Handwriting Recognition Tool

Takehiro Matsuzawa, Emma Weil, Sofia Shapiro, Thomas Waite

LINK TO DEMO VIDEO:
https://www.youtube.com/watch?v=fU-Uym7US5g

Overview

We have implemented a program that recognizes alphabetic characters in handwritten text and return the text in a .txt file. The first major component of our program is the extraction component which separates and input image of handwritten letters into individual character images, returning these images with relative position and spacing data so the letter can later be reconstructed into a text file. The second key component is the neural network, which, after learning from a large database of handwritten letter images, can recognize the characters extracted from the input image. Our program also utilizes a GUI and a number of supporting scripts and formatting modules that allow for the generation and appropriate formatting of data to be fed through our algorithms.

Planning

Here is our Draft Specification and Final Specification, we had originally planned to have the program recognize more complex handwritten sequences like mathematics notation as well as punctuation and numbers. We were eventually limited by the size of data sets required and the time it takes to train a network. As such, we focused on upper and lowercase letters alone, allowing us to spent a little more time developing the network rather than gathering data. The second part of our initial proposal that was not completely fulfilled is the complexity of our outputted text. We had intended to output the data in a more beautiful format, but again, our focus was directed toward developing our core functions.

Design and Implementation

Formatting

The next step in our software, object separation, expects the image to be in the correct format with only white (255 bytes) or black (0 bytes) pixel data. In order to achieve this we made a formatting function using PIL and Numpy image processing libraries to upload an image, convert it to an array, change an uploaded image into black and white, and reconfigure the image. We wanted our algorithms to work on images taken with a camera which meant the background could be an off-white and the black text would not necessarily appear black. To compensate for this our function found the darkest pixel value and compared every pixel to it. If the pixel was within a reasonable amount of darkness away from this value it converted it to black (0) and everything else was converted to white (255). Then this list of pixel data was fed back into an image readable format.

Object separation

In order to make the recognition part work, we need to separate objects and make a numpy array. The "runExtr" function inside extraction3.py returns a numpy array. Numpy contains 0 if the color is black and it contains 1 if the color is white. In order to make the Numpy objects of each character, we need to remove white space of each character. We first checked a row of pixels in this numpy array to see if they were completely white. If so, we remove them. Then we rotate the chopped row image by 90 degrees and repeat the same process (thus addressing white columns). By repeating the same entire process again, we can extract each individual character. We also need to make sure that in the final text output the characters maintain the same order and line placement.
Based on the white space between letters, we try to recognize the connectivity between letters (main2 function inside the extraction3.py). If there is no white-space before a letter, we assign 1 to that character, and if there is white-space before a letter we assign 0 to that character. We then compose a tuple of the image array and a tuple of its row coordinate and leading white-space (0 or 1): **([numpyarray],(row,whitespace))** for each pixel.
The recognition part works using 28 by 28 pixel arrays therefore we need to modify the numpy arrays of each character to be 28 by 28 pixels if they are not already. As a result, the numpy array contains 28*28=784 numbers - which is helpful to know later in our recognition algorithm. By looking at the row and whether a character has white space before itself, we see whether we need to put space before the character in the final output.

Neural Network

There were two main design-heavy elements to this project. The first was the object separation and the second is the neural network algorithm. We developed our network relying heavily on the example and explanations presented by Micheal Nielson at neuranetworksanddeeplearning.com [1]. We adapted the network and some of its optimizations as we trained it to recognize the 52 character set of lower and upper case letters. This was an extension of the network's example structure, which only identified the 10 digits. One of the added challenges of expanding to this larger set of characters was getting a sufficiently larger data set to train and test the set. We accomplished this by downloading the NIST SD19 data set. this gave us access a series of images from numbers and upper and lower case letters. However, all of this data needed to be preprocessed before being placed in the network, and as a network requires a large training set and many training session, after we loaded and processed our 1000 distinct uppercase and 1000 distinct lowercase alphabets to for the training and testing data, the runtimes for training quickly became unwieldy on our machines, so we were never able to train a network for more than 30 training session or so, with training data sets of 800 distinct sets of characters. This shortcoming shows in our maximal identification accuracy of about 61%. However, we were able to see dramatic improvements in our network's learning rate of this early phase of learning with the addition of several enhancements. Some of the optimizations we were able to implement were better initializations of the weights and biases as well as the introduction of a parameter to helper normalize the learning rate and variance as well as a cross entropy cost function, rather than the more simplistic quadratic cost function. These improvements showed massive improvements in accuracy over the time scale of about 30 training sessions from 10% with the nascent method to ~60% with the optimized method. These improvements operated primarily by preventing the network from too quickly getting "saturated, meaning the weights and biases lie so high on or low on the sigmoid function that they do not respond well to small changes in input, leading to a decreased learning rate. At the core of this algorithm were two key ideas: gradient descent and backpropagation. A simple method for numerically finding the minimum of our cost functions and a method for "teaching" the network retroactively given the amount of error is associated with the calculated output and the real output. These were very interesting to learn about since they are rather simple ideas mathematically, but allow such a complex and powerful tool as a neural network.

GUI

The GUI was created using a Python library called Tkinter, which allows for the implementation of various "widgets" that interact with the program and user in various ways. The main functions of our GUI is to actually run the program given input from the user, which is the file path/image name to be converted into text. The GUI accepts this path name and will start running the conversion functions when the "Convert" button is clicked. After a few moments, the text discovered will appear on the right side of the GUI window. The text can then be exported into a .txt file with the name entered by the user at the bottom right of the window. Each of these fields is made with the Entry widget from Tkinter, and the text and images that appear in the window are Label widgets.

Reflection

How good was your original planning?
While we were not able to make a interface with the accuracy we had hoped for a the start and failed to accurately estimate the time required for seemingly simple interfaces like GUI, we were still able to plan a project and prove it's core concepts in a new language.
How did your milestones go?
By working on the milestone seriously, we were able to set our goal and work toward our project. We made algorithm and pseudocode in the milestone. We all had a good goal of what we need to do and a good sense of which part each person is responsible for. In short, by keeping with our milestones as best we could, we were able to learn a great deal about the material while still producing a product that proves our concepts.
What was your experience with design, interfaces, languages, systems, testing, etc.?
No one in our group was particularly experienced with Python, so it was a bit of a challenge in that we had to pick up a new language and learn how to use it well while also implementing many different libraries to assist us. Creating a GUI was also new to us, and although it seemed fairly straightforward, it ended up being more complicated than we realized because of the limited capabilities of various GUI libraries and complication with running the entire program through it.
What surprises, pleasant or otherwise, did you encounter on the way?
One of the exciting things was seeing our algorithms actually working as expected, which as a group of fairly new programmers was surprising. It was great to work on the algorithm on object separation. It was tough to keep the order of objects. I ended up in keeping the order of objects with an element of values.
What choices did you make that worked out well or badly?
We split the work pretty equally. As a result, everybody almost had the same amount of work and could progress as a team.
Bad decision: We began to combine our work pretty late. As a result , we encountered the problems that we did not expect. Not noting how long the train would take until after the time.
What would you like to do if there were more time?
We would run our neural network over more training sets and increase the accuracy. Unfortunately, since we did not have a big computer, it took an enormously long time to go over training sets and we had no other reliably faster method for doing this. We would also like to add in digits and more logical checks to distinguish between lowercase and uppercase letters. For example, distinguishing between zero, lowercase "o," and uppercase "O," seems like a nightmare. To help our algorithm out we would put in context checks. If the character is found amongst a string of other numbers then we would be more likely to rate the zero with a 1 and the O's as lower.
How would you do things differently next time?

We would make a GUI earlier than we did. GUI is more complicated than I expected and it took a long time to finish the part. It would have been nice to have time to let the neural networks train for many training sessions to reach their potential. As it is, they are sort of half-baked, even though they showed great potential.

What was each group member's contribution to the project?

Thomas Waite: Working on the recognition part, building neural net (recognition.py)
Takehiro Matsuzawa: Working on the object separation part (extraction.py)
Sofia Shapiro: Working on the format.py, video and final report
Emma Weil: Working on the gui.py, formatting.py

What is the most important thing you learned from the project?

We learned the difficulty of combining our work together. We also realized the applicability of simple mathematical concepts to produce complex and accurate software. It was also a useful exercise in learning from the existing literature in computer science and applying it to our own projects, a skill that will come in handy for the rest of our lives in computer science.

Advice for Future Students

Start the project and combine your work early because combination of your work is as important as separation of objects. I would consider installing the development tools in a more powerful machine if you are interesting in truly optimizing the performance of your network. Otherwise, it takes hours to run just 20 training session, which is somewhat of a letdown when most reach their optimal accuracy much later.

Citations:

[1] http://neuralnetworksanddeeplearning.com/

This website, and it's attached code, were our inspiration and example in the development of our own neural network interface.

[2] http://en.wikipedia.org/wiki/Connected-component_labeling

We used the two-pass algorithm in extraction3.py in order to separate different characters in inputted images.