Sudoku Solver with Image Processing

Beyazıt ISIK[1], Doğukan GÖZLER[2], Cihan Berk ELMAS[3]

[1,2,3] *Eskişehir Technical University, Electrical and Electronics Engineering*

Abstract

Sudoku Solver with Image Processing is important for the future artificial intelligence problem recognition and troubleshooting algorithms. Various methods have been combined to get absolute results with small loss. In this paper, we propose OpenCV, Backtracking, Number Detection OCR based approach for recognition and analyzation of sudoku puzzle. In the proposed model, our approach is to consecutively, (i) Explore and visualize the dataset, (ii) Develop a CNN model (iii) Train and validate the model, (iv) Test the model with the test dataset, (v) Develop a algorithm to find boxes in the puzzle, (vi) Predict the number in each cell with the OCR model, finally, (vii) Solve the sudoku matrix with backtracking algorithm. These tasks carried out by Python programming language with varying algorithms and additional libraries. Dataset for number detection OCR provided by The Chars74K dataset with sorting only [0-9]. Experiments demonstrate that, constructed model gives high accuracy for detecting sudoku numbers and solving sudoku puzzles efficiently.

1. Introduction

Puzzles play an important role in protecting people's mental health. According to researchers, some puzzles reduce risk of Alzheimer's and improve your verbal skills [1]. Crossword, Jigsaw and Sudoku can be given as examples of puzzles. The objective of sudoku is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 sub-grids that compose the grid (also called "boxes", "blocks", or "regions") contain all the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution.

The goal of this project is to reach maximum possible accurate and efficient solutions of sudoku puzzles with Image Processing. In order to achieve the goal, we choose method of sudoku solver with image processing based on deep Convolutional Neural Networks (CNNs) [2,3], a biologically inspired, multilayer feed-forward architecture that can learn multiple stages of invariant features using a combination of supervised and unsupervised learning (see Figure 1). Each stage is composed of a (convolutional) filter bank layer, a non-linear transform layer, and a spatial feature pooling layer. In addition, with the backtracking algorithm, the generated outputs are solved within the framework of sudoku rules.

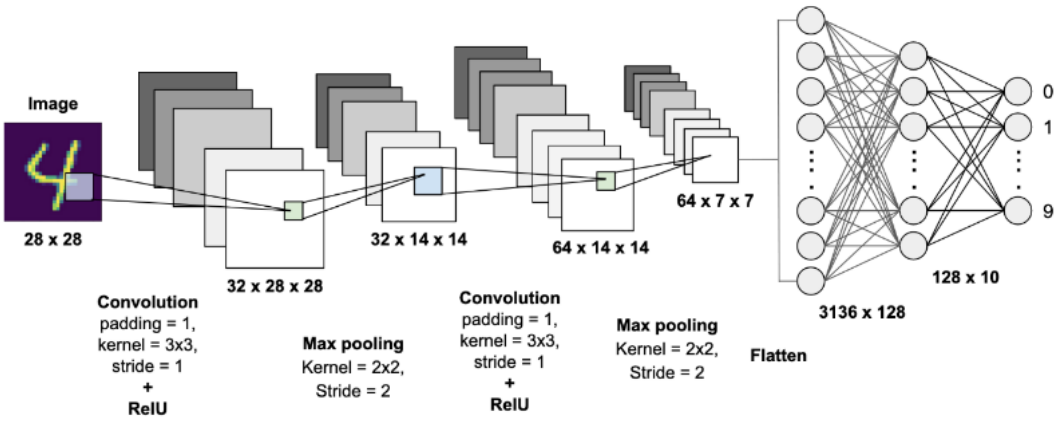

Figure 1: *An example of CNN Architecture*

The following sections of this paper explain the outline of the model, detailed architecture of the networks employed in each stage, implementation and training procedure of the networks, and the results obtained after testing the OCR in Sudoku puzzles.

2. Materials and Methods

The methods we used are Artificial Intelligence, Deep Learning, Artificial Neural Networks and Convolutional Neural Networks, Canny Edge Detection and Back-Tracking Algorithm.

2.1 Contours

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition. There are many algorithms for this purpose. These algorithms based on approximation to the desired contour. Most popular algorithm called as Ramer–Douglas–Peucker algorithm. The aim of this algorithm is, given a curve composed of line segments to find a similar curve with fewer points. Basically, it simplifies the curve with a subset of the points. from the original curve. Various methods can be used to select the point where to divide a subset into two subsets. The division at the point of maximum distance from the approximating straight-line results in a low number of vertices and very little extra computation. [4-5]

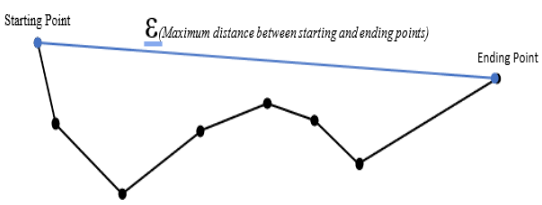

Figure 2: *A piecewise linear curve*

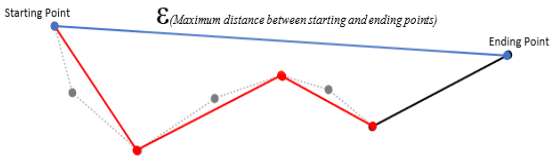

Figure 3: *A simplified version of piecewise linear curve using Douglas–Peucker algorithm*

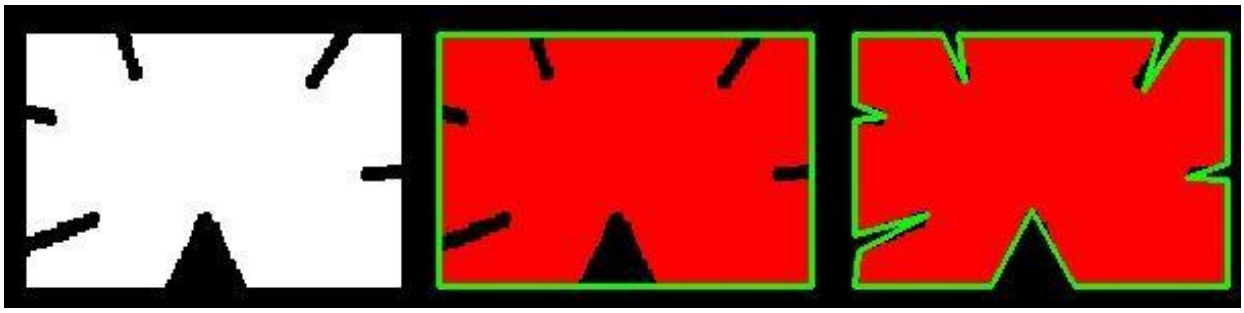

Figure 4 *An approximated image with Douglas–Peucker algorithm*

In figure [4], there is an approximation to the outer box. Image in the left is the original image, center image is the approximated curve for epsilon =10% of arc length and the last one is for epsilon = 1% of the arc length. Decreasement in the epsilon causes more indentation on the image.

2.2 The Canny Edge Detection

First of all, edge detection is one of the most commonly used operations in image analysis, and there are probably more algorithms in the literature for enhancing and detecting edges than any other single subject. The reason for this is that edges form the outline of an object. What is an edge? An edge is the boundary between an object and the background and indicates the boundary between overlapping objects. This means that if the edges in an image can be identified accurately, all of the objects can be located and basic properties such as area, perimeter, and shape can be measured. [5] Since computer vision involves the identification and classification of objects in an image, edge detections are an essential tool. The traditional edge detection algorithms are done through detecting the maximum value of the first derivative or zero crossing of the second derivative. Although the representative first order differential operators (Roberts operator, Prewitt operator, Sobel operator, etc.) and second order differential operators (Laplace operator, LOG operator, etc.) have many advantages such as simple computation, rapid speed and easy to implement, they are more sensitive to noise and their detection effect are not perfect in engineering application. The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. In 1986, Canny proposed three criteria to judge edge detection operator' performance: SNR criterion, localization precision criterion and single edge response criterion, and deduced the best Canny edge detection operator. Compared with common edge detection algorithm, in most cases, the Canny algorithm has the best performance. Owing to its optimality to meet with the three criteria for edge detection and the simplicity of process for implementation, it became one of the most popular algorithms for edge detection. The Process of Canny edge detection algorithm can be broken down to 5 different steps:

2.2.1 Apply Gaussian Filter to Smooth the Image in Order to Remove the Noise

The first step of the traditional Canny algorithm is to smooth image. Canny deduced the first derivative of Gaussian function, which is the best approximation of the optimal edge detection operator. Choose appropriate 1-d Gaussian function to smooth the image according to the row and column respectively, that is, execute convolution operation to image matrix. Since the convolution operation satisfies commutative law and associative law,

Canny algorithm generally uses two-dimensional Gaussian function to smooth image and get rid of the noise.

$$G(x, y)=\exp[-(x^2+y^2)/2\sigma^2]/2\pi\sigma^2$$

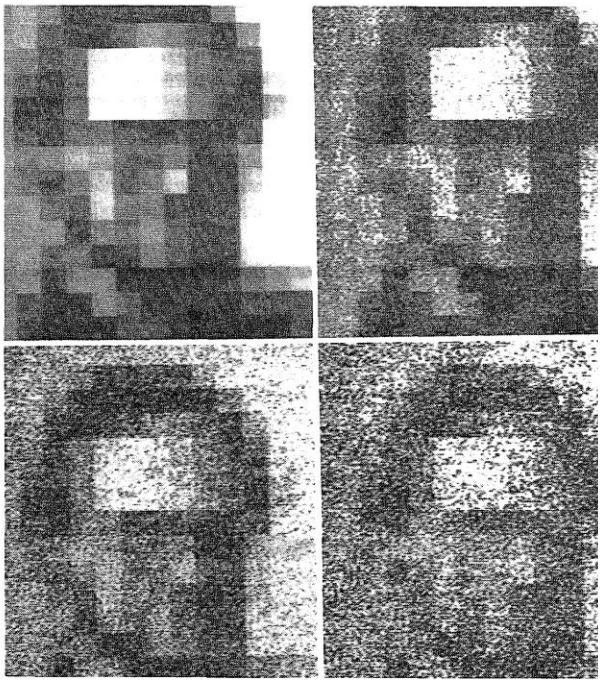

Figure 5 *Varying Amount of Additive Gaussian Noise*

2.2.2 Find the Intensity Gradients of the Image

The second step is to calculate the magnitude and direction of image gradient. The traditional Canny algorithm adopts limited difference of 22 neighboring area to calculate the value and direction of image gradient. The first order partial derivative's approximation on the X and Y directions can be got from these following formulas:

$$E_x[i,j]=(I[i+1,j]-I[i,j]+I[i+1,j+1]-I[i,j+1])/2$$

$$E_y[i,j]=(I[i,j+1]-I[i,j]+I[i+1,j+1]-I[i+1,j])/2$$

Therefore, the templates of the image gradient calculation operator are:

$$G_x=\begin{pmatrix}-1 & 1\\ -1 & 1\end{pmatrix}$$

$$G_y=\begin{pmatrix}1 & 1\\ -1 & -1\end{pmatrix}$$

The magnitude and direction of gradient can be calculated. The image gradient magnitude is:

$$\|M(i,j)\|=\sqrt{E_x[i,j]^2+E_y[i,j]^2}$$

The azimuth of the image gradient is:

$$\theta(i,j)=\arctan\left(E_y[i,j]/E_x[i,j]\right)$$

2.2.3 Apply Non-Maximum Suppression to Get Rid of Spurious Response to Edge Detection

After acquired the gradient magnitude image M [i, j], it's needed to perform non-maximum suppression on the image to accurately position edges. The process of NMS can help guarantee that each edge is one-pixel width. Canny algorithm uses 33 neighboring area which consists of eight directions to execute interpolation to the gradient magnitude along gradient's direction. If the magnitude M [i, j] is bigger than the two interpolation results on the gradient direction, it will be marked as candidate-edge point, otherwise it will be marked as non-edge point. Therefore, the candidate edge image is acquired through the process.

2.2.4 Apply Double Threshold to Determine Potential Edges

The Canny algorithm adopts double-threshold method to select edge points after carrying on non-maximum suppression. The pixels whose gradient magnitude is above the high-threshold will be marked as edge points, and those whose gradient magnitude is under the low-threshold will be marked as non-edge points, and the rest will be marked as candidate edge points. Those candidate edge points who are connect with edge points will be marked as edge points. This method reduces the influence of noise on the edge of the final edge image.

2.2.5 Track Edge by Hysteresis

Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges. The method of thresholding used by the Canny Edge Detector is referred to as "hysteresis". It makes use of both a high threshold and a low threshold. If a pixel has a value above the high threshold, it is set as an edge pixel. If a pixel has a value above the low threshold and is the neighbor of an edge pixel, it is set as an edge pixel as well. If a pixel has a value above the low threshold but is not the neighbor of an edge pixel, it is not set as an edge pixel. If a pixel has a value below the low threshold, it is never set as an edge pixel. [6]

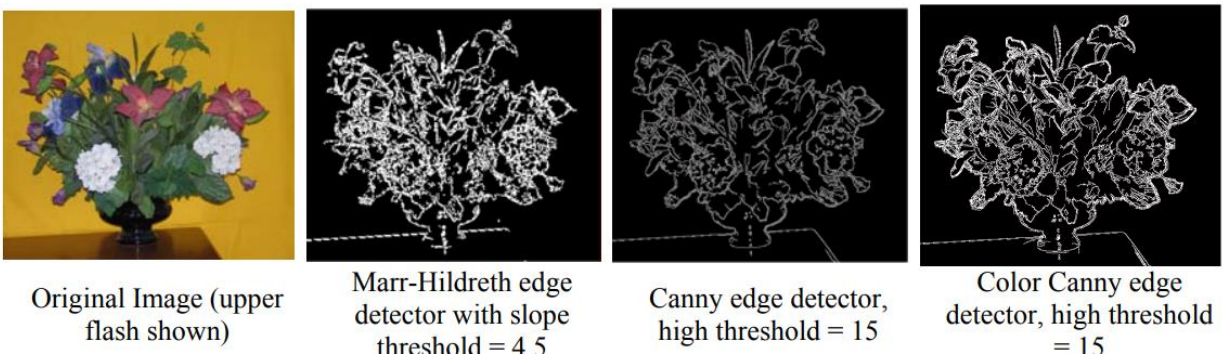

Figure 6 *Canny Edge Detection Example [7]*

2.3 Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains [8]. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs [9]. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. An ANN (Figure 7) is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it. In common ANN implementations, the signal is that a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs. The connections between artificial neurons are called 'edges'. Artificial neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Typically, artificial neurons are aggregated into layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

The original goal of the ANN approach was to solve problems in the same way that a human brain would. However, over time, attention moved to performing specific tasks, leading to deviations from biology. Artificial neural networks have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games and medical diagnosis.

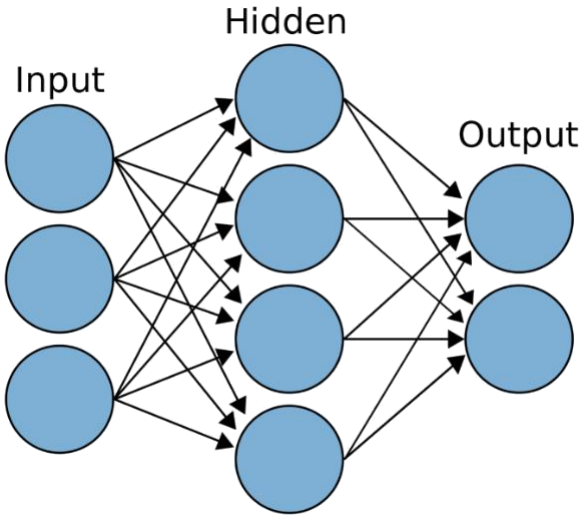

Figure *7* *ANN Architecture*

2.4 Convolutional Neural Networks (CNN)

Convolutional neural networks (CNN) is a special architecture of artificial neural networks, proposed by Yann LeCun in 1988. A convolutional neural network is a class of deep neural networks, most applied to analyzing visual imagery. CNNs use a variation of multilayer perceptions designed to require minimal preprocessing. [10] Convolutional networks were inspired by biological processes [11] in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons

respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. There are numerous ways of connecting artificial neurons together to create a CNN. One such topology that is commonly used is known as a feed-forward network (Figure 8).

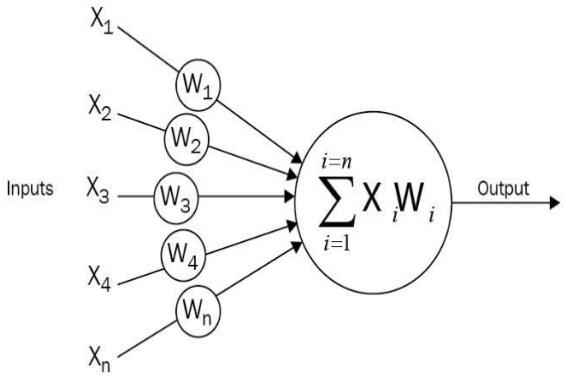

Figure 8 *Feed-Forward Network*

CNNs use relatively little pre-processing compared to other image classification algorithms. CNN has three important building blocks; a convolutional layer that extracts features from the image or parts of an image, a subsampling or pooling layer that reduces the dimensionality of each feature to focus on the most important elements and a fully connected layer that takes a flattened form of the features identified in the previous layers and uses

them to make a prediction about the image.

2.5 Applying CNN

There are many CNN architectures such as Darknet, Alexnet, Vgg to make recognition process. One of the widely used software tools in deep learning is TensorFlow. As an open-source artificial intelligence library, TensorFlow uses data flow charts to create models and allows developers to create multi-layered and large-scale artificial neural networks. With the correct data feed, artificial neural networks with convolutional neural network are capable of detecting the desired image. However, TensorFlow is not that easy to use. Keras is high-level an API built on TensorFlow. Keras follows best practices for reducing cognitive load: it offers consistent and simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error. Also, Keras easy to learn and easy to use. Therefore, in this project we chose TensorFlow Library with Keras.

2.6 Backtracking Algorithm

The "Backtracking" algorithm is the assignment of certain partial candidates for the solution of a problem. When it is noticed that the assigned solution is not suitable, follow-up is done, the remaining candidates are appointed, and this process is repeated until a suitable solution is obtained. Therefore, the "backtracking" algorithm can be used in structures where the partial candidate solution approach can yield effective results. While this condition cannot give an effective result as a sorting algorithm, it can be used effectively in puzzle problems. Examples: include maze puzzle, sudoku puzzle, Constraint satisfaction problems and combinatorial optimization problems. The term "backtracking" was coined by the American mathematician D. H. Lehmer in the 1950s. The first algorithmic definition was made by R.J Walker in 1960. SNOBOL, the pioneering string processing programming

language, became the first language to allow the use of "backtrack". The "backtracking" algorithm tries to find the solution by going through a tree structure in depth, from the root. In this case, if we consider every possible path as a node of the tree, we can consider the depth we are in as each branch leading to a solution. At each correct step, the function calls itself and descends to a new depth, if the node reached does not provide the desired solution, that node is truncated and the paths to the solution will be limited. In this way, the solution of the problem is reached. Below is an illustrated example of this operation [12].

```
procedure EXPLORE(node n)
    if REJECT(n) then return
    if COMPLETE(n) then
        OUTPUT(n)
    for n_i : CHILDREN(n) do EXPLORE(n_i)
```

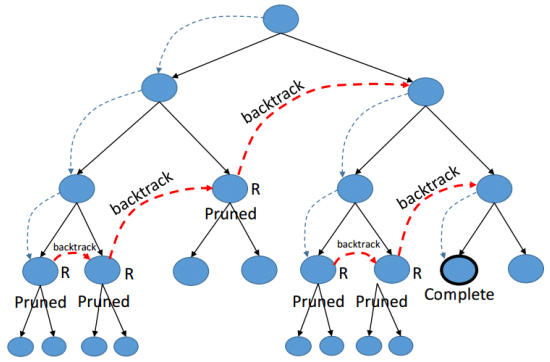

Figure 9 *Analogy of Backtracking to Tree Structure*

While implementing the "backtracking" algorithm with recursion, it is important to fulfill two conditions that are a requirement for recursion. Firstly, the calling of the function itself must be limited by the conditions. For Sudoku, this condition is whether the assigned value is the solution of the subset of the problem. Secondly, Sub-solution sets should take us step by step to the solution of the problem. The shatter-divide-conquer approach is a good example for this.

To understand the Backtracking approach more formally, given an instance of any computational problem ***P*** and data ***D*** corresponding to the instance, all the constraints that need to be satisfied in order to solve the problem are represented by ***C*** [13]. A backtracking algorithm will then work as follows:

The Algorithm begins to build up a solution, starting with an empty solution set ***S***.

1-) Add to ***S*** the first move that is still left (All possible moves are added to ***S*** one by one). This now creates a new sub-tree ***S*** in the search tree of the algorithm.

2-) Check if ***S*** + ***s*** satisfies each of the constraints in ***C***

- If Yes, then the sub-tree ***s*** is “eligible” to add more “children”.
- Else, the entire sub-tree ***s*** is useless, so recurs back to step 1 using argument S.

3-) In the event of “eligibility” of the newly formed sub-tree ***s***, recurs back to step 1, using argument ***S*** + ***s***.

4-) If the check for ***S*** + ***s*** returns that it is a solution for the entire data ***D***. Output and terminate the program. If not, then return that no solution is possible with the current ***s*** and hence discard it.

3. Experiment

The main steps of this project are to develop a CNN model to recognize numbers in cells, train, validate the model and test the model with the test datasets and apply the model on the sudoku puzzle.

3.1 Explore and Visualize the Dataset

The dataset we used to train is the The Chars74K dataset. We only got the numbers from this dataset. A sample of the dataset can be seen in Figure 10-11 the number's images have been pre-cropped, implying that the datasets have labeled numbers in the images. Our dataset consists of 10 number classes and nearly 6000 images that size is 28x28x3. Where 28 is width, next 28 is height and 3 is depth [14].

Figure 10 *A sample of the dataset*

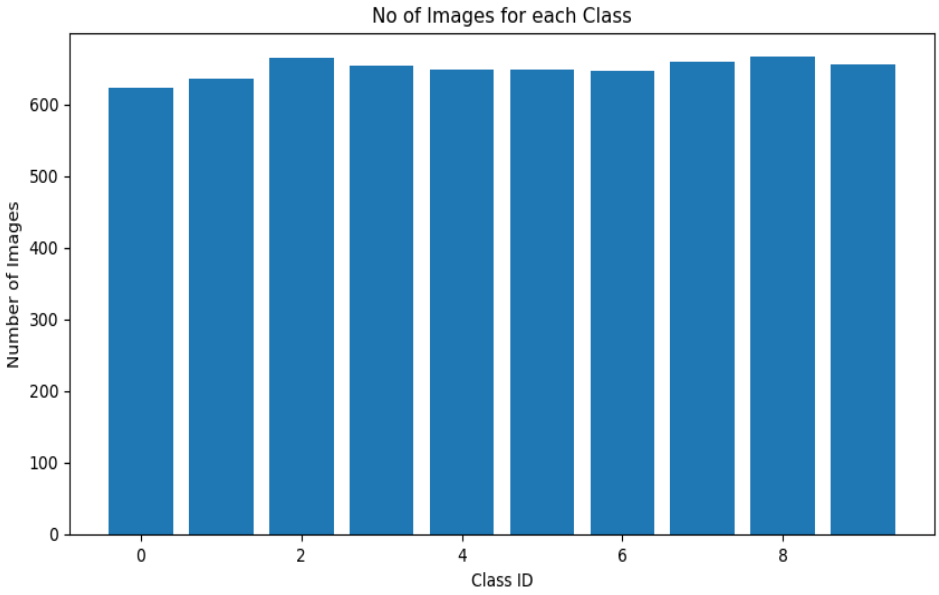

Figure 11 Number of Images for each Class 0 to 9

3.2 Develop a CNN Model

This step is the most important step for our project. The image is passed through a series of convolution, nonlinear(activation), pooling layers and fully connected layers, and then generates the output. This is shown by the CNN code we generated in Figure 12.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 60)        1560
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 60)        90060
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 60)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 30)        16230
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 30)          8130
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 30)          0
_________________________________________________________________
dropout (Dropout)            (None, 4, 4, 30)          0
_________________________________________________________________
flatten (Flatten)            (None, 480)               0
_________________________________________________________________
dense (Dense)                (None, 500)               240500
_________________________________________________________________
dropout_1 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5010
=================================================================
Total params: 361,490
Trainable params: 361,490
Non-trainable params: 0
```

Figure 12 *CNN Structure*

The Convolution layer is always the first. The image is entered into it. Imagine that the reading of the input matrix which is zero-padded (Figure 13) begins at the top left of image. Next the software selects a smaller matrix there, which is called a filter (or neuron, or core). Then the filter produces convolution, i.e. moves along the input image. The filter's task is to multiply its values by the original pixel values. All these multiplications are summed up. One number is obtained in the end. Since the filter has read the image only in the upper left corner, it moves further and further right by 1 unit performing a similar operation.

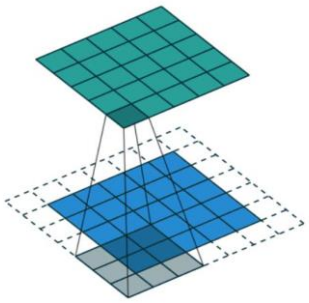

Figure 13 *Basic padding example*

After passing the filter across all positions, a matrix is obtained same input matrix. This operation, from a human perspective, is analogous to identifying boundaries and simple colors on the image. The network will consist of several convolutional networks mixed with nonlinear and pooling layers. When the image passes through one convolution layer, the output of the first layer becomes the input for the second layer. And this happens with every further convolutional layer. The nonlinear layer is added after each convolution operation. There are many activation functions; sigmoid, tanh, ReLU e.g. We choose Rectified Linear Unit (ReLU) function because of computationally efficient and non-linear. Without non-linear property a network would not be sufficiently intense and will not be able to model the response variable (as a class label).

The pooling layer follows the nonlinear layer. It works with width and height of the image and performs a down- sampling operation on them. As a result, the image volume is reduced. This means that if some features have already been identified in the previous convolution operation, then a detailed image is no longer needed for further processing, and it is compressed to less detailed pictures. Indeed, we used batch normalization that allows each layer of a network to learn by itself a little bit more

independently of other layers because it tends to stabilize training and make tuning hyperparameters easier. This means that it can double or triple your training time. Also, ReLU activation is applied along with batch normalization and dropout.

After completion of series of convolutional, nonlinear and pooling layers, it is necessary to attach a fully connected layer. This layer takes the output information from convolutional networks. The output after these 4 blocks is then flattened, and a full connected layer is used to map this flattened feature vector to 10 classes. These 10 classes represent the 10 types of challenge that are to be detected. Finally, a "SoftMax" classifier is added to the network the output of this layer are the prediction values themselves.

Deep neural networks contain multiple non-linear hidden layers, and this makes them very expressive models that can learn very complicated relationships between their inputs and outputs. With limited training data, however, many of these complicated relationships will be the result of sampling noise, so they will exist in the training set but not in real test data even if it is drawn from the same distribution. This leads to overfitting problem. To reduce this potential problem, we used Drop Layer, which developed methods. Dropout's purpose is to help your network generalize and not overfit. Neurons from the current layer, with probability p, will randomly disconnect from neurons in the next layer so that the network has to rely on the existing connections. In the tests, the Dropout rate was chosen 0.5 and the accuracy rate was maximized.

3.3 Train and Validate The Model

When we finish creating our model, we divided our dataset into two parts: The first is the train dataset we use to train, and the second is the validate dataset we use for forecasting. After separating these two datasets, created model was trained and the following validation results were obtained (Figure 14-15):

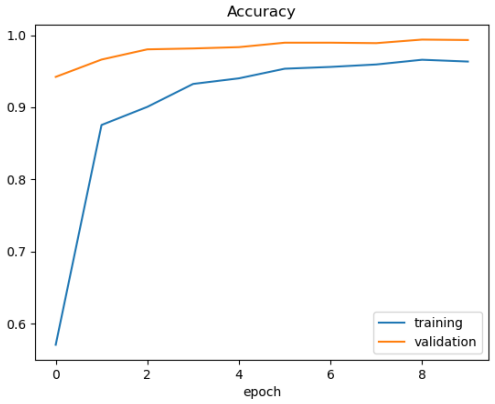

Figure 14 *Accuracy of OCR per Epoch*

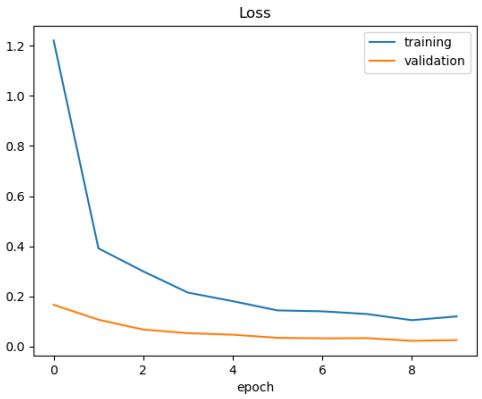

Figure 15 *Losses of OCR per Epoch*

3.4 Sudoku Cell Detection and Seperation

Contour area can be calculated to catch a specific shape in a canny image. This area can be filtered with area thresholds depending on image's resolution to reduce unwanted results. After that, contour's

curve length or perimeter should be calculated to use this value in contour approximation after Contour Approximation approximates a contour shape to another shape with a smaller number of vertices depending upon the precision we specify.

Firstly, our algorithm tries to detect the rows and columns in a sudoku puzzle's canny image (Figure 16). Then, it focuses on smaller cells. When it once detects corners of each cell, our algorithm identifies the box (Figure 17). By calculating the areas of these boxes, we make sure that they are at certain threshold values and we detect 81 cells. Then we crop the positions of these cells from Sudoku Puzzle and send them to the OCR model that we created. We create a 9x9 matrix with the feedback from OCR and send this matrix to the Sudoku Solver Algorithm with "backtracking".

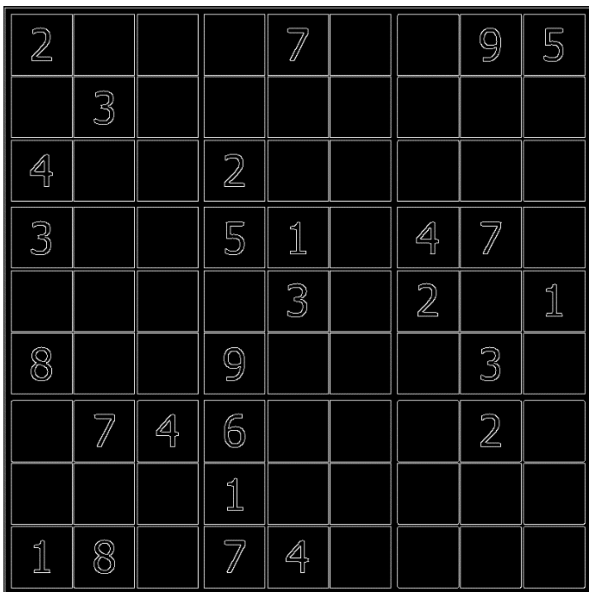

Figure 16 *Canny image of Sudoku Puzzle*

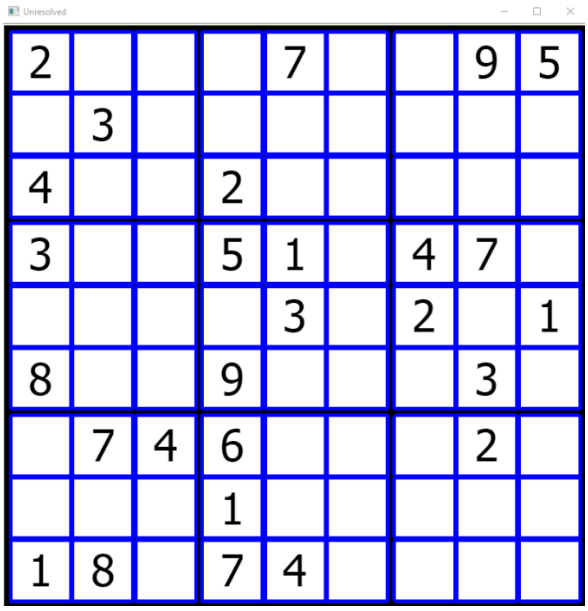

Figure 17 *A Sudoku that all 81 boxes detected*

3.4. Sudoku Solving Algorithm with Backtracking

In pseudocode, our strategy is [15]:

1. Find row, column of an unassigned cell
2. If there is none, return true
3. For digits from 1 to 9
4. if there is no conflict for digit at row, column
5. assign digit to row, column and recursively try fill in rest of grid
6. if recursion successful, return true
7. if not successful, remove digit and try another
8. if all digits have been tried and nothing worked, return false to trigger backtracking

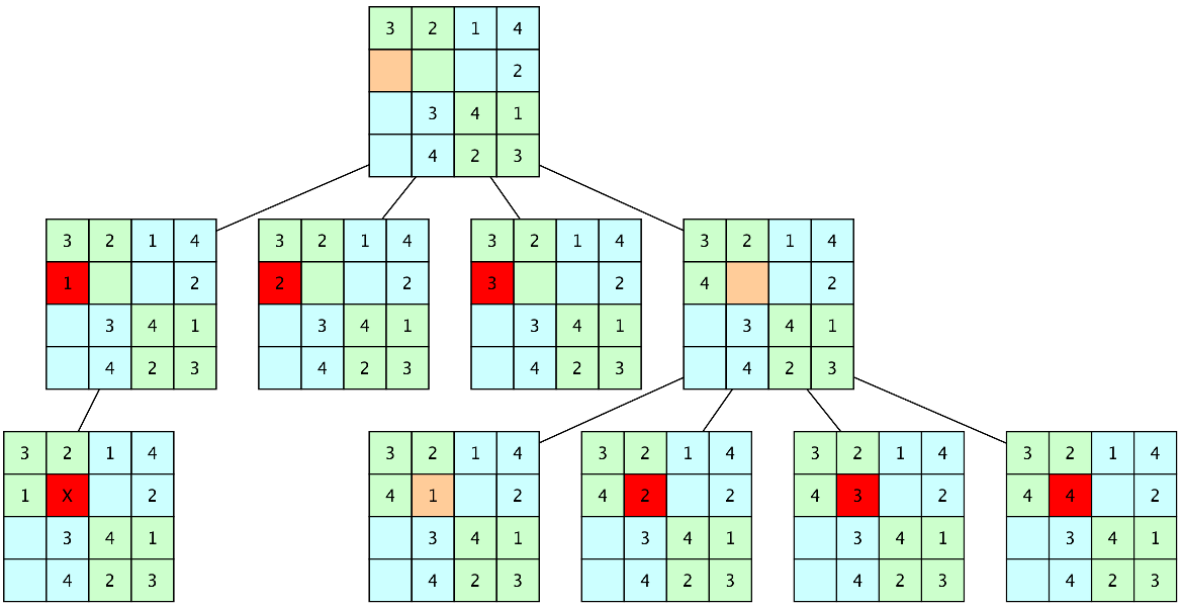

Figure 18 *Sudoku solving with tree structure bases on Backtracking algorithm*

Backtracking algorithms are usually used to solve hard problems – i.e. such that we don't know whether a significantly more efficient solution exists. Usually, the solution space is quite large, and uniform and the algorithm can be implemented so that its time complexity is close to the theoretical lower bound. To get an upper bound it should be enough to check how much additional (i.e. unnecessary) work the algorithm does. The number of possible solutions, and thus the time complexity of such algorithms, is usually exponential – or worse. Time complexity can be given as Big O(9^n*n) [13].

	Number of Steps	Time(ms)	Number of Clues
1st Puzzle	221	2.992	29
2nd Puzzle	2405	37.891	28
3rd Puzzle	1575	21.314	28
4th Puzzle	1241	21.940	29
5th Puzzle	4133	57.842	33
6th Puzzle	4552	66.820	27
7th Puzzle	27368	370.010	24
8th Puzzle	156682	2169.477	25
9th Puzzle	12207091	$0.162e^3$	17
10th Puzzle	176	2.990	29

Figure 19 *Number of steps and relation between time*

4.Conclusion

As a result, certain steps were followed in the Sudoku Solver with Image Processing project. These are the steps below respectively.

First, dataset was used to train the model. Then, using this dataset again, the OCR model, which was designed in Keras, was used to determine the numbers inside the boxes in the Sudoku Puzzle. The determined numbers were placed in a 9x9 matrix and the solution of Sudoku Puzzle was solved by using the "backtracking" algorithm Figure(20-21).

Some important implications have been made.

After increasing the variety of samples in the data in which our model was trained, it was determined that OCR perceived the numbers more accurately, and the algorithm responded faster if the solution set of Sudoku Puzzle was odd. Thus, the lower the known number of digits, the faster the algorithm proved to respond. From another point of view, when the known number increases, the resolution speed of the algorithm slows down (Figure 19).

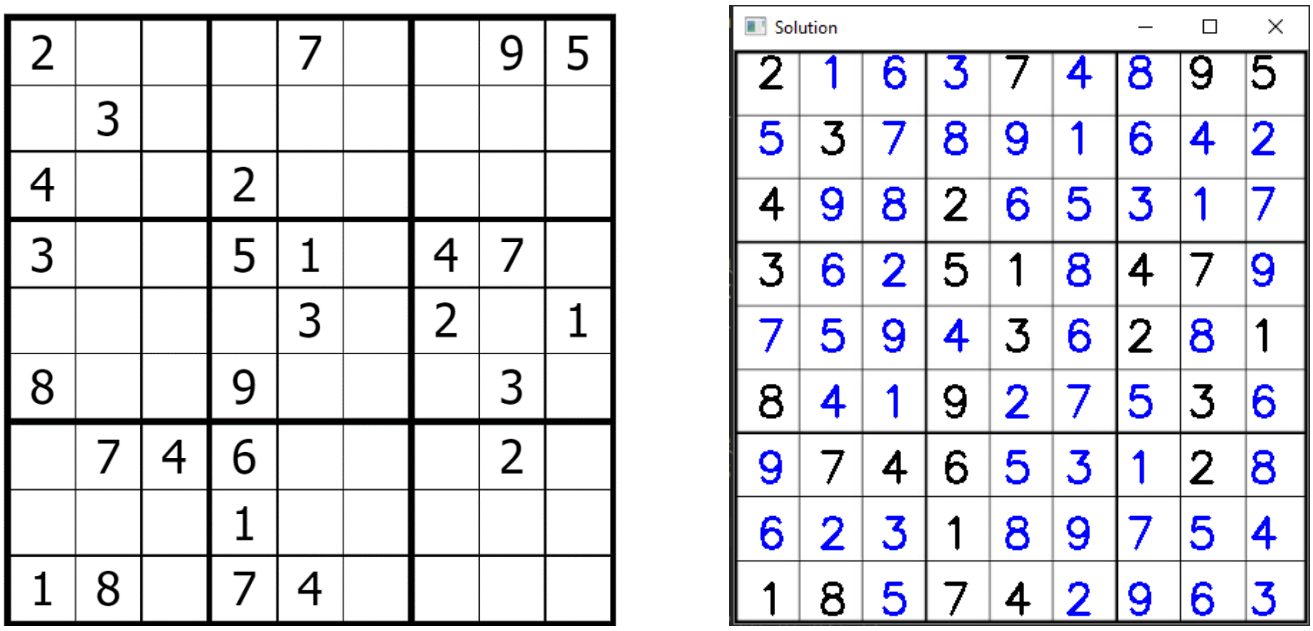

Figure 20-21 *Sudoku puzzle and solution of the puzzle*

5.References

[1] Verghese J, Lipton RB, Hall CB, Kuslansky G, Katz MJ, Buschke H. Abnormality of gait as a predictor of non-Alzheimer's dementia. The New England Journal of Medicine. 2002;347(22):1761–1768. doi:10.1056/NEJMoa020441.

[2] LeCun, Y, Bottou, L, Bengio, Y, and Haffner, P. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, November 1998.

[3] Jarrett, K, Kavukcuoglu, K, Ranzato, M, and LeCun, Y. What is the best multi-stage architecture for object recognition? In Proc. International Conference on Computer Vision (ICCV'09). IEEE, 2009.

[4] Ramer, U. (1972). An iterative procedure for the polygonal approximation of plane curves. Computer Graphics and Image Processing, 1(3), 244–256. doi:10.1016/s0146-664x(72)80017-0]222222

[5] Saalfeld, A. (1999). Topologically Consistent Line Simplification with the Douglas-Peucker Algorithm. Cartography and Geographic Information Science, 26(1), 7–18. doi:10.1559/152304099782424901

[6] Rong, Weibin, et al. "An improved CANNY edge detection algorithm." 2014 IEEE International Conference on Mechatronics and Automation. IEEE, 2014.

[7] Nadernejad, Ehsan, Sara Sharifzadeh, and Hamid Hassanpour. "Edge detection techniques: evaluations and comparisons." Applied Mathematical Sciences 2.31 (2008): 1507-1520.

[8] "Artificial Neural Networks as Models of Neural Information Processing | Frontiers Research Topic". Retrieved 2018-02-20.

[9] "Build with AI | DeepAI". DeepAI. Retrieved 2018-10-06.

[10] LeCun, Yann. "LeNet-5, convolutional neural networks". Retrieved 16 November 2013

[11] Matsugu, Masakazu, et al. "Subject independent facial expression recognition with robust face detection using a convolutional neural network." Neural Networks 16.5-6 (2003): 555-559.

[12] Baker B. Andrew,Intelligent Backtracking on Constraint Satisfaction Problems: Experimental and Theoretical Results,March 1995

[13] https://www.geeksforgeeks.org

[14]http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/

[15] Julia Zelenski,CS106B Handout #19,February 2008