Using Deep Learning to Read Sudoku Puzzles and Provide the Solution

Richard Gower, Bryon Mosier, Joshua Roach, Veronica Stephens, Donald Villarreal, Aaron Wright

***Abstract*-The purpose of the project was to identify the potential methods for importing images of unfinished or partially finished sudoku puzzles and provide the solution. The first issue was identifying the grid of the puzzle so the program looks in the correct areas to find the numbers. We examined the MNIST dataset to analyze handwritten and printed numbers and provide the correct output values using machine learning. This proved rather inaccurate for our tests so images were compiled and labelled with the correct value to create our own dataset which was a significant improvement. An interface then needed to be created in order to manually make corrections to the interpretation of the puzzle. Once any corrections are made, then the user can select to solve the puzzle. The logic of the solution itself was very straightforward and simple to code. The most difficult aspect was reading the image of the puzzle grid and interpreting the printed text to populate the grid in the app.**

I. Introduction

Deep learning is far and away the fastest growing segment of machine learning, but the concept has been around for decades. This tool, which was once a fantasy, has been brought to fruition by the exponential leaps forward in processing power, especially in the area of GPUs. Now, deep learning can be used, not just by massive companies with the infrastructure to support it, but also by the individual with a decent laptop. This is what has allowed us to use deep learning, in the form of a convolutional neural network (CNN), to read the numbers in a sudoku puzzle and complete it.

We began with the task of converting an image of a sudoku puzzle into a useful numpy array. This was completed by determining the largest square represented in the array, and then determining the smaller squares that comprised to largest. Next, we had to determine which of the 81 smaller squares were blank or had a number. Lastly, and quite possibly most importantly, we had to use a CNN to determine what numbers were inside the squares.

II. Dataset Sources

The MNIST handwritten number dataset was used to train the model, at first. While a fantastic dataset with thousands of samples, it was not ideal for the determining the numbers in a sudoku puzzle. It was very readily apparent that the CNN model was not predicting the true values of the numbers in the sudoku squares very well, and the cause was discovered to be the fact that the code used to scrape the individual numbers from the sudoku squares was not able to do so without grabbing some or all of the edges of the square around the number. So, while the training dataset was very clean, the actual inputs were nothing of the sort. Two methods were attempted to remedy the issue, clean the sudoku numbers to make them look more like the training dataset, or make the training dataset dirty to make it more the like the sudoku numbers. Ultimately, it was decided to make the training set dirty.

To create a new training set that would more closely align with our imported images, a new dataset consisting of 37,800 labeled images was created using Adobe InDesign. Utilizing fonts rather than the handwritten samples from MNIST yielded significantly better results. To further improve the accuracy, the numerals were blurred, skewed, off-centered and had gridlines added to the edges of some of them. The end goal being to have a dataset of images that looked as much like the numbers from the sudoku boards as possible.

III. Machine Learning

The CNN we used started with a 2D convolution using 64 filters, a kernel size of 3, and relu activation. After maxpooling, the outputs from the first convolution were run through another 2D convolution, this time with 32 filters and the same kernel size and activation method. After that, it was maxpooled one more time, flattened, and the dense function was applied twice, with relu activation first and softmax second. It is important to note that the arrays run through the CNN were 28x28x1, and padding was used to ensure that arrays maintained that size as they progressed through the layers.

IV. Application

The Sudoku application was created to tie together the different pieces of the solution into a single comprehensive application. It evolved during the project into a tool that helped identify issues with the image extraction and helped understand how the deep learning models were working. The application is written in Python with the GUI elements written using the Tkinter framework. It was designed to be somewhat modular to allow separate development on the image extraction and the number identification by different people.

A. Sudoku Application – Technical Details

The application is made up of 4 Python programs:

- sudokuBoard.py – responsible for displaying the GUI and handling the user interactions
- imgRead.py – responsible for analyzing the initial image, breaking it into the component number images, cleaning the number images, passing them into the CNN model to identify the number, and return a matrix to the GUI for display
- sudokuSolver.py – responsible for solving a partially filled out Sudoku board.
- sudokuClasses.py – holds any shared classes used between the programs. Currently only includes the analysis parameter object used for communicating to the image processing, the analysis configuration settings input by the user

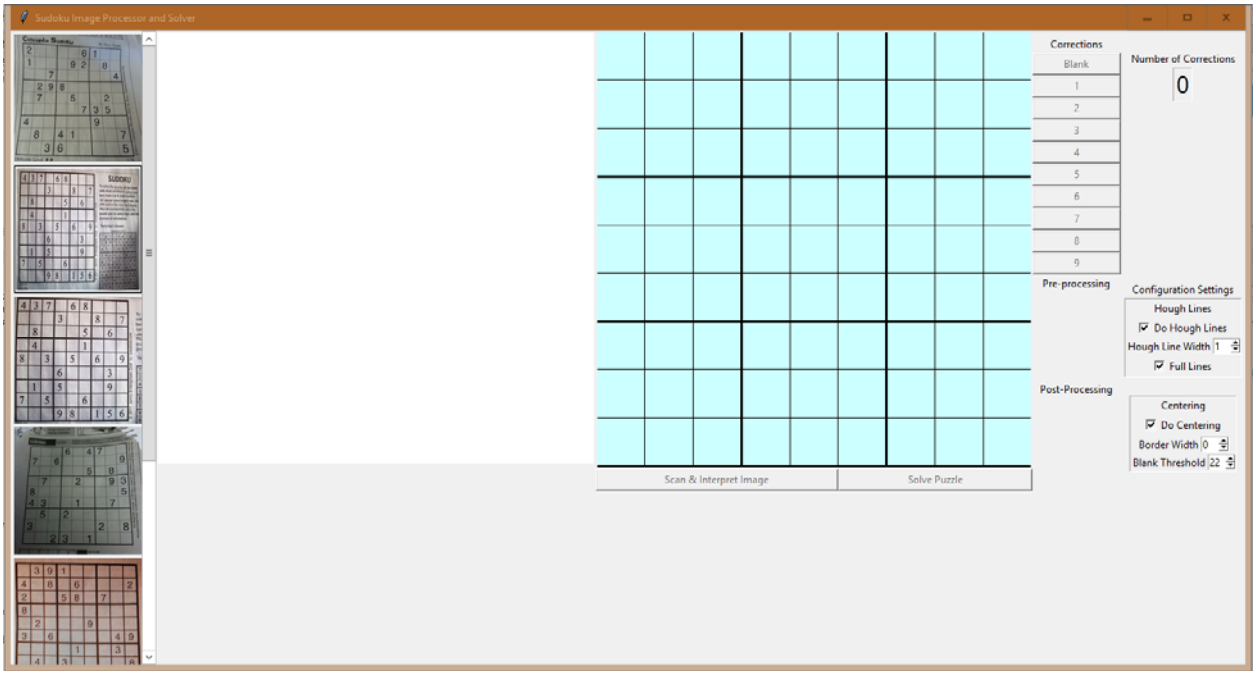
The image handling performs the following functions:

- Identifies the largest square or object in the image and assumes that this is the Sudoku board
- Attempts to make the largest square into an actual square by straightening out the image
- Optionally attempts to eliminate many of the lines from the image
- Divides the resulting square into a matrix of 9 by 9 smaller squares
- Based on configuration settings, cleans each image number by removing noise, lines that stretch across the full image, items that are smaller than a number would be removed, and center the final image both horizontally and vertically
- Use the CNN model to identify the number represented by each of the 81 squares. This could be a blank or a valid number 1-9.

The Sudoku solver was an algorithm found on Stack Overflow by an anonymous user that uses a simple backtracking algorithm to solve the puzzle. Comprehensive testing proved this code to be very accurate and very robust.

B. Sudoku Application – Functional Details

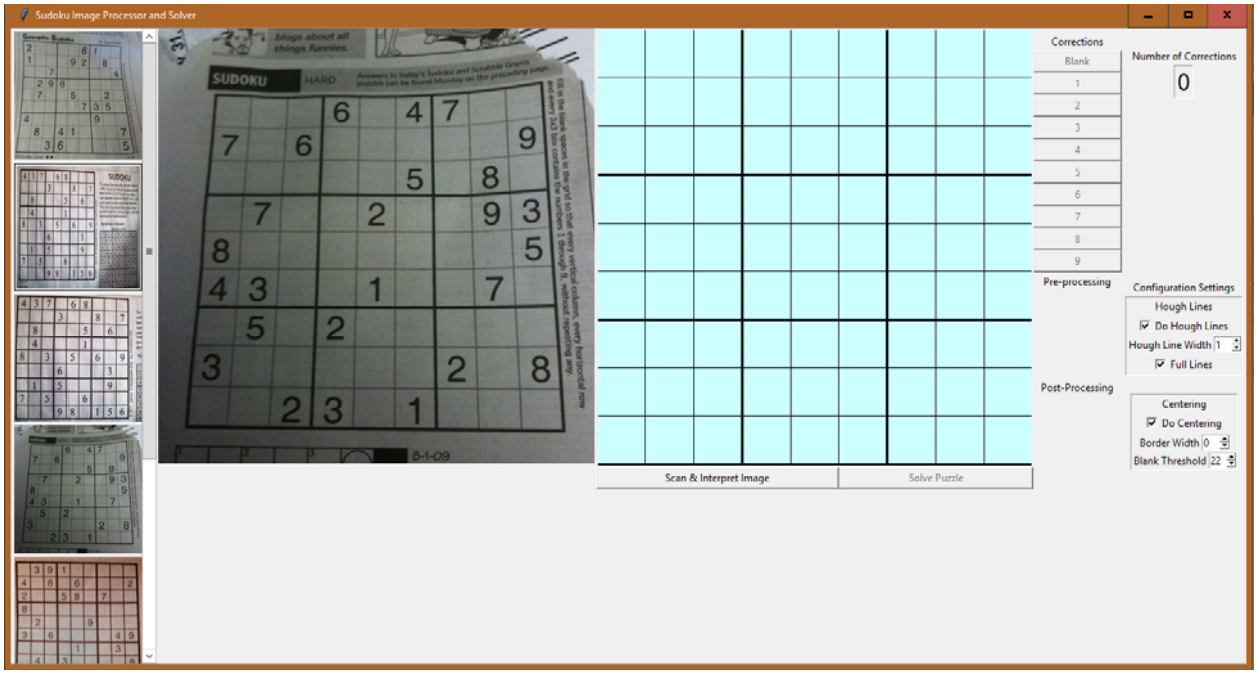
The application searches for all jpg images in the same directory as the program. It also pulls in the CNN model from this directory. The application displays the available images in a scrollable frame on the left of the window.



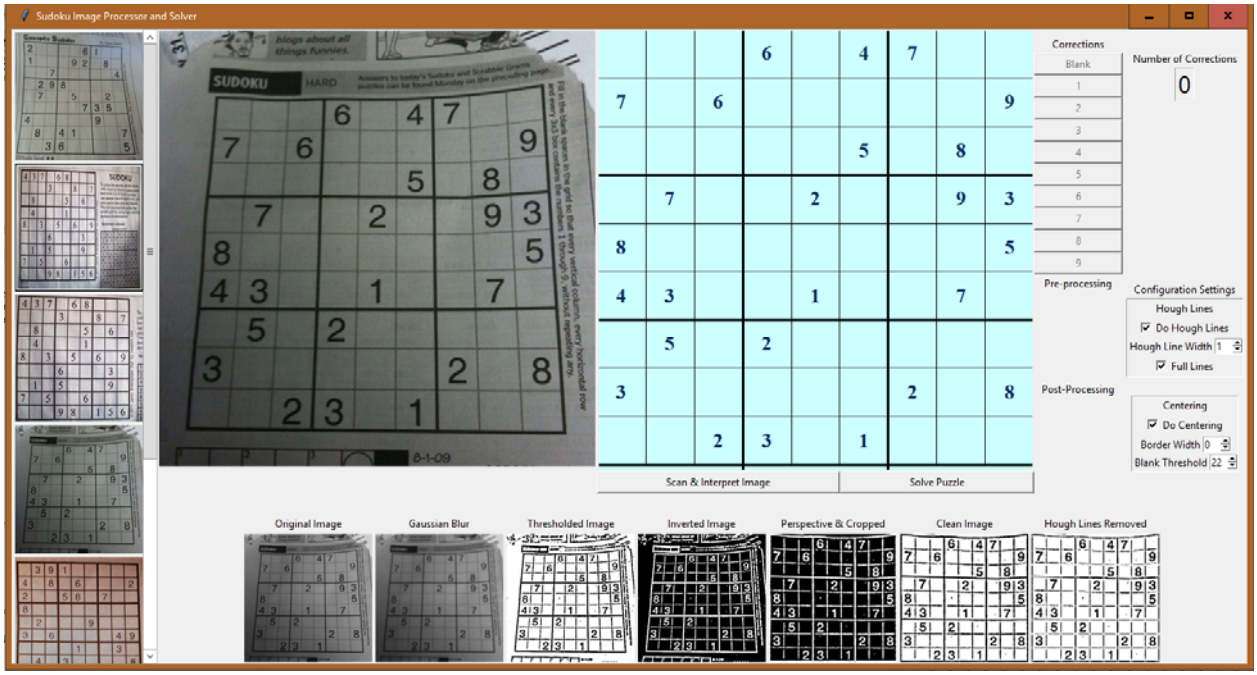
On the right of the window are the configuration settings that will be used for extracting the image. There are 2 main sections to the configurations:

- Hough Lines
  - There is the option to remove lines using Hough computation or not
  - If the lines are to be removed the following options are allowed:
    - The width of the line to be replaced – default = 1
    - Whether to use full lines or partial lines. The computation identifies multiple lines. If using the full lines option, the program identifies the start and end for each horizontal or vertical line and replaces everything in between
- Centering
  - Centering deals with the individual number images. There is an option to not do centering in which case the number image is left alone
  - If centering is selected, the numbers will be centered both horizontally and vertically once the following cleanup has been performed
    - Added a border of blank characters based on the border width setting
    - Deleted lines that have a nonblank object greater than the blank threshold value

Clicking on an image in the left-hand frame, selects that image and displays it in the larger box to the left of middle.

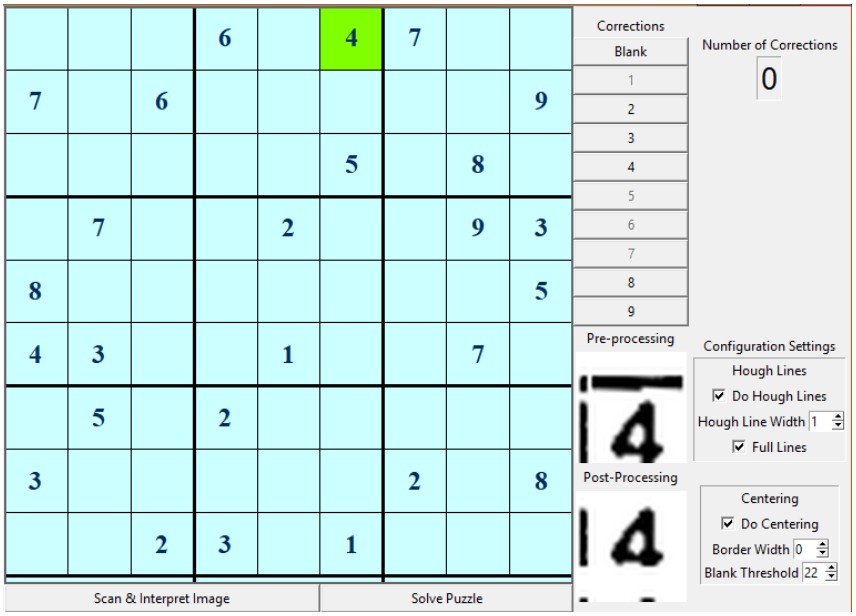


Clicking the Scan & Interpret Image button results in the image being analyzed and the results posted in the blue grid. The intermediate images used during the image processing are displayed in the bottom frame. This proved useful in identifying situations where the image processing failed to correctly identify the Sudoku board.

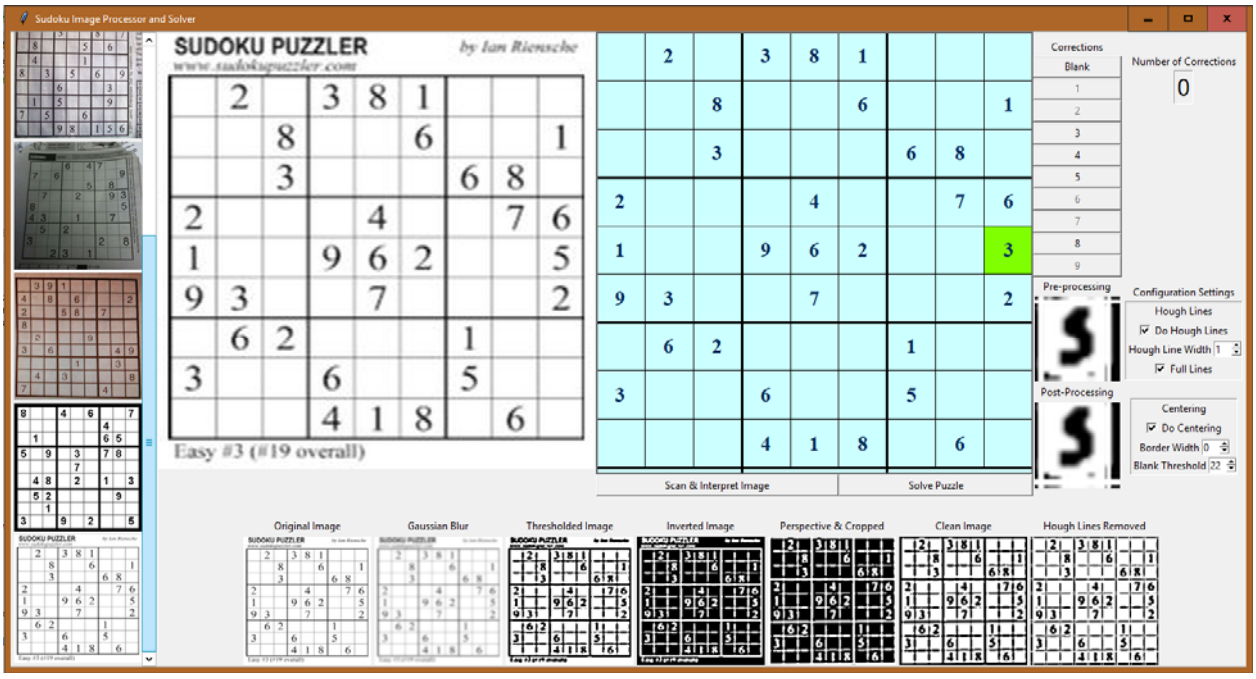


Clicking on a square in the blue grid results in two images being displayed:

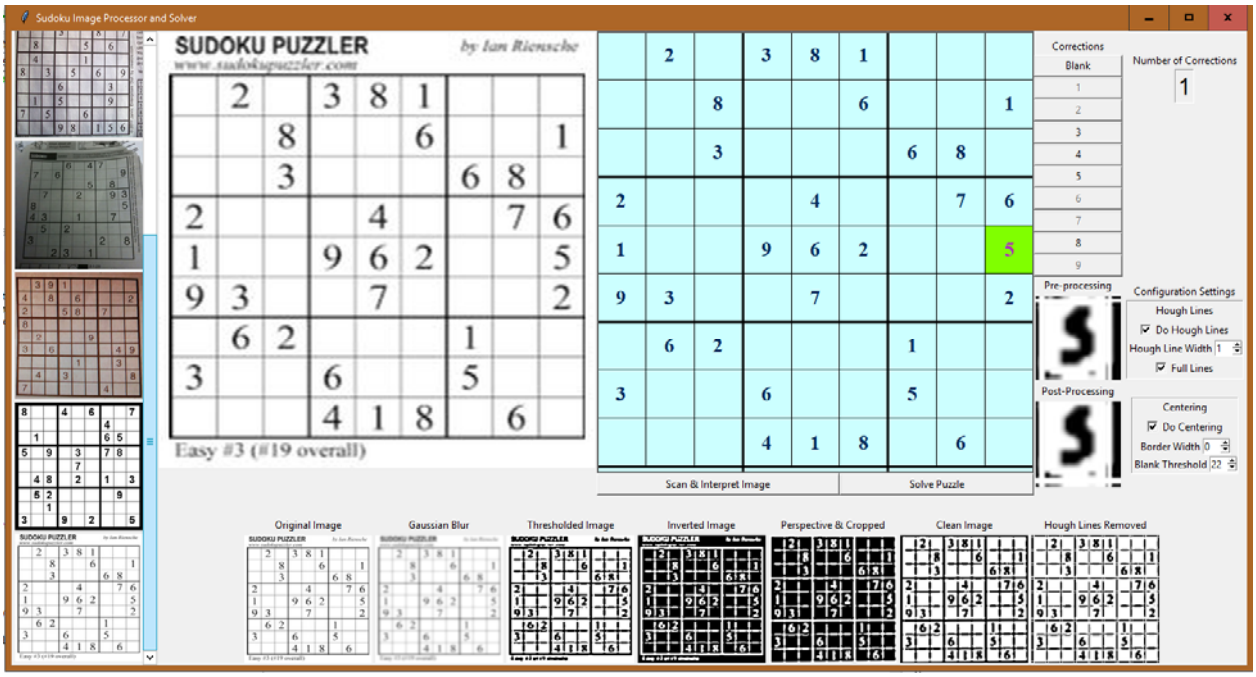
- Pre-Processing – the number image without the centering processing being applied
- Post-processing – the number image with the centering processing applied



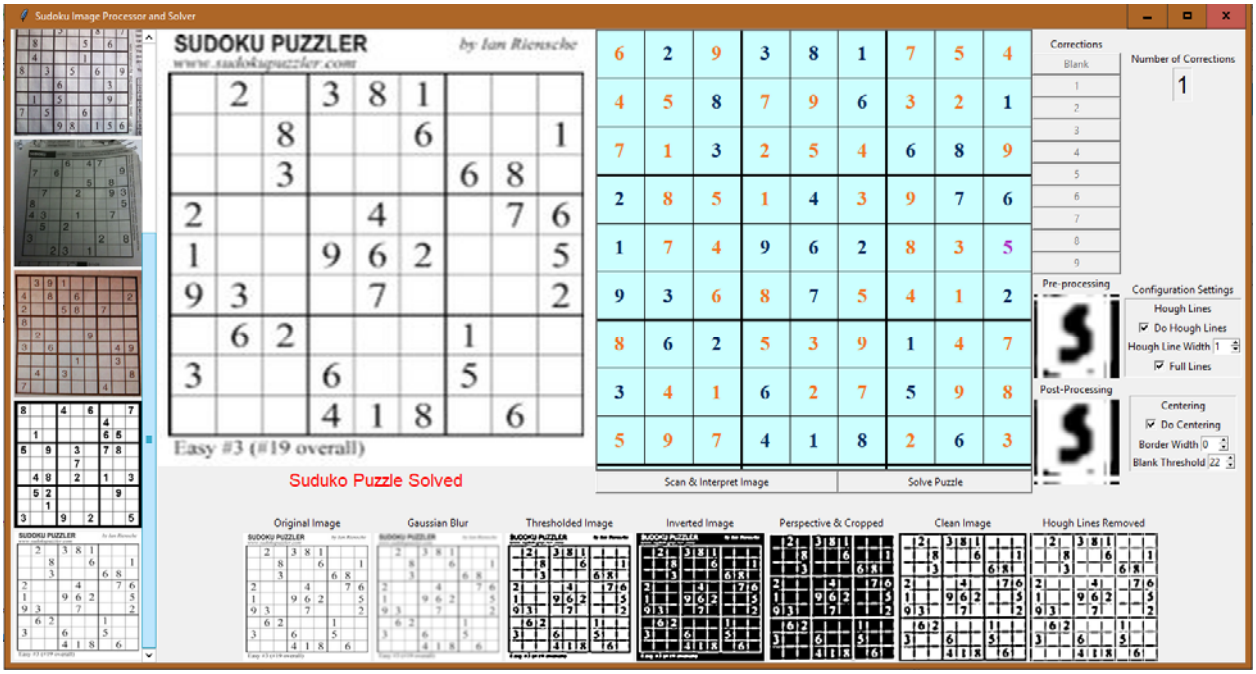
If the number is incorrect, as shown in the image below, the number can be corrected using the number buttons in the Corrections section of the screen. Only the numbers that are valid based on Sudoku rules can be selected. Note, it is possible to select the blank squares to and manually solve the Sudoku puzzle. The number of corrections made is displayed in the upper right of the window.



In the puzzle above, the 5 has been incorrectly interpreted as a 3. Clicking the 5 Correction button changes the value in the puzzle.



Once the misinterpreted numbers have been corrected, clicking the Solve Puzzle button will pass the current numbers into the Sudoku solver and display the results.



The color scheme for the numbers in the blue grid are:

- UTSA Blue – originally interpreted number from the grid
- Purple – manually corrected number
- UTSA Orange – number returned from the Sudoku solver

If the puzzle cannot be solved, the message “This puzzle cannot be solved” is displayed in red under the large image of the board.

V. Challenges

Many new tools had to be adopted and learned in order to complete this project. OpenCV is an extremely useful tool, but it is not without its learning curve. The "tkinter" package was most aptly described as being "painful" by those working with it. Even the neural networks seem to be more of an art than a science.

The seemingly simple task of converting a sudoku image into a useful numpy array was a project in itself. Recognizing what comprised the square and identifying smaller sub-squares was challenging, but even more so was taking an image that was not shot directly above the sudoku puzzle and converting it to look like it was. In other words, taking a trapezoid and converting it to a square.

The most time-consuming issue was the issue of the training set not matching the actual sudoku numbers being run through the model. As stated before, the script for pulling individual numbers from the sudoku puzzle was keeping the edges of the squares that surrounded the numbers. This caused to the MNIST dataset, which is very clean, to be a very ineffective training set. Many dead ends were pursued for a time before an entirely new dataset was created to account for the edges.

Even with the model correcting more accurately, there were still some errors, and any error would make solving the sudoku puzzle impossible. A GUI was creating from the python package "tkinter" that allowed the predicted values numbers to be verified and changes if needed. Once numbers were verified and modified as needed, the sudoku solver script was executed and the puzzle was completed.

VI. Conclusion

The final result is a functioning application which can identify sudoku puzzle boards, interpret the numbers, then provide the solution to the puzzle. The accuracy of the model is 96.2% which is quite impressive considering the 23.1% from the original run using the MNIST dataset. The few remaining errors can now be corrected by the user in

the interface prior to initiating the solver. The biggest lesson learned from this project was that rather than trying to just fix the dataset or the input, it is best to work from both ends in to the solution.

For future improvements, it would be ideal to allow the user corrections to be saved to the dataset so that it continually gets "smarter" by adding new examples. A stretch goal would be to create an app for mobile devices which uses an AR interface so the user positions a square over the sudoku puzzle and marking the corners of the outer box. Not only would this be more fun for the user than uploading a picture, it would also help the accuracy of the app by taking away some of the guesswork to pinpointing the board and individual numbers. As long as there are sudoku puzzles, then there will be people looking for help to solve them.

REFERENCES

[1] T. Guo, J. Dong, H. Li, and Y. Gao, "Simple convolutional neural network on image classification," 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)(, 2017.

[2] N. Jmour, S. Zayen, and A. Abdelkrim, "Convolutional neural networks for image classification," 2018 International Conference on Advanced Systems and Electric Technologies (IC_ASET), 2018.

[3] J. Schmidhuber, "Deep learning in neural networks: An overview," Neural Networks, vol. 61, pp. 85–117, 2015.

[4] F. Wang, H. Liu, and J. Cheng, "Visualizing deep neural network by alternately image blurring and deblurring," Neural Networks, vol. 97, pp. 162–172, 2018.

[5] H. Zhan, S. Lyu, and Y. Lu, "Handwritten Digit String Recognition using Convolutional Neural Network," 2018 24th International Conference on Pattern Recognition (ICPR), 2018.