Ryan Helmlinger

Period 5

SysLab Paper Draft

The main goal of this research project was to create an online crossword puzzle solver. It would be able to take in an image of a crossword puzzle and display it to the user. Then the puzzle would be solved in the background and the user could ask for hints if needed when solving the puzzle themselves. The inspiration for this came from my grandma. She loves crossword puzzles; she usually does the daily one in the newspaper, plus extra ones from books. However, she has arthritis in her fingers, so trying to write and erase in small squares can be difficult for her. This program and application would allow her to continue to do what she loves without provoking her arthritis.

There currently isn’t anything like what I’m attempting to create. There are some websites that list answers to crossword clues and some separate websites that display a crossword user interface. But none of them combine the two. Also, there isn’t any current program that reads in an image of a crossword and converts it to digital form.

My first step was to have a program that could look at a hint and determine a list of words with the highest probability of being the answer to that hint. At first I was attempting to understand in depth semantics in order to parse the hints and then use the Google API. I looked around for other APIs as well, but none were better than Google’s. However, after a while I wasn’t making much progress with that strategy, so I tried to figure out a new one. I eventually came across the idea of scraping synonyms and matches of crosswords from online websites, especially from Dictionary.com. This strategy worked very well and allowed me to solve the hints fairly accurately. I used pip to install and import lxml and requests.

The next step was to represent a picture of a crossword puzzle digitally. I started with a basic image of a crossword with straight lines and no blurred sections. Since all the lines were straight, or at least close to straight, I essentially looped through the picture horizontally and vertically to find a certain number of dark pixels that were in a row so that I knew it would be a line that’s part of the grid. I also made sure that no two lines were too close together because most grid lines on pictures were greater than 1 pixel wide. While doing this, I kept track of the location of and number of lines that were detected. The number allowed me to determine the size of the grid. The location allowed me to loop through each line, horizontal and vertical ones, and find the average between each consecutive one. When doing this, it gave me the center point of each box in the grid. If the center point was a dark color, then the spot was filled. Otherwise, the spot was empty. Since the numbers in the top left corner are really small, it is hard for the computer to detect them. So instead, since numbering works the same for all typical crosswords, I wrote a method that numbered the puzzle itself. I used pip to install numpy and openCV2 for this portion of the program.

After detecting the grid, reading in the actual hints was the next step. I used Pytesseract and PIL for this step. I was able to install PIL from pip, but since I use a Windows computer, Pytesseract didn’t work right away when I installed it. I ended up having to install it from an unofficial provider at UB Mannheim. Using openCV2, I erased the grid so only the text remained. Then I used Pytesseract to read in the text. Next, I wrote a program that went through the hints and separated them by number and between across and down sections.

The final step of this was bringing all the parts together into a web application. I created a crossword grid and sections for across and down. I also added features where when you typed in text at automatically took you to the next square, depending on which direction you had selected. It also highlighted the entire space of the current word you were trying to input. Beneath, there is a section for asking for the possible word clues from the hints of the puzzle. This part was a combination of html, css, javascript, and jquery. It was a little tedious setting it all up to look aesthetically pleasing, but it wasn’t too complicated, especially since I had experience with it. The tricky part, however, was getting my python files and the website to talk. I tried googling this, but didn’t find any good methods or programs. I even tried some but they didn’t work. However, when talking with a friend a couple weeks later about a separate Computer Science project, he mentioned flask. I briefly asked him about it, and it was the perfect program for me to use for this project. I looked up how to use it and I was able to get the python files and website to work together.

At first it seemed like I was hitting a dead end with the semantics. I couldn’t get my program to work well at guessing answers to hints. But then once I changed my strategy to scraping information from websites, my program started doing a great job of guessing the clues. When looking at results from Dictionary.com, my program did really well, especially when it looked at clues from previously used New York Times puzzles. I believe this is because Dictionary.com’s algorithms looked through those puzzles when creating the results. The computer vision portion, however, didn’t work as well. While the algorithm to detect the grid worked extremely well and is efficient, detecting the actual words did not work nearly as smoothly. For example, with words like “Without”, the program would read them as “V1thout”. I tried looking up ways to make Pytesseract more accurate, but this is the best it can do. It also occasionally messed up the clue numbers, which also made it difficult to translate it to the across and down sections and correctly number each hint. I attempted to do this, and left the hints blank that the program wasn’t sure about.

Originally, I wanted the program to be able to completely solve a picture of a crossword puzzle. But, after I realized that the text detection didn’t work well, the puzzle couldn’t possibly be solved because not all of the hints were clearly legible. I wrote an algorithm that could complete the puzzle when given options for every clue. However, it only worked when the correct word was one of the options for each word. Since the clues were slightly messed up from the text detection, there are definitely some clues won’t have the correct word.

Overall, I has able to solve most of the problem that I wanted too. However, not being able to completely solve an image of a crossword puzzle from scratch was the biggest failure in this research project. For improvements, text detection would be the most important item to increase accuracy. However, being able to read in grids that are blurred, smudges, and angled, would also be an important addition. Also, making the interface available on tablets and phones as an application would be an overall improvement for usability and popularity. This was a fun project to work on, and I believe I made some good headway, but there is still a lot that can be done to improve it.