Character Segmentation of Cursive Handwritten Words

Jordan McCaskill, *Undergrad Student, MacEwan University*

1 Introduction

THIS paper seeks to explore various methods of segmenting cursive handwritten words into individual characters. Segmentation of words is the evaluation of an image of a word and determination of where the word can be split into individual characters. Text segmentation of handwritten letters is a field which is seeing lots of experimentation and research. Over years of research, many different methods have been developed for handwritten characters; however, the number of methods that can be applied to cursive handwriting remains much smaller. This paper seeks to determine the usefulness of an additional method for cursive letters specifically. Experimentation with an existing pixel projection analysis method and a new top line tracing method are examined in order to present alternatives for cursive text segmentation with a comparison of results.

The main focus of the research is the implementation and evaluation of the quality of the projection analysis and top-trace methods. The projection analysis method is an existing method of segmentation which has been in use with cursive and non-cursive writing. It is used as a primary segmentation method to begin thinking about the problem in a more simple manner and ultimately as a baseline test for evaluating the quality of the segmentation generated by the secondary, top-trace, method. The Top-line Trace is a novel idea for segmenting cursive writing. The other methods which were investigated as options for implementation and used to develop the idea for the top-trace method are described in more detail below.

2 Background Research

Background research was primarily conducted to determine existing methods which were of interest as a topic of research. The projection analysis method was selected for implementation during this time. The idea for the top-trace method was also developed during this research period while reading about many different methodologies for segmenting handwritten characters. All the methods, as well as their final impact on the experiment and implementation, are described in detail below.

Many of the segmentation methods outlined are investigated as a result of the use of a survey of segmentation methods done by R. G. Casey and E. Lecolin [1].

Additional research was also done to investigate possible tools and languages which were useful in developing and testing the chosen methods. Several tools where examined, however, only those chosen will be identified as the other options investigated made insignificant contributions to the project, if any at all.

2.1 Methods

The following methods were investigated and contributed to the overall success of the research, either directly or indirectly:

2.1.1 Over-Segmentation

Over segmentation, as described in [4], is the overarching methodology of almost all other methods examined. For the subject of this paper, over-segmentation is mainly a guiding principle for the segmentation of the words into their individual characters. It is the idea that it is more helpful to generate more segmentation points than necessary, as opposed to lacking segmentation points that are needed. In other words, over segmentation focuses on eliminating false negatives (losing points which should exist) at the expense of generating more false positives (points that exist but are not needed).

Once an image has been put through a method that follows the premise of over segmentation, the results can be trimmed down through various other methods. One such example is eliminating any points which span a gap in a letter with a hole, such as 'a', 'o', 'g', etc. Other examples include eliminating a point which is too close to another or one that intersects a letter for too much vertical distance. Once one or several of these trimming methods have been used, the false positives in an over segmentation method can be reduced drastically in order to improve run time and efficiency.

2.1.2 Projection Analysis

Projection analysis is the use of pixel density in an image in order to determine certain information about the image without looking at the image itself in detail. Two main forms of projection analysis are presented in [1], along with several reasons why they are useful. These two differing methods are Vertical Projection and Horizontal Projection Analysis. These two methods are very similar in form but seek to generate very different information.

Both of the projection analysis methods work in the same way, but vary slightly in implementation. The projection method generally seeks to generate a histogram of pixel density which can be used to find out certain information. Pixel density is a sum of all of the pixels in a given row or column. A line is then drawn across the histogram at a specific threshold and the segmentation points are picked based upon whether the line overlaps the histogram's peaks or not. The vertical and Horizontal Projection Analysis techniques are explained in further detail in the implementation section.

2.1.3 Average Longest Path

The average longest path method is described in [3] and based upon the idea that the word can be viewed in parts, and that each part can be evaluated separately to decide the likelihood of a letter being present. The algorithm starts out by arbitrarily separating the word into many small parts, based on the likely distance that a letter might span, without any attention to the word itself. For example, the word might be segmented into parts which have a width of one quarter of the likely width of a letter. After this initial separation is complete, a support vector machine (SVM) is used to determine the likelihood that any specific section, or combination of sections, contains a letter. An SVM is a tool which is trained on images and then uses data from what it has learned to attempt to match parts of another image to the training data. After the SVM has run over all of the parts individually, a second algorithm looks at the likelihood of the parts being letters and tries to determine the most probable place for segmentation. This is done by determining the largest average likelihood across a certain area and using that result find the start and end point of each letter.

2.1.4 Ball-Tracing

Described in [5], the ball-tracing method can be understood by imagining a ball moving down a track. In this example the ball will have momentum and at each intersection, it will attempt to continue to travel in the direction that it was previously headed. The algorithm is based on the principle that tracing letters could be handled similarly. When a tracing algorithm gets to any point of intersection, it attempts to continue in the same direction before evaluating any other options. This method was shown to be very successful when evaluating handwritten characters with overlap; however, one major issue with the use of ball tracing for cursive letters is that the connections between letters cannot be ignored because it is part of the font itself. This method was extremely useful in designing the top-trace method as the principles which govern each method are similar in nature.

2.1.5 Landmarks

This method is described briefly in [1], and in more detail in [2] and [6]. It is simply looking for key characteristics that certain letters contain, such as a tail for ’d’ or ’q’, or a dot for ’i’ and making segmentation points based upon those details. It generates many possible points and then uses some trimming techniques to eliminate extras. This method was modified for use in designing the top-trace.

2.1.6 Top-line Trace

The Top-line Trace was developed by the author while trying to determine a way to overcome some of the shortcomings of the other methods. It is based heavily on the ball tracing and landmark methods. The overall idea is to view the word based on only the information contained in the top line of the image, not considering the word in its entirety. This is accomplished by generating the information of the top line of the image first, then using a series of functions to determine potential break points in the word. Examples of usage and implementation details will be provided in more detail in the Experiment section.

2.2 Tools

The following tools are used in the implementation of the methods and the verification of the characters:

2.2.1 Python

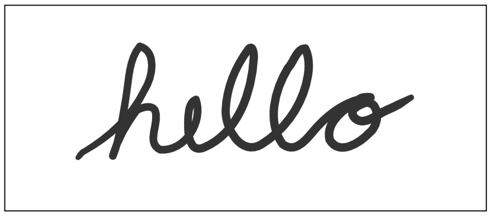
All algorithms and methods were implemented using the python programming language. Details on python can be found in [8]. The PIL library was used.

2.2.2 Tesseract OCR

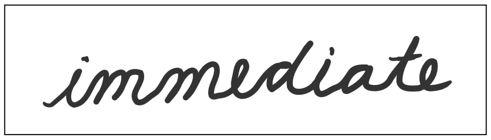
Tesseract is an Optical Character Recognition (OCR) tool. OCR is the process by which text of any nature can be converted into a machine-encoded text for use by a computer. The Tesseract OCR engine is an open-source project sponsored by google. Details on Tesseract can be found in [7]

2.2.3 PyTesseract

PyTesseract is a python wrapper for Tesseract OCR. This wrapper was used in an early implementation of the program but was abandoned for TesserOCR. PyTesseract can be found at https://github.com/madmaze/pytesseract.



(a)



(b)

Fig. 1. Images for (a) ”hello” and (b) ”immediate”.

2.2.4 TesserOCR

TesserOCR is a Python wrapper for Tesseract OCR. This wrapper was used in place of the PyTesseract after an initial implementation due to better customization and the ability to pull more useful information of of the results generated. TesserOCR can be found at https://github.com/sirfz/tesserocr.

3 Implementation

Three different methods were implemented over the course of this experiment. A Horizontal Projection Analysis was implemented to obtain additional meta-information about the word. A Vertical Projection Analysis was implemented as a baseline for comparison. Lastly, a Top-line Trace was implemented to evaluate the viability of the method in determining character segmentation points. All implementation descriptions will be based on fig.1a, with fig.1b used as an additional example where necessary.

3.1 Horizontal Projection Analysis

The Horizontal Projection Analysis is created to determine useful information about the image. This information includes the baselines of the image (the top and bottom line of the image with letter tails excluded) and the height of the main body of the letters. The information gained from this method is used in both the Vertical Projection Analysis and the Top-line Trace methods. The use of this information in the other methods will be explained in detail in their respective sections.

The Horizontal Projection Analysis in this implementation begins with a picture of a single word. Starting with the picture of ”hello” as shown in fig.1b, all of the pixels are slid to the left to generate the image shown in fig.2a. This image shows the horizontal pixel density for the entire image. In other words, each row contains all of the individual pixels from the original image placed next to each other, starting from the left hand border and moving to the right. This allows information to be pulled from the image without the need to look at the complex line data from the original image. By looking at the height of the entire word on the far left border and comparing it to other areas of the projection, certain sections of the images height can be ignored. Assuming that a cursive word has both tails up and down, in most cases any part of the projection which spans more than roughly 70% of the left border height can be ignored, as it is likely including a tail. Similarly, anything which is less than roughly 30% of the left border height can also be ignored, as it is likely all tail. This leaves a valid section of between roughly 70% and 30% of the image height which allows baselines to be drawn in as shown in fig.2b. Final implementation includes a step which accounts for words with no tails. This step ignores the 70% limit if the histogram value is higher than the limit for more than a certain proportion of the histogram, 30% in implementation.

Once these baselines are determined, the height of the base of the image can be determined by subtracting the bottom baseline height from the top baseline height. The base height of the image also provides an approximate width of one character given that a letter such as ’o’ is roughly the same width as its height. Larger letters such as ’m’ or smaller letters such as ’i’ need to be resolved as special cases.

3.2 Vertical Projection Analysis

The Vertical Projection Analysis used to determine potential segmentation points for splitting the word. The process is the same as the

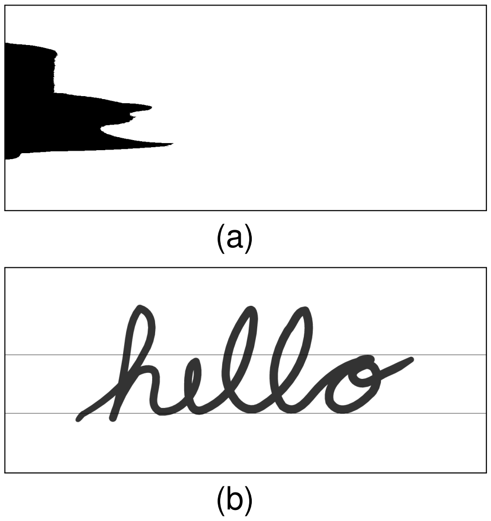


Fig. 2. Images of (a) the Horizontal Projection of ”hello” and (b) ”hello” with the top and bottom baselines drawn in.

Horizontal Projection however the gathering of the information is slightly different. Horizontal Projection gathers meta information about the image and seeks to help other methods work better, whereas Vertical Projection is only used for finding segmentation points for use in identification. These points are found using the vertical projection histogram and a threshold value, which will be discussed further.

The Vertical Projection Analysis starts with the image of the word and slides all of the pixels to the bottom of the image so the pixels in each column are stacked directly on top of one another. Starting with ”hello” from fig.1a this gives the image as shown in fig.3a. The image can then be analyzed by looking at the any line which runs horizontally across the projection. Looking for any changes from white to black or black to white allows identifies potential segmentation points. Any area of the image which has the transition of white-black-white is likely a letter and any area which has the transition of black-white-black is likely a potential segmentation point. The specific height at which the algorithm looks is defined as the threshold value.

The threshold value for the Vertical Projection Analysis is the horizontal line which runs across the image that the algorithm follows. An example of this is shown in fig.3b. The value is

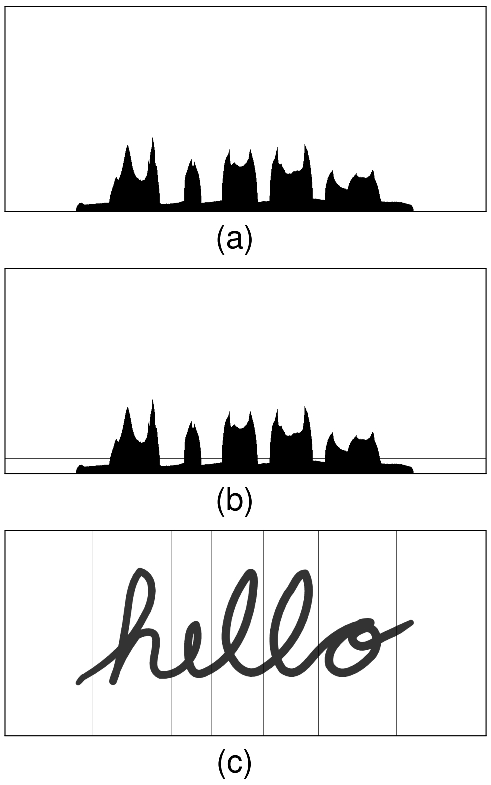


Fig. 3. Images of (a) the Vertical Projection of ”hello”, (b) Vertical Projection of ”hello” with a 20% threshold value line drawn and (c) the segmentation generated at a 20% threshold value.

a percentage value of the highest peak on the projection histogram. For example, an image which has a height of 450px and maximum projection height of 50px will have a 20% threshold at 10px based on the histogram height, not at 100px based on the image height. While traversing the threshold line the algorithm is looking specifically for black-white-black transitions in order to find segmentation points. When one of these transitions are encountered the algorithm will determine the x-value in the picture where the black-white and following white-black transitions occurred and average the x-values to find the segmentation point. White-black-white transitions are ignored as it is looking for only segmentation points and not letters. An example of segmentation is given for ”hello” using a 20% threshold in fig.3c.

Some experimentation was done to find a sufficient threshold value for the Vertical Projection Analysis. The original test value was set at 20% as shown in fig.3b and 3c. This provided

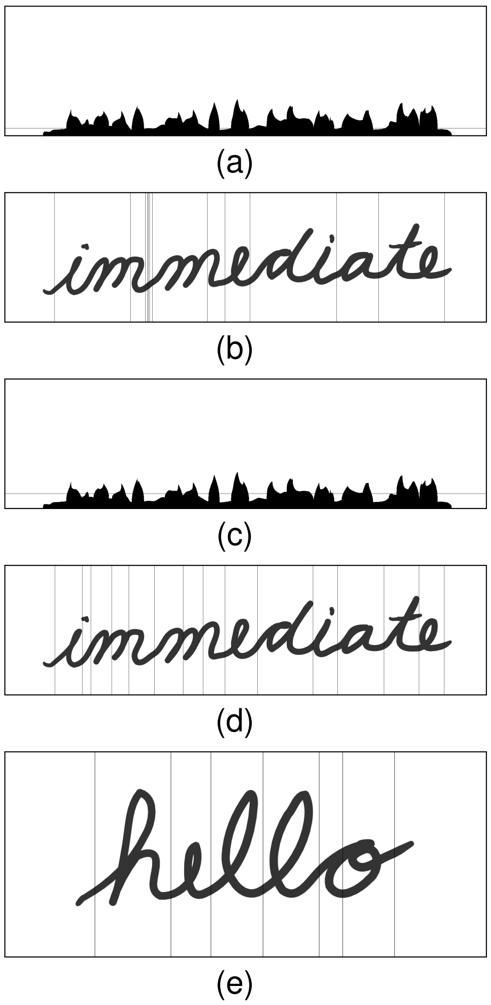


Fig. 4. Images of (a) the Vertical Projection of ”immediate” with a 20% threshold value line drawn, (b) the segmentation generated at a 20% threshold value, (c) Vertical Projection of ”immediate” with a 40% threshold value line drawn, (d) the segmentation generated at a 40% threshold value and (e) segmentation of ”hello” generated at 40% threshold.

excellent segmentation when used with simple words; however, the value proved entirely ineffective when used on more complicated words. When applied to a more complicated word such as ”immediate”, fig.1b, the segmentation generated with a 20% threshold was useless for identification purposes.

As seen is fig.4a, a 20% threshold value sits very low on the histogram and runs through primarily black areas on the image. This results in very few black-white-black transition, thus very few segmentation points are generated.

The 20% threshold worked well for ”hello” as the peaks are more distinct, but not for ”immediate” which has lower and less distinct peaks. Testing was done with a much higher threshold value, resulting in a similar problem for the opposite reason. In that case the lines spend too much time in the white area and give very random segmentation points due to the averaging of the x-values. In many cases a much higher threshold value yields poorer results than a lower value. Testing proved a mid-range value to be most effective in the majority of cases. An example of a 40% threshold is shown in fig.4c. At 40% the segmentation provided by the algorithm is quite accurate, as shown in fig.4d. Extra points are added in certain locations, such as in the ’m’s; however, this is an expected outcome as an ’m’ is not very different than several ’i’s in a row. Generally, the over-segmentation is minimal and all of the key points are present. The segmentation of simplistic words are accurate with the higher threshold value as well. Fig.4e shows ”hello” segmented at 40% with the only difference being an additional line through the ’o’, which is easily removed by a thinning algorithm. Final testing proved to be most effective with a threshold at 35%.

3.3 Top-line Trace

The Top-line Trace method is fundamentally different than the other two projection methods. The Top-line Trace, as the name suggests, seeks only to gather information about the top-line of the word. This is done by starting at one side of the image and working across the image to find only the very top pixels of the image. The algorithm starts from a single pixel rotates around it to find the next pixel. The starting position is one step more than the relative position of the previous pixel. When rotating clockwise around the current pixel, it looks at each of the surrounding eight pixels until it finds one that is black. The step required to get from a single pixel to the next is recorded and the position is moved to the next. For example, if the current pixel is (100, 100) and the next black pixel is (100, 101), then (0, 1) is recorded. An example of a top-line is given in fig.5. The algorithm is as follows:

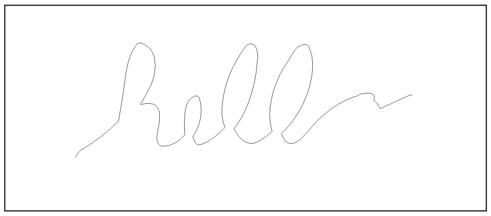


Fig. 5. The top-line for the image ”hello”

```
previous = first
steps is empty list of steps

while current pixel
    is not end pixel

    next = find_next_pixel
        (current,previous)

    next minus current add to steps
    previous = current
    current = next

return steps

function find_next_pixel
    (current, previous)

    for each pixel surrounding
        current starting at
        previous + 1

        if pixel is black
            return pixel
```

The top line trace is based on the idea of using the steps to determine where the possible segmentation points might occur. The initial implementation used only the x-value steps to determine possible segmentation, but this implementation gives no value whatsoever. The final implementation, uses a combination of the x-value steps and the absolute y-value of the pixel in order to generate information. In order to make this work, three values are used: high, low and threshold. The high and low values are used in conjunction with the top and bottom baselines and the threshold is based on the height. All of these values are generated by the Horizontal Projection Analysis. The high and low values are multipliers which are given to the top and bottom baselines respectively. The y-value of the pixel determines the value of the multiplier. A scale is generated between the top and bottom baselines which begins at the low value on the bottom baseline and scales linearly to the high value on the top baseline. Any value which is below the bottom baseline receives the value of the bottom baseline and any value above the top baseline receives the value of the top baseline. These values can be anything, with better results being obtained with the high value being lower than the low value. The threshold value is a percentage of the height of the letters. This threshold represents the distance in the x-direction that needs to be traveled in order for a likely width to be reached.

When traversing the top-line, the multiplier for each pixel is determined by finding its y-value location on the scale made by the high and low values. Once the multiplier is determined, the value of the horizontal movement is determined. The values for horizontal movements are -1 for backwards one step, 0 for no movement and 1 for forward movement. The horizontal movement value is then multiplied by the multiplier to get the final value of the step. The final value of the step is then added to the total value up to that point. Once the total value passes the threshold value, a segmentation point is registered and the total value is reset to 0. The process then continues until the last step is reached and a final segmentation point is registered. This implementation allows for ignoring purely vertical movements because the x-value is 0 and a value of 0 is added to the total after multiplication. It also accounts for negative direction movement and allows it to ignore abrupt swings in a negative direction.

Some experimentation was done with the high, low and threshold values to determine which are best for segmentation. Although the preferred values are somewhat dependent on each individual word, some overall trends were noticed. Simple words, such as "hello", are impacted less by smaller changes to any of the values than the more complicated words, like "immediate". A threshold value of between 80% and 85% worked best in the majority of cases. Having the high value be between 0.4 and 0.6 worked the best in most cases. The low value was much more flexible than the other two values with experimentation between 2.5 and 5.0 resulting in different segmentation points which were applicable depending on the word. Initial testing with a low value of around 2.5 resulted in less over segmentation; however, the words were likely to have their segmentation points shifted to the left or the right based upon previous letters. One fix to this problem was to make the low value closer to 5.0. This change allowed the segmentation points to reset quickly during the low portions of the word where segmentation points are likely to occur; however, over-segmentation increased drastically with an increased low value and ended up not being worth the extra overhead in the majority of cases.

TABLE 1
Reason for picking each testing word

Test Word	**Reason for choosing**
hello	Easy word for baseline
immediate	Longer, more difficult word for baseline
peak	Word with upper and lower tails
people	Word with multiple lower tails
same	Word with no tails
school	Word with tails and a mid-line letter-letter transition
sore	Word with no tails and a mid-line letter-letter transition

4 Results

4.1 Number of Segmentation Points

Testing is done using various words which are used to test multiple different letter layouts. Seven words are chosen for testing purposes and the reason for choosing each word is found in tab.1. Algorithm parameters are chosen based upon several factors. While a reduction of the over-segmentation is seen as preferable, segmentation points which are missed entirely are considered a complete failure. Missed segmentation points are not completely avoidable in some words when maximizing results across all test words. Overall quality of the segmentation points is evaluated by looking at images of the words with segmentation points

TABLE 2
Number of segmentation points

Test Word	Ideal	Vert. Proj.	Top-Trace
hello	6	7	9
immediate	9	16	13
peak	5	8	10
people	7	9	13
same	5	8	11
school	7	11	11
sore	5	6	11

and targeting the best overall quality across all words.

After testing is completed for all words and the results are compiled, the following optimal parameters are obtained:

1) Vertical Projection threshold: 35% of histogram height
2) Top-line Trace threshold: 80% of baseline height
3) Top-line Trace high value: 0.4
4) Top-line Trace low value: 2.0

The optimal number of segmentation points with no over-segmentation is equal to the number of letters in a word plus one. As an algorithm segments more, the recursive steps to determine letters becomes much more lengthy. For this reason it is important to attempt to reduce over-segmentation without missing any actual points. Given the above parameters, the algorithms give a certain amount of segmentation points more than the expected values. All of the results for each word in each algorithm as well as the ideal values are given in tab.2.

Perfect segmentation, while ideal, is not possible in all situations. A certain amount of over-segmentation is expected in order to minimize missed segmentation points to a near zero value. The amount of over segmentation for the Vertical Projection Analysis and Top-line Trace are 45% and 82% respectively. This means that for the average word, Vertical Projection will have less over-segmentation than the proposed Top-line Trace as it will be roughly 40% less segmentation above the ideal value. However, the Top-line Trace results in a less than 100% increase, on average, which is a respectable value in terms of results.

4.2 Quality of segmentation

Appendix A contains images of all the test words with their segmentation points drawn in and will be referenced heavily in this section. As noted there, the Vertical Projection segmentation is on the top and the Top-line Trace is on the bottom. All references are to sub-figures of Appendix A.

4.2.1 Vertical Projection Analysis

Vertical Projection Analysis works fairly accurately with one major exception. Simple words such as ″hello″ (a) or ″peak″ (c) match close to the ideal, but the addition of more complicated letters such as ′m′ produces many more segmentation points. This is apparent in both ″immediate″ (b) and ″same″(e). The addition of letters with tails appears to cause some issues for the projection analysis. In ″peak″ (c) the failing is masked due to the letter being at the beginning of the word. In ″people″ a segmentation point was missed entirely between the ′o′ and second ′p′ and was not correctable without affecting multiple other words. This failing is likely due to the slanted tail of the ′p′, not the different transition of the ′o′ as ″school″ (f) has two ′o′ and the transitions are both handled correctly, as well as in ″sore″ (g). Another minor failing is shown in ″school″ (f) as the segmentation of the ′c′ and ′h′ is skewed to the middle of the ′c′. Overall, the segmentation points of the Vertical Projection Analysis are very accurate when they are present. They are generally found in the center of the letter gaps and provide very good segmentation, however; the algorithm appears to fail when the letters are less perfect or more slanted, providing inaccurate segmentation or no segmentation at all.

4.2.2 Top-line Trace

Top-line Trace appears to work quite well across all testing cases. In testing, Top-line Trace does not appear to favour any specific type of letter over another, resulting in segmentation being fairly consistent across all test cases. The algorithm does not outright miss any segmentation points, however, it appears to exhibit a skew in one direction or another in some circumstances. For example, ′hello′ (a) shows a right skew on many of the letters. This skew is not to the point of ruining the segmentation but might limit it's effectiveness depending on application. Another example is the skew of the 'o' and 'p' segmentation point in "people" (d) which could be considered a failure in certain circumstances even though the segmentation point is not completely missed.

4.2.3 Comparison

In comparison there appears to be no clear winner. The Top-line Trace has better segmentation in that it loses less segmentation points, while, the Vertical Projection is more accurate when it determines a segmentation point. Top-line shows a larger margin of error when placing its segmentation points, and may not be expressly useful if exact segmentation is needed. On the other hand, complicated or dirty words will limit the effectiveness of the Vertical Projection method much more than the Top-line Trace, potentially making the Vertical Projection method next to useless in some circumstances.

5 Conclusion

The purpose of this paper is to evaluate a novel idea for segmentation of cursive handwritten letters, the Top-line Trace. It is shown that the Top-line Trace is an effective way to segment cursive words. While it demonstrates slight inaccuracy in some situations, it maintains quite high accuracy in most situations and is shown to be more adaptable than the Vertical Projection Analysis. The Top-line Trace shows the ability to ignore slants in letters to a far greater degree than the Vertical Projection Analysis. It is shown that the Top-line Trace may prove to be a very valuable solution for the segmentation of cursive handwritten words in the future.

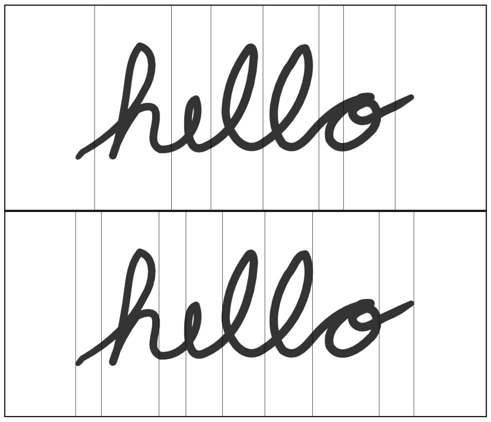
6 Future Work

Future work would include:

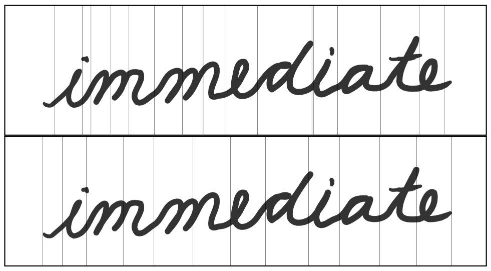
1) Modifying the algorithm to work better on angled words
2) Using the Top-line Trace as a gap removal technique as well as a segmentation technique
3) Use a machine learning algorithm with training data to determine the best parameters for the algorithm
4) Train Tesseract OCR better and integrate it into the project more completely
5) Modify the Top-line Trace to work with unexpected gaps in individual words

Appendix A
All segmented test words

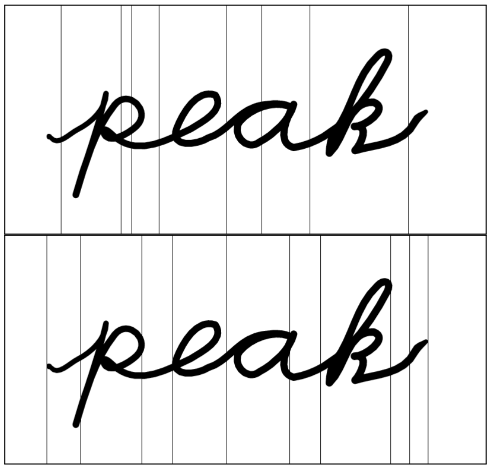
*Each grouping shows the Vertical Projection on top and the Top-line Trace segmentations underneath.

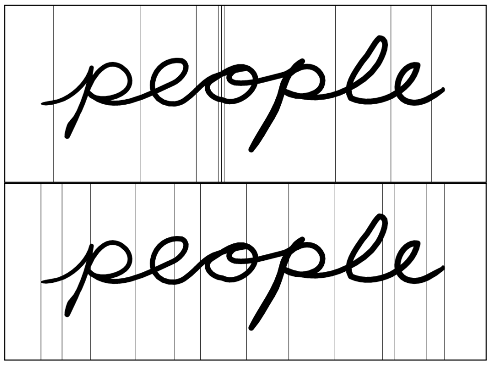


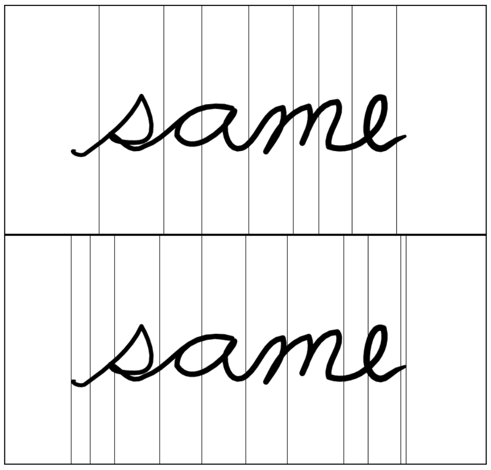
(a)

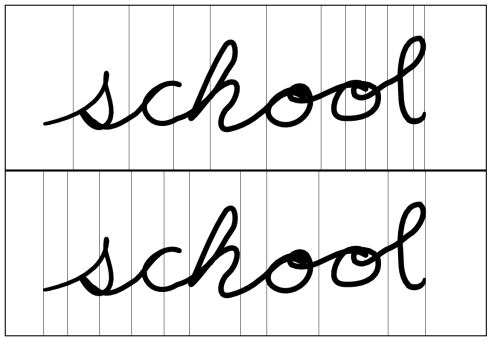


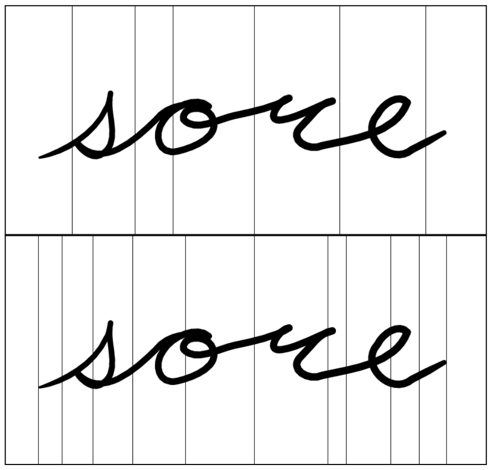
(b)


(c)


(d)


(e)


(f)


(g)

Appendix B Code Examples

As of writing, all code is held at https://github.com/mccaskillj/Capstone. The open repository will be updated with any future developments however will likely not open up to additional contributors unless high interest is shown in the project.

As outlined in the tools section, Tesseract can be found at https://github.com/tesseract-ocr and TesserOCR can be found at https://github.com/sirfz/tesserocr.

In order to run the code as is, the following is needed:

1) Python 3.6 or higher
2) PIL library for python
3) Libleptonica 1.74.2 or higher
4) Tesseract OCR 4.0 or higher
5) TesserOCR

If you wish to run it without tesseract then comment out or remove all code associated with segmenter.py. If you wish to run it with tesseract then the steps are below and the training data for a basic cursive handwriting font is in the git repository. The steps were executed on a linux Lubuntu machine and the author was not able to get a windows machine to function within a reasonable time so those steps are not found in this document. The steps below were found in either https://www.linux.com/blog/using-tesseract-ubuntu or in the official documentation or forums on github for the respective package.

In order to install libleptonica run:

```
sudo apt-get install autoconf
    automake libtool

sudo apt-get install libpng12-dev
    libjpeg62-dev libtiff4-dev
    zlib1g-dev

wget http://www.leptonica.com/
    source/leptonica-1.74.4.tar.gz

tar xvf leptonica-1.74.tar.gz

cd leptonica-1.74

./configure

make

sudo make install
```

In order to install Tesseract run:

```
git clone https://github.com/
    tesseract-ocr/tesseract.git

cd tesseract

./autogen.sh

./configure --enable-debug

LDFLAGS="-L/usr/local/lib"
    CFLAGS="-I/usr/local/include"
    make

sudo make install

sudo ldconfig
```

In order to install tesserocr for python:

```
git clone https://github.com/sirfz/
    tesserocr

cd tesserocr

pip install .
```

References

[1] R. G. Casey and E. Lecolinet, *A Survey of Methods and Strategies in Character Segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, v.18 n.7, p.690-706, July 1996.

[2] Ke. Han and I. K. Sethi, *Off-Line Cursive Handwriting Segmentation*, Proceedings of 3rd International Conference on Document Analysis and Reconition, v.2, p.894-897, Aug 1995.

[3] D. Salvi, J. Zhou, J. Waggoner and S. Wang, *Handwritten Text Segmentation Using Average Longest Path Algorithm*, 2013 IEEE Workshop on Applications of Computer Vision, p.505-512, Jan 2013.

[4] H. Lee and B. Verma, *Over-Segmentation and Validation Strategy for Off-Line Cursive Handwrting Recognition*, 2008 International Conference on Intelligent Sensors, Sensor Networks and Information Processing, p.91-96, Dec 2008.

[5] C. A. B. Mello, E. Roe and E. B. Lacerda, *Segmentation of Overlapping Cursive Handwritten Digits*, Proceedings of the Eigth ACM Symposium on Document Engineering, p.271-274, Sept 2008.

[6] K. M. Sayre, *Machine Recognition of Handwritten Words: A Project Report*, Pattern Recognition, v.5, p.213-228, Sept 1973.

[7] R. Smith, *An overview of the Tesseract OCR engine*, Ninth International Conference on Document Analysis and Recognition, pp. 629-633, 2007.

[8] G. van Rossum, *Python tutorial, Technical Report CS-R9526*, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.