End-to-end system for recognizing and solving Kakuro puzzles

Sameep Bagadia
Stanford University
sameepb@stanford.edu

Nihit Desai
Stanford University
nihit@stanford.edu

ABSTRACT

Kakuro is a type of logic puzzle that is often referred to as a mathematical equivalent of the crossword. In this project, we build an end-to-end system for recognizing Kakuro puzzles from real images that one might take from a cellphone camera, and solving them. The final solution is overlaid on top of the original image. This problem provides us the opportunity to work on a wide variety of computer vision problems such as binarization, segmentation and recognition. While there has been some prior work in building similar systems for Sudoku, as far as we know, ours is the first attempt to build such a system for Kakuro puzzles.

1. INTRODUCTION

Mathematical puzzles and games have often been used to showcase various techniques and algorithms in Computer Science because of the well-defined nature of the problem domain. In this project, we explore the problem of recognizing and solving Kakuro puzzles [1] taken from real images that one might take from a cellphone camera. After solving the puzzle we aim to overlay the solution on top on the photograph of the unsolved puzzle.

Our motivation for building a system for recognizing and solving Kakuro puzzles is two fold: firstly, we would love to use such a system ourselves! We see puzzles like Kakuro in newspapers and puzzle books, but many times they come without a solution. Instead of manually feeding in the puzzle setup to a computer, it is faster, easier and more intuitive to be able to see the solution of the puzzle by just taking a photograph. Secondly, this project will provide us the opportunity to work on problems related to to image binarization, perspective correction, image segmentation and multi-digit number recognition, all of which are relevant problems in the field of Computer Vision.

We devise our technical solution as a sequence of steps, each of which can be seen as a module or a lego block. Our aim is to have each module solve a specific problem and operate as independently as possible from other modules. This allows us to easily change the implementation inside any module without affecting the system. A detailed explanation of our technical approach is in Section 4. We finally test the performance of our number recognition model, as well as the entire system on a test dataset of Kakuro puzzle images. Our system is fairly robust to variations such as different puzzle sizes, lighting conditions, rotations and variations in fonts. Quantitative and qualitative results from our evaluation are discussed in Section 5.

2. PROBLEM DOMAIN

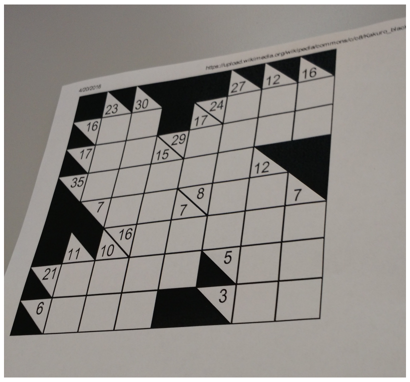


Figure 1: Kakuro puzzle photograph

Kakuro is a type of logic puzzle that is often referred to as a mathematical equivalent of the crossword. An example puzzle is shown in Figure 1. The Kakuro puzzle is a 2-D grid of cells where each cell can be a “Blank” cell (which need to be filled in as part of the puzzle solution), or a “Brick” cell (which contains a diagonal slash from upper-left to lower-right and a number in zero or one or both halves). Each horizontal array of Blank cells have a number in the Brick half-cell to its immediate left and each vertical array of Blank cells has a number in the Brick half-cell immediately above it. These numbers provide important clues in the puzzle. The objective of the puzzle is to insert a digit from 1 to 9 into Blank empty cell such that the sum of the numbers in each array of Blank cells equals the clue associated with it and that no digit is duplicated in an array.

In this project, we aim to build an end-to-end system for solving these Kakuro puzzles. This system can best be thought of as a pipeline, consisting of a sequence of steps, each of which solves one part of the problem - image denoising and binarization, puzzle detection, perspective transformation, number recognition and searching for a valid solution. We provide details about these steps in Section .

3. RELATED WORK

Image Thresholding: Thresholding is one of the simplest method of image segmentation. From a grayscale image, thresholding can be used to create a binary image. In their survey [15], Sezgin and Sankur categorize thresholding methods into various groups - local methods (adapt the threshold value on each pixel), spatial methods (using higher-order probability correlation between pixels), clustering methods (where background and foreground pixels are modeled as a mixture of two Gaussians) to name a few. Among the last group of techniques, Otsu's image thresholding method [14] is by far the most commonly used. This is also the technique we use in this work. The algorithm assumes that the image contains two classes of pixels following bi-modal histogram and calculates the optimum threshold separating the two classes.

Feature Detection: Feature detection refers to methods to compute abstractions that represent an image, starting from raw pixels. Broadly, the aim is to capture information about "interesting" parts of the image, which can then be used downstream for tasks such as recognition, segmentation etc. The geometrically precise structure of Kakuro puzzles means that we can leverage edge features and corner detectors quite substantially - for detecting the puzzle grid, for cell type classification and so on. For edges, we use the Canny edge detector [4], while for corners, we use the Harris corner detector [8].

Convolutional Neural Networks: Visual recognition is a well studied problem in Computer Vision. Deep Convolutional Neural Network (CNN) architectures [11] have recently become popular as they eliminate the dependence on hand-designed explicit features and instead directly learn good feature representations from raw data [24]. CNN-based models have advanced state-of-the-art performance in various computer vision tasks, including visual recognition (which is the application we focus on in this project) [10], [9]. There has also been considerable work in enhancing individual components of CNN architectures, including pooling layers [23] and activation units [7]. Additionally, there is ongoing work in improving training of CNN models, such as by Dropout[17]. In this project, we use CNN models for the problem of number recognition in order to identify constraints of the given Kakuro puzzle. There has been some prior work in using CNN architectures for this problem by Niu and Suen [13].

Sudoku Solvers: There has been some work previously to recognize sudoku puzzles - [16] use some computational geometry techniques to detect thepuzzle orientation, followed by template matching as a method for digit recognition. While this method works well for puzzles of a fixed size and fonts, these techniques don't generalize well to handwritten puzzles or fonts that are not present in the template database. Recently, [22] has proposed using a Deep Belief Network (DBN) to recognize digits from Sudoku puzzles. While these bear some similarities with the problem at hand, it is important to note that Sudoku puzzles are easier to recognize and solve. Firstly, there can be only two types of cells (unlike 5 as in case of Kakuro), a cell can at most have one digit (unlike of multiple digits in case of Kakuro).

4. TECHNICAL APPROACH

In this section, we outline the main steps and technical challenges involved in our project. Since our system consists of sequence of steps. We will explain each step in sequence. Throughout this section we use a running example of kakuro puzzle shown in figure 2. We implement our system in Python, and use OpenCV [3] for binarization and feature detection steps (such as Cannry edge detection, Harris corner detection, detecting contours in the image etc). We implement the CNN model training and testing in Tensorflow [2].

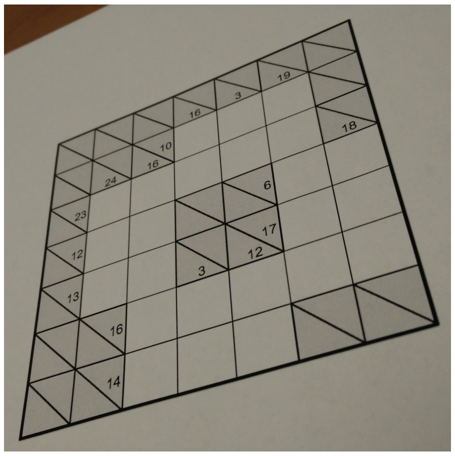


Figure 2: Original Image

4.1 Denoising and Binarization

We first need to separate the background (non-puzzle parts of the image) from the foreground (puzzle) by binarizing the image. As a preprocessing step, we convolve the image with a Gaussian filter to reduce noise. This technique, also called Gaussian smoothing, is widely used in graphics software to reduce image noise. As a next step, we use Otsu's adaptive thresholding method [14] for binarizing the image. Otsu's method performs a histogram-based image thesholding. The algorithm assumes that the image to be thresholded contains a bi-modal histogram of pixels (foreground and background). The algorithm then calculates the optimum threshold separating those two classes. The resulting binarized image is shown in figure 3.

4.2 Puzzle Corners Detection

In this step, we want to detect the puzzle in the photograph by detecting the four corners of the puzzle. The approach we use is to detect contours in the image using the algorithm

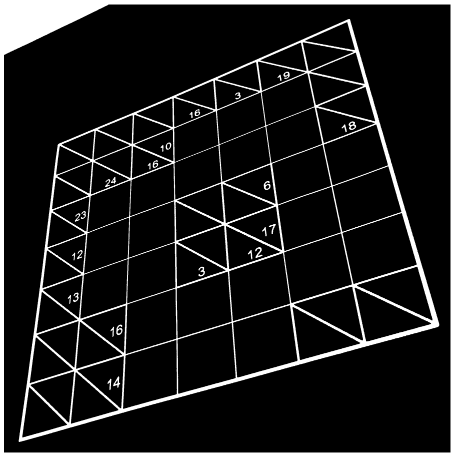


Figure 3: Binarized Image

of [19]. An implementation of this is already available in OpenCV, as part of the the `findContours` function. Contours are a useful tool for shape analysis and object detection and recognition. From all the contours we retrieve, we consider the one with the largest area to be the contour corresponding to the puzzle grid. From this contour, we will find the 4 corners of the puzzle.

In order to detect the 4 corner points from the list of points defining the puzzle contour, consider line of slope -1 drawn from each point. Consider their intersection with the y axis as shown in the figure 4. The point of intersection of lines corresponding to two of the corners with y-axis will be either maximum or minimum value. The value for rest of the points on the contour will lie between them. Similarly other two corners will be detected using line of slope 1. We make use of this idea to detect the corners.

Concretely, for each point (x_i, y_i) we calculate two values:

1. $x_i - y_i$
2. $x_i + y_i$

The points at which maximum and minimum values are obtained for the above two equations give us the four corner points of the puzzle. Figure 5 shows the corners detected for our running example.

4.3 Perspective Correction

We do not assume the image to be taken from any particular viewpoint. This means the puzzle will not always be perfect centered rectangle but can be off-centered and can have skew and rotations. We transform the detected puzzle grid into a standardized top-down viewpoint for the task of recognition with a two step process:

1. Using the corners of the puzzle grid and the mapping of each corner to the grid in the desired viewpoint, we compute a generic 2D to 2D perspective transformation matrix.

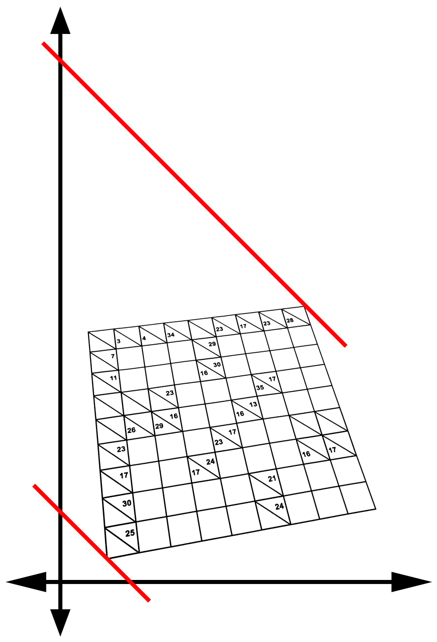


Figure 4: Corner Detection for inferring the orientation of the puzzle. These corners are used for perspective correction

2. We apply this perspective transformation to the original puzzle grid to get the desired top-down view.

In summary, we calculate a 3x3 perspective matrix so that a point (x_i, y_i) in the original image is transformed to (x'_i, y'_i) in the transformed image, such that:

$$\begin{bmatrix} t_i.x'_i \\ t_i.y'_i \\ t_i \end{bmatrix} = H * \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Figure 6 shows the transformed image.

4.4 Grid Detection

Once we have an aligned image, we detect the lines in the image to detect the grid of the puzzle. In order to do this, we first apply Canny edge detector on the image and then we use Hough transform [6]. Canny edge detection helps in improving the results of Hough transform. The image obtained after apply Canny edge detection is shown in figure 7.

After we have the lines from Hough transform, we post-process them to remove duplicate lines by eliminating lines that are very close to each other. Once we have the unique set of lines, we remove the boundary lines. The grid lines detected from Hough transform are shown in figure 8. From the grid lines we can infer the size of the puzzle. The example image is a 7×7 puzzle.

4.5 Cell Type Classification

Once we know the grid and size of the puzzle, we can focus on individual cells. For each cell in the grid, we detect the type of cell it is. In Kakuro, we have blank cells where the

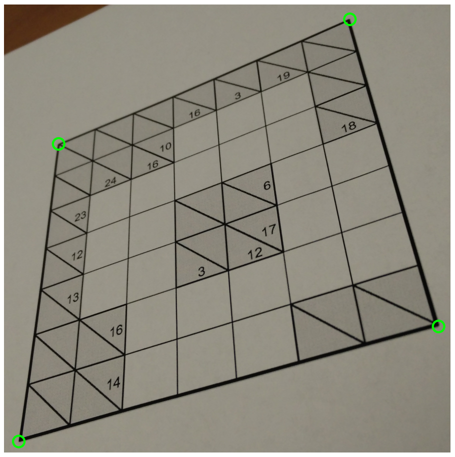


Figure 5: Detected puzzle corners

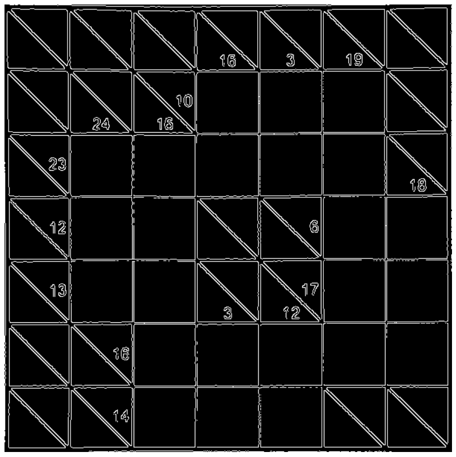


Figure 7: Edge features after applying Canny Edge detection

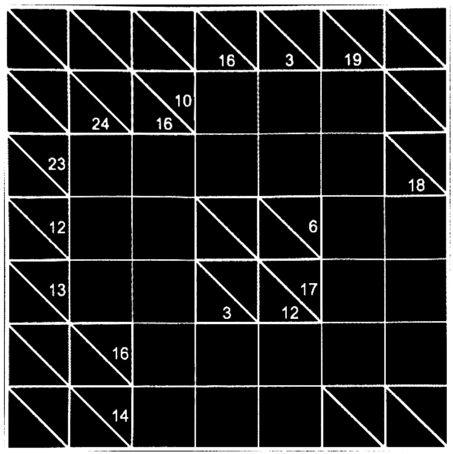


Figure 6: Perspective corrected

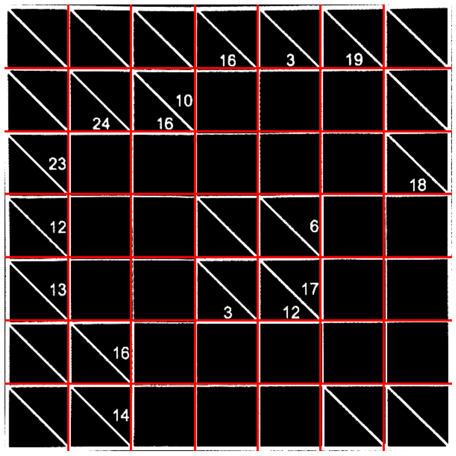


Figure 8: Detected Grid lines overlaid on top of the puzzle

solver needs to put in a number and brick cells which are not to be filled. We distinguish between them based on the diagonal line present in the brick cells. This line is detected using Hough transform [6]. Figure 9a shows the diagonal line detected in a Brick cell.

Brick cells are further divided into 4 categories based on whether there is a number present on either side of the diagonal. In order to detect this, we first apply Harris corner detection to cell image. We expect a large number of corners if there is a number present. Figure 9b shows Harris corner applied on one of the cells. We check each half of the diagonal and if the number of corners detected there is greater than a threshold, we mark it as a number.

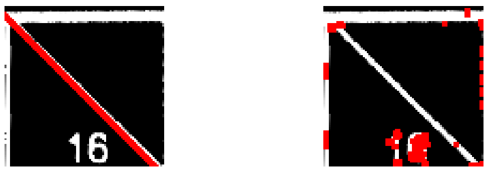
At this point we can completely classify each cell into Blank or Brick and also each brick into various sub-types. The cells and their types are shown in table 1.

4.6 Multi-Digit number recognition

Our next task to to recognize the numbers present in the brick cells. Note that some of the numbers contain 2 digits whereas some numbers contain single digit. We further divide the number recognition problem into finding the digit bounding boxes, digit recognition and combining the digits to obtain the number.

4.6.1 *Digit bounding box detection*

The problem here is to find bounding boxes for individual digits so that we can extract them and pass it to Convolutional Neural Network (CNN) for digit recognition. For each brick cell containing numbers, we first find contours in the image. We then eliminate false positives using heuristics on the area and location of the contour. We also match



(a) Diagonal line detection (b) Harris corner detection

Figure 9: Features for cell type classification

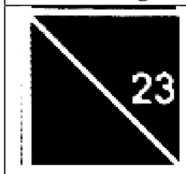
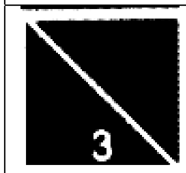
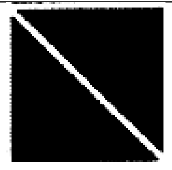
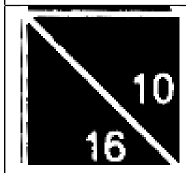
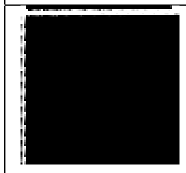
Cell Image	Cell type obtained
	Brick01
	Brick10
	Brick00
	Brick11
	Blank

Table 1: Example cells and cell detection output

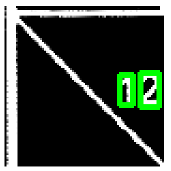
the contour location with the brick type to make sure we are detecting bounding box for digits in the correct side of the diagonal. Figure 10a shows the bounding boxes detected for one of the cells from the puzzle.

Using the bounding boxes, we crop the digits. We then resize the digits and add padding around them to match the size of the images used for training the CNN. Figure 10b shows one of the extracted digits after resizing and padding. We first extract all the digits in the puzzle and then pass them in a batch to the CNN for recognition.

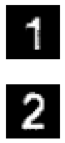
4.6.2 Digit recognition using CNN

This is a supervised learning task - Given an image (a grid of pixels of fixed size), we want to classify it into one of the 10 labels. After evaluating some possible approaches to solve this problem, we finally decided to use a Deep Convolutional Neural Network (CNN) model: CNNs can learn highly non-linear functions unlike linear classifiers, CNNs are fast at test-time compared to models like $k-$nearest neighbors, and CNNs generalize very well to unseen data such as different fonts compared to template matching techniques.

We experimented with different architectures (varying the number of layers, type of non-linearity, size of convolving filter etc), but the final architecture layout of our model is given in Figure 11. We use a 6-layer CNN (1 input layer, 3 convolutional layers, 1 fully-connected hidden layer, 1 output layer) for this task. It is important to note that all layers except the input consist of an affine transformation, a non-linearity following by a pooling operation layer.



(a) bounding box around digits



(b) Extracted digits

Figure 10: Extracting digits from each brick type cell

- **Input Layer**: Input layer expects a fixed size grayscale image of 28x28 pixels.
- **Convolutional Layers**: These are 2D convolutional layers. Each layer consists of a convolution, followed by a ReLU (Rectified Linear Unit) non-linearity, followed by max-pooling. The convolution uses padding around the edges the maintain input dimensions, while the 2x2 max-pooling layer reduces the width and height by a factor of 2. Since our network has 3 convolutional layers, the input images of spatial dimension 28x28 are finally reduced to 7x7. This is matched by increasing the depth of the output volume of a convolutional layer (i.e. 32 for the first layer, and 64 for the next two layers). We also added dropout after the last convolutional layer to better regularize our model.
- **Fully Connected layers**: We flatten the output of the last convolutional layer and use this "feature vector" as input to a fully-connected layer with 512 hidden units. This hidden layer uses a ReLU non-linearity and dropout for better regularization. The output of the hidden layer is fed into a softmax classifier which produces class probability scores as the final output.

To train our model, we used a combination of two datasets - MNIST, which is a dataset of handwritten digits [12], and the Chars74K dataset [5], which is a dataset of digits from various computer fonts. The total size of our dataset is 80000 images, which we split into 60000 training images (for training), 10000 validation images (for optimizing hyperparameters), and 10000 test images (for final testing). We noticed that using only the MNIST dataset did not generalize as well to printed numbers, and that the combination of datasets gives us a better test performance.

We implemented our model using Tensorflow [2] and trained it on a g2.2xlarge GPU instance (NVIDIA Tesla K40 GPU) on Amazon Web Services. These instances have 15GB of memory and 4GB of GPU memory. We used a cross-entropy loss function (with l-2 regularization) and minibatch gradient descent using Momentum optimizer [18] for the training

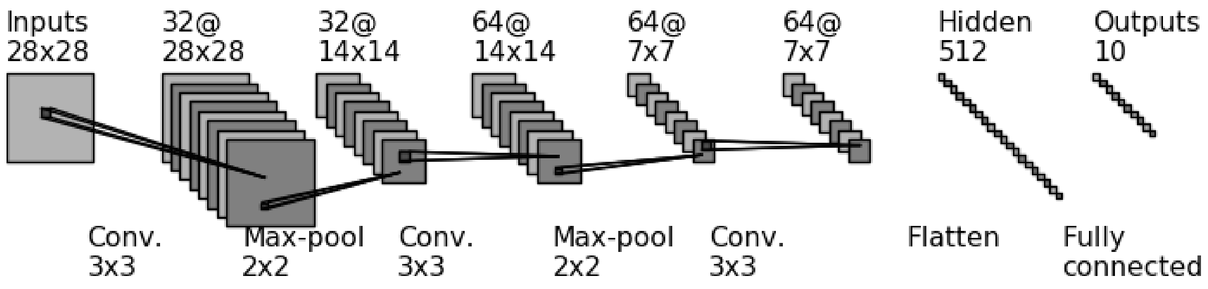


Figure 11: CNN architecture used for digit classification

process. Details about hyperparameters and evaluation are given in Section 5.

4.6.3 *Combining digits*

Once we have recognized all the digits in the puzzle, we map them back to their respective cells. If we have two digits belonging to the same number then the original number is constructed from them based on the location of the bounding box corresponding to the digits.

4.7 Solve

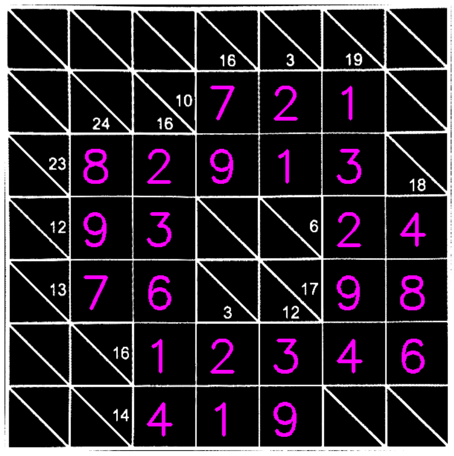


Figure 12: Solution

Once we have detected all the bits and pieces of the Kakuro puzzle, we put them together to solve the puzzle and get the digit to be filled in each blank cell. The board is solved using an iterative refinement process by eliminating possible values from entries and candidate values for semi-rows or semi-columns used in sums. When pure refinement yields no more improvement, the algorithm uses backtracking, branching on an entry of fewest possible values.

4.8 Overlay solution on top of the image

Once the puzzle is solved successfully, we print the solution on the unsolved puzzle grid. Figure 12 shows the solution for our example puzzle. We then re-transform it back to original image so that digit appears to have been filled in the original image which will be shown to the user. In order to retransform it back, we apply the inverse of the transformation matrix we applied in section 4.3. The final image obtained is shown in figure 13.

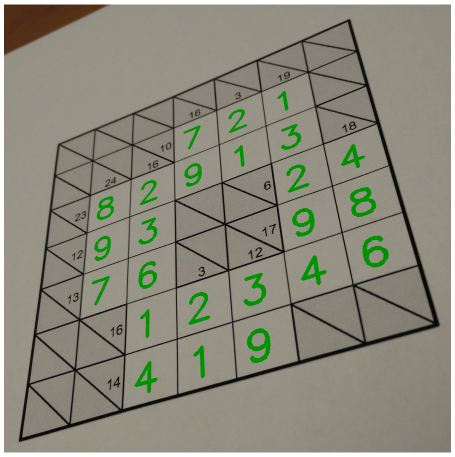


Figure 13: Solution transformed and overlaid on original image

5. EXPERIMENTS & EVALUATION

Our evaluation consists of two main components. Firstly, we evaluate the the performance of the Convolutional Neural Network model on the training and validation dataset quantitatively and qualitatively. Secondly, we evaluate the performance of our system by evaluating various steps individually.

5.1 CNN Evaluation

As mentioned in Section 4.6, we use a combination of two datasets - MNIST, which is a dataset of handwritten digits [12], and the Chars74K dataset [5], which is a dataset of digits from various computer fonts. The total size of our dataset is 80000 images, which we split into 60000 training images, 10000 validation images, and 10000 test images. In order to train the model, we used a cross-entropy loss function (with l-2 regularization) and minibatch gradient descent using Momentum optimizer [18]. Our models take about 20 epochs to converge. The optimal hyperparameters values we arrived at after grid search and some manual testing are given below:

- Learning rate: $1e$-2, decays exponentially after each epoch.

- L2 regularization: $5e$-4
- Dropout keep probability: 0.5
- Convolution filter size: 3x3
- Convolution output depth: 32, 64, 64 respectiely in the three convolution layers
- Densely connected layer hidden dimension: 512

While training, we track training loss curves as a yardstick to ensure proper training and convergence of our models. Training loss curve for our model training phase is given in Figure 14.

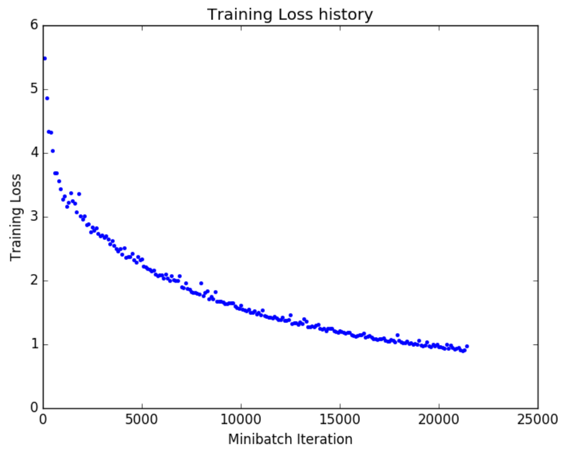


Figure 14: Loss history for 6-layer CNN

After training is completed, we test the performance of our model on our dataset. Since our dataset is balanced across all classes, we use accuracy as our evaluation metric at this stage (the next stage of system evaluation also looks at class-wise performance). Performance numbers of our model are given in Table 2. Our model, while only 6 layers deep, achieves close to state of the art performance on this dataset ([21] achieve a test accuracy of 0.998).

Dataset	Accuracy
Train	0.998
Validation	0.996
Test	0.995

Table 2: 6-layer CNN evaluation results on the MNIST + Chars74K dataset

Lastly, we perform some qualitative evaluation. We extract features from the hidden layer (layer 5 of our model), apply a dimensionality reduction technique and then cluster our data points in the reduced dimensionality space using a well-known technique called t-SNE (t-Distributed Stochastic Neighbor Embedding). Figure 15 shows a t-SNE [20] plot of the layer-5 representation of data points in the validation set. We observe that data points belong to different classes are quite well separated, an insight that is also borne out by the quantitative results in Table 2.

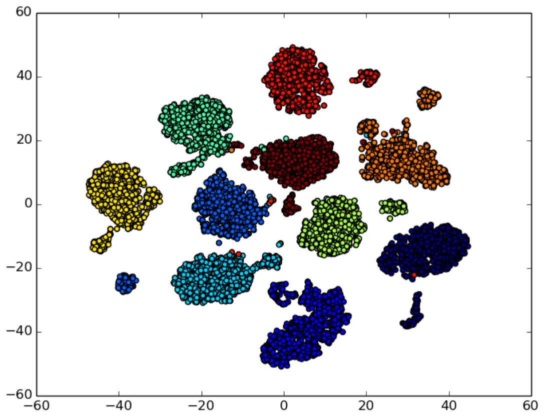


Figure 15: t-SNE visualization of the MNIST validation set. Points with a particular color all belong to the same labels (i.e. they are the same digit).

5.2 System Evaluation

Since our pipeline consists of a series of geometric transformation and recognition tasks, we think it is important to rigorously evaluate the performance of each step in the pipeline. We explain our system evaluation setup and results in this subsection.

We used puzzles of varying sizes for evaluation. For each of the puzzles, we clicked multiple images by varying the angle, skew and lighting conditions between them to give us a wide range of test examples. In total we have 40 different kakuro puzzle test images. Since each puzzle has many cells, numbers and digits we have a good size of test data. Out test set consists of a total of 2020 cells, 1180 digits and 700 numbers. We use this test set to test the following steps of our system:

1. Corner detection
2. Puzzle size
3. Cell type classification
4. Bounding box detection
5. Digit recognition
6. Number recognition

These evaluation metrics helped us in evaluation of our system and to improve our performance by helping us find the major error sources. Our final test accuracies for the above metrics are shown in table 3

We see that our system is pretty robust in corner detection as well as detecting the size of the puzzle. It achieved 100%

Type of test	Test accuracy
Corner detection	1.000
Inferring size (nxn puzzle)	1.000
Cell type classification	0.991
Digit bounding box	0.989
Individual digit recognition	0.981
Number recognition	0.952

Table 3: System evaluation results

accuracy in both the tasks. For cell type classification it achieved an accuracy of 99.1%. It does fairly good job at this. Figure 16 shows the confusion matrix for cell type classification. We see that sometimes it confuses Brick00 as other brick types. This means it thinks there is a number sometimes when there is none. This is because of some noise in the image making Harris corner detection showing some noisy corners. Also, most of these errors are in images with higher puzzle size since number of pixels per cell is smaller for larger puzzle sizes.

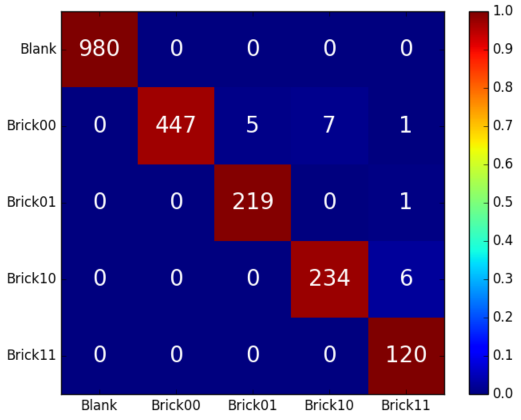


Figure 16: Cell type classification confusion matrix on the kakuro puzzle test set

Bounding box detection accuracy is 98.89% and out of those digits for which we have correct bounding box, digit recognition accuracy is 98.11%.

Since digit recognition is a multi-label classification problem, we also present a confusion matrix which shows the classwise performance of our model on the test. Figure 17 shows the confusion matrix for digit recognition. We see that in general the model does a good job across the board - it mostly classifies the digits correctly. But there is occasionally a confusion between the pairs of digits (1,7), and (8, 9). This is intuitively easy to see, since these pairs of digits are written in a similar fashion.

A number will be detected correctly only if the following criteria is satisfied:

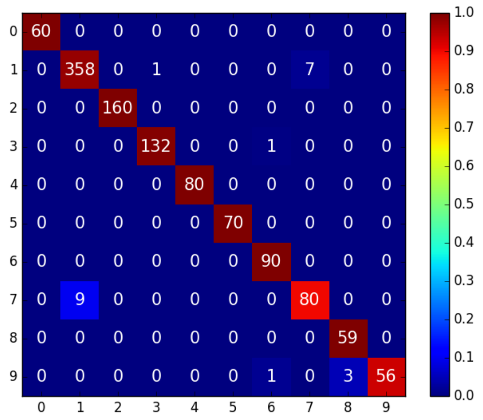


Figure 17: Digit recognition confusion matrix on kakuro puzzle test set

- cell type is correctly classified
- bounding boxes of all digits of the numbers are correctly detected
- all digits in the number are correctly recognized.

Overall number detection accuracy on our test set is 95.28%

In general, our system works pretty well but makes few mistakes as seen in the above results. Also we went ahead to analyze when the mistakes are made. We notice that most of the images have no mistake in any of the steps and give the correct final solution image. Only some images are the source of the errors. Most of the images that had errors were because of slightly out of focus or blurred image. Thus if the photo is taken with proper focus then our system does a pretty good job at completely recognizing everything correctly. We also notice some errors in images with higher puzzle sizes. This is because as the puzzle size increases, number of pixels per cell decreases. This means at cell level we have slightly inferior image leading to slightly worse results

6. CONCLUSION AND FUTURE WORK

As part of this project, we created a system to recognize and solve Kakuro puzzles. Through this project, we had the opportunity to work on a wide variety of computer vision problems (each of which play a role in solving a sub-problem) - binarization, perspective correction, feature detection, segmentation and recognition. While there has been some prior work in building similar systems for puzzles such as Sudoku, as far as we know, ours is the first attempt to build a recognition and solving system for Kakuro puzzles.

We would like to share some insights we came across when working on this project:

- We observed that traditional Computer Vision techniques, despite being ovetaken by neural network approaches in popularity in recent years, actually still worked well for our case. Our solution for perspective correction, cell type classification and extracting digits from the puzzle (in order to feed it to the classifier) all use techniques that are well-established and were taught in the class. From our tests results (given in Table 3), we conclude that the techniques generalized fairly well.

- On the other hand, for tasks such as recognition, convolutional neural network achieve state of the art performance and we chose to use them for digit recognition. However, neural network models are sensitive to hyperparameter settings, parameter initialization schemes and can overfit very easily to training data as we observed during the initial stages. We had to experiment with hyperparameters to find ranges where the model performed well.

There are multiple possible directions to extend this work in the future. We would like to make the system faster and more robust - currently, the end-to-end latency for an image is between 2-3 seconds (depending on puzzle size) and we think w can get it to under 2 seconds with optimizations. Secondly, we would like to extend the user interface in the form of a smartphone application and try to have the entire system running on the client for an even more responsive user experience. Lastly, we would like to extend this to recognize different variations of Kakuro puzzles (which may not be rectangular grids, or may operate with a different set of constraints etc).

7. REFERENCES

[1] Kakuro puzzle. *https://en.wikipedia.org/wiki/Kakuro*.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] G. Bradski. *Dr. Dobb's Journal of Software Tools*.

[4] J. Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

[5] T. E. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In *Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal*, February 2009.

[6] R. O. Duda and P. E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, Jan. 1972.

[7] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[8] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Citeseer, 1988.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[12] Y. Lecun and C. Cortes. The MNIST database of handwritten digits.

[13] X.-X. Niu and C. Y. Suen. A novel hybrid cnn–svm classifier for recognizing handwritten digits. *Pattern Recognition*, 45(4):1318 – 1325, 2012.

[14] N. Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[15] M. Sezgin et al. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic imaging*, 13(1):146–168, 2004.

[16] P. J. Simha, K. Suraj, and T. Ahobala. Recognition of numbers and position using image processing techniques for solving sudoku puzzles. In *Advances in Engineering, Science and Management (ICAESM), 2012 International Conference on*, pages 1–5. IEEE, 2012.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[18] R. S. Sutton. Two problems with backpropagation and other steepest-descent learning procedures for networks. In *Proc. 8th annual conf. cognitive science society*, pages 823–831, 1986.

[19] A. K. Suzuki S. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics and Image Processing*, 1985.

[20] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[21] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066,

2013.

[22] B. Wicht and J. Hennebert. Camera-based sudoku recognition with deep belief network. In *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of*, pages 83–88. IEEE, 2014.

[23] M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

[24] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer vision–ECCV 2014*, pages 818–833. Springer, 2014.