OCR Sudoku Solver

COMP 4102

April 18, 2020

Student #	Name	E-mail
101018032	Vineel Boddula	vineel.boddula@carleton.ca
101011084	Ashton Woollard	ashton.woollard@carleton.ca

Abstract

A OCR sudoku solver application is capable of taking raw images or video, capture an unsolved sudoku puzzle within the data, and solve it. Broken down into steps, the program finds a puzzle within an image, recognizes the digits within the puzzle, solves the puzzle, and then displays the solved solution. Typically, many users do sudoku puzzles to consume time while waiting in a lobby for an appointment or pass time during travel. Thus, most of the puzzles found during these situations are in magazines or newspapers. It is very hard for a user to input an np problem sudoku puzzle in an online solver to check for correct results or even hints when a user is stuck at a point. Thus, OCR sudoku solver is a good candidate for being placed into a vision application.

Introduction

Our OCR sudoku solver application finds a puzzle within an image, recognizes the digits within the puzzle, solves the puzzle, and then displays the solved solution. Most of the project was worked on by both parties, but Ashton worked primarily on detecting puzzle within image, transforming/un-transforming puzzle, and OCR. While Vineel worked on the sudoku solving and print results to the image.

Originally, we planned to solve puzzles and display the results in Augmented Reality, where the results are displayed over a live video stream. The challenge with this is that there is a limited amount of training data we could put together. With our limited training data, we need a very high resolution original image in order to detect digits. We found an original image with roughly around 4k pixel quality was necessary for a high accuracy reading. Our webcams on our laptops are far from 4k quality and we could not continue testing in a live video environment. We instead use still images to process our results. Our project has been built with a video in mind, and thus should work with minimal code changes for processing live video. We accomplish this by waiting until we find a puzzle within the frame, capture the puzzle, OCR the puzzle using pytesseract trained on our custom training data, solve the puzzle using basicsudoku

(BasicSudoku, n.d.), and then cache the results to display each frame over the live video. However, as a result of removing live video processing, we decided to remove live AR results and moved on to instead outputting the results onto the image as regular text.

Background

There are many sudoku solver applications that exist in various forms. One that uses a similar approach to ours can be found in the publication of sudokugrab (Chris, 2009), which uses many of the principles and knowledge learned in COMP 4102. Since the objective of this assignment is related to the field of computer vision, and sudoku solving is obviously an NP problem, we instead opted to use public library basicsudoku (BasicSudoku, n.d.) to solve our sudoku solutions. The library we use for solving the puzzle is not particularly fast, but since we had to cut live video solving, this library is good enough. Implementation of the application is done using the sudoku solving algorithm in Python 3.7 using the OpenCV library for detecting the puzzle, normalizing the image, identifying all the numbers in the puzzle and displaying solutions on the image, as well as the aforementioned basicsudoku library, and pytesseract library.

Approach

For our approach we follow a 4 step approach. First, we find a puzzle within an image. Second, we run an OCR on the puzzle to find the given digits in the unsolved solution. Third, we solve the solution. And finally, we print the solved solution back onto the original image for viewing.

First, the program gets an image from the user input or video stream and then processes the image in order to find the puzzle within the image. This is done by using a similar process to edge detection. Using the built-in OpenCV functions, we perform a gaussian blur for noise smoothing, and edge enhancement and localization with adaptive threshold. After we process the image, we find the contours within the image (OpenCV term meaning enclosed shape within the image) and sort the contours by size. We assume the largest enclosed shape within the image is our puzzle. At this point we need to calculate the perspective transform that transforms the potentially skewed puzzle into a square image and transform the image. At the same time, we cache the perspective transform that undoes this transformation for later.

After finding the puzzle within an image, the next step is to read the data from the image. To do this first we divide each section of the puzzle into its own individual cell. The reason for using this approach is because pytesseract is very limited in its ability to read digits, and as such we must reduce noise and unexpected artifacts as much as possible. To minimize errors, the decision was made to perform an OCR on every cell of the image individually, while much slower, this method is much more accurate, which is extremely important for reasons that will be explained later. The OCR process is first done by using a simple blur and thresholding to reduce noise, then we use pytesseract (pytesseract, n.d.) along with our custom training data to detect what each digit is. Among high resolution images using the same font we trained with, we can almost always find the digits provided. The only circumstances we fail to find the digits is when the puzzle image plane is massively skewed compared to the camera, or the image is not of a sufficient resolution.

After reading the digits from the image, the next step is solving the puzzle. Since Sudoku is obviously an NP problem, and studying NP problems is not the point of this class, we instead opted to use a public library to solve our sudoku solutions. It is not particularly fast, but since we cut live video solving, this library is good enough. First we check if the digits read from the image are valid to use for solving the puzzle. We do this by checking that all columns, rows, and boxes have only digits 1 through 9, with no repetition of any digits.

Finally we take the solved sudoku results, and print the results on to the puzzle image. Before un-warping the image using the perspective transform from earlier, and overlaying the solved puzzle on the original image.

Results

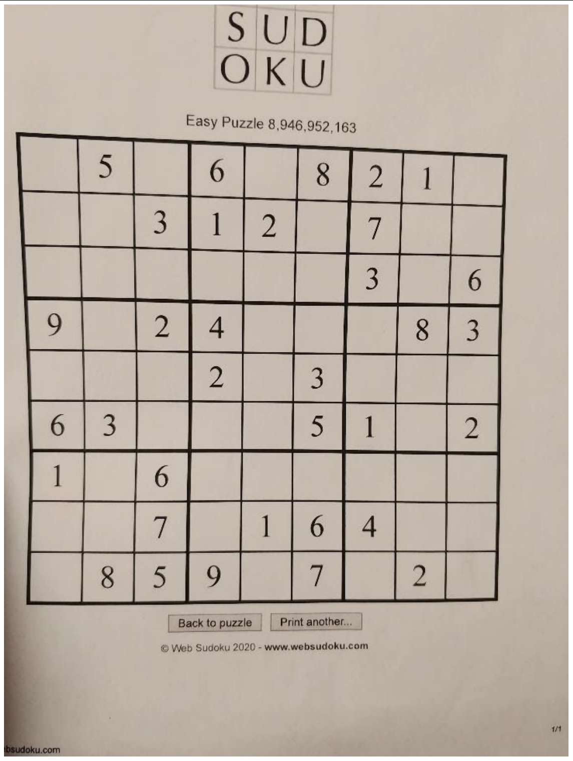
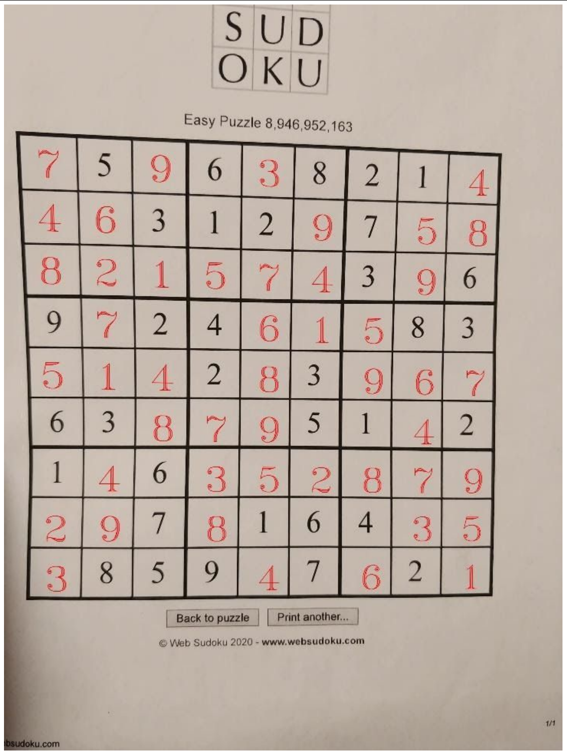
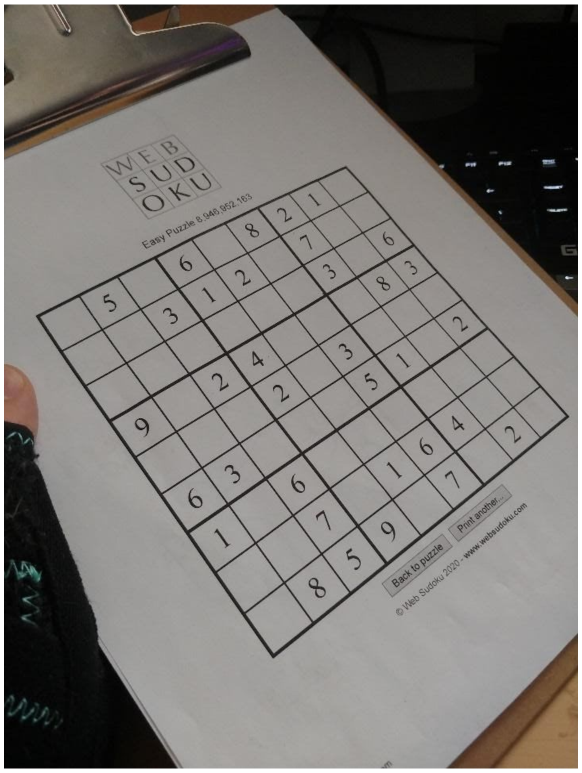
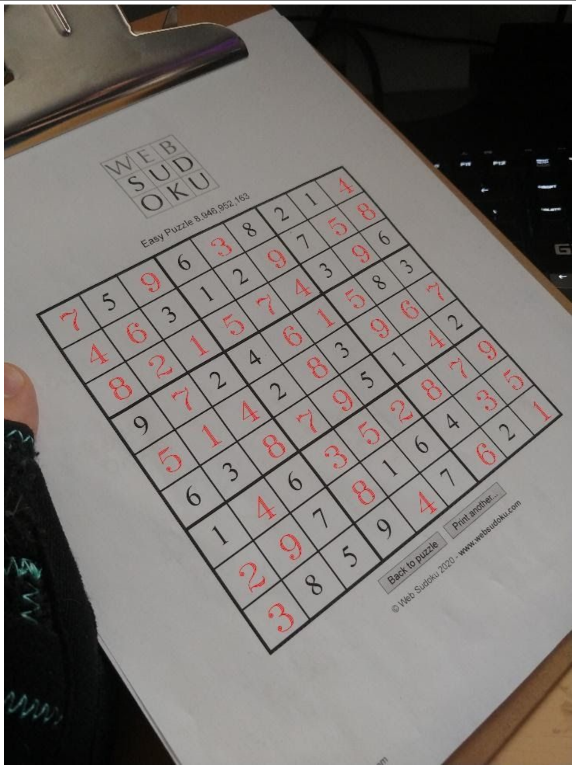
When processing high resolution images (~4k), our success rate is very high under normal use conditions. However two primary failure cases present themselves under specific criteria.

Our training data's sample size is relatively small compared to what a fully trained data would use. And in order to solve a puzzle, we need to have a 100% success rate when reading digits from the original puzzle. If a single digit is missed or misread, the puzzle has a very high likelihood of becoming invalid. In order to accurately read every digit without ever misreading, we need very high-resolution images (~4k). If an image with insufficient resolution is used, the program will fail to read every digit, and will then fail to find a solution to the puzzle.

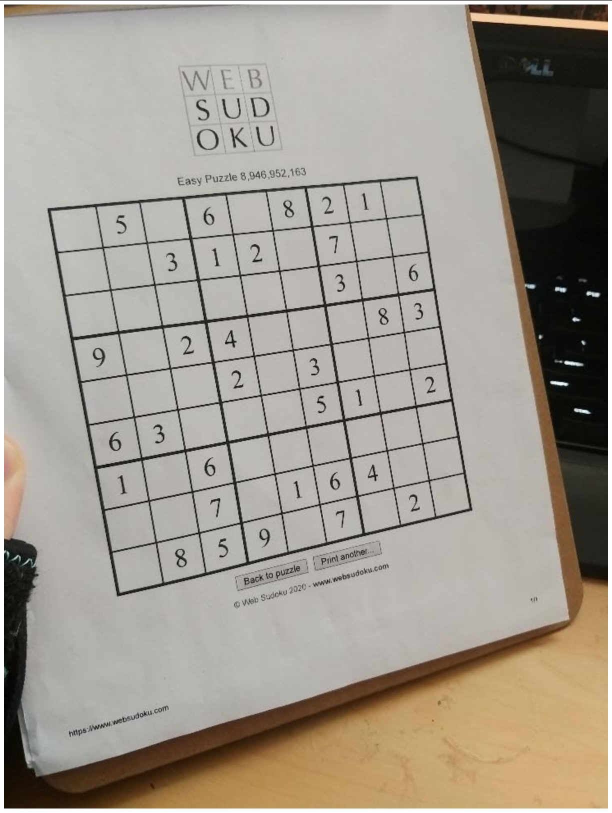
The second failure case presents itself when the puzzle is imaged from an extreme angle (more than 45 degrees at the resolution we test at). Our first step in processing the image, is to find the puzzle within the image and un-skew the puzzle so that it is square. If the original image is sufficiently warped, then the processed puzzle image will have too little resolution, and we will fail to find every single digit accurately under the same conditions as above.

Below is Table 1: Original Images and Processed Image Results, shows samples of the input image and the processed image. Note that due to ongoing pandemic and lack of printers at our home residences, we could not access more physically printed puzzles to take high resolution images of (Images found online and images of a screen were both insufficient). Thus, all the provided images are of a single puzzle, captured from a variety of angles.

Table 1: Original Images and Processed Image Results

SUDOKU
Easy Puzzle 8,946,952,163
Back to puzzle
Print another...
© Web Sudoku 2020 - www.websudoku.com



WEB SUDOKU

Easy Puzzle 8,946,952,163

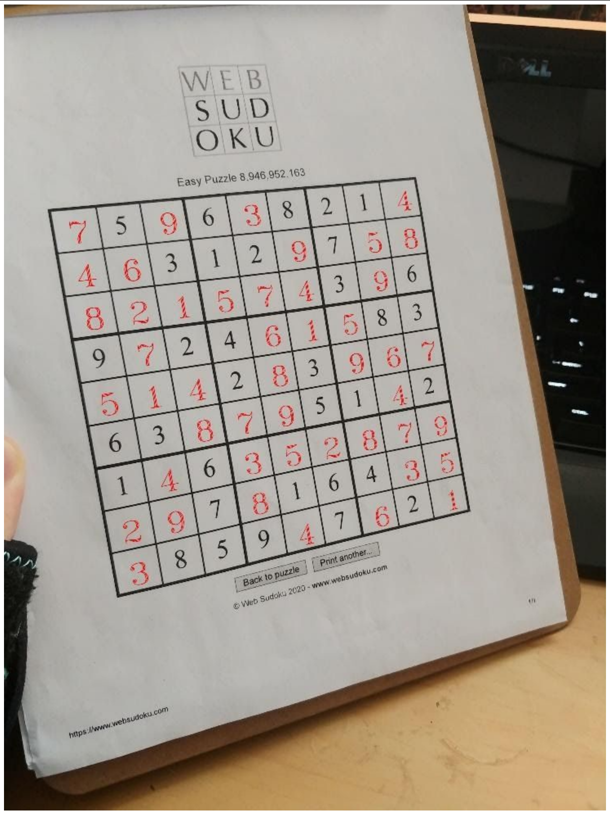
	5		6		8	2	1	
		3	1	2		7		
						3		6
9		2	4				8	3
			2		3			
					5	1		2
6	3							
1		6			1	6	4	
		7			7		2	
	8	5	9					

Back to puzzle Print another...

© Web Sudoku 2020 - www.websudoku.com

1/1

https://www.websudoku.com



WEB SUDOKU

Easy Puzzle 8,946,952,163

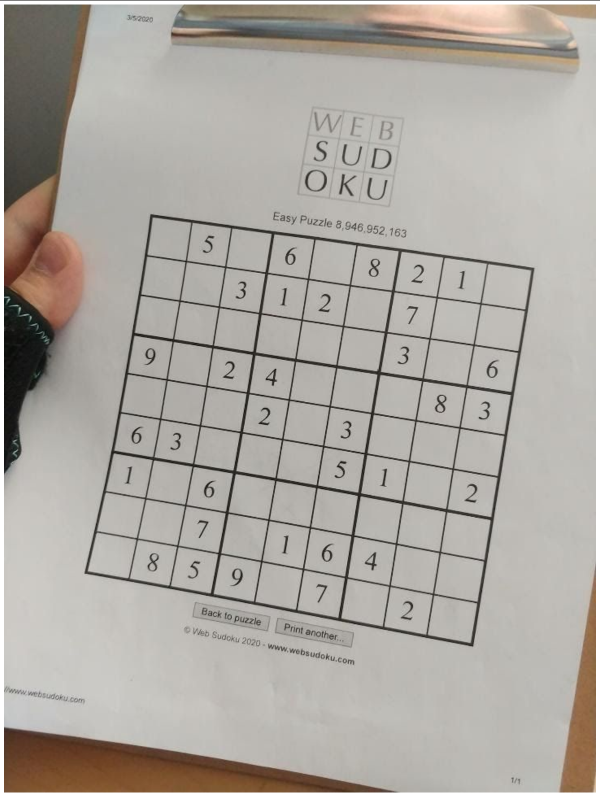
7	5	9	6	3	8	2	1	4
4	6	3	1	2	9	7	5	8
8	2	1	5	7	4	3	9	6
9	7	2	4	6	1	5	8	3
5	1	4	2	8	3	9	6	7
6	3	8	7	9	5	1	4	2
1	4	6	3	5	2	8	7	9
2	9	7	8	1	6	4	3	5
3	8	5	9	4	7	6	2	1

Back to puzzle Print another...

© Web Sudoku 2020 - www.websudoku.com

1/1

https://www.websudoku.com



3/5/2020

WEB SUDOKU

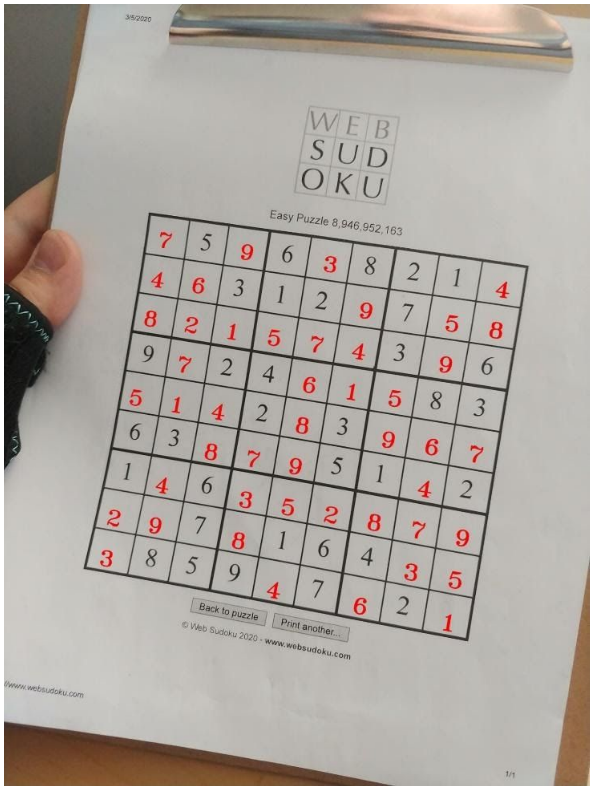
Easy Puzzle 8,946,952,163

Back to puzzle Print another...

© Web Sudoku 2020 - www.websudoku.com

//www.websudoku.com

1/1



3/5/2020

WEB SUDOKU

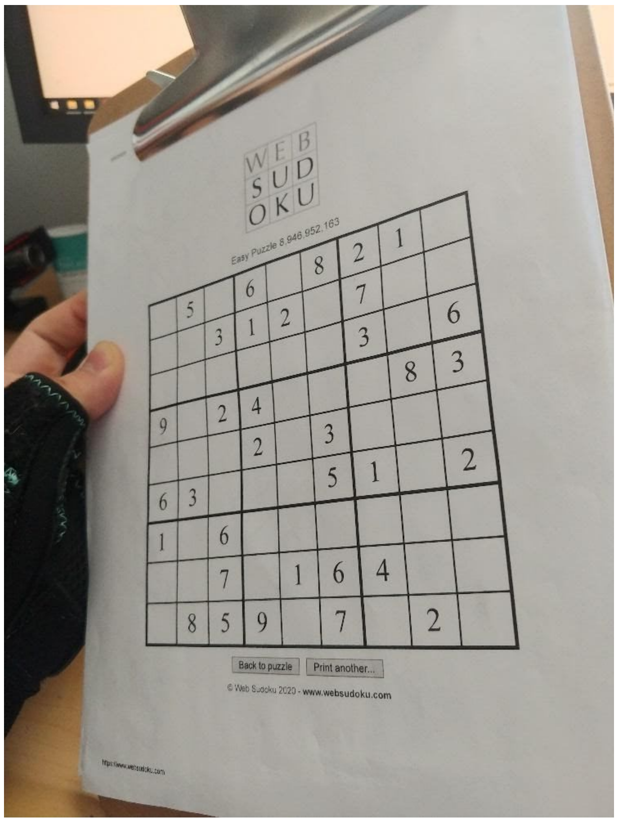
Easy Puzzle 8,946,952,163

Back to puzzle Print another...

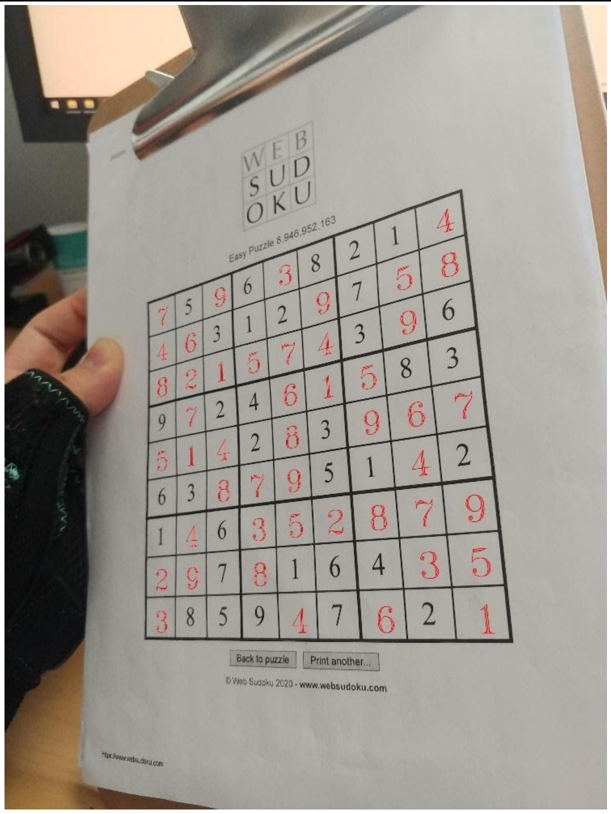
© Web Sudoku 2020 - www.websudoku.com

//www.websudoku.com

1/1



WEB SUDOKU
Easy Puzzle 8,946,952,163
Back to puzzle
Print another...
© Web Sudoku 2020 - www.websudoku.com

 | 

WEB SUDOKU
Easy Puzzle 8,946,952,163
Back to puzzle
Print another...
© Web Sudoku 2020 - www.websudoku.com

 |

List of Work

This application can be broken down into 4 distinct deliverables. A calendar for the deliverables is included in Table 2: Schedule of Deliverables.

The first deliverable was a program that takes a live video of a square, normalizes the perspective, visually modifies the shape (eg. drawing an X on the shape), then 'un-normalizes' the perspective. This is a simple augmented reality program.

The next deliverable was a program that can take a static image of a sudoku puzzle, detect the 'puzzle square', run an OCR on the image, and was then able to format the data to feed to a sudoku solving algorithm.

The third deliverable was a program that took a sudoku solution, parsed it, and displayed the solution over a static sudoku image.

The fourth deliverable implemented the 2nd and 3rd deliverable, combined with the perspective normalization from the 1st deliverable, and was capable of taking a static sudoku image from a variety of perspectives, solve it, and then display the solution over the original image.

Table 2: Schedule of Deliverables

Task	Completed by
Installing library OpenCV and getting better understanding of the research publication.	Ashton and Vineel
Deliverable 1, a program that takes a live video of a square, normalizes the perspective	Vineel
Deliverable 2A, using threshold, OCR to detect edges/shape of the puzzle. (Borders of the puzzle)	Ashton
Deliverable 2B, using OCR to identify numbers.	Vineel
Deliverable 3, implementing sudoku solver algorithm for getting a solution to the problem.	Vineel
Deliverable 4, putting deliverable 1, 2 and 3 together.	Ashton
Deliverable 4, displaying the solution over a static image.	Ashton
Testing and final touches	Ashton and Vineel
Working on presentation and demo	Ashton and Vineel

GitHub Page

Link to public Repo: https://github.com/krashton1/Sudoku_OCR_Solver

References

BasicSudoku. “Basicsudoku.” n.d., *PyPI*, pypi.org/project/basicsudoku/.

Chris. “How Does It All Work?” *How Does It All Work?*, 19 July 2009, sudokugrab.blogspot.com/2009/07/how-does-it-all-work.html.

pytesseract. “Pytesseract.”n.d., *PyPI*, pypi.org/project/pytesseract/.