been done for many different written languages, to braille.

Braille is a code used to represent written language in a format accessible to blind people through tactile raised dots. Unified English Braille (UEB,) the main focus for this project, assigns a symbol to each of the letters in the English alphabet, different punctuation marks, and specific braille symbols which have no corresponding symbol in written English. Braille does not transcribe to English in a 1 to 1 character to character manor. A braille cell, or character, is made of six possible dots, labels 1-6 as in Figure 1. Which dots are present in a given cell determines what symbol it is. Six dots which are present or not limits braille to 64 possible permutations for a braille cell. Therefore, some braille symbols are multi-cell. Braille also is very space expensive, so uses contractions to reduce the number of cells needed to represent language. Common words like "the" and "and" and common letter groups like "ing" and "ar" are represented by single cells.

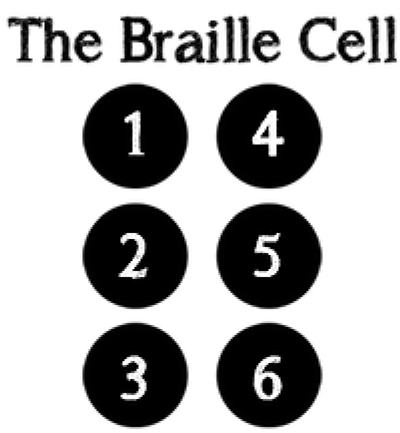

Figure 1: Structure of a braille cell [1]

Since braille does not transcribe directly to written English, an electronic braille format called Braille Ready File (BRF) was developed. This gives a 1 to 1 mapping where each of the 64 possible braille cell permutations map to an ascii character. This file format is used by embossers, braille printers, and electronic braille displays. The braille characters for the 26 letters map to the ascii symbol for the letter, but symbols for contractions and braille specific symbols map to various alternative ascii characters. This makes BRF files not quite human readable. The BRF file format is the target for this project, since it is a 1 to 1 mapping to hard copy braille and programs to convert from a BRF file to plain text are readily available.

Approach

The approach taken for optical character recognition in this project follows the traditional OCR procedure of image acquisition, preprocessing, text recognition, and postprocessing [2]. The text recognition step has been further broken into two steps, segmentation and recognition, following an older, simpler approach to OCR compared to modern methods [3].

Image Acquisition

For this project, image were acquired using the feeder scanner on a HP Officejet 250 Mobile All-in-One Printer. No special step were taken to improve the scan, the braille page was simply fed through the scanner. The image used for validation of the complete system is shown in Figure 2.

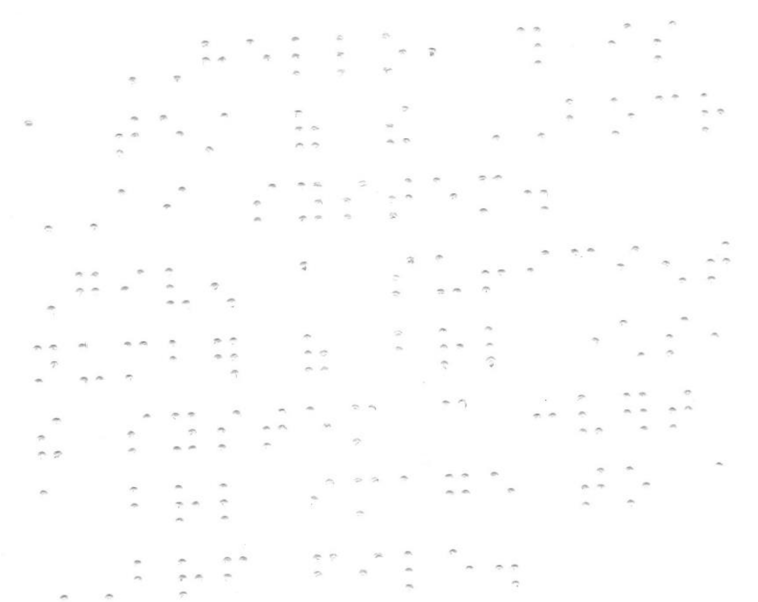

Figure 2: Short braille paragraph used for system validation

Preprocessing

Preprocessing is a series of data preparation steps taken to improve the quality of the scanned image for OCR. For BOCR, the following preprocessing steps are taken: conversion to grayscale, filtering, thresholding, and deskewing.

Grayscale

A traditional image has three color channels, red, blue, and green. For the purposes of OCR, it is much simpler to use a single grayscale intensity channel. The conversion to grayscale from color was accomplished using OpenCV's load as grayscale function and the results on the validation image are shown in Figure 3.

Figure 3: Grayscale image

Filtering

The grayscale image is then filtered to smooth edges and gain a better representation of the braille characters. Open CV's implementation of a Bilateral Filter was used. A Bilateral Filter blurs a pixel with pixels that are both nearby and similar in intensity . This has the effect of smoothing the image while preserving edges [4]. Figure 4shows the results of the filtering operation.

Figure 4: Result of Bilateral Filter Applied to validation image

Binary Thresholding

Binary thresholding is the process of replacing every pixel in an image with two possible values. For this project, each pixel in the image is marked as either black or white, using Open CV's implementation of Otsu's thresholding algorithm. This algorithm finds a threshold value which minimizes the variances of the two classes, black and white pixels [5]. The result of Otsu's thresholding applied to the filtered image is shown in Figure 5.

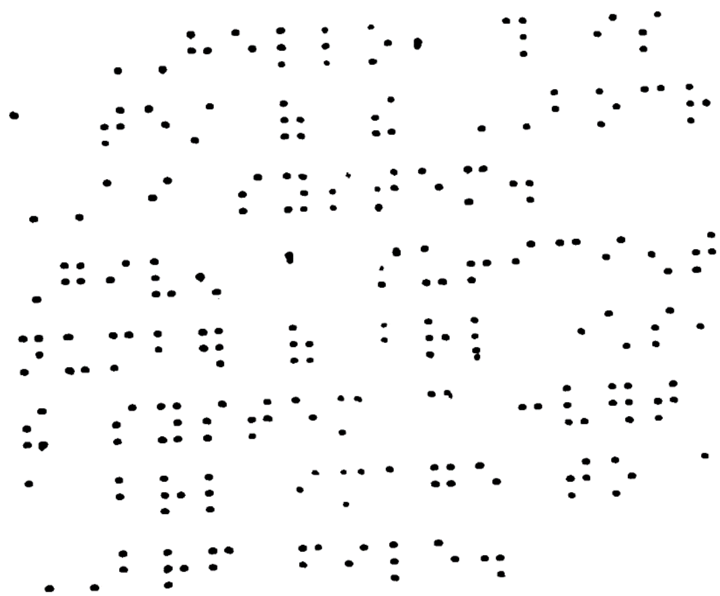

Figure 5: Binary Image from Otsu's Thresholding

Deskew

The final step in the preprocessing state is to correct for any possible rotation in the image. This is a very important step because the segmentation algorithm used assumes the braille characters are aligned orthogonally. The approach for deskewing taken was to maximize the variance in the row and column weights of the image. The assumption is that if the dots are aligned, more white pixels will be in the rows and columns that are whitespace and more black pixels will be in the rows and columns which are not. This is accomplished by searching an evenly spaced discrete set of possible rotation angles. For each angle, the rotation is applied to the image. The row and column weights are then calculated, and their variances are summed to get the combined variance. The rotation angle which produces the largest combined variance is kept and applied to the image. The results of this operation are shown in Figure 6. For the validation image shown, the correction was about 2.5 degrees, and the segmentation algorithm did not work before the correction but did after the correction.

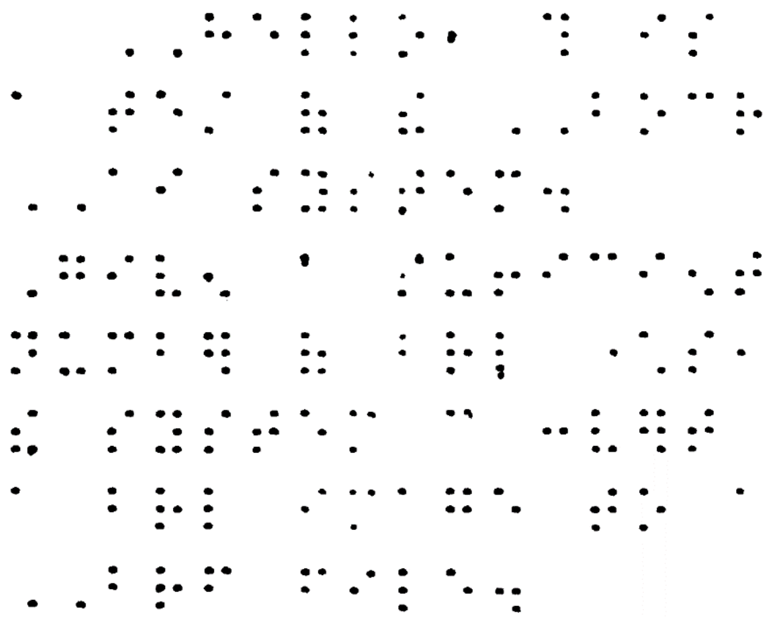

Figure 6: Final Product of the Preprocessing procedure

Segmentation

It was determined that recognizing individual braille characters would be the simplest solution to converting from a braille image to a BRF file. So, a method for extracting, or segmenting, individual characters from the image needed to be developed. Because braille is written completely on a grid, there is no chance of characters running together or being misplaced relative to the rest of the characters. The structured nature of braille was taken advantage of to perform the segmentation into equal-sized, equal-spaced rectangular segments.

Row and Column weights

The first step in the segmentation algorithm is to calculate the row and column weights. This is the average pixel intensity across an entire row or column of the image respectively. The row weights are shown in Figure 7 and the column weights are shown in Figure 8 for the validation image.

Whitespaces

Otsu's thresholding algorithm is applied to the line weights to mark each row and each column as whitespace or not. Figure 9 shows the whitespace representation of the grid the algorithm believes the braille characters to be on.

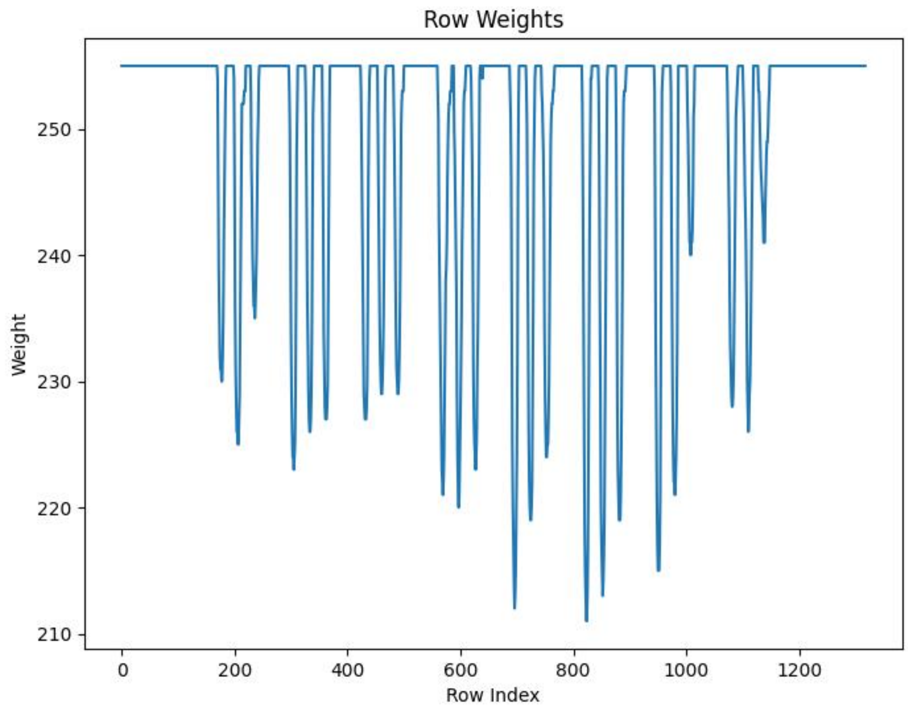

Figure 7: Average pixel intensity across each row

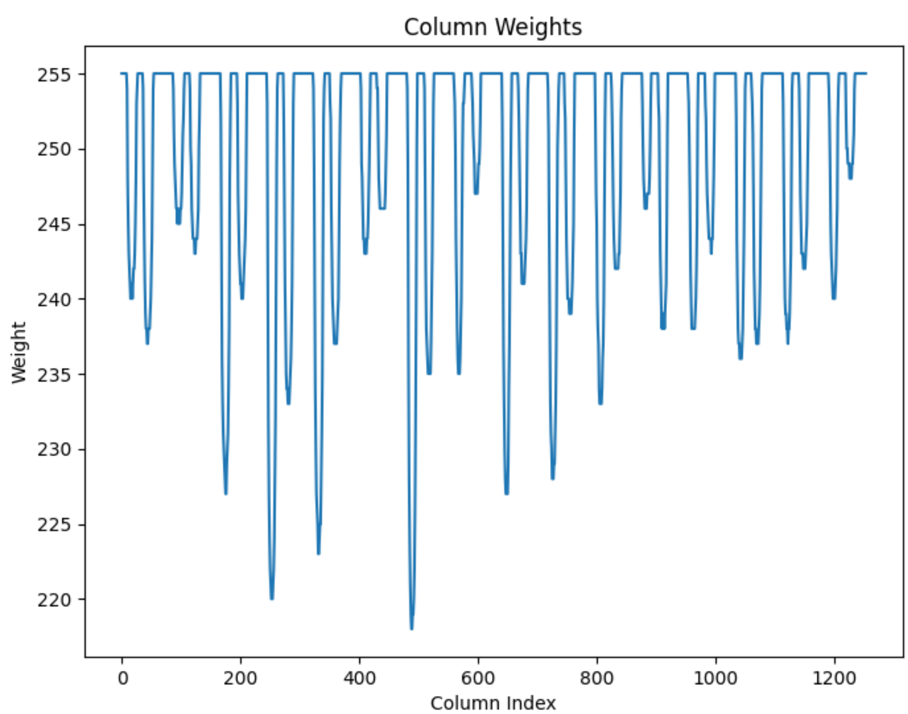

Figure 8: Average pixel intensity across each column

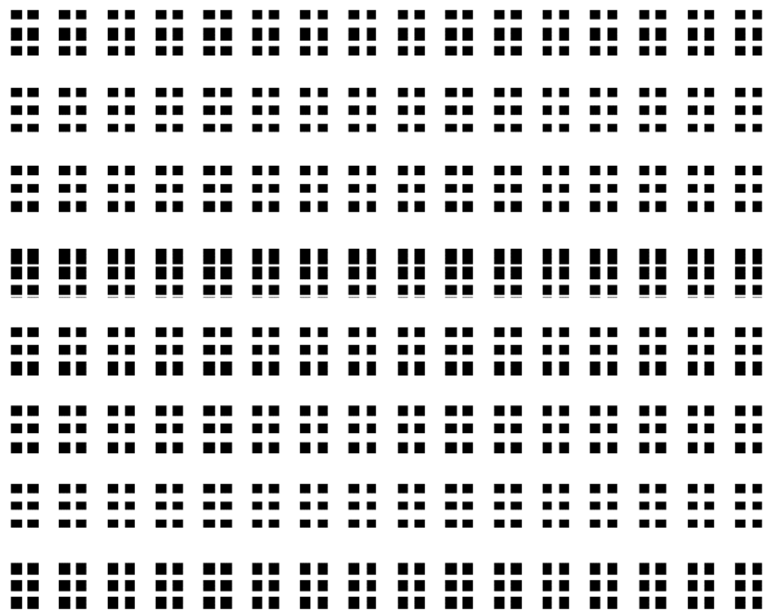

Figure 9: Whitespace representation of image

Delimiter Whitespaces

The whitespaces are collapsed into a representation where each set of whitespace lines without black space to separate them is considered part of the same piece of whitespace. Ignoring the margins, Otsu's thresholding algorithm is used to mark the larger whitespaces as delimiters. The midpoint of each delimiter space identified is shown in Figure 10. These midpoints are then fitted to a linear model, since it is known that the spacing should be even. The linear model is then used to generate new evenly spaced cut points and to extrapolate the location of the outer cuts. The evenly spaced cuts are shown in Figure 11.

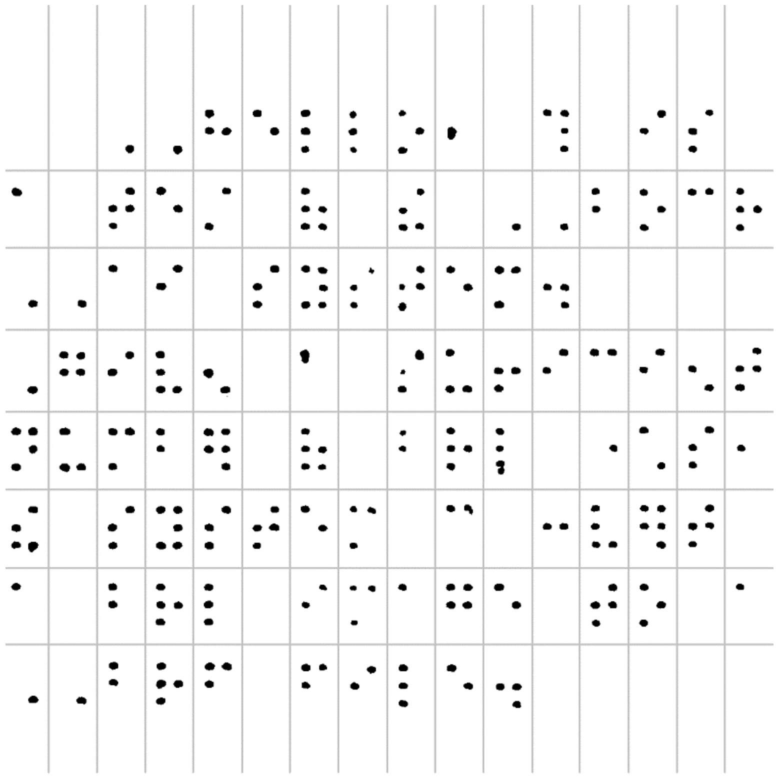

Figure 10: Identified delimiter whitespaces

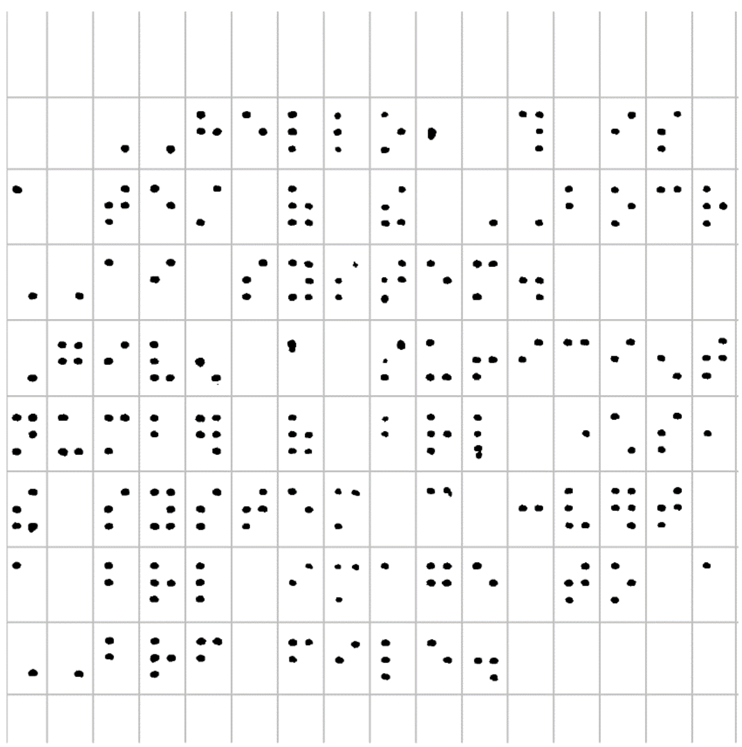

Figure 11: Segmentation Cuts

Adding Context

Braille characters require context from surrounding characters to be interpreted. A single dot with no positional context could in any of six locations, and could be interpreted as the letter a, a comma, or the capital sign. The segmentation into individual characters discards the positional information gained from surrounding characters. The solution used in this project was to add a row of pixels along the bottom and right edges of the segmented images which contain information from the rest of the image. These additional pixels have the line weight from the original image scaled so that the minimum weight is black and the maximum weight is white. Figure 12 shows and example character with no context and Figure 13 shows the same character with context.

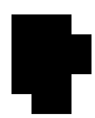

Figure 12: Example dot with no context

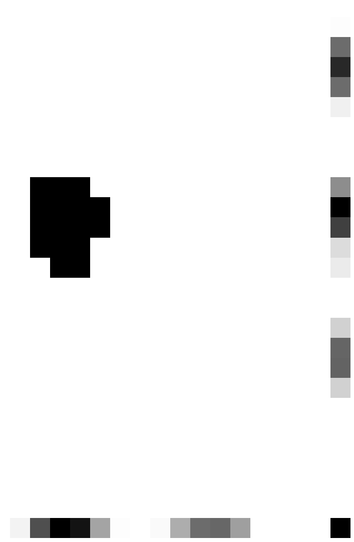

Figure 13: Same dot with context

Character Recognition

Character recognition of a segmented image is done by a neural network. The problem was formulated as a multi-label classification problem. Each character is represented by a 6D multi-hot vector, where each bit represents one of the possible 6 braille dots. The network consists of two convolutional layers and two dense layers. The network was trained on a data set created for this project. The data set was three scans of different papers, where each paper contained all 64 possible permutations of a braille cell. Each of the scans was mirrored horizontally, vertically, and both, effectively quadrupling the data set. In the training, random noise was added to the images and dropout layers were added to the model to encourage generalization. The output layer of the network had six neurons, and used the sigmoid activation function. This means each neuron represents one of the six possible dots. The multi-hot vector representation is converted to an integer to be passed along the OCR procedure.

Postprocessing

The postprocessing step has two components. First, It removes trailing whitespace from the predictions. Since the segmentation assumes an even grid of cells, any line of characters which did not reach the furthest column has trailing spaces. Second, The integer representations of the characters are used to look up in a table what the ascii representation of that character is. All the ascii characters are packaged into a single string, with line breaks added to preserve the original document formatting.

Chat Bot

The chat bot interface used in this project is a simple rule based chat bot. It was determined that putting more time an energy into the OCR algorithm than the chat bot would produce a better project given the nature of the problem chosen.

Evaluation

Validation Test

The image shown in Figure 2 was used for validation of the system. The chat bot conversation was as follows:

User: Hello
Bot: Hello! I am a chat bot who can assist you in converting hard-copy braille into electronic braille.
User: What can you do?
Bot: I am your interface to perform Braille Optical Character Recognition (BOCR) on an image file. Ask me to OCR a file for you.
User: OCR a file for me.
Bot: What is the file path to the photo you wish to BOCR.
User: ../data/ validation.png
Bot: I encountered an error attempting to open the file. Maybe the file does not exist or is not a supported file type.
User: OCR a file for me.
Bot: What is the file path to the photo you wish to BOCR.
User: ../test/validation.png

Bot: Thank you, I will begin the BOCR process.
Bot: The BOCR process has finished. You can find the BRF file at: ../test/validation.brf
User: quit
Bot: Goodbye!

The contents of the generated BRF file are as follows:

```
  ,,hello1 ? is
a te/ ( ! ,,bocr
,,ai system4
,giv5 b su6ici5t
numb] ( brl "*s1
! system c 3v]t
a brl image to a
,,brf file4
```

Translated to plain text this is:

```
  HELLO, this is
a test of the BOCR
AI system.
Given b sufficient
number of braille characters,
the system can convert
a braille image to a
BRF file.
```

There is one error on line 4 of the file. An “a” was misrecognized as a “b.” This is an accuracy of 127 out of 128, or 99.21%. The misrecognized character is shown in Figure 14. The dot appears vertically elongated, and since the “b” character in braille is the two vertical dots where the single dot of the image has been elongated, it makes sense that “b” was the prediction. However, there is clearly only one dot in the image and therefore the model should not have predicted a character with two dots.

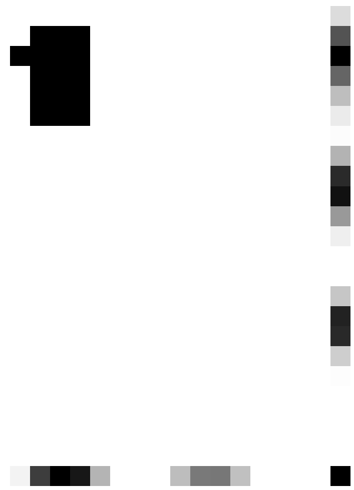

Figure 14: Misrecognized character

Evaluation Against ChatGPT

ChatGPT can hold a much better conversation than the chat bot of this project. When asked what braille is, ChatGPT responds:

> Braille is a system of raised dots that can be felt with the fingertips and is used by people who are blind or visually impaired to read and write. Named after its creator, Louis Braille, the system is a tactile method of communication that allows individuals with visual disabilities to access written information.
>
> The Braille system is based on a grid of six dots arranged in two columns of three dots each. Each character or letter is represented by a unique combination of these dots. Braille can be used for representing not only the alphabet but also numbers, punctuation marks, and even musical symbols. It provides a means for people with visual impairments to read and write books, documents, and other texts.
>
> Braille is widely used around the world and has been adapted for various languages. It plays a crucial role in promoting literacy and independence among individuals who are blind or visually impaired. Modern technology has also integrated Braille into electronic devices, making it more accessible in today's digital age.

ChatGPT could not explain what a BRF file is unless prompted that it is in the domain of electronic braille. Its description is as follows:

> The BRF file format is indeed associated with electronic Braille. BRF stands for Braille Ready Format. It is a file format designed to store electronic Braille documents. Electronic Braille is a

digital representation of Braille text that can be used with Braille displays and Braille embossers. BRF files contain Braille characters encoded in a specific format to ensure compatibility with assistive technology devices used by individuals with visual impairments.

These files typically include metadata, the text content in Braille, and information about formatting. BRF files are used to make digital texts accessible to people who read Braille, and they can be converted from various document formats into a Braille-ready format.

Software tools and converters are available to generate BRF files from standard text documents, allowing accessibility for individuals who rely on Braille for reading. It's worth noting that BRF is just one of several formats used for electronic Braille, and others, such as BRL and BRFPlus, are also utilized in different contexts.

The chat bot in this project cannot answer the questions that ChatGPT could. Since ChatGPT is a natural language model, it can provide general knowledge on a wide array of topics. However, without access to GPT4, ChatGPT cannot interact with images and therefore has no way to attempt the OCR of braille. I am on the waiting list for GPT4 so I could not evaluate GPT4's capabilities at recognizing braille characters.

References

[1] "Learn Braille with DOTS," DOTS, 2021. [Online]. Available: https://www.dotsrpg.org/learn-braille. [Accessed November 2023].

[2] "What is OCR (Optical Character Recognition)?," Amazon Web Services, [Online]. Available: https://aws.amazon.com/what-is/ocr/#:~:text=Optical%20Character%20Recognition%20(OCR)%20is,words%20in%20the%20image%20file.. [Accessed September 2023].

[3] R. G. Casey and E. Lecolinet , "A SURVEY OF METHODS AND STRATEGIES IN CHARACTER SEGMENTATION," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 18, no. 7, pp. 690-706, 1996.

[4] "Smoothing Images," Open CV, [Online]. Available: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html. [Accessed November 2023].

[5] A. Murzova, "Otsu's Thresholding with OpenCV," Learn OpenCV, 5 Auguts 2020. [Online]. Available: https://learnopencv.com/otsu-thresholding-with-opencv/. [Accessed November 2023].