Project Title: Sudoku Solver

Results So Far: So far we have been able to do a number of things. We are able to take in an input image of a Sudoku puzzle and threshold it using adaptive Gaussian thresholding. From there, we can detect the largest square in the image- the outline of the puzzle-, find the corners of this square, and create an orthophoto based on these corner locations. Then, we are able to partition the whole puzzle into individual tiles, recognize if a digit is present in each tile and then detect which digit is present using a neural network that we have trained against the MNIST handwritten digit data set. This detection works very well for handwritten digits and decent for printed digits in the puzzle. Once all of the digits in the tiles have been identified, we have implemented a graph search algorithm that returns whether the puzzle is solve-able and, if it can be solved, returns the completed puzzle. Last, we are able to output the completed puzzle on top of the orthophoto of our input image that we created previously.

Problems Encountered: The first problem that we encountered was in thresholding our input image of the Suduko puzzle. We intended to use Otsu's binarization in order to handle abnormalities such as shadows in our image. This form of binarization, however, proved to be ineffective because our images required a more regional thresholding instead of the global thresholding that Otsu's uses. To solve this, we ended up using Adaptive Gaussian thesholding instead.

Another issue that we encountered was recognizing the digit within each tile of the puzzle compared to other contours that exist in and around each tile. So far, to solve this, we have found the contour in each tile that is relatively square and fills most of the bounding box. This works about 90% of the time, but does not work well when detecting digits such as the number one.

Additionally, we have run into issues in properly formatting inputs for our neural network from digits that we have identified within our tiles. These inputs are required to be 28 by 28 pixels in order to match the input format of the MNIST data set. We have found, however, that if the digits in the input image are at the borders of the 28x28 image, the neural net fails to properly identify them, as most of the digits in the data set have a bit of padding around each of the digits. In order to solve this, after we identify the location of a digit within a tile, we extract this digit from the image and add padding around it to make it a square before re-sizing it and inputting it into the neural network. This has greatly improved the accuracy of our digit classifier.

Plan for Remaining Work: The remaining work to do for our project is to make it much more robust. This robustness must be implemented in two areas: the digit classifier and the digit recognizer. The digit classifier is currently trained off of the MNIST data set, which works well for recognizing handwritten digits, but does not perform well on 'cleaner' fonts that exist on the given numbers in the Sudoku puzzle. We plan to inject new data into the MNIST data set of the same format for a variety of fonts such that the model can better generalize the digits it is attempting to recognize to common font styles. In order to better detect the digits, we are going to take advantage of the fact that the digit is the most centered contour in each tile and choose that contour to run through our classifier. This position is not simply the center of mass (since a full box outline around the tile also has a center of mass around the middle of the tile) so we will rather compute which contour has values that exist closer to the center.

Additionally, we plan on making our application more user friendly. To this this, we plan to make a simple user interface that will allow the user to upload an image of their Sudoku puzzle, choose to solve it, and either output an error message if the puzzle is unsolvable or a nicely formatted image of the solved puzzle if solve-able. We will likely use the Tkinter UI for python to achieve this task if the built in UI tools for OpenCV are found to be insufficient for what we want to implement.