Mathematics Clinic

Final Report for
EDR

Detecting Environmental Hazards in Historical Maps

October 4, 2023

Team Members

Nathaniel Diamant
Mackenzie Kong-Sivert
Jacky Lee (Spring Project Manager)
Vivaswat Ojha
Kinjal Shah (Fall Project Manager)

Advisor

Nicholas Pippenger

Liaisons

Zachary Fisk
Paul Schiffer
Richard White

Abstract

Our team was given a large database of digitally scanned images of hand drawn maps. Our aim was to come up with a solution to detect and highlight possible sources of environmental contamination from these maps to aid in the tedious human process of searching by hand. We accomplished this using image processing to isolate text, OCR to interpret characters, and word finding to group characters into words. Our solution provides clients with a list of words extracted from the maps that could be used as indicators of environmental hazards.

Contents

List of Figures

List of Tables

Acknowledgments

We would like to thank our liaisons Zachary Fisk, Paul Schiffer and Richard White for their valuable guidance throughout the process and our advisor, Professor Nicholas Pippenger, for supporting us during this project. We are also grateful to Clinic Director Professor Weiqing Gu and Clinic Coordinator DruAnn Thomas. We would also like to thank the people who helped label training data for our OCR model.

Chapter 1

Executive Summary

1.1 Our Problem

The main problem for this project is to find a way to highlight and identify environmental features indicated on scanned maps from the Sanborn map data set of approximately 1.2 million historical maps. Each of these maps depicts a part of a city in the United States in a given year in the late 19th or early 20th century. While our sponsor EDR has access to this enormous data set of maps, the relevant information stored in these maps is not easily accessible. While relevant environmental features are often marked by the text on the maps, it is very difficult to perform a manual search for a given feature over all the relevant maps. For instance, if one wants to determine whether an oil well ever existed in a certain area of a city using the map data, it would require having a person look for related words in potentially hundreds of maps in order to cover the entire geographic region and all the years of map data. This can be a time-consuming and resource-intensive task.

1.2 Our Solution

Our solution to the problem of highlighting and identifying environmental features takes an individual map as an input and outputs a JSON file marking the locations of potentially relevant features and information about them. This process is composed of six different stages:

1. Line Removal

2. Pixel Island Detection
3. Junk Classification
4. Optical Character Recognition
5. Symbol Grouping
6. Word Recommendation

The first step in the process is to remove the long lines from the original map. The next step is to find groups of contiguous black pixels that we term pixel islands. At this point, we have a set of pixel islands, with only some of them representing relevant alphanumeric symbols. We use a classifier that eliminates most of the junk pixel islands to get the resulting set of alphanumeric symbols. We then use a convolutional neural network to classify each of these symbols into classes that correspond to the underlying letters or numbers they represent. At this point, we employ a KD-tree to help us group these symbols together, forming words, acronyms or abbreviations as necessary. Finally, we take these output symbol groups and use a custom word finding system to find the most closely corresponding words that are common in the maps. This gives a final output of a dictionary that associates the location of the symbol group to the most likely words that might exist at that location.

Chapter 2

Introduction

2.1 Our Sponsor

When companies and institutions buy land, they are required by law to do their "due diligence" and look into whether the land is likely to have been contaminated by hazardous chemicals. If they don't and contaminants are found on the land, the company or institution is required to pay to clean it up. Many companies want to avoid having to pay this fine, but some other companies make a business out of buying contaminated land, cleaning it up, and selling it for a higher price. Regardless of their intentions, researching the land use history of a site is an important part of the process of buying and selling land. Our sponsor, EDR, seeks to assist in this process by providing easily interpreted and user-friendly resources like land-use maps indicating possible sources of contamination.

One particularly useful resource that EDR relies on for this process is its collection of Sanborn maps. These are a style of map that was used from the early 19th century until 2007 to mark how the land was being used. Some aspects of the map—such as street names—have a very consistent style, while other annotations—such as markings for manholes—do not. Most of the information falls between these two extremes. For example, some of the more relevant labels distinguish residential buildings from buildings with other uses, so they are likely not to be contaminated. These are written in a trained style neater and more consistent than most handwriting, but still not as clear as typed words. The lines of markings also sometimes intersect each other or intersect the lines separating buildings, which complicates matters. Customers can look through these maps in order to find businesses that

were likely to contaminate the ground, such as dry cleaners, gas stations, and coal storage.

2.2 Previous Work

Though the maps are a useful resource for many reasons, anyone who wants to interpret these maps will have to read through them themselves because they are not constructed in a way that is conducive to electronic searching. Since they started being drawn in the 19th century, it should be no surprise that these maps exist primarily on paper. In the 20th century, many of them were converted to microfilm, and these microfilm copies have since been scanned into computerized images, but they were still somewhat cumbersome to search.

The clinic team that worked for EDR last year was tasked with automating this process to some extent. Specifically, they made it possible to search the scanned maps for specific street names and house numbers. The team used image processing methods to isolate clumps of text and then used optical character recognition (OCR) methods to recognize individual characters to be clustered with neighboring characters into words. Given that we are working with the same data, some of these image processing techniques that the previous team had already implemented have been of great use to us.

Furthermore, previous year's work generated a codebase containing a variety of tools that we have been able to use. Firstly, we been able to reuse and repurpose some of the image processing code in order to help us isolate text relevant to our project. Next, we have been able to make some use of the existing OCR model as well. While our task is different from last year's, their model has proven useful as a first-pass attempt for recognizing objects like digits. Furthermore, our final OCR model could potentially benefit from using transfer learning based on the old model. Additional tools developed last year, such as a GUI for hand-labelling of data for training purposes, have also been useful.

Chapter 3

Problem Statement

Our project focuses on analyzing EDR's vast collection of Sanborn maps. Although the boundaries of the maps have been georeferenced, the individual components of the maps, such as symbols denoting different hazards and words representing other relevant environmental phenomena, have not been associated with a global coordinate system. The identification of these phenomena is important because EDR would benefit from being able to identify environmental phenomena that have been in the nearby area in the past. As the maps stand today, there are several challenges a user might face. A user might find it difficult to discern what the markings on the maps mean. A large proportion of markings are handwritten and clumped together at unpredictable angles, and it can be difficult to discern the markings. Moreover, a lot of markings are symbols that correspond to something. For instance, 'D' on the maps stands for 'dwelling' - a residential tax parcel. It would be cumbersome to manually read through a legend to understand what the markings mean, especially since there are so many markings on the maps. Most importantly, it is impossible for someone using these maps to quickly search for the existence or lack thereof of a particular phenomenon in an area as one must manually look through decades worth of maps.

Our project attempts to make all this easier by developing a system which will identify key information from maps along with its real-world coordinates, returning an output file containing the actual words on a given map and making the maps much easier to search through.

Our primary goal is therefore to identify key information from the Sanborn maps and associate it with the location so that clients of EDR can easily find important context when evaluating an area using the maps.

Chapter 4

Challenges

We encountered a number of challenges in the course of this project. A number of these emerged as we better understood the scope of our task and had to be dealt with in order to accomplish our final goals. Other challenges related to the specific techniques we chose to employ.

Many of the important steps in the pipeline were developed with the intent of overcoming the challenges presented by the problem.

Our initial approach of trying to locate all the relevant symbols by putting bounding boxes around each pixel island was immediately met with challenges. One of these was the presence of many long lines in the maps and the fact that they made it much harder to isolate the symbols we wanted to find. We resolved this challenge by using code from last year's EDR clinic project to remove such lines. The other main challenge for this part was the fact that, even after this, the pixel islands we isolated were often random clumps of pixels from irrelevant details in the map as opposed to relevant symbols. This problem was resolved by introducing the junk classifier, which eliminates many of these noisy symbols.

The next step of the project was to try and interpret the alphanumerical symbols, which also involved overcoming several challenges. We wanted to use a neural network-based optical character recognition model to identify each of the symbols based on the model built for last year's project. However, due to various deprecation-related issues and differences between the tasks, it was too difficult to adapt from the previous model.

Our next strategy was to adapt from an existing model that had been trained on the EMNIST data set, which is composed of handwritten letters and digits. In this case, the challenge was that some letters and digits in the

maps look very different from the EMNIST examples. We accounted for this using transfer learning, which allowed us to benefit from the EMNIST model and then adapt to our data. At this point, we were confronted by the lack of labeled data of map digits to actually train a model. We addressed this by developing a procedure to label data and asking a number of students to help out with manually assigning labels to pixel islands to prove training data. Our next major challenges were going from identified symbols to meaningful words or abbreviations. We accomplished this by developing symbol grouping and word recommendation algorithms.

Chapter 5

Methods

Our process for detecting features on Sanborn maps can be divided into three main portions: image processing, optical character recognition (OCR), and word finding. The image processing portion is used for isolating pixel islands, OCR is used for text recognition, and word finding is used to group symbols and suggest what words they represent. For each of the three main portions, we explored various methods. We will describe each of these methods and processes in detail.

5.1 Image Processing

This portion of our pipeline takes in an image and returns a list of extracted pixel islands. Pixel islands are defined as contiguous regions of pixels of the same color, which may take the form of letters, symbols, or irrelevant stray marks. Whether these shapes are relevant to our process is a matter to be addressed further down the pipeline; here, we are only concerned with extracting shapes which might be of interest to EDR's clients.

5.1.1 Image Binarization

The Sanborn maps are given to us in a raster format, which is a matrix of grayscale pixel values ranging from 0 to 255. We convert the maps to binary images, assigning each pixel a value of either 0 or 1. This makes processing the images easier for various reasons. For example, as can be seen in Fig. 5.2, the binarized image demonstrates a higher contrast between lines and symbols and the background in cases where the background is shaded. In general, the binarized image is cleaner and clearer, not only to the human

observer, but to the computer reading it, as there is less ambiguity between the foreground and the background of this image.

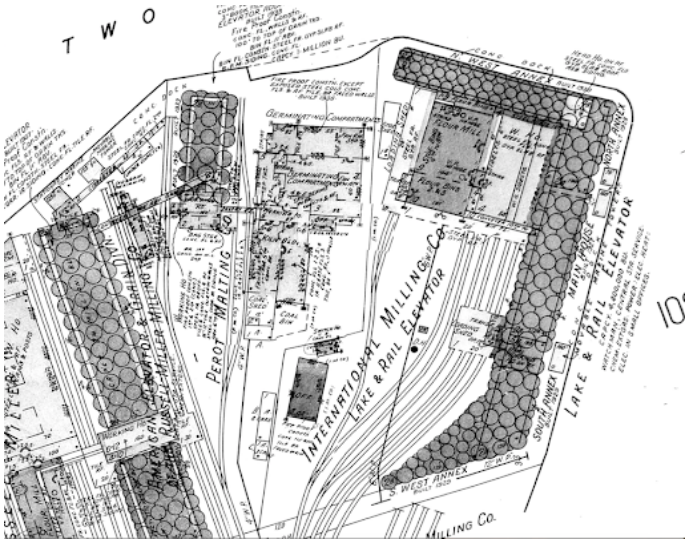


Figure 5.1 A map fragment before binarization.

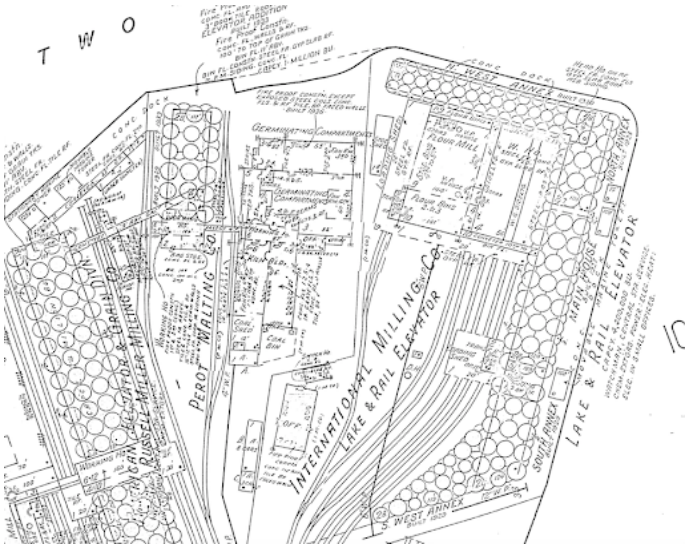


Figure 5.2 A map fragment after binarization.

We perform this binarization using adaptive Gaussian thresholding. This

means that for every grayscale pixel value v, the corresponding pixel in the binarized image will be white if $v > T$ for some threshold T and black otherwise. The value of T is calculated as a weighted sum of all the pixels within some defined neighborhood of the pixel in question. The astute reader will note that this inverts the image, so that the resulting image in our case would be white-on-black rather than the black-on-white shown in Fig. 5.2. Thus, the black-on-white representation in this image is for display purposes only.

5.1.2 Line Removal

The reader may observe from map samples such as the one in Fig. 5.2 that even when the lines and markings are made more distinct, the maps are cluttered with information that is irrelevant to us, even though it may have been of interest to the Sanborn surveyors. One of the main sources of such clutter is boundary lines for features like streets and buildings. These lines can cause problems for our pixel island detection because if a word is written such that it touches the line, our algorithm will recognized the line along with the word as a single pixel island. This is obviously bad since this means we will be unable to detect the word. These lines (that remain straight for a long period) can be distinguished from other markings using several different image processing methods which we will describe below.

Seam Carving

Seam carving is an algorithm most often used to detect parts of images that are largely the same, e.g., unimportant pieces of background, so that a computer program can remove them without removing any important content from the image. This algorithm starts with calculating an energy value for each pixel, which is a representation of how different it is from the pixels around it. At many points along the top of the image, this algorithm traces a line from the top of the image to the bottom, but not necessarily as a straight line. At every pixel on its way down, the path it carves can veer left or right or continue straight down. The path accumulates the energy of the pixel it chooses to reach next. The direction it chooses at every step is the one that will give it the least energy.

As a result, the seam carving algorithm finds a path of similarly-colored (or similarly-valued, for monochrome images) pixels. Since these are not very different from their surroundings (as they were chosen for their low

energy), they can be removed without much change to the content of the image in question.

For our purposes, this algorithm can also be used to detect long lines because if we start on a line, the path will want to stay on that line because moving into the negative space will add too much energy. If the path starts on part of the white background and comes close to a black line, it will move out of the way to avoid this line, and when the line veers off in the same direction, the path will continue veering as well, so the path will essentially trace the outline of the black line. Having found these lines, we can then proceed to remove them.

Hough Transform

One very common method for finding straight lines in image processing is the Hough Transform. The following shows the results of using the Hough transform on a sample map.

Figure 5.3 A map before applying the Hough transform.

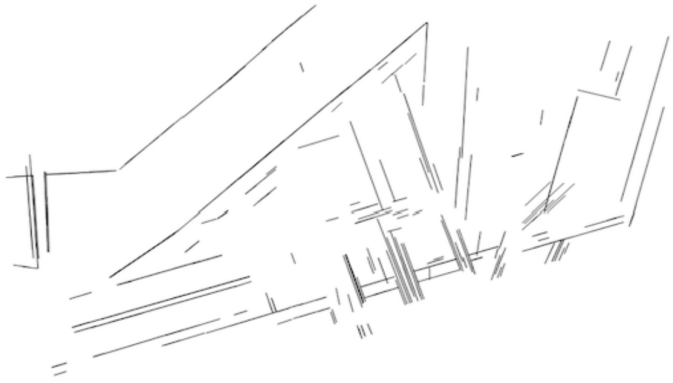


Figure 5.4 A map after applying the Hough transform.

Thinning

Another approach that we tried is called thinning. As the name suggests, this involves making long, thin lines even thinner, removing imperfections until the lines are mostly straight. This makes the straight lines easy to detect and remove by applying a threshold. The approach used for this was the Zhang-Suen algorithm, which operates in two stages. In the first stage, a pixel is marked for removal if it meets the following conditions:

- The pixel is black and has eight neighbors (i.e., it is not on a corner or an edge).
- The pixel has between two and six (inclusive) black neighbors.
- If you travel in a circle around the pixel through all of its neighbors, there will be only one transition from white to black.
- At least one of the north, east, and south neighbors is white.
- At least one of the east, south, and west neighbors is white.

The algorithm then makes another pass through the image, marking pixels for removal according to a very similar set of criteria, outlined below:

- The pixel is black and has eight neighbors (i.e., it is not on a corner or an edge).

- The pixel has between two and six (inclusive) black neighbors.
- If you travel in a circle around the pixel through all of its neighbors, there will be only one transition from white to black.
- At least one of the north, east, and west neighbors is white.
- At least one of the north, south, and west neighbors is white.

Note the difference in the last two criteria. The results of this algorithm are demonstrated in Fig. 5.5.

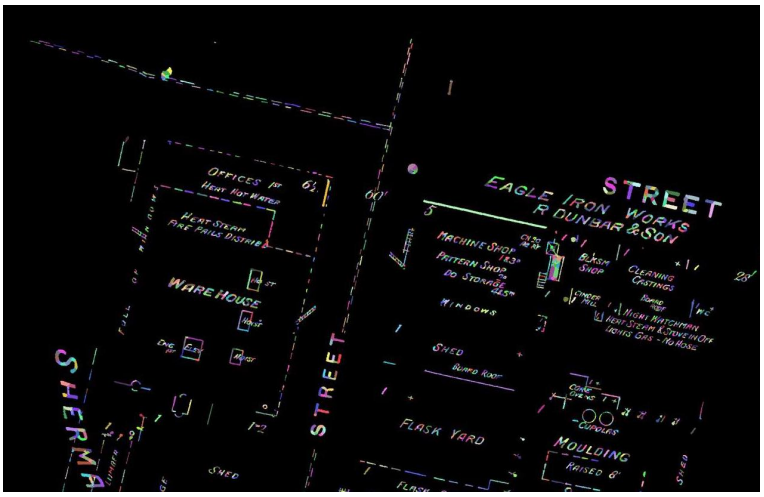


Figure 5.5 Result of running the thinning algorithm on a map. You can see the segments of the skeleton by how each segment is colored differently.

5.1.3 Pixel Island Detection

In this stage of the processing, we isolate pixel islands, or connected groups of pixels, and preprocess them so we can send it through a junk classifier.

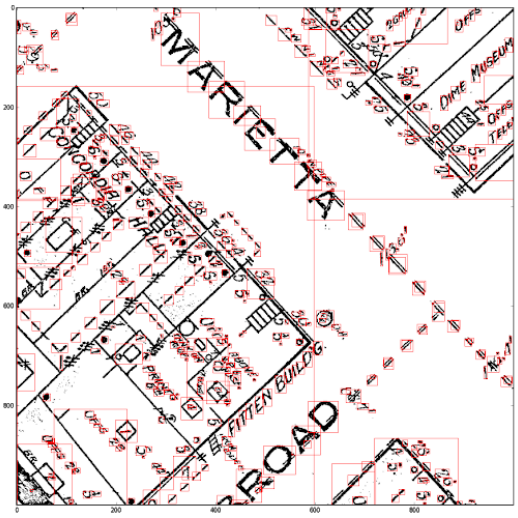


Figure 5.6 A map fragment with boxes around the isolated pixel islands.

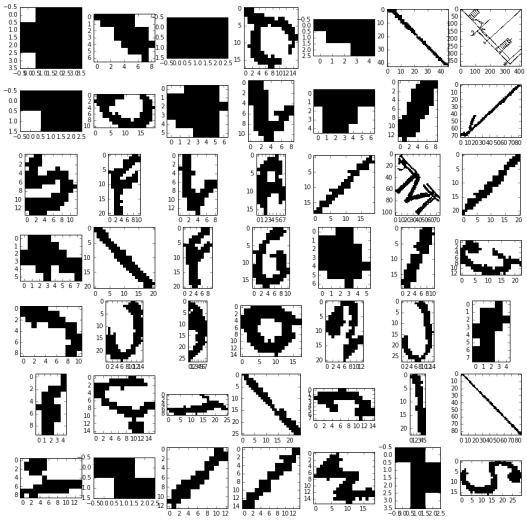


Figure 5.7 An array of isolated pixel islands.

5.1.4 Symbol Orientation

To orient our symbols, we put a minimum bounding rectangle around the symbol and rotated the symbol such the the minimum bounding rectangle was parallel to the x and y-axes. We expected that symbols representing the

same letter would look similar.

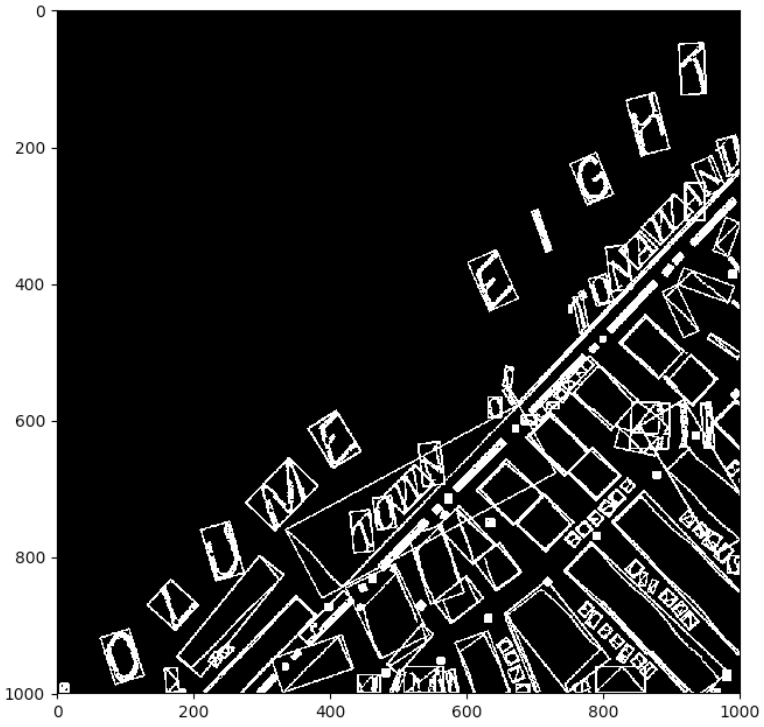


Figure 5.8 A map fragment with the minimum bounding rectangles shown around the pixel islands found.

5.1.5 Junk Classification

We trained a simple convolutional neural network to predict whether a pixel island is alphanumeric or not. The input is an oriented symbol (see 5.1.4), and the output is the probability the island is alphanumeric. We hand labeled around 5000 pixel islands as either alphanumeric or not to train the network. Since lowercase 'L's, and sans serif '1's are impossible to distinguish from unimportant marks on the page, we labeled them as non-alphanumeric.

5.2 Optical Character Recognition

Once we extracted the symbols from our maps, we used OCR to interpret them. There were various methods that we tried to interpret the symbols.

5.2.1 Circular Hough Transform

Just like the Hough Transform that detects lines in an image, there is a slightly modified version of this function that detects circles in images. This is useful to us because, as you can see in Fig. 5.9, many of the symbols used by the Sanborn surveyors incorporate some sort of circle. A particularly good example of the results of this portion of our code is shown in Fig. 5.10.

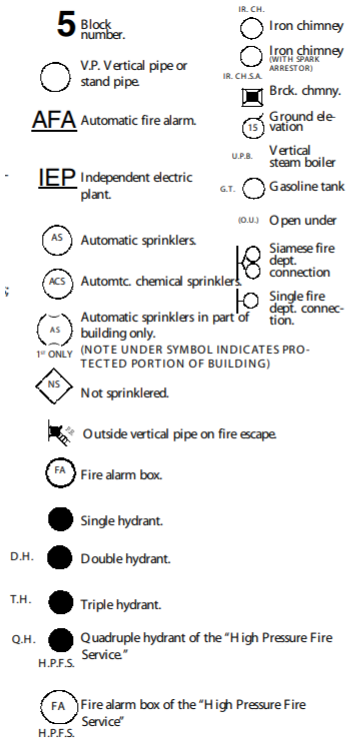


Figure 5.9 A sample of a key for a Sanborn map showing non-textual symbols, many of which are circles.

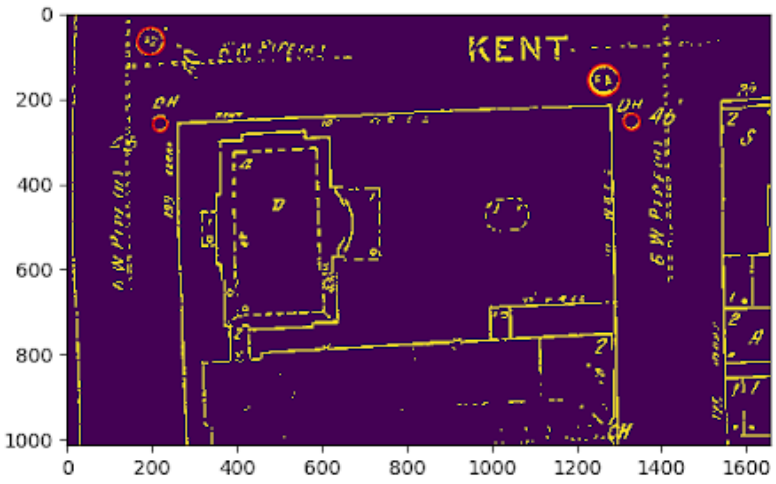


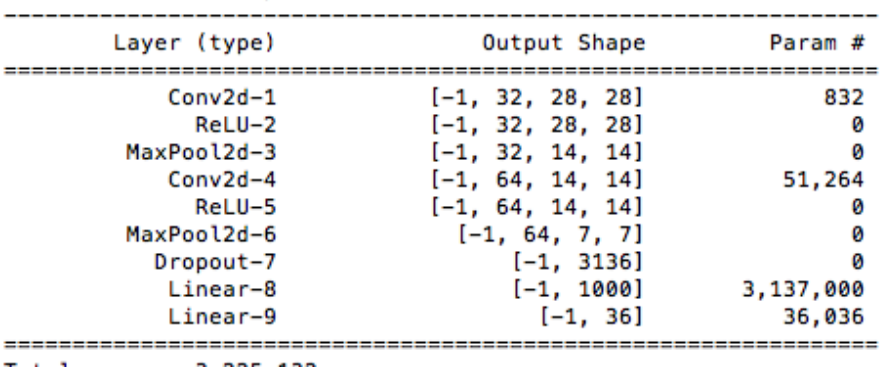
Figure 5.10 A map fragment with the code for the circular Hough transform run on it. The circles found are highlighted in red.

5.2.2 Neural Networks

The state of the art for handwritten character recognition is to use a neural network. We tried three different neural network approaches.

Convolutional Neural Network

We trained a Convolutional Neural Network (CNN) on symbols extracted from Sanborn maps. An overview of the model architecture is shown in Fig 5.7. We also merged some classes that are indistinguishable under a rotation invariant model. Below is a table showing the class number and the corresponding letters/numbers in that class.



Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 28, 28]	832
ReLU-2	[-1, 32, 28, 28]	0
MaxPool2d-3	[-1, 32, 14, 14]	0
Conv2d-4	[-1, 64, 14, 14]	51,264
ReLU-5	[-1, 64, 14, 14]	0
MaxPool2d-6	[-1, 64, 7, 7]	0
Dropout-7	[-1, 3136]	0
Linear-8	[-1, 1000]	3,137,000
Linear-9	[-1, 36]	36,036

Figure 5.11 Overview of the model architecture used for the CNN.

Table 5.1 Sanborn Classes

Class Number	Symbols	Class Number	Symbols
0	’0,o,O’	18	’k,K’
1	’1,i,l,I,L’	19	’m,w,M,W’
2	’2’	20	’r’
3	’3’	21	’t’
4	’4’	22	’v,V’
5	’5,s,S’	23	’x,X’
6	’6,9’	24	’y,Y’
7	’7’	25	’z,N,Z’
8	’8’	26	’A’
9	’a’	27	’B’
10	’b,q’	28	’D’
11	’c,n,u,C,U’	29	’E’
12	’d,p,P’	30	’F’
13	’e’	31	’G’
14	’f’	32	’H’
15	’g’	33	’Q’
16	’h’	34	’R’
17	’j,J’	35	’T’

We trained two sequences of convolution layer, followed by a ReLU activation function and a max-pooling function. We then add a dropout layer as a form of regularization and to prevent overfitting. Finally, we add two fully connected layers. We trained this model on 15046 training

examples and 3762 test examples. We also rotated an input symbol by 90, 180, 270 and 360 degrees and chose the prediction that was most confident. We did this because although we rotate our symbols based on minimum bounding rectangle, they can be on either x or y axis. We saw that the model was very confident in predicting a class - the output probability of class predictions was 1.0 for one class and 0 for all other classes. This isn't a problem when the OCR model makes the right prediction, however, in the cases when it makes an incorrect prediction, the model doesn't give any additional information about other possible classes. To tackle this issue, we introduced another form of regularization.

We modified the loss function to include the entropy of the output class probability distribution. The entropy of a probability distribution measures the uncertainty of the distribution. The more uniform a probability distribution is, the higher the entropy. Therefore, we maximize entropy in addition to minimizing loss.

Transfer Learning

Another OCR approach we tried was using transfer learning on EMNIST symbols. We trained a Convolution Neural Network on EMNIST symbols and then froze the gradients of all layers except the last fully connected layer. We tried this because EMNIST has about 2800 images of alphanumeric characters so we were hoping that the model can learn to capture the vague features of alphanumerics. However, we decided to use a basic convolution neural network because its accuracy was higher than the transfer learning model.

Autoencoder

The pipeline is capable of producing hundreds of thousands of pixel islands fairly easily given enough input maps. We used the junk classifier to get around 100,000 pixel islands representing characters with 90% confidence. We trained a variational auto encoder (see Tutorial on Variational Autoencoders) to encode filtered pixel islands. Once the encoder was trained, we trained a two layer dense net to classify encodings as one of the 36 alphanumeric classes we built.

90 Degree Rotation Invariance

The images we pass to the neural networks are rotated to align with their minimum bounding rectangle. For any given character, that could be one of four different orientations. Our networks try all four rotations, and output a class activation vector for each. Then we take the pointwise max across all four rotations, and take the log softmax of the result to get a log-probability vector.

5.3 Word Finding

Once we interpret each symbol individually, we need to interpret them together as words. This can be challenging since some symbols are interpreted incorrectly and it's difficult to recognize which symbols belong to the same word. Our approaches for grouping the symbols then interpreting the words they represent are detailed below.

5.3.1 Symbol Grouping

Once the symbols have been located, the pipeline groups them into "words". The words can be used to update the OCR output using nearby symbols. Combining the found words with OCR and spell-checking yields the final output of words present in the map.

Gaussian Blur

The first approach we tried was to blur symbols on the map so that nearby symbols would overlap. Ideally, the bounding box around connected symbols would surround individual words. This approach assumes words are differentiated solely by symbol nearness, which turns out to be a flawed assumption (Figure 5.12). Another drawback is it does not give the order of the symbols in the words.

Figure 5.12 A section of a binarized map. Note that the '3' in 'R113' is as close to the 'S' in 'Hampshire' as it is to the '1' to its left.

Minimum Spanning Tree

Another approach we tried was to build a minimum spanning tree using the coordinates of the symbols. The distance between symbols is calculated using a Euclidean distance weighted by differences in symbol size and direction. Once the tree is built, large distances are pruned, leaving sub-trees. The sub-trees ideally would be words, though they might have multiple branches The minimum spanning tree approach requires direction to be determined fairly accurately to work. We achieved some preliminary results, but decided the iterative approach was more promising (5.3.1) partially due to the difficulty of pruning the sub-trees to words.

Iterative Directional Symbol Grouping (IDSG)

The final pipeline uses an iterative approach to symbol grouping. It assumes symbols in a word are of similar size, near to each other, and arranged mostly

linearly. The algorithm is as follows:

```
for symbol in symbols:
    nearby_symbols = find_nearby(symbols)
    candidates = []
    for nearby in nearby_symbols:
        if nearby already used in recorded word:
            continue
        cutoff = RADIUS * size(symbol)
        if distance_between(symbol, nearby) < cutoff:
            candidates.append(
                nearby, direction_from(symbol, nearby))
    for nearby, direction in candidates:
        search around nearby in direction for rest of word
        if len(found word) > MIN_WORD_CUTOFF:
            record found word
```

This approach is fairly successful and takes less than one second per map. It fails when letters are broken up into two symbols or joined as one symbol in the map, or the letters have inconsistent sizing (Figure 5.13). It also sometimes improperly separates or combines words if they are together in a line with inconsistent spacing.

Figure 5.13 The red lines show grouped letters. Note that 'Ou.b.' is not detected because the 'O' is too large relative to the 'u' and 'b' and the 'ub' falls below the minimum word length cutoff applied of three.

5.3.2 Word Recommendation

After the pixel islands have been grouped together, we can take each island's highest probability predicted class from the OCR to form an entire word. However, the resulting groups are not yet words because the predicted classes in OCR often group together similar-looking symbols like 'I' and '1'. Furthermore, since the OCR system's accuracy is not perfect, it is quite likely to have a small number of incorrectly predicted symbols, meaning that the word that's put together will have mistakes in it - this makes it much harder to search for specific words. Additionally, the symbol grouping does not have an orientation, meaning that the order of the words could easily be backwards.

In order to resolve all of these problems, we developed a manual word-recommendation algorithm that uses spell-checking techniques and a custom-made Sanborn word bank to return the words that a given group of symbols is most likely to represent. The first step in developing this system was to

look through about 25 Sanborn map images and identify approximately 200 words that appeared to be most common in these maps. These words form the word-bank used for the spell-checking.

For each symbol group, the algorithm determines the words from the word bank that have the smallest weighted edit distance from the group. The weighted edit distance is based on the default edit distance algorithm but adjusts it by accounting for classes that tend to be confused with each other often. The cost for replacing a letter with another is reduced based on how often they are confused by each other in the confusion matrix from the OCR model's test evaluation. Furthermore, the algorithm checks if the reversed word matches the group as well, ensuring that potential reversals from the symbol grouping step are accounted for. Since each of the word bank words is originally something found in a map, the resulting recommended words are all in fact actual words from maps as opposed to just groups of classes. When giving a final output of recommended words, the algorithm only returns words that had less than 40% edits relative to the original - empirically, it was observed that most actual words that could be guessed from the OCR output would not have more edits than this.

Chapter 6

Results

The following describes the methods that we use in our solution, along with images that demonstrate the result of our processing. The result of each step of our processing will be shown visually through debug images of the following map as it is processed. The performance of each of these methods is also discussed.

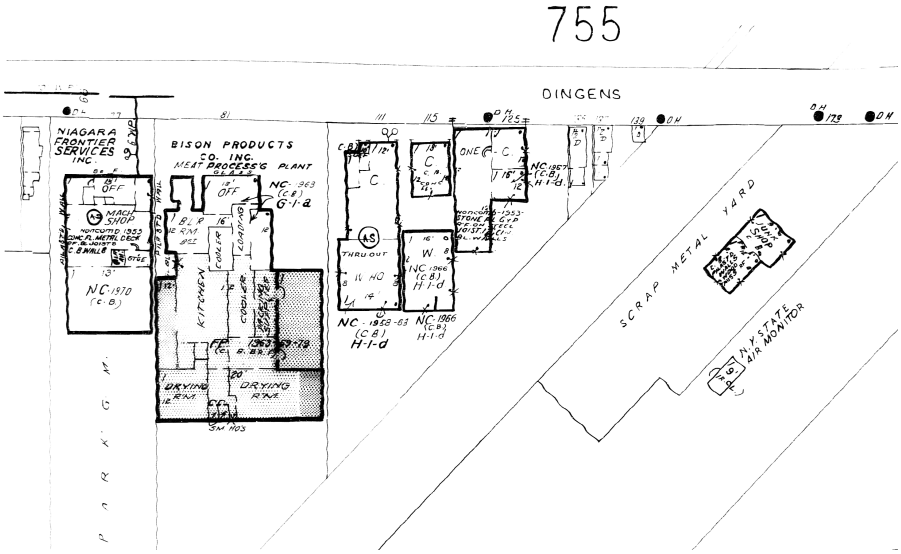


Figure 6.1 Original map.

6.1 Image Processing

We begin our image processing with a binarization of our input map. The adaptive Gaussian thresholding mostly does a good job.

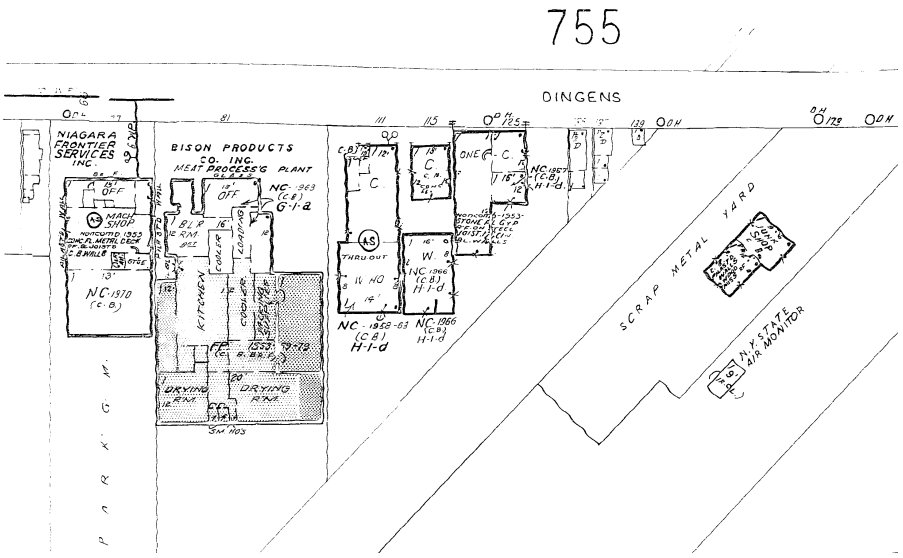


Figure 6.2 Binarized map.

We then apply our line removal algorithm. Although the line removal algorithm is able to remove many of the parcel boundaries and street lines, it is unable to remove all the lines. This could be due to the line being too short, bent, or cluttered with other markings.

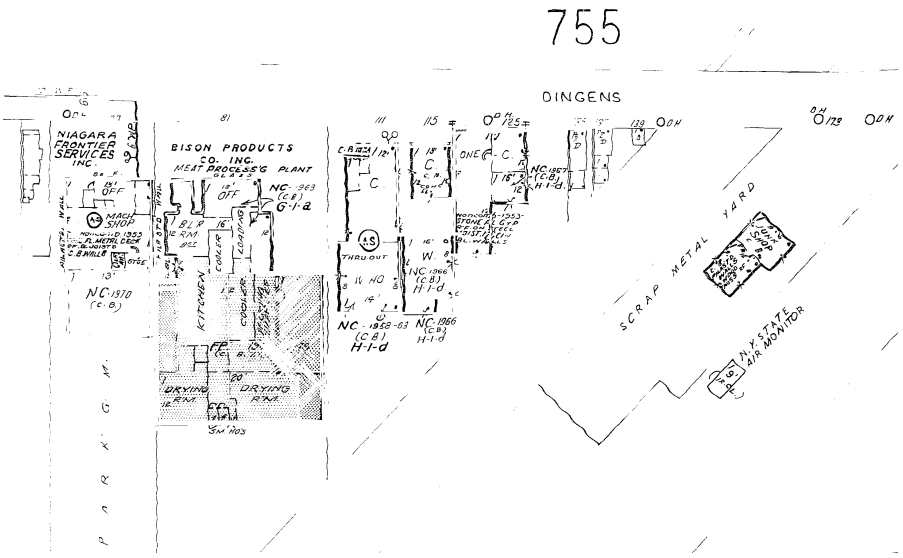


Figure 6.3 Thinned map.

Once lines are removed, we isolate all the remaining pixel islands and put bounding boxes around them to orient them.

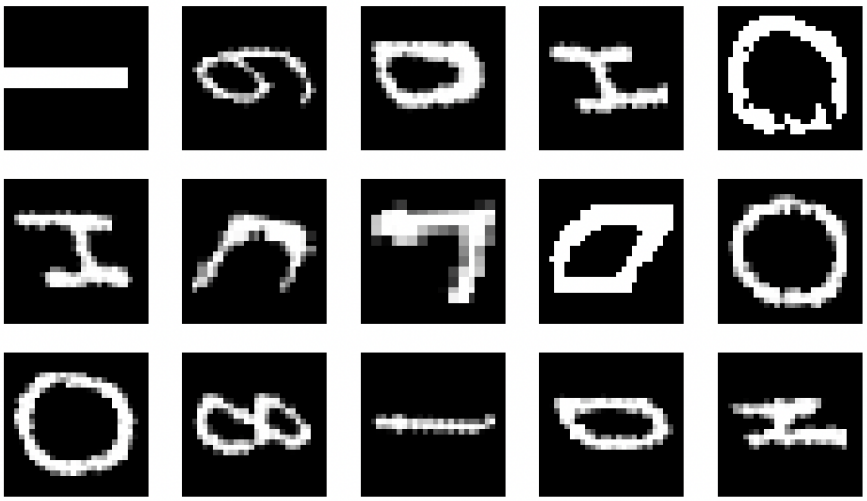


Figure 6.4 A sample of pixel islands isolated from the map.

These pixel islands are then sent through a junk classifier which leaves

us with a list of useful symbols. These are then passed to the OCR to be recognized.

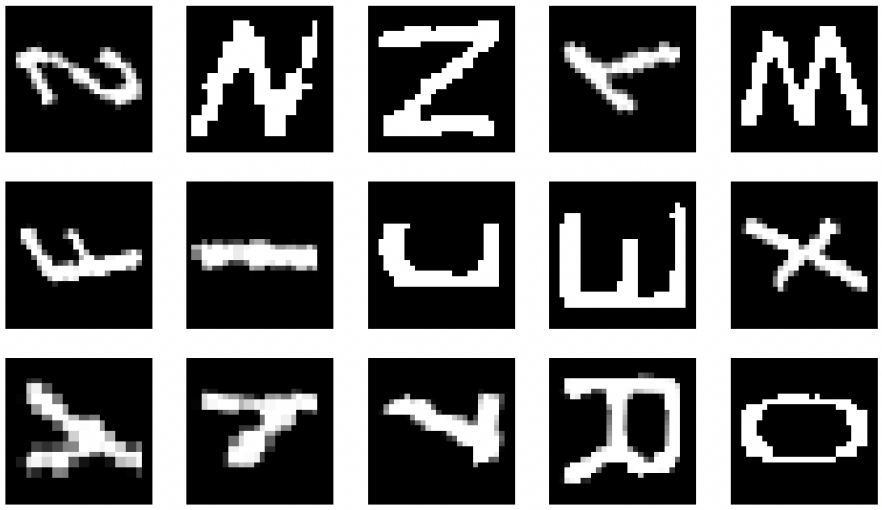


Figure 6.5 A sample of symbols filtered from the collection of pixel islands.

6.2 Optical Character Recognition

Figure 6.6 OCR results of each symbol superimposed on the map.

We trained our model using a convolution neural network (CNN) along with a loss function which incorporates entropy of the predicted class probability distribution. The training and test accuracy using each of the three approaches described in the methods section is shown below.

Table 6.1 OCR Results

Model	Training Accuracy	Test Accuracy	Learning Rate
CNN with Entropy	86.9%	85.3%	1.00E-04
Transfer Learning	85.02%	81.3%	1.00E-04
Autoencoder	92.0%	83.3%	1.00E-04

One limitation of the model is that we have an imbalanced representation of classes. We can see in the figure below that there are a few classes that have fewer than 500 examples.

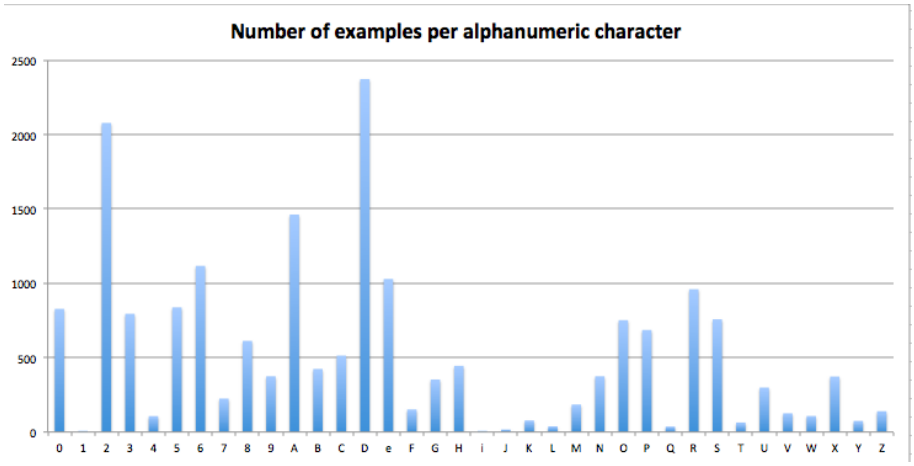


Figure 6.7 Graph of number of examples in each class.

We have found that characters with few examples are the ones that are likely to be incorrectly classified. Therefore, having a greater set of training data should further improve our accuracy.

6.3 Word Finding

We grouped nearby symbols using iterative directional symbol grouping.

Figure 6.8 Grouped symbols from the map where each group is connected by a gray line along with their OCR results.

Once we got these groups of symbols representing words, we sent them through the word recommender. The final word recommendation algorithm performs quite well when the output from the previous step - after OCR and symbol grouping - can be reasonably interpreted. There are almost no cases when the algorithm fails when the output from the previous step would be easy to understand for a human.

However, in many cases, when symbols are put into incorrect groups or there is a significant incorrect classifications of symbols, the algorithm cannot correct for these. The final output is also subject to a certain number of false positives when multiple words get recommended. Finally, the big limitation of the algorithm emerges from the word bank, which only covers a subset of the numbers that will be encountered. We believe that the results from this step can be augmented significantly by expanding the word-bank as necessary if any important words from maps are being consistently missed.

The following is an example of a portion of the JSON output for one word recognized from the map and spell checked.


```
{
  "word_centroid": [
    1442.5979381443299,
    824.8453608247422
```

```
    ],
    "word_letter_classes": [
      "0",
      "z",
      "E"
    ],
    "spell_checker_suggestions": [
      [
        "STONE",
        2
      ]
    ]
}
```


We see that 'word_centroid' gives the centroid of the word on the map, 'word_letter_classes' gives a list of the class each symbol in the word belongs to, and 'spell_checker_suggestions' gives a list of possible words along with their Levenshtein distance from the OCR result.

6.4 Overall Performance

On average using a computer with 4 GPUs, each map takes approximately 10 seconds to process completely. Running the pipeline with the debug option on, i.e. debug images are also generated, does not significantly slow the processing time.

Chapter 7

Future Work

7.1 Image Processing

The image processing can be improved further. For example, binarization sometimes breaks up symbols, making them unrecognizable in the future portions of the pipeline.

Also, our motivation behind implementing thinning to remove lines was that some symbols were drawn over lines. This is problematic since then the symbol would be joined together with that line as a single pixel island. If we are able to remove these lines, then we will be able to extract more symbols. Furthermore, currently when the lines are removed, they get erased completely, meaning that if a symbol had a line drawn straight through the middle, it would get split in half. This is also not good. Thus, we need to have a way to remove these lines without erasing portions of symbols.

Another issue that came up was being able to tell the difference between short lines and the letters 'i', 'l', and 'I'. This is something that might be able to be solved by analyzing the surrounding context or providing special treatment for these symbols.

Being able to detect the rotation of symbols is also something that can greatly improve the pipeline. Currently, most of our OCR and word recommendation inaccuracies stem from the fact that we don't know the absolute rotation of the symbols. Through symbol orientation we are only able to reduce it to four rotations. This means that 'c', 'n', 'u', 'C', and 'U' can't really be told apart by the OCR, leading to a lot of merging of classes.

7.2 Optical Character Recognition

One limitation of the OCR model is that we have an imbalanced representation of classes. We can see in the figure below that there are a few classes that have fewer than 500 examples.

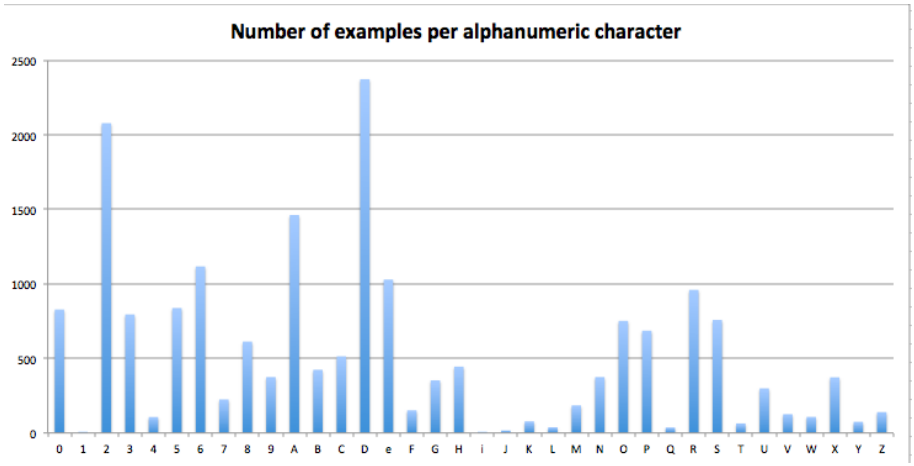


Figure 7.1 Graph of number of examples in each class.

We have found that characters with few examples are the ones that are likely to be incorrectly classified. Therefore, having a greater set of training data should further improve our accuracy.

7.3 Word Finding

We added symbol grouping to the pipeline fairly late, so there are a lot of ways it could be improved. One possibility is to use the distance between the convex hulls of symbols instead the distance between their centroids. This could help us better find words since it better reflects how humans read words on a map.

Another possibility is to incorporate bigram probabilities to decide which groups of symbols is most likely. For example, it is much more likely to see an ‘e’ than an ‘x’ immediately after a ‘c’.

We can also build a “training” data set of maps with known groupings to tune the symbol grouping parameters. This can help us better adapt our code to Sanborn maps.

Chapter 8

Conclusion

Our goal this year was to highlight and interpret symbols on the scanned Sanborn maps that may indicate an environmental hazard. Many of the challenges that we faced had to do with variations in the data. Despite being drawn using a process that was remarkably consistent, these maps were drawn by hand on paper, scanned and printed onto microfiche, and then scanned into an electronic format. Even if the maps had been scanned at a higher resolution, this opens up many sources of error for our data. There could be minor differences in the handwriting or scratches on the microfiche, for example. While our data were significantly more uniform than might be expected for a handwriting recognition project, they were much messier than what might be expected for a typical OCR project. As a consequence of this, we spent the majority of the first half semester trying to extract the relevant data and make them clean enough to be interpreted by the computer.

The question was posed to this team of whether it would have been beneficial to have been given images in color, since the images that we used were in grayscale. It is difficult to answer this question, given that we have not seen many examples of the colored maps in order to be able to determine what context clues might be useful from them. However, we speculate that, since not much information is encoded in the actual color values on the maps, it may just have served as a distraction.

The astute reader may notice that we tried many strategies were discarded for the reason that we did not think we had enough time to implement them or improve them. In many of these cases, we settled on the simplest solution. This may be for the best because these are more time-tested and well-established methods and tend to require less overhead in terms of

both memory and time. However, these methods have not been definitively shown to be ineffective or even less effective, and if a team were to continue this project, they may find it worthwhile to revisit these strategies.

As for the rest of the project, our results seem promising. We have not had the time to test the accuracy our final result more rigorously, but our preliminary tests show promise. As with all projects, there are still some markings that our model misses or misinterprets; the model could definitely be refined and the results greatly improved. However, the results that we have thus far are promising. Also, the framework that we have put in place is highly modular, so new models and algorithms can easily be interchanged for the old ones.