An End-to-End System to Solve Sudoku Puzzles

Elliott Jobson
Stanford University
emjobson@stanford.edu

Andres Hernandez
Stanford University
andresh@stanford.edu

Connor Normand
Stanford University
cnormand@stanford.edu

Abstract

Sudoku is a popular math puzzle where each column, each row, and each 3 x 3 grid in a larger 9 x 9 grid must contain all numbers 1 through 9 once. In this report we will describe our process developing an end-to-end system that receives, as input, an image of a sudoku puzzle captured by a digital camera from some angle and solves the puzzle. Our overall approach to the problem involves tackling each step with robustness and efficiency as priorities. We plan for the system to be able to handle variations in lighting, camera angles, fonts, and states of completion.

1. Introduction

We divide the problem pipeline into three main sections: board recognition and manipulation, digit recognition, and game logic for the solution. At a high level, board recognition and manipulation involves converting an image of the Sudoku puzzle into a format that is usable by the program. After manipulating the board and extracting the various cells and digits, we apply a classifier to determine the digit values. Finally, we use a recursive depth-first search strategy to either solve the board, or determine that no solution is possible (this may occur if the user inputs a partially-solved board).

2. Background and Related Work

For a long time now, puzzle solving has been a novel way to test the intelligence of computer systems and software. Several approaches have been taken to solve this problem, in particular, as well [1, 2, 3].

Differences arise at many stages in the traditional pipeline, and they begin with the thresholding method chosen. We found that one of the most popular forms of image thresholding is Otsu's Method, which is implemented using OpenCV in [4]; however, we elected to use the same library's Adaptive Threshold technique to binarize the image instead for reasons described below[6].

Rectifying the image is a crucial step to ensure that the extraction of cells is consistently accurate. Many implementations of end-to-end puzzle solvers rectify by extracting and multiplying by a 3x3 transformation derived from the 4 corners of the largest contour found in the image and the 4 new desired corner coordinates [4, 7].

One major assumption made by this method is that puzzle in question lies flat on a table or smooth surface-- in other words, that it isn't wrinkled or otherwise distorted itself.

2.1 Image manipulation

The challenge for the puzzle manipulation section is to build a system that can handle variation in the quality of the images that are presented to it. Because most Sudoku puzzles look fairly similar, we will need to find photos to test varying kinds of conditions.

Our model will be able to handle non-ideal conditions in terms of lighting, camera angles, and orientations. In order to test for these qualifications, we printed an assortment of Sudoku boards, and took photos of them under a variety of conditions.

We test our process on a number of different Sudoku images that we deem reasonable, printing out their results at all steps of the board manipulation pipeline for our own visual verification. Upon achieving a successful implementation of this step, we will further experiment with ways to improve the robustness of our manipulation process. Later in the paper, we will review and evaluate various computer vision techniques used at different steps in our pipeline.

2.2 Digit recognition

A number of methods have been used to build classifiers for digit recognition. For example, a simple K-nearest neighbors classifier can achieve an error rate of around 2.4% on the MNIST database of handwritten digits by operating directly on raw pixels – without even considering image features. [9] Although the implementation of such classifiers is relatively simple, the classifier must compute the Euclidean distance between the input image and all training images at test time. Therefore, the test time memory and computation requirements are relatively large. [8]

In the same study, a large fully-connected network, consisting of alternating linear and nonlinear functions, achieved an error rate of 1.6% on the MNIST. A downside of using large fully-connected networks for digit classification, however, is the sheer number of trainable parameters. Because the network initially operates directly on the raw pixels, a huge number of parameters are introduced in each of the weight matrices.[9]

Lastly, in the same study, a large convolutional network called LeNet5 achieved an error rate of under 1%, using only 60,000 trainable parameters. By comparison, the deep fully-connected network used in the same study had 123,300 trainable parameters. In general, convolutional neural networks have enjoyed a high level of success when applied to image-related tasks, as the convolution operation can effectively detect features at various levels. The first filters detect low level features in the networks, while later convolutions detect increasingly higher level features. Though the convolution operation is more computationally-expensive than fully-connected operations, CNNs are still efficient at test-time. [10]

2.3 Sudoku Solution

Given the recognized values of an unsolved or partially-completed puzzle, we use a depth-first search recursive strategy to solve the puzzle.

3. Approach

As stated previously, our first problem is to convert the image of the Sudoku board into a format that is recognizable by the computer. Overall, this involves binarizing the image, detecting its corners, correcting the perspective, and classifying cells as blank or filled. In order to carry out this section of the pipeline, we utilize the OpenCV Python library, which contains a number of Python wrappers for C++ functions for computer vision applications.

3.1 Board manipulation

3.1.1 Image binarization.

The first step in our pipeline involves binarizing the image. In order to reduce noise, we apply a Gaussian blur to the image. This acts as a low pass filter for image information. Gaussian blur is effective at removing noise, such as the noise that may occur within cells. [11]

As mentioned above, Otsu's method for applying image thresholding is used a substantial amount in applications like this one; however we found that adaptive thresholding techniques are more robust to varied input, specifically varied lighting conditions. OpenCV's adaptive thresholding algorithm opts to choose a different binarization threshold for different regions of an image rather than a global value as implemented in Otsu's method. Finally we dilate one more time. [12]

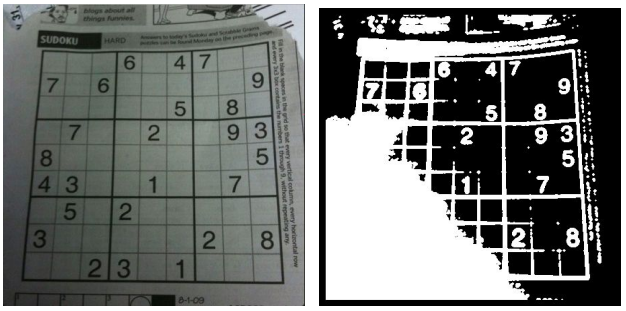

Figure 1: Otsu's thresholding method in non-ideal lighting

3.1.2 Corner detection

The next step of the pipeline is to determine the corners of the grid. To do so, we use OpenCV's findContours function to find the largest contour in the binarized image. We assume that in each image the largest box shape will be that of the puzzle's bounding box. The corners on the contour are defined by minimums and maximums of the sums and differences of the of the 'x' and 'y' coordinates. [13, 14]

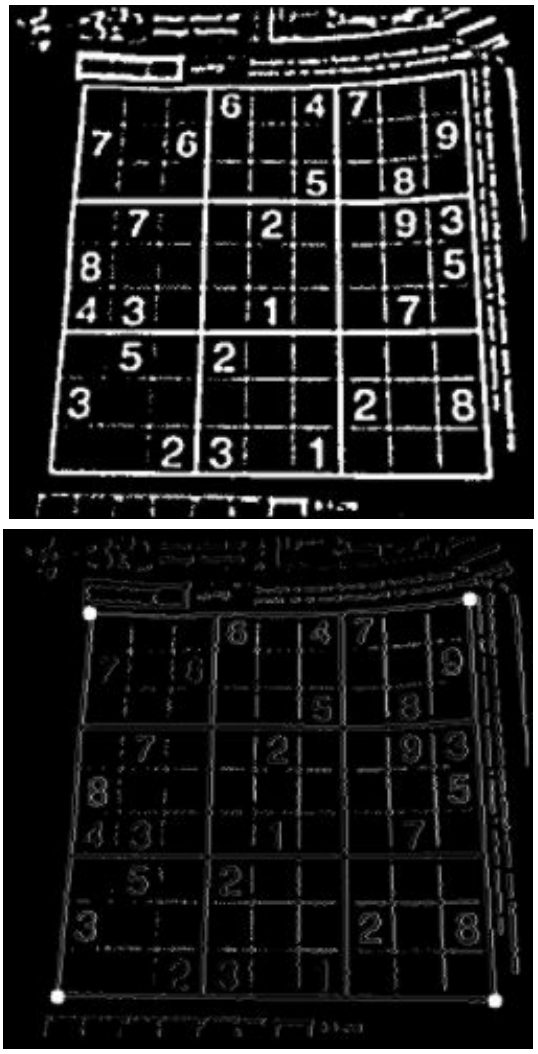

Figure 2: Corner detection

Concretely, the contour coordinates are organized in a tall, Nx2 matrix P, where the first column defines the x-coordinate and the second column defines the y-coordinate. Two more column vectors can be created by the sum and difference of the columns of P. The maximum value of the sum vector yields the index of the bottom right corner's coordinates. The minimum value of the sum vector yields the top left corner. The maximum value of the difference vector yields the top right corner, and finally the bottom left remains. Note, this step is highly dependent on the successful extraction of the longest contour, which is why dilation is a key step in the previous section of this pipeline. [15]

3.1.3 Perspective correction

Next, we transform our image to be "flat." This can be done by finding some matrix A that transforms, rotates, and scales our image into the dimensions that we desire. We require 4 correspondences (the corners of the puzzle's bounding box) to map all of the points in the image to a new perspective defined by the maximum length. This is found by finding the maximum distance between two corners. Using OpenCV, we achieve this by calling the functions getPerspectiveTransform and warpPerspective. [16]

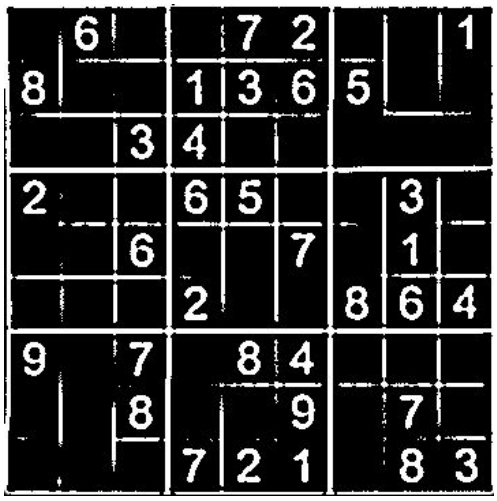

Figure 3: Rectified Image

3.1.4 Cell classification and division

After applying the transformation to our binarized image, we must classify the cells as blank or filled. In this section of the pipeline, we divide the rectified image of the board into its 9 by 9 cells. Then, we determine whether or not there is a digit in the cell – if there's a digit in the cell, we pass it to the digit classifier.

To divide the image into its cells, we simply divide the rectified image by area, since we know that each cell on the real puzzle is the same size. In order to remove the lines that separate cells from our divided cells, we crop the cells slightly.

Finally, we must determine whether or not the cell contains a number. In order to do so, we apply a Harris corner detector to the cropped cell. Cells that contain a threshold number of corners contain a digit, so they are passed to the digit classifier. [15, 17]

3.2 Digit classification

For the task of digit classification, we use a convolutional neural network. As discussed earlier, we considered a number of other methods for designing our

classifier. Though KNN classifiers are simple to implement and have achieved good results on classifying both handwritten and printed digits, they require a lot of memory and are computationally-expensive at test time. Because we are aiming for both robustness and efficiency, we looked for other classification methods. [18, 19]

As mentioned earlier, CNNs have achieved a high level of success when applied to image classification tasks due to their efficiency at runtime and ability to extract image features at various levels. The latter ability helps with generalization to unseen data, whereas classifiers such as KNNs may struggle when applied to unseen data that greatly differs from the training data.[18, 19]

Aiming to achieve high classification accuracy by starting with a simpler model and adjusting as necessary, we trained our first model on the MNIST using a 5-layer CNN [cite MNIST]. As we will discuss later in the experiment section, this model achieved a classification accuracy high enough that any improvements would be marginal, so we did not experiment with other models. Our model accepts batches of grayscale images of shape 28x28 and has the following architecture:[9]

2 Convolutional Layers: Both convolutional layers consist of the convolutional filter operations followed by a ReLU non-linearity and a max-pooling operation. The first convolution consists of 30 (5,5) filters, producing an output of shape (30, 24, 24). The ReLU nonlinearity maintains the output shape and helps separate the data, while the max-pooling layer, with stride 2 and shape (2,2), downsizes the input shape by a factor of 2. After the first layer, we have shape (30, 12, 12). The second convolutional layer is similar, except that the filter size is (3,3). After the second convolution, ReLU, and max-pooling, we have an output of shape (15, 5, 5).[20]

Dropout: At training time, we randomly set 20% of the input units to 0. Dropout can be thought of us a form of regularization, as it helps prevent the model from overfitting to the training data. One can also think of dropout as similar to ensembling within a single model, as it forces the model to learn a number of different features. [21]

3 Fully-Connected Layers: Finally, we have three fully-connected layers – the output of the last layer produces raw scores for each digit class. We then use a softmax classifier to produce a normalized probability distribution over our outputs. [22]

Before we used this model in our pipeline, however, we realized that the majority of digits that the model sees at test time would be printed rather than handwritten. Hypothesizing that training the model on both handwritten and printed digits would improve the generalizability of our model, we retrained the model on both MNIST and Chars74K. Overall, we used 60,000 28x28 images from both datasets as our training set and 10,000 28x28 images from both datasets as our test set. In the following section, we will evaluate the performance of the model.[23]

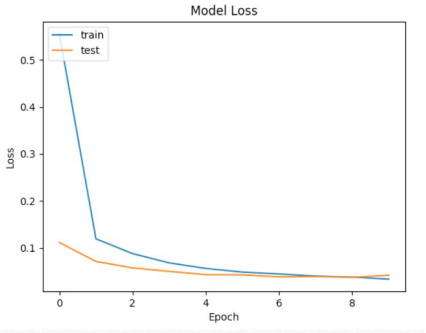

Figure 4: Loss of our model over 10 epochs

We wrote the model using Keras wrappers over a TensorFlow backend. We trained the model for 10 epochs on our local CPU. Training for only an hour on a CPU, the model was able to achieve high classification accuracy.

4. Experiment

Since we are creating an end-to-end system, we evaluate the model by gauging its effectiveness at each step. As described above, we have successfully made the pipeline robust to certain inputs, while it fails on other inputs. For the CNN, we are able to give a more quantitative evaluation. Our evaluation of the board manipulation sections of the pipeline is more qualitative, since we had control over the inputs and stats on the number of boards that the pipeline solves would be useless anyways.

For our board manipulation dataset, we used a number of printed Sudoku puzzles. We varied the lighting conditions, camera angle, and board style, ultimately achieving varied results depending on these conditions. In the following sections, we will analyze the strengths and weaknesses of our model.

4.1 Board Manipulation Evaluation

4.1.1 Image Binarization

As discussed above, Otsu's thresholding was not robust enough to varied input, so we chose to use an

adaptive thresholding algorithm. The Gaussian blur step introduces certain issues when applied to especially dark images. We briefly considered brightening darker images before applying the Gaussian kernel; however this solution would have be based on a heuristic rather than empirical properties of the images being manipulated. Furthermore, the parameters for a Gaussian blur that work for one image may not work for a similar, darker image. [6]

4.1.2 Corner Detection

By the method described above, corner detection is geometrically sound and completely reliable barring any secondary transformations made to the image, so long as the longest contour can be assumed to be the bounding box for the puzzle. [13, 14]

4.1.3 Image Rectification

While the rectification method used is effective on sudoku puzzles that lay flat, it is not robust to substantial warping of the page. To mitigate this, more than 4 corner correspondences should be used when calculating the transformation matrix. Several document unwarping strategies exist, however none that would have been useful for us in varied lighting situations. As can be observed in Figure 3, the lines on the sudoku puzzle do not always meet the adaptive threshold value, and simple parameter adjustment tends to bias toward favoring the sample in use at that time. Furthermore, most document warping strategies rely heavily on applying horizontal morphologies to small lines of text, not large boxed values. This makes it difficult to find a reference for horizontal and vertical lines. In fact, the only sure reference we have for x- and y-axes would be the bounding box of the puzzle. [7]

We made the following hypothesis: If the bounding box for the puzzle could be reliably extracted, then, instead of simply creating 4 corner point correspondences, we could map all points of the warped bounding box to evenly spaced points on a square, and then repurpose our method of finding a Fundamental Matrix for this transformation, as if we had a second camera perspective. [7]

Unfortunately, this task's success also proved to be heavily vulnerable to variability of the input. In several cases the largest contour, our bounding box for the puzzle, also included points on an inner sudoku box. This merging of contours was likely due to the dilation applied earlier, and could even exist without dilation if the picture is taken at a shallow enough angle. While this artifact of the bounding box extraction still would not interfere with the corner detection, determining what points the inner contours for various unseen inputs would map to increasingly became a task outside of the focus of this project. [24]

4.1.4 Cell and Digit Extraction

Our cell division achieved varying levels of success. As discussed in the previous section, we attempted to make the solver robust to images that were wrinkled or slightly warped. However, failing to do so, we were unable to properly isolate our cells for wrinkled puzzles. Cells extracted from wrinkled boards looked like the following:

Figure 5: Cell extracted from warped Sudoku board

On flat Sudoku boards, we successfully isolated the cells by dividing the board by area, then cropping the resulting cell to ensure that the there is no border line in the image. This gives us a much cleaner image:

Figure 6: Cell extracted from flat Sudoku board

Originally, our approach was to crop a small frame area of the picture and count the number of white pixels contained within; however, we found that all too often our warped cells would contain puzzle borders where we expected an value, and numbers such as 7 and 1 were often missed due to not having sufficient white pixels to pass the given threshold.

Opting to use the Harris corner detector was highly successful in determining whether or not a digit was present in the cell. For cells without digits, the detector usually reports that the number of corners is at or near 0. For cells with digits, the number of corners always exceeds 120. [15, 17]

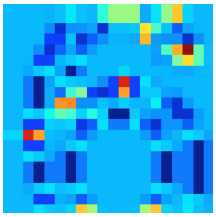

Figure 7: Corners detected by the Harris corner detector

Therefore, our function for determining whether or not a digit is present in the cell achieves near 100% accuracy.

4.2 Digit Classifier Evaluation

Ultimately, by training our model on both the MNIST and Chars74K dataset, we were able to achieve a low error rate on both the training and test sets. We reached a final test error of 1.01%, where the test set was a batch of 10,000 holdout images from both MNIST and Chars74K.[9, 23]

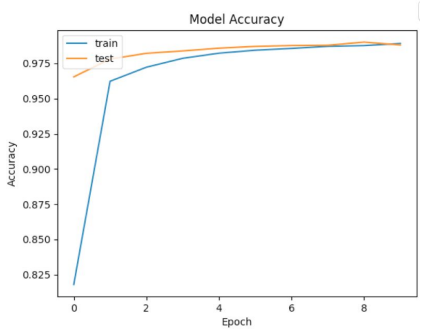

Figure 8: Model accuracy at each epoch

The last test of whether or not the digit classifier works occurred when it came time to test our pipeline. Though the classifier achieved a high test accuracy on the images pulled straight from the dataset, there were instances of it failing on certain Sudoku cells. However, the cells for which it would fail were always cells that contained an extra line because they contained part of a cell border line. For example, on cells with a line above the digit itself, the network would often classify the digit as a 7. Ultimately, we do not attribute these failures to the model.

4.3 Sudoku Game Logic Evaluation

The Sudoku solver itself can efficiently fill in the game board for any board with a solution. Using a depth-first search recursive strategy, it solves the puzzle in between .004 and .04 seconds.

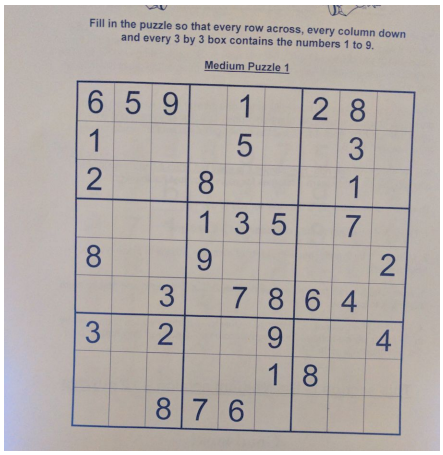

Figure 9: Starting puzzle

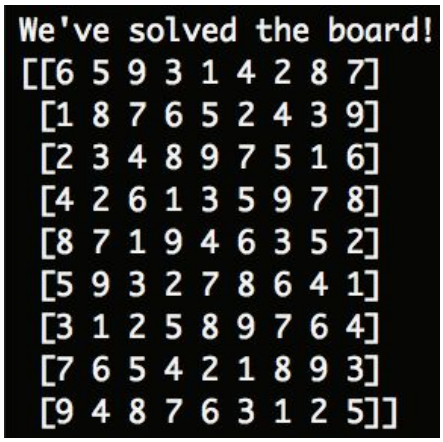

Figure 10: Solved board

5. Conclusion

The task of creating a Sudoku solving pipeline has many design considerations. While one filtering method may work for one image of a puzzle, it may not necessarily behave the same way for the next. Creating a pipeline that is robust to document warping, low light conditions, varying lighting conditions within the same image, etc. proves to be a intricate balance between when heuristic approaches are useful and when a technique that leverages the empirical properties is necessary.

In subsequent attempts at end-to-end Sudoku solving, one very useful feature would be puzzle unwarping with robustness to various types of input. This would make cell extraction a much simpler problem and further improve the strategy of dividing the rectified image into 81 equally-sized squares.

Notes:

1. We have added Boris Ivanovic to our private BitBucket repository.
2. Elliott and Andres are no longer double counting this project with CS231N.
3. The strange formatting at the end of page 2 is unintentional – we could not figure out how to get rid of it.
4. https://bitbucket.org/CNormand/cs231a_project

References

[1] Pramod J Simha, Suraj K V, and Tejas Ahobala. Recognition of numbers and position using image processing techniques for solving Sudoku Puzzles. Advances in Engineering Science and Management, 2012.

[2] Yixin Wang. Sudoku Solver. Stanford University, EE368. Spring 14-15.

[3] Alejandro Hernandez Cordero. Sudoku Recognizer. shogun-toolbox.org/static/notebook/current/Sudoku_recognizer.html.

[4] Sameep Bagadia and Nihit Desai. End-to-end system for recognizing and solving Kakuro Puzzles. Stanford University, CS 231A. Spring 15-16.

[5] Actual Author Name. Frobnication tutorial, 2014. Some URL al tr.pdf.

[6] Chan, Fhy; Lam, Fk; Zhu, H. Adaptive thresholding by variation method. IEEE Transactions on Image Processing, 1998, v. 7 n. 3, p. 468- 473

[7] Shaodi You, Yasuyuki Matsushita, Sudipta Sinha, Yusuke Bou, Katsushi Ikeuchi. Multiview Rectification of Folded Documents. https://arxiv.org/pdf/1606.00166.pdf

[8] Heinz Breu, Joseph Gil, David Kirkpatrick, and Michael Werman. Linear Time Euclidean Distance Transform Algorithms. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE. VOL. 17, NO. 5, MAY 1995

[9] Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research. IEEE Signal Processing Magazine. Volume: 29 Issue: 6

[10] LeCun, Yann. "LeNet-5, convolutional neural networks." *URL: http://yann. lecun. com/exdb/lenet* (2015).

[11] Hummel, R. A., Kimia, B., & Zucker, S. W. (1987). Deblurring gaussian blur. *Computer Vision, Graphics, and Image Processing*, *38*(1), 66-80.

[12] Moghaddam, Reza Farrahi, and Mohamed Cheriet. "AdOtsu: An adaptive and parameterless generalization of Otsu's method for document image binarization." *Pattern Recognition* 45.6 (2012): 2419-2431.

[13] Sojka, E. (2002, April). A new algorithm for detecting corners in digital images. In *Proceedings of the 18th spring conference on Computer graphics* (pp. 55-62). ACM.

[14] Shen, Fei, and Han Wang. "A local edge detector used for finding corners." *Proc. ICICS*. Vol. 2001. 2001.

[15] Harris, Chris, and Mike Stephens. "A combined corner and edge detector." *Alvey vision conference*. Vol. 15. No. 50. 1988.

[16] Carroll, Robert, Aseem Agarwala, and Maneesh Agrawala. "Image warps for artistic perspective manipulation." *ACM Transactions on Graphics (TOG)*. Vol. 29. No. 4. ACM, 2010.

[17] Derpanis, Konstantinos G. "The harris corner detector." *York University* (2004).

[18] Tan, Songbo. "An effective refinement strategy for KNN text classifier." *Expert Systems with Applications* 30.2 (2006): 290-298.

[19] Acuna, Edgar, and Caroline Rodriguez. "The treatment of missing values and its effect on classifier accuracy." *Classification, clustering, and data mining applications* (2004): 639-647.

[20] Rennie, Steven J., Vaibhava Goel, and Samuel Thomas. "Deep order statistic networks." *Spoken Language Technology Workshop (SLT), 2014 IEEE*. IEEE, 2014.

[21] Wu, Chunpeng, et al. "Handwritten character recognition by alternately trained relaxation convolutional neural network." *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*. IEEE, 2014

[22] Zeng, Daojian, et al. "Relation Classification via Convolutional Deep Neural Network." *COLING*. 2014.

[23] de Campos, T. E., R. B. Bodla, and M. Varma. "The Chars74K dataset." (2009).

[24] Xu, Ning, Narendra Ahuja, and Ravi Bansal. "Object segmentation using graph cuts based active contours." *Computer Vision and Image Understanding* 107.3 (2007): 210-224.