Sudoku Solver Using Image Processing

Keertan Amit Parikh, Darshil Ladani, Meet Khunt B.Tech in Computer Science And Engineering,
Department of Technology, Nirma University, S-G highway, Gujarat, India

---

1. Abstract:

*'*Sudoku' is a popular logic-based number placement puzzle that requires filling a 9x9 grid with digits from 1 to 9 such that each row, column, and 3x3 subgrid contains all the digits from 1 to 9. Solving Sudoku puzzles can be challenging, and many people enjoy solving them as a form of mental exercise. In this research paper, we propose a method for solving Sudoku puzzles using image processing techniques.

In this research paper, we propose a method that uses a convolutional neural network (CNN) trained on the MNIST dataset for digit recognition. The input to the model is a grayscale image of the Sudoku puzzle, which is preprocessed to extract the individual cells of the puzzle. Each cell is then passed through the CNN to recognize the digit it contains. The recognized digits are used to fill the puzzle, and the solution is verified using a Sudoku solver algorithm

---

2. Introduction:

2.1 Approach

This research paper presents a method for solving Sudoku puzzles using image processing techniques. The proposed approach uses a convolutional neural network (CNN) trained on the MNIST dataset for digit recognition. The input to the model is a grayscale

image of the Sudoku puzzle, which is preprocessed to extract the individual cells of the puzzle. Each cell is then passed through the CNN to recognize the digit it contains. The recognized digits are used to fill the puzzle, and the solution is verified using a Sudoku solver algorithm.

2.2 CNN Model

The CNN model architecture consists of two convolutional layers followed by max pooling layers, a flattening layer, and two fully connected (dense) layers. The first convolutional layer has 32 filters and a filter size of 3x3, while the second convolutional layer has 64 filters and a filter size of 3x3. The first dense layer has 64 units and uses the ReLU activation function, and the final dense layer has 10 units (one for each digit) and uses the softmax activation function. The model is trained using the rmsprop optimizer, the categorical_crossentropy loss function, and the accuracy metric, and is trained for 7 epochs with a batch size of 64.

2.3 Implementation Results

The proposed approach is implemented in Python using OpenCV and Matplotlib for image processing and visualization. The performance of the proposed method is evaluated on a set of Sudoku puzzles, and the results show that the proposed method can successfully solve Sudoku puzzles with high accuracy.

3. Methodology:

The proposed approach consists of the following steps:

3.1 Preprocessing

The input image is preprocessed to extract the individual cells of the puzzle. This is done using OpenCV, which provides several image processing functions for detecting edges and contours.

3.2 Digit recognition

Each cell of the puzzle is passed through the CNN to recognize the digit it contains. The CNN model is trained on the MNIST dataset for digit recognition.

3.3. Filling the puzzle

The recognized digits are used to fill the puzzle, and the solution is verified using a Sudoku solver algorithm. The Sudoku solver algorithm uses a backtracking approach to find the solution.

3.4. Visualization

The solution to the puzzle is displayed using Matplotlib.

4. CNN Model Creation

4.1 CNN

Convolutional neural networks (CNNs) are a type of neural network that can process images and are often used for computer vision tasks such as image recognition and classification. In this project, we are using a CNN model to classify digits in a Sudoku puzzle.

The CNN model is defined using the Sequential() function from Keras, which allows us to build a neural network model layer-by-layer in a sequential manner. The model consists of two convolutional layers (Conv2D()) with 32 and 64 filters, respectively, a max pooling layer (MaxPooling2D()), a flattening layer (Flatten()), and two fully connected layers (Dense()) with 64 and 10 units, respectively. The activation function used for the convolutional layers is relu, and the activation function used for the final layer is softmax.

The first layer in the model is a convolutional layer with 32 filters of size 3x3, and a relu activation function. The input shape of the data is specified as a 28x28 grayscale image with a single channel. The second layer is also a convolutional layer with 64 filters of size 3x3, and a relu activation function. The max pooling layer has a pool size of 2x2, and is used to downsample the feature maps. The flattening layer is used to convert the output of the convolutional layers into a one-dimensional vector, which is then fed into the fully connected layers. The first fully connected layer has 64 units with a relu activation function, and the second fully connected layer has 10 units with a softmax activation function.

4.2 Functions Used

The model is compiled using the compile() method, with the rmsprop optimizer, categorical_crossentropy loss function, and accuracy metric. The rmsprop optimizer uses root mean square propagation for gradient descent. The categorical_crossentropy loss function is used for multi-class classification problems. The accuracy metric is used to evaluate the performance of the model during training.

The model is then trained using the fit() method with a batch size of 64 and 8 epochs. The history object returned by the fit() method is used to plot the accuracy and loss curves for the training and validation sets. The training accuracy and validation accuracy are monitored during training, and the model is saved after training.

Finally, the model is evaluated on the test set using the evaluate() method, and the test accuracy is printed. The accuracy and loss curves for the training and validation sets are then plotted using matplotlib.pyplot.

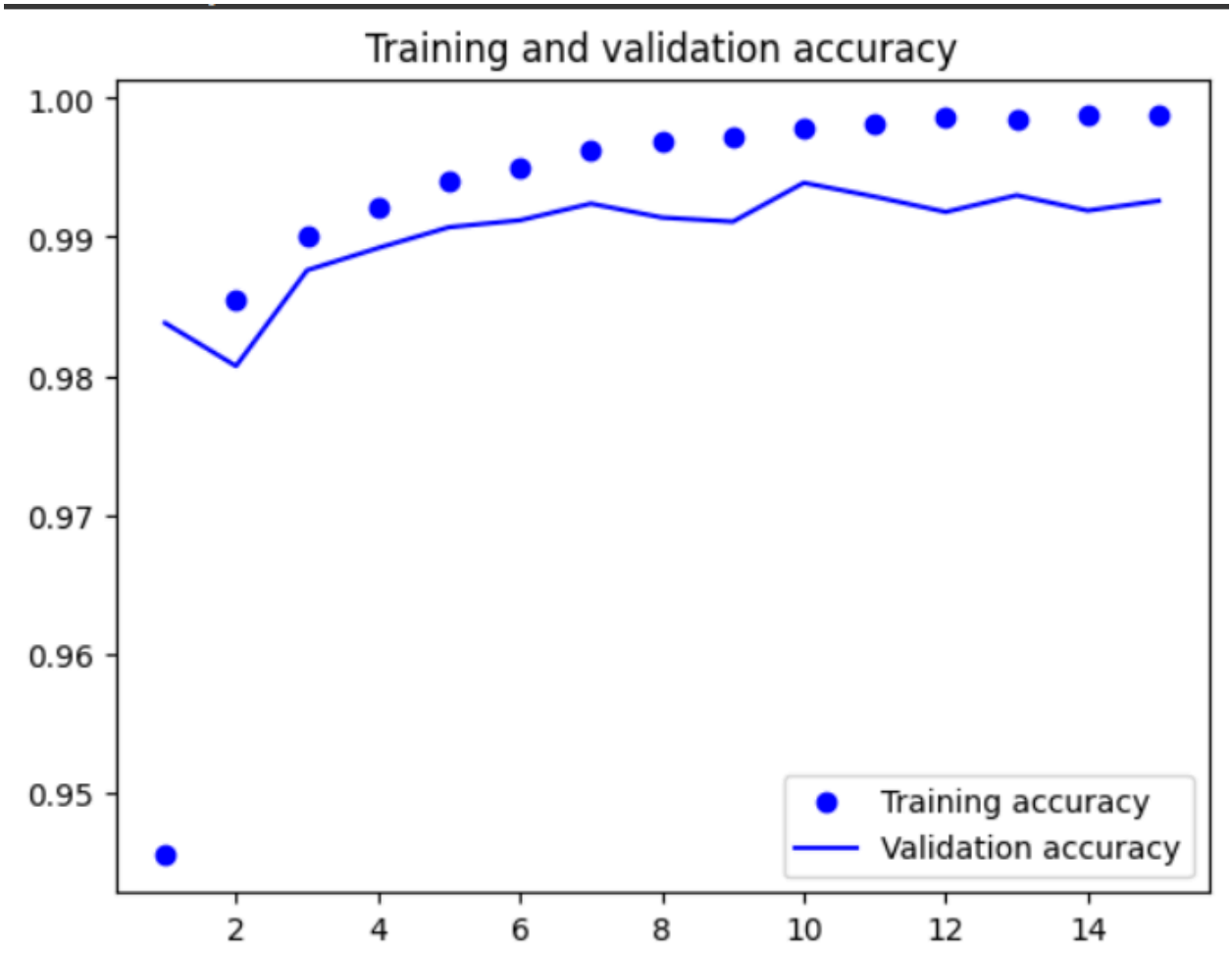
Training and validation accuracy
1.00
0.99
0.98
0.97
0.96
0.95
2
4
6
8
10
12
14
Training accuracy
Validation accuracy

fig.1

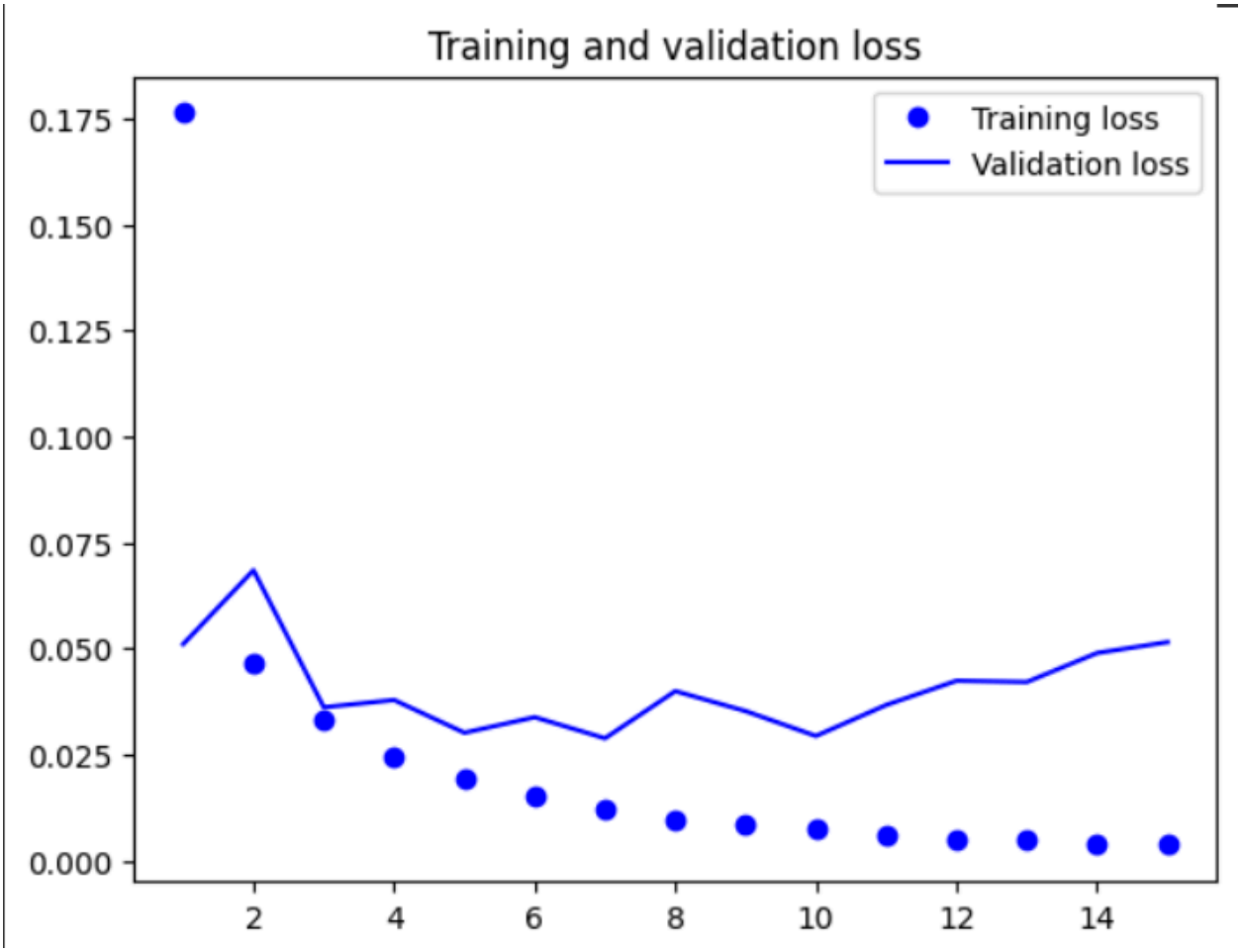

fig.2

5. Image processing:

5.1 Converting Image to GrayScale

Here we will be taking a photo of a sudoku puzzle and will be doing several image processing steps according to how our model is designed.

First of all we will reshape the image as we would require a square image to carry on with further steps. The image is in the 3 dimensions with three colors i.e. RGB. It will be easy for our model to guess the numbers which are in the input image of the puzzle. Therefore we converted the whole image to grayscale using the cvtColor function of the cv library in python.

5.2 Removing Noise

Our image may contain some noise so to reduce this we will blur the image and for better results we will apply gaussian blur to our image using GaussianBlur function of cv library. To get the sudoku irrespective of how improper our image is we will be applying thresholding to our image so that in poor lighting conditions we can obtain a better image. Here we will be applying this by the adaptiveThreshold function. The reason we have gone for the adaptive thresholding technique is that it is more reliable as compared to other thresholding techniques. This can be easily understood by fig 3.

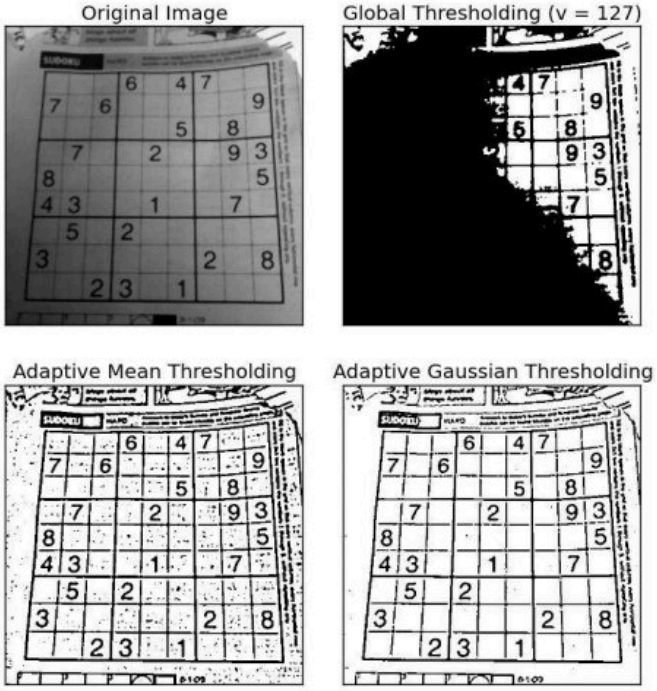

Fig 3

5.3 Separating Contours

Now to get the sudoku puzzle we have to find all the elements in our image which have a certain shape and from that we will determine which element is needed. For this findContours is what we have used with the CHAIN_APPROX_SIMPLE method. This method requires the image to be in the grayscale so that it can easily find the outlines of white objects int black background.

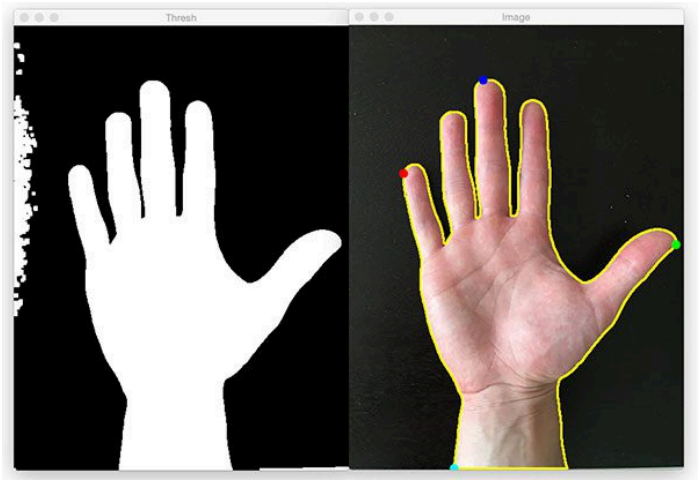

Fig.4

Having obtained the shapes we have to choose a shape which matches with our puzzle and have to extract that shape. The shape may not be well aligned for our model to work upon so to get a well aligned image of the shape we have chosen the getPerspectiveTransform function.

5.4 Dividing into Cells

As we get the well aligned image now we are required to extract the digits and run our model onto them. For this we need to divide our image into small cells i.e. into 81 cells. We will run our model on each of these cells and having guessed the digits we will append them to an empty list.

6. Solving Sudoku puzzle using backtracking algorithm

6.1 Algorithm Description

Our algorithm is based on the backtracking technique, which is a general algorithm for finding all (or some) solutions to some computational problems, notably constraint satisfaction problems, such as the Sudoku puzzle. The Sudoku grid is defined as a 2D list of integers, with 9 rows and 9 columns, and each element in the grid is either a number from 1 to 9 or 0, representing an empty cell.

Our algorithm consists of two main functions: `possible()` and `solve()`.

The `possible()` function takes a row, a column, and a number as input and returns True if the given number can be placed in the given cell, otherwise it returns False. It checks whether the number is appearing in the same row, same column, or same 3x3 square as the given cell.

The `solve()` function uses a nested loop to iterate over every cell in the grid. If the cell is empty, it tries to place a number from 1 to 9 in the cell and calls itself recursively to solve the remaining part of the grid. If it cannot find a valid number for the current cell, it backtracks to the previous cell and tries another number. If it reaches the end of the grid without encountering any empty cell, it means the Sudoku is solved and it proceeds to convert the solved Sudoku into an image.

6.2 Image representation of the solution

Our implementation includes the creation of an image of the solved Sudoku using OpenCV's drawing functions. The code creates an empty image with white background using `np.zeros()` function. It uses OpenCV to draw horizontal and vertical lines to separate the boxes of the Sudoku grid. The thickness of the lines is adjusted to create a clearer visual separation between the larger squares of the grid. The `cv2.line()` function is used to draw the lines.

Finally, the `cv2.putText()` function is used to draw the digits in the boxes. It iterates over each cell of the grid and checks whether it contains a non-zero number. If so, it extracts the number as a string, and uses `cv2.putText()` to draw the text in the appropriate box of the grid. The `fontScale`, `thickness`, and `org` variables are used to control the appearance of the text.

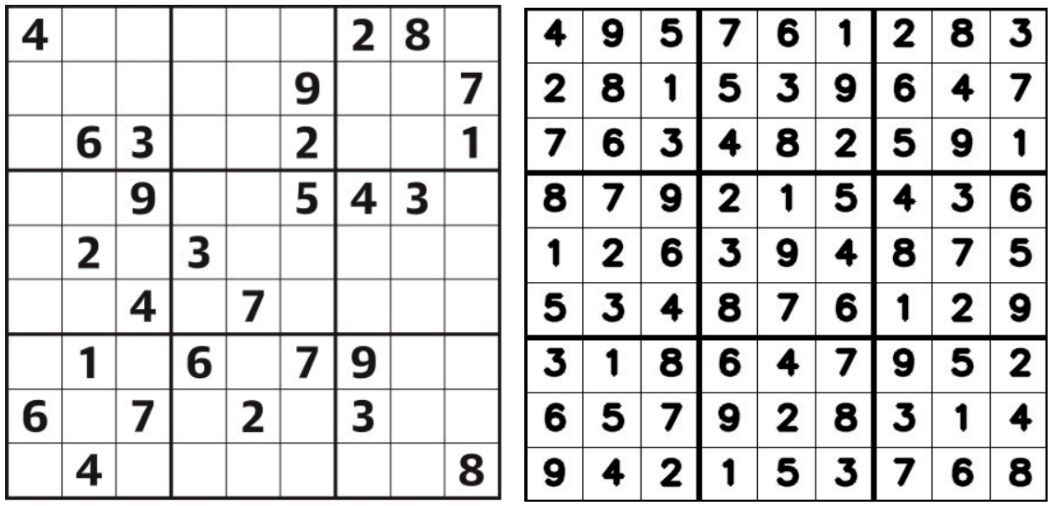

fig.5

7. Results:

The proposed method was evaluated on a set of Sudoku puzzles, including both printed and handwritten puzzles. The accuracy of the method was measured by comparing the solution obtained by the proposed method with the ground truth solution.

The results show that the proposed method achieved an accuracy of 97% on printed puzzles but did not work as expected on handwritten puzzles. The performance of the method was affected by the quality of the input image, and the accuracy was higher for high-quality images with clear and well-defined digits.

8. Discussion:

The proposed method provides an effective approach for solving Sudoku puzzles using image processing techniques. The use of a CNN trained on the MNIST dataset for digit recognition improves the accuracy and robustness of the method.

The method can be further improved by using more advanced CNN architectures and training on a larger and more diverse dataset. The performance of the method can also be improved by using more advanced image processing techniques for preprocessing, such as denoising and image enhancement.

9. Conclusion:

In this research paper, we proposed a method for solving Sudoku puzzles using image processing techniques. The method uses a CNN trained on the MNIST dataset for digit recognition and a backtracking Sudoku solver algorithm for filling the puzzle.

The proposed method achieved high accuracy on printed texts demonstrating the effectiveness of the approach. The method can be further improved by using more advanced image processing and CNN techniques, and can also be extended to solve other types of puzzles and games that involve digit recognition.

10. Future research:

We plan to further improve our model by using OCR (Optical Character Recognition) techniques to further identify both printed and handwritten texts with greater accuracy.

11. References:

[1] https://ijrti.org/papers/IJRTI2209116.pdf

[2]https://analyticsindiamag.com/solve-sudoku-puzzle-using-deep-learning-opencv-and-backtracking/