Integrating of Image Processing and Number Recognition in Sudoku Puzzle Cards Digitation

Pu-Sheng Tsai[1], Ter-Feng Wu[2*], Jen-Yang Chen[1], Jen-Feng Huang[3]

[1] Department of Electronic Engineering, Ming Chuan University, Taiwan
[2] Department of Electrical Engineering, National Ilan University, Taiwan
[3] Department of Electronic Engineering, China University of Science and Technology, Taiwan
pusheng@mail.mcu.edu.tw, tfwu@niu.edu.tw, jychen@mail.mcu.edu.tw, gofon.gofon@msa.hinet.net

Abstract

The aim of Sudoku puzzle is to fill in the blank cells in a square matrix consisting of nine 3x3 blocks with the numbers 1-9 in a way that no number appears more than once in any row, column, or block. We combined image processing, a convolutional neural network (CNN), and a Sudoku game algorithm to automatically place the numbers 1-9 in the blank cells of a Sudoku square matrix. An image of the Sudoku square matrix is first captured using a camera, and then the vertical lines, horizontal lines, and outer frame of the Sudoku square matrix are detected using Hough transform (HT). Based on the OpenCV module, we proposed an image processing algorithm that captures the numbers in the image and calculates the location coordinates of the numbers in the image. We trained the CNN using the MNIST handwritten digit dataset to recognize the numbers in the Sudoku square matrix. Finally, we used the Python programming language to design a Sudoku puzzle backtrace algorithm that automatically deduces and fills in the blank cells in the square matrix. This study provides further understanding of the critical operating principles of CNNs and lays down a foundation for future research.

Keywords: Sudoku, Convolutional neural network, Hough transform, MNIST dataset, OpenCV

1 Introduction

Sudoku, meaning “number place” in its original language, is an intellectual game involving logical thinking to fill in numbers. The rules of the game are simple, and it requires only a pen to write the numbers 1-9 on a paper card. No other equipment is needed. The game swept the world over two decades ago and remains one of the most popular games. Sudoku involves a 9×9 grid, that is, a number puzzle with nine rows and nine columns containing a total of 81 cells. The grid also comprises nine 3×3 subgrids, also known as “blocks”, each of which consists of nine cells. At the beginning of the game, a portion of the numbers is already filled in as hints. This is called the “base grid”, as shown in Figure 1. The objective is to fill the cells with the numbers 1 through 9 so that each number appears only once in each row, column, and block [1]. Each base grid has a single solution, and for this reason, it is called Sudoku, which means “single number”. The completed Sudoku grid with all of the cells filled is called the “solution grid”, as shown in Figure 2, which means that this level has been successfully cleared [2].

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

Figure 1. Base grid of Sudoku

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

Figure 2. Solution grid of Sudoku

We adopted the Hough transform (HT) to search for the lines bordering the rows, columns, and frames in the image of a Sudoku game and then covered them using white lines so that the base grid looked like the grid in Figure 3. HT was first proposed by Paul Hough [3] in 1962 and then promoted by Duda and Hart [4] in 1972. HT is a feature extraction technique in image processing and is a way of detecting geometric shapes within images. It maps feature points in the image space to a parameter space for voting, and by testing the regional extreme points of the accumulated results, a set of points that fit a certain shape is obtained. For instance, curves or straight lines with the same shape in a space are mapped to

*Corresponding Author: Ter-Feng Wu; E-mail: tfwu@niu.edu.tw
DOI: 10.53106/160792642022122307012

a point in the parameter space to form peak values, thereby converting a problem in arbitrary shape detection into a problem in peak value statistics. The classical HT is used to detect straight lines in images. HT was later expanded to identify circles [5], ellipses [6], and even arbitrary shapes.

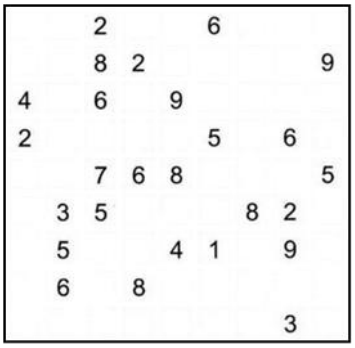

Figure 3. Image of covered lines

HT has always held an important place in image processing because it perfectly combines image data and mathematical geometry. It uses the cold and massive pixel data in the image space, maps the data to the parameter space using mathematical deduction, and accumulates the local maxima via voting to obtain feature parameters that fit certain shape points. For instance, lines such as slopes and intercepts require a two-dimensional parameter space. Detecting circle patterns, including the center and radius, requires a three-dimensional parameter space. Detecting an ellipse, including the center, radius, and the major and minor axes, requires a five-dimensional parameter space. Clearly, the conventional HT would require massive spatial complexity when it comes to detecting lines, circles, and ellipses. Voting in a multidimensional parameter space requires tremendous memory space and imposes a heavy computing load. Random Hough transform (RTH) [7] and gradient Hough transform [8] are basically modifications of the conventional HT, developed to reduce the computing load. For instance, the basic thought process of using conventional HT to detect circles is based on the premise that all of the non-zero pixels in the image are believed to be potential points on the circle. Thus, all non-zero pixels must be mapped via the parameter space to form a parameter trajectory, and votes are then accumulated to determine the local maxima of the three-dimensional space parameters. The maxima can be considered the center and radius of the detected circle. RHT first analyzes geometric properties to determine candidate circles and avoids wasting time on accumulating votes for each non-zero pixel. This effectively shortens the processing time and reduces the amount of memory storage space needed.

In the field of image processing, numbers or letters are often the targets of detection and capture, such as license plates [9], invoice numbers, checks, or check digits. There are a number of ways to identify digits and letters in images. Akhtar (2020) [10] proposed an approach to processing vehicle license plates; after an image is captured using a camera, it is subjected to a process of four steps: pre-processing, plate positioning, digit segmentation, and digit recognition. Pre-processing augments the image to facilitate subsequent processing. Plate positioning extracts the license plate region from the image. Digit segmentation separates the individual digits from the extracted plate region. A random forest classification algorithm then identifies the digits. Experimental results have indicated that this approach could reach an accuracy rate of 90.9%. Regarding the image segmentation methods in image processing, histogram-based methods are highly efficient, needing only to scan an entire image once and total the pixels in all of the rows (x coordinates) to establish a histogram of the image before classifying the peaks and valleys in the image. The contour extraction algorithm proposed by Suzuki (1985) [11] is also an effective approach, often used to capture the digits in images and screened using the area of the contour. For instance, all 81 cells can be obtained in the grid displayed in Figure 1, and any black pixels in each cell are then detected. If the pixels are greater than the set threshold (eliminating noise), then whether a digit exists within the cell can be confirmed. Finally, contour extraction is again ap-plied to the cells containing digits to obtain the digital contours and contour bounding rectangle [12].

Machine learning (ML) and deep learning (DL) were both derived from artificial intelligence. The relationships among the three can be expressed using the yin-yang symbol in Figure 4. The circle of the yin-yang symbol is divided into two interlocking “fish”, called the yin fish and the yang fish, by an S-like curve serving as the boundary. The head of the yang fish contains a yin eye, and the head of the yin fish contains a yang eye, signifying that mutual conversion exists among all things. This also means that there is yang in yin and yin in yang in this symbol, conveying that a mutual trade-off also exists among all things. This figure indicates that not only does a mutual trade-off exist between ML and DL during their development processes, but there is also mutual conversion.

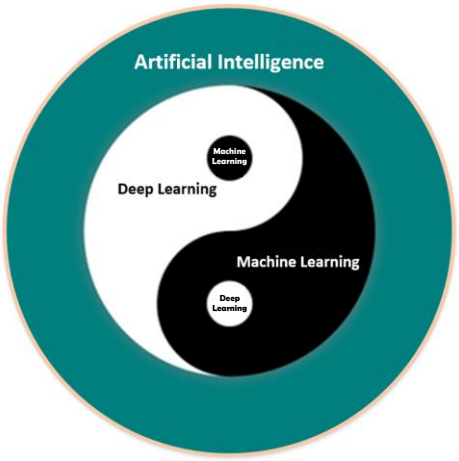

Figure 4. The relationship of ML/DL

In the 1980s, research on artificial neural networks stagnated due to limitations in the computing capabilities of computers. Research on ML, in contrast, became extremely fruitful. Nine classic algorithms, including linear regression, logistic regression, support vector machine (SVM), k-nearest neighbors (KNN), k-means, naive Bayes classifiers, decision trees, random forest, and gradient boosting trees (GBT) [13], were all proposed during this period. In recent years, AI has led to the rapid enhancement and popularization of computer hardware, the internet, various sensors, cloud computing, and big data. After NVIDIA created Compute Unified Device Architecture (CUDA), computers were able to perform rapid

calculations using graphics processing units (GPUs), thereby enabling DL to flourish.

Minsky (1969) [14] proved that in essence, single-layer sensors are just one type of linear model and cannot accurately classify (e.g., XOR classification) nonlinear problems. Rumelhar and Hinton (1986) [15] proposed multilayer perceptrons (MLPs), adding a hidden layer between input layers and output layers, as shown in Figure 5. Using MLPs to construct backpropagation (BP) algorithms can effectively solve the problem of nonlinear classification and learning. Hecht-Nielsen (1989) [16] demonstrated that the continuous functions in any closed interval can be approximated using a BP net-work containing a hidden layer. LeCun (1989) [17] proposed the convolutional neural network (CNN) framework LeNet-5, which uses BP for training. Reference [18] used the LeNet-5 model to establish a DL machine that can recognize handwritten text and successfully applied it to a check recognition system, again demonstrating the feasibility of the CNN framework. Salakhutdinov et al. (2006) [19] proposed the Restricted Boltzmann machine (RBM) model, and Hinton et al. (2006) [20] proposed the Deep Belief Network (DBN), in which multilayer neural networks were successfully trained. This was officially named deep learning (DL).

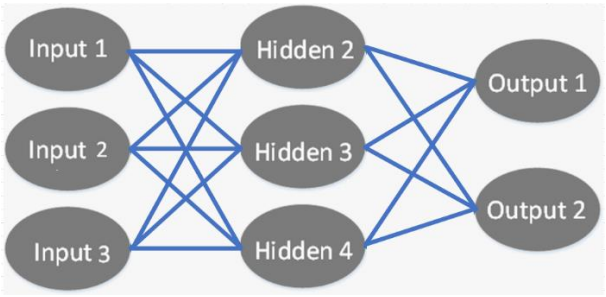

Figure 5. A multilayer perceptron (MLP)

Yann LeCun, the founding father of CNNs, was the first to use CNNs and the Modified National Institute of Standards and Technology (MNIST) database [21] to achieve handwritten digit recognition. The MNIST dataset, collected by LeCun et al., comprises 60,000 items of training data and 10,000 items of test data. Each data item in the MNIST dataset contains images (images of digits) and labels (the true digits). Each digit image consists of 28×28 pixels, which equals 784 pixels. Figure 6 shows some examples in the MNIST dataset. To reduce data collection time and accelerate the application needs of real-time recognition, we used the MNIST dataset and computer fonts that we established ourselves (shown in Figure 7) as samples for neural network training.

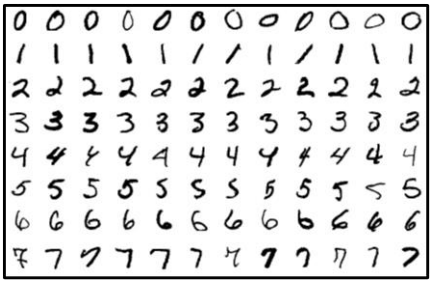

Figure 6. MNIST handwritten dataset

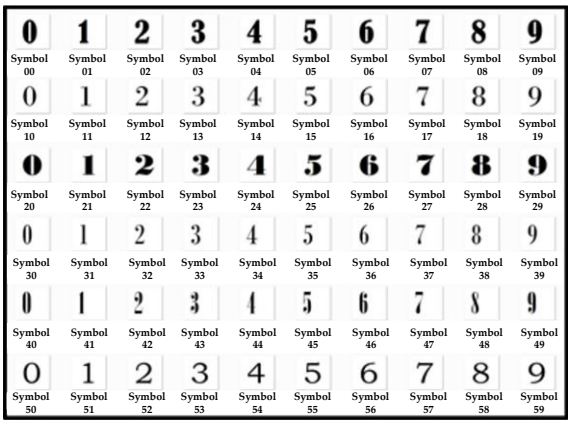

Figure 7. Self creating computer font dataset

CNNs imitate the way that the human brain works and are neural networks consisting of multiple layers. An entire CNN mainly consists of several parts: convolution layers, pooling layers, and fully connected layers [22]. The purpose of the convolution layers is to preserve the spatial arrangements of the image and obtain local images to serve as input features, which then take the CNN model from point comparison to two-dimensional local comparisons. Via the gradual stacking and comprehensive comparison of the features in a region, better recognition results can be derived. The purpose of the pooling layer is to compress images while preserving the important in-formation in the images, which can gradually eliminate redundant information and lighten the computing load on the neural network. The principle of convolutions is to slide a window with designated dimensions from left to right and from top to bottom in an orderly fashion to obtain the sum of the products of local regions in the images. This sliding window is referred to as a convolution kernel, also known as a filter, which can obtain features in the images. In Figure 8, for instance, there are five "▚" feature symbols that can be detected by the 3×3 kernels (or filters) in the image. With the original image, window sliding generates the extracted feature map, which contains the locations of the "▚" features. The operations of CNN convolution layers involve two important characteristics: local connectivity and weight sharing, which can reduce the computing load of the neural network [23]. The purpose of the pooling layer is to obtain the maximum value wherever the sliding window passes to achieve image downsampling so that the original image is scaled down n times, thereby shortening the time needed for subsequent calculations. Figure 9 illustrates that the pooling results enable the CNN to only consider whether recognition features appear in the image, regardless of their locations. The "▘" feature appears in the upper-left corner, the lower-right corner, and the middle of the image. After calculations are performed for the convolution layer and pooling layer, ❹ is similarly derived, meaning that the feature can be detected wherever it appears [24].

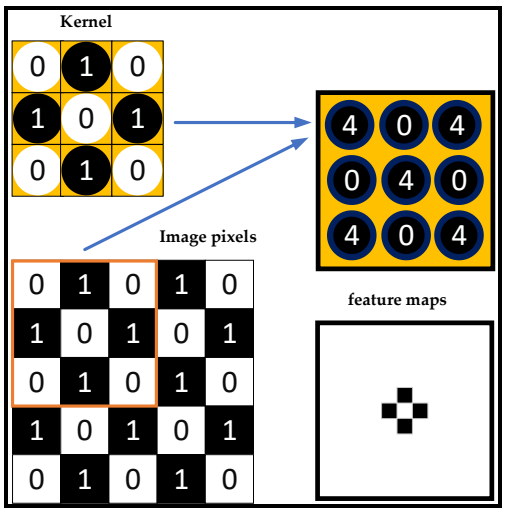

Figure 8. Convolution layer extracting image features

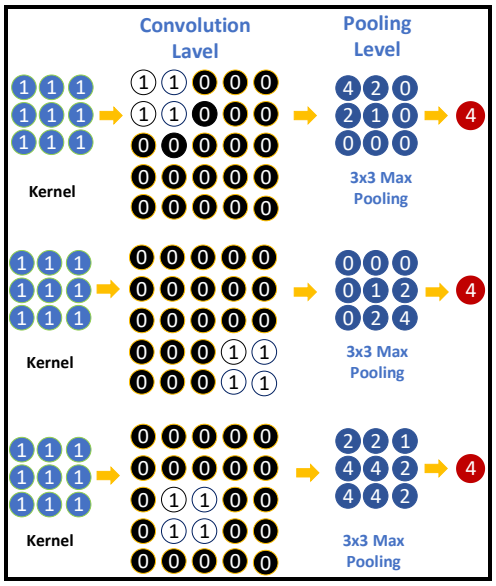

Figure 9. Characteristics of pooling layer

2 Research Methods

2.1 System Structure

This paper combined HT, CNN, and a Sudoku game algorithm to design a system that automatically places the numbers 1-9 in the blank cells in the nine blocks of a Sudoku game card. It then completes the solution and automatically displays it on the computer screen. The structure of the system is shown in Figure 10. Using a webcam, the image of a Sudoku puzzle card is first captured, as shown in Figure 10. In a Python platform environment, the image undergoes preprocessing such as grayscale conversion and binarization to produce the image of a Sudoku square matrix (base grid).

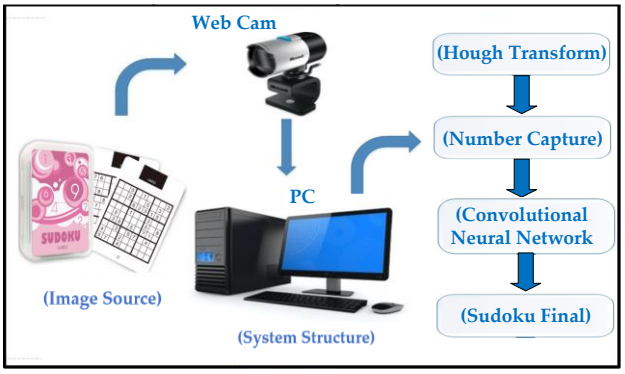

Figure 10. Diagram of system structure

Using HT, the vertical and horizontal lines within the Sudoku square matrix and the outer frame are obtained. White lines are used to cover these lines so that only the numbers remain. Based on the OpenCV module, we proposed an image processing algorithm for digit capture to capture each of the numbers in the image and calculate their coordinates in the image. The captured numbers remain in jpg format and are stored in a designated disk directory. Using the Python tensorflow keras package, we trained the CNN with the MNIST handwritten dataset. Once the parameter training in the CNN is complete, it automatically reads the images of numbers in the designated disk directory for recognition. Finally, we used the Python programming language to design a Sudoku game solution method that automatically deduces and fills in the blank cells in the square matrix.

2.2 Hough Transform (HT)

In real-life applications, Cartesian coordinate systems cannot be chosen for image spaces because any special lines that are perpendicular to the x-axis in the image space, such as $x = c$, will have infinite slopes that cannot be expressed in the parameter space. Using the polar coordinates (ρ, θ) to process HT problems is therefore the best choice as it avoids the issue of an infinite slope. Here, we used the Hesse normal form to express the polar coordinate equation and define a straight line, as shown in Figure 11. The ρ represents the vertical distance between this straight line and the origin O, and θ indicates the angle between the x-axis and the vertical line. From the figure, we can obtain the following relationship:

$$\begin{aligned}\rho &= \alpha\cos\theta = (x+\beta)\cos\theta = (x + y\tan\theta)\cos\theta \\ &= x\cos\theta + y\sin\theta.\end{aligned} \quad \textbf{(1)}$$

There are roughly three types of straight lines that may appear in the image space: (1) lines with negative slopes and positive intercepts, (2) lines with positive slopes and positive intercepts, and (3) lines with positive slopes and negative intercepts, all of which are respectively shown in Figure 12(A), Figure 12(B), and Figure 12(C). Eq. (1) shows how polar coordinates are expressed using the Hesse normal form, where represents the vertical distance between the straight line and the origin O, and θ indicates the angle between the x-axis and the vertical line in the clockwise direction. As shown with the straight line in Figure 12(A), $\rho > 0$ and $0 < \theta < 90^o$. With the straight line in Figure 12(B), $\rho > 0$ and $90^o < \theta < 180^o$. However, with the straight line in Figure 12(C),

$\theta < 180^o$. To keep θ within 180 degrees, such lines are set as $\rho < 0$ and $90^o < \theta < 180^o$.

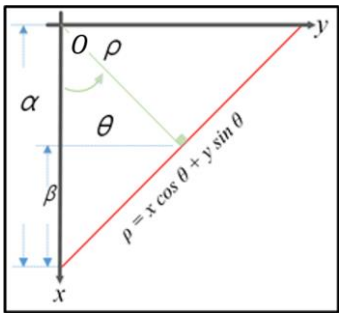

Figure 11. Expression using Hesse normal form

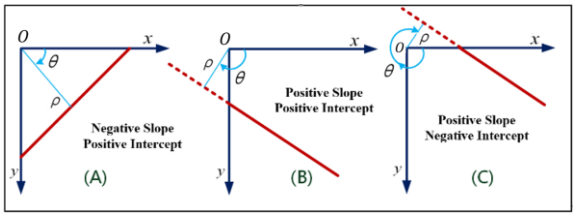

Figure 12. Types of straight lines that could appear in the image space

The algorithm of conventional HT while processing votes to search for local peak values is as follows. First, a cumulative voting matrix M with a two-dimensional array is established and initialized so that all of the elements are 0. Based on the schematic of cumulative voting in the parameter space in Figure 13, the horizontal axis is set as θ, and the vertical axis is set as ρ. The size of the array exerts a direct impact on the resolution. If θ is precise to 1 degree and the range of the horizontal coordinates reaches 3.14 rad, then 0-180 graduations (n = 181) are needed. For ρ, pixel m on the diagonal of the image can serve as the number of rows in order for ρ to be precise to the pixel level. Based on the specifications above, the size of the accumulator voting matrix in the parameter space is $M \in R^{m \times n}$.

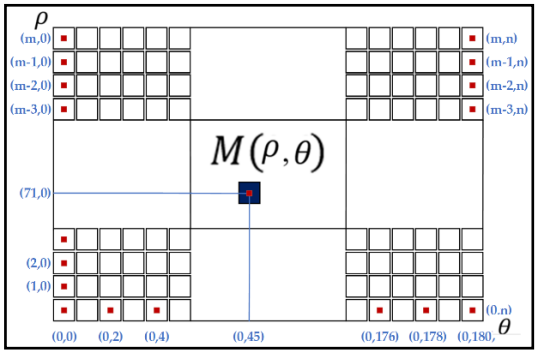

Figure 13. Accumulator voting matrix M

Suppose that there is a straight line consisting of N pixel points in the image, including the two ends of the straight line, which are $A = (a_1, b_1)$ and $B = (a_N, b_N)$, as shown in Figure 14. Take the first point $P_1 = A = (a_1, b_1)$ on the straight line and substitute it into Eq. (1), which gives:

$$L_1 : \rho = a_1 \cos\theta + b_1 \sin\theta \,. \tag{2}$$

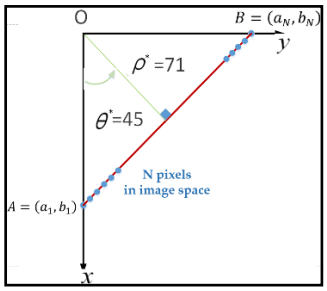

Figure 14. A straight line in the image space

Because θ is precise to 1 degree, substitute $0-180$ into Eq. (1) to obtain 181 ρ values. The values pairs $(\rho_0, 0)$, $(\rho_1, 1), (\rho_2, 2), \ldots, (\rho_{180}, 180)$ correspond to locations in the accumulator voting array M, and 1 is added to the content of this location, as shown in Figure 13. If substituting $\theta = \theta^*$ into Eq. (1) gives $\rho = \rho^*$, then the accumulator voting array $M(\rho^*, \theta^*) = 1$. Take other pixels on the straight line, such as $P_2 = (a_2, b_2)$,..., $P_N = (a_N, b_{N2})$, and repeat the above process to continuously update the content of accumulator voting array M. Once the operations of a point on the straight line have been completed, 1 will be added to the 181 elements in the accumulator array. The voting process of conventional HT can be analyzed as follows:

(1) total number of votes: $n \times N$,

(2) candidates (size of accumulator array): $m \times n$.

The statistics of the contents of the accumulator voting array can be compiled to find the maximum value.

$$(\rho^*, \theta^*) = \arg\max\left(M(\rho, \theta)\right), \tag{3}$$

where $0 \le \rho \le m$ and $0 \le \theta \le n$.

If the value of $M(\rho^*, \theta^*)$ exceeds a specific threshold, indicating the existence of a straight line in the image space. Suppose that the statistical result is (ρ^*, θ^*) = (71,45). It represents that in the accumulative voting process of the accumulative voting array M(71,45) coordinates received the highest number of votes. This indicates that there is a straight line in the image space, and the distance from the straight line to the origin is 71 pixels, and the angle between its vertical line and x-axis being 45°, as shown in Figure 14.

2.3 Number Capture

To enable the computer to understand the numbers on the Sudoku card, we must capture the numbers on the card and store them in a directory folder in jpg format. Aside from capturing the images of the numbers, we must also confirm where the numbers are located in the image. Our approach was to store the coordinates of the upper-left corner (x_1, y_1) and the coordinates of the lower-right corner (x_2, y_2) on a three-dimensional array $pnt[k][2][2]$, where the coordinates of captured number k are expressed as follows:

$$\begin{aligned} pnt[k][0][0] &= x_1, \quad pnt[k][0][1] = y_1 \\ pnt[k][1][0] &= x_2, \quad pnt[k][1][1] = y_2 \,. \end{aligned} \tag{4}$$

First, obtain the size of the image as $col \times row$ pixels. The direction of scanning to capture the numbers is from left to right and from top to bottom, as shown in Figure 15.

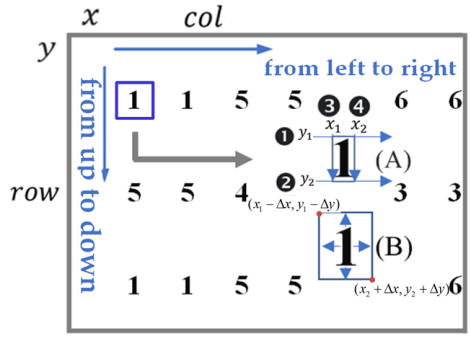

Figure 15. Scanning for number capture

[Step I] Define the initial values of the three variables as $x = 0$, $y = 0$, $c = 0$. Beginning from the top of the image ($y = 0$), scan from left to right ($i = 0,1,2, \dots, col - 1$) and scan a total of col pixels. If the entire horizontal line consists of white pixels or if the black pixels are less than the default threshold value ($c < thd$), then it is determined that no numbers appeared during the scanning process. Add 1 to the content of y and continue to scan the next row. Repeat the left-to-right scanning process until the black pixels are greater than the default threshold value ($c \geq thd$), which means that numbers appeared on that horizontal line. The value of y at this point equals the vertical coordinate above the number block ❶. Define it as y_1, and then stop scanning. Write the above process into a function $obtain\_y_1(col)$.

[Step II] Redefine the initial values of the three variables as $x = 0, y = y_1, c = 0$. Beginning from the top ❶ of the number block ($y = y_1$), scan from left to right ($i = 0,1,2, \dots, col - 1$) and scan a total of col pixels. If the black pixels in the entire horizontal line are greater than the default threshold value ($c \geq thd$), the scanning process spans the number part. Add 1 to the content of y and continue to scan the next row. Repeat the left-to-right scanning process until the black pixels are less than the default threshold value ($c < thd$), which means that the horizontal line has moved to the blank space below and that no numbers appeared during the scanning process. The value of y at this point equals the vertical coordinate below the number block ❷. Define it as y_2, and then stop scanning. Write the above process into a function $obtain\_y_2(col)$.

[Step III] Define the initial values of the three variables as $x = 0$, $y = 0$, $c = 0$. Beginning from the left of the image ($x = 0$), scan from top to y_2 ($j = 0,1,2, \dots, y_2 - 1$) and scan a total of y_2 pixels. If the entire vertical line consists of white pixels or if the black pixels are less than the default threshold value ($c < thd$), then it is determined that no numbers appeared during the scanning process. Add 1 to the content of and continue to scan the next row. Repeat the top to y_2 scanning process until the black pixels are greater than the default threshold value ($c \geq thd$), which means that numbers appeared on that vertical line. The value of at this point equals the coordinate to the left of the number block ❸. Define it as x_1, and then stop scanning. Write the above process into an $obtain\_x_1(y_2)$.

[Step IV] Define the initial values of the three variables as $x = x_1$, $y = 0$, $c = 0$. Beginning from the left of the number block ❸ ($x = x_1$), scan from top to y_2 ($j = 0,1,2, \dots, y_2 - 1$) and scan a total of y_2 pixels. If the entire vertical line consists of white pixels or if the black pixels on the vertical line are greater than the default threshold value ($c \geq thd$), then it is determined that the scanned line passed through numbers. Add 1 to the content of, and continue to scan the next row. Repeat the top to y_2 scanning process until the black pixels are less than the default threshold value ($c < thd$), which means that the vertical line moved to the blank space to the right and that no numbers appeared during the scanning process. The value of x at this point equals the coordinate to the right of the number block ❹. Define it as x_2 and then stop scanning. Write the above process into a function $obtain\_x_2(y_2)$.

[Step V] From [Steps I-IV], we can obtain the number "1" in the upper-left part of the image in Figure 15(A) and place the coordinates of the upper-left corner (x_1, y_1) and the lower-right corner (x_2, y_2) in a three-dimensional array $pnt[k][i][j]$, where k represents the number of digits captured, $i = 0,1$, $j = 0,1$. Currently, the algorithm cuts out images of the numbers using the coordinates of four points. If the image just precisely covers the number, it could affect the subsequent recognition performance of the neural network. We therefore moved the edges of the left, right and top, bottom to a distance $(\Delta x, \Delta y)$ outward before cutting for better recognition performance, as shown in Figure 15(B).

[Step VI] After cutting out the number in the upper left corner of the image, store it in a jpeg file. According to the upper-left corner coordinates (x_1, y_1) and the lower-right corner coordinates (x_2, y_2) of the number, create a white block to cover the original digit.

[Step VII] Repeat [Steps I-VI] to capture the next number.

2.4 Convolutional Neural Networks (CNNs)

CNNs are multilayer perceptrons (MLPs) similar to neural networks. Conventional DL networks use fully connected networks to recognize images. Two dimensional images must be decomposed into one dimension, and then each pixel is regarded as a feature and input into a deep neural network framework for training and learning. However, these input pixels lose their original spatial arrangement information. The purpose of the convolution layer in CNNs is to preserve the spatial arrangements of the image and obtain local images to serve as input features, which then take the CNN model from point comparison to two-dimensional local comparisons. Though the gradual stacking and comprehensive comparison of the features in a region, better recognition results can be derived. An entire CNN mainly consists of several parts: convolution layers, pooling layers, and fully connected layers.

The convolution layer uses a window with designated dimensions sliding from left to right and from top to bottom in an orderly fashion to obtain the sum of the products of local regions in the images. A ReLU activation function outputs the feature values and then provides them to the next network. This sliding window is referred to as a convolution kernel, which is also known as a filter. Based on the original image, an-other simplified image (the feature map) is generated to extract image features. The operations of CNN convolution layers involve two important characteristics: local connectivity and weight sharing, which can reduce the

computing load of the neural network. The pooling layer is a way of compressing images while preserving important information. Via image down-sampling, the original image is scaled down. The objective of the down-sampling in the pooling layer is to reduce redundant information in the feature map without affecting the performance of image feature recognition and to shorten the time needed for subsequent computation.

Figure 16 displays the structure of a CNN. The source images were 28×28 pixel images, and Convolution layer 1 was given six kernels or filters. These six filters also generated images of six corresponding feature maps. The convolution operations did not change the size of the images; thus, they were still 28×28 pixels. Pooling layer 1 performed down-sampling once, shrinking the six 28×28 pixel images to six 14×14 pixel images. Convolution layer 2 was given 16 kernels, generating 16 feature maps that were still 14×14 pixels. Pooling layer 2 performed the second downsampling, shrinking the sixteen 14×14 pixel images to sixteen 7×7 pixel images.

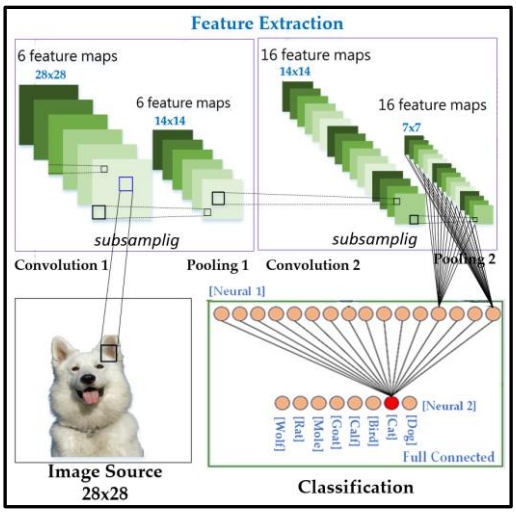

Figure 16. Structure of CNN

2.5 Sudoku Puzzle Solving

To simplify the explanation of the algorithm, a 9×9 Sudoku grid is used, as shown in Figure 17. The easiest way for a computer to solve a Sudoku game is by trial and error, trying the numbers 1-9 for each cell and then continuing on to the next blank cell until all of the cells have been filled so that the rules of Sudoku are met. This is clearly not an efficient approach. Slightly modifying the trial and error method would greatly enhance the operating efficiency of the program. Making a list of all the possible numbers, or a candidate number list, for each blank cell. The values in the list are tested, and if they satisfy the conditions, then the program goes on to the next blank cell. If there are any contradictions, the program backtracks. This approach is called the candidate backtrack solution. The program defines the list $sudoku[\,]$ to store the numbers in the Sudoku grid. If a cell is blank, then 0 fills the spot, refer to Figure 17.

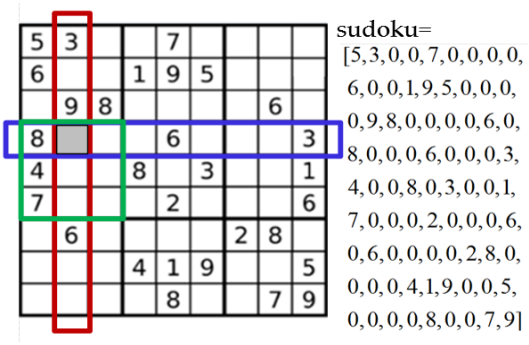

Figure 17. 9×9 Sudoku card

The program next establishes the five following functions: (1) $row\_Num()$, which searches for the numbers appearing in an entire row in the grid and stores them in the list $row[\,]$; (2) $col\_Num()$, which searches for the numbers appearing in an entire column in the grid and stores them in the list $col[\,]$; (3) $blk\_Num()$, which searches for all of the numbers appearing in the (3×3) blocks and stores them in the list $blk[\,]$; (4) $show\_Board()$, which display the Sudoku puzzle grid in the graphical user interface window showing the solution process to players; and (5) $check\_Num()$, which detects whether any numbers appear more than once in any rows, columns, or (3×3) blocks; if so, then the False response is given, and if not, then the True response is given.

The entire base grid is searched for at which index k the blank cell is located. For each blank cell, the three functions $row\_Num()$, $col\_Num()$ and $blk\_Num()$ are called. Following the rule that no number can appear more than once in any row, column, or (3×3) block, all of the numbers that can be placed in the cell are identified, and the list $available[\,]$ is established. One by one, all of the blank cells that have not been filled are added to the list $pointList[\,]$. The function $show\_Board()$ is called, and the base grid of the Sudoku game is displayed on the screen for the player to see.

We employed the candidate backtrack solution method to solve Sudoku puzzle. All of the numbers that may be placed in all of the blank cells were tested by trial and error. With the base grid in Figure 17, there are 51 blank cells. Thus, there are 51 physical objects in $pointList[\,]$. Take one physical object p_1 from $pointList[\,]$, the x and y coordinates of Sudoku grid can be determined by the k value.

$$(p_1.x, p_1.y) = (k\%9, k\,//\,9)\,, \tag{5}$$

where % is the remainder of the division, // is the integer of the division.

Suppose that its coordinates are (1,3), the grey grid shown in Figure 17. Therefore, $p_1.x = 1$, $p_1.y = 3$, and $p_1.available[\,] = [1, 2, 5]$, which means that the numbers that may be filled in this position are 1, 2 and 5, which are called candidate numbers. Test with the first possible number, and call $check\_Num()$ to check whether any numbers appear more than once in the row, column, or (3×3) block. If the response is False, it means that there are numbers that appear more than once, and the test must then be repeated with the next possible number. If the response is true, it means that no numbers appear more than once in the row, column, or (3×3) block; therefore, the number is placed in the Sudoku grid. If none of numbers of $available[\,]$ is valid, previous call of the function will reset the value of the cell to 0 and continue

iterating to find the next valid number. Whether $pointList[\,]$ is empty is then examined. If that is the case, then all of the physical objects have been processed, and $show\_Board()$ is called to display the completed Sudoku grid on the screen. If *pointList*[] is not empty, then the next physical object p_2 is taken from $pointList[\,]$ to confirm the number that should be placed in the next blank cell. The method for this is the same as above.

3 Experiment Results

We divide the process of solving the Sudoku game in the camera captured image into four steps. In Step (1), the vertical lines, horizontal lines, and outer frame are re-moved using HT. In Step (2), the left-to-right, top-to-bottom scanning method proposed in this paper is used to capture the numbers in the images, which are then stored in a directory folder. The locations of the numbers in the images are also obtained. In Step (3), the module kit of Tensorflow and Keras is used to construct a CNN, and the training samples in the MNIST handwritten number dataset are used to train the model. The images of numbers in the directory folder are introduced as test data to assess model performance. In Step (4), the candidate backtrack algorithm is used to automatically infer and fill in the blank cells in the grid, completing the solution to the Sudoku puzzle. In this section, we will present the experiment results of the four steps separately and evaluate the performance of the CNN.

3.1 Straight Line Detection using HT

The image in Figure 18 shows two intersecting straight lines. Line L_1 passes through points (0,100) and (0,500) whereas Line L_2 straight line passes through points (100,0) and (0,500). Conversion to grayscale, binarization, and edge detection produced the image in Figure 19. The lines have widths, and the aforementioned image processing then generates four line segments: L_{11}, L_{12}, L_{21} and L_{22}. The line segments detected using the HT straight line detection algorithm are presented in Table 1.

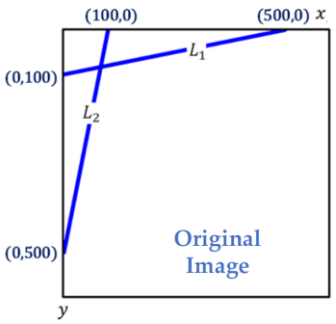

Figure 18. Two straight lines

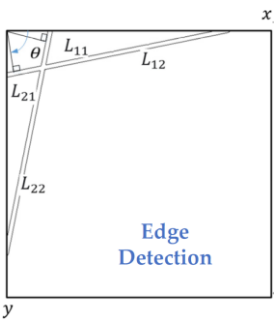

Figure 19. Four lines after edge detection

Table 1. Four straight lines resulting from HT detection

	L_1		L_2	
	L_{11}	L_{12}	L_{21}	L_{22}
ρ	92	102	92	102
θ	1.3788	1.3788	0.1920	0.1920
θ	79^o	79^o	11^o	11^o

The results in Table 1 reveal that these two straight lines are both 10 pixels wide. Note that the axis definitions of Figure 11 and Figure 18 differ. As θ defines the angle between the x-axis and the vertical straight line, the θ in Figure 11 is the angle between the vertical axis and the vertical straight line. In contrast, the θ in Figure 19 is the angle between the horizontal axis and the vertical straight line. The θ angle of Line L_1 derived using HT is 79 degrees, whereas the θ angle of Line L_2 derived using HT is 11 degrees. In the cumulative voting matrix M, elements $M(92,79)$, $M(102,79)$, $M(92,11)$, $M(102,11)$ received the highest number of votes and can be regarded as the regional peak values.

Figure 20 displayed an image of a 4×4 grid similar to a 4×4 Sudoku grid. We detected the horizontal lines, vertical lines, and outer frame of the image. Conversion to grayscale, binarization, and edge detection produced the image in Figure 21. The results of straight line detection using HT are shown in Table 2. A total of 16 straight lines were detected, including 8 horizontal lines and 8 vertical lines. Their angles were 0 degrees and 90 degrees, respectively.

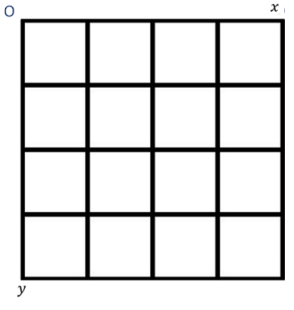

Figure 20. 4×4 grid

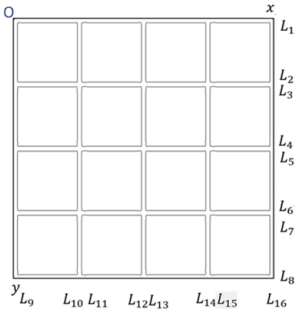

Figure 21. Lines of 4×4 grid following edge detection

Table 2. Four lines of 4×4 grid resulting from HT detection

	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8
ρ (pixels)	7	11	12	23	24	35	36	46
θ (degrees)	0^o	0^o	0^o	0^o	0^o	0^o	0^o	0^o

	L_9	L_{10}	L_{11}	L_{12}	L_{13}	L_{14}	L_{15}	L_{16}
ρ (pixels)	7	11	12	23	24	35	36	47
θ (degrees)	90^o	90^o	90^o	90^o	90^o	90^o	90^o	90^o

3.2 Scanning for Number Capture

The image processing algorithm proposed for number capture in this paper uses the Python program as its work platform to program four functions:

$$obtain\_y_1(col),\ obtain\_y_2(col)$$
$$obtain\_x_1(y_2),\ obtain\_x_1(y_2) \quad \textbf{(6)}$$

The numbers in the image are captured from left to right and from top to bottom, and the coordinates of their upper-left corner and their lower-right corner are stored in a three dimensional array $pnt[k][i][j]$, where k denotes the number of digits in the image and col represents the column pixels of image. With the image in Figure 22 as an example, there are four numbers. The scanning goes from

left to right and from top to bottom, and the four following functions are called to capture numbers:
$obtain\_y1(col)$ ➡ $obtain\_y2(col)$ ➡
$obtain\_x1(y_2)$ ➡ $obtain\_x2(y_2)$
Assume that the dimensions of the image are 600×800 pixels, so $col = 800$.

❶ $obtain\_y1(800)$ ➡ obtains $y1 = 50$,
❷ $obtain\_y2(800)$ ➡ obtains $y2 = 150$,
❸ $obtain\_x1(150)$ ➡ obtains $x1 = 50$,
❹ $obtain\_x2(150)$ ➡ obtains $x2 = 150$. **(7)**

By repeating the steps in Section 3.2, the numbers "1" ➡ "2" ➡ "3" ➡ "4" can be captured.

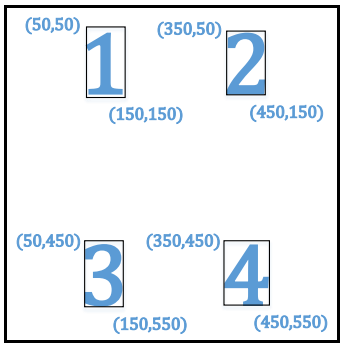

Figure 22. The numbers are aligned left and right as well as up and down

3.3 Digit Recognition using Neural Network

Because each of the MNIST images is 28×28 pixels, they occupy 784 bytes of memory. They are moderate in size and are all binarized black and white images, making them extremely suitable for modeling, training, and prediction in this study. The MNIST dataset comprises 60,000 items of training data and 10,000 items of test data. Generally, 80% of the sample data serve as training data, and the other 20% serve as validation data. Thus, with 60,000 items of data in total, we divided the data into 60,000 × 0.8 = 48,000 items as training data and 60,000 × 0.2 = 12,000 items as validation data. We performed training and prediction validation of an MLP model and a CNN, explained separately as follows.

(I) Multilayer perceptron (MLP) model

The input layer of the MLP model has 784 input neurons, which is a one-dimensional array (of 784 items of data) converted from 28×28 two-dimensional images, serving as the input neurons of the network. The hidden layer has a total of 256 hidden neurons. The output layer contains a total of 10 output neurons that correspond to the 10 prediction results 0-9, the structure shown in Table 3.

The feature values and true values of the 60,000 items of pre-processed data in the MNIST dataset were input into the MLP model for training. The trained model could then be used to predict number digits in various forms (including handwritten and computer fonts). We adopted four computer fonts to test the trained model: Arial Black, Arial Unicode MS, Hanyi Senty Finger Painting, and Hanyi Senty Garden Handwriting. The total number of weight parameters to be trained in the MLP model framework is as follows:

$$256 \times 784 + 256 + 256 \times 10 + 10 = 200,960 + 2,570 = 203,530 \quad \textbf{(8)}$$

The accuracy rates of the MLP model in predicting numbers in Arial Black computer font, Arial Unicode MS, Hanyi Senty Finger Painting, and Hanyi Senty Garden Handwriting were 0.63, 0.54, 0.48, and 0.42, respectively. None of the results were very satisfactory.

Table 3. The structure of MLP model

MLP model		
Layer (type)	Output shape	Parameter
dense_5 (Dense)	(None, 256)	200960
dense_6 (Dense)	(None, 10)	2570
Total parameters: 203,530 Trainable parameter: 203,530 Non-Trainable parameters: 0		

(II) CNN

The structure of the CNN adopted in this study included the following.

Convolution layer 1: The size of the images of the input numbers was 28×28 pixels. The first convolution layer was given sixteen 5×5 kernels (also referred to as filters). In the other words, there are 25×16+16 = 416 weights that need to be updated at the same time in the same layer, and these 16 filters will also generate images of 16 corresponding feature maps. The convolution operations do not change the size of the images; therefore, they are still 28×28 pixels.

Pooling layer 1: This layer performs the first down-sampling, shrinking the sixteen 28×28 pixel images to sixteen 14×14 pixel images.

Convolution layer 2: The size of the images of the input numbers was 14×14 pixels. The second convolution layer was given thirty-six 5×5 kernels. This means that there are 25×36×16+36=14,436 weights that need to be trained and updated in the second convolution layer, and these 36 filters will also generate images of 36 corresponding feature maps. The size of the images is still 14×14 pixels.

Pooling layer 2: This layer performs the second downsampling, shrinking the thirty-six 14×14 pixel images to thirty-six 7×7 pixel images. In the neural network, Dropout(0.25) is added to the model to randomly abandon 25% of the neurons in each training iteration in order to prevent overfitting.

Flattening layer: This layer converts the thirty-six 7×7 images output by Pooling layer 2 into a one dimensional array (including a total of 1,764 items of data), which correspond exactly to the 1,764 neurons in the flattening layer.

Hidden layer: A hidden layer with 128 neurons is established, and a total of 1,764×128+128=225,920 weights need to be trained. Dropout(0.5) is added to the model to randomly abandon 50% of the neurons in each training iteration in order to prevent overfitting.

Output layer: This layer contains a total of 10 neurons corresponding to the 10 digits 0-9. The feature values and true values of the 60,000 items of pre-processed data in the MNIST dataset were input into the CNN model for training, the CNN model is shown in Table 4. We adopted four computer fonts

to test the trained model: Arial Black, Arial Unicode MS, Hanyi Senty Finger Painting, and Hanyi Senty Garden Handwriting. The accuracy rates of the model were all around 80%. From the experimental results, we can conclude that under the same training conditions, the CNN model had better predictive capabilities towards computer fonts than did the MLP model; however, the results were still not satisfactory.

Table 4. The structure of CNN model

CNN model		
Layer (type)	Output shape	Parameter
conv2d_40(Conv2D)	(None, 28,28,16)	416
max_pooling2d_40 (Maxpooling2D)	(None, 14,14,16)	0
conv2d_41(Conv2D)	(None, 14,14,36)	14436
max_pooling2d_41 (Maxpooling2D)	(None, 7,7,36)	0
dropout_40(Dropout)	(None, 7,7,36)	0
flatten_20(Flatten)	(None, 1764)	0
dense_40(Dense)	(None, 128)	225920
dropout_41(Dropout)	(None, 128)	0
dense_41(Dense)	(None, 10)	1290
Total parameters: 242,062 Trainable parameter: 242,062 Non-Trainable parameters: 0		

(III) Self-made dataset with 4,580 training samples

The MNIST dataset used so far contained 60,000 training images and 10,000 test images, which was an ample number of samples. However, this dataset contained handwritten digits, as shown in Figure 6, whereas the recognition targets on our Sudoku cards were computer script fonts. The substantial differences between them could explain why the accuracy of the CNN in digit prediction could not be increased. We therefore collected 4,580 images of digits in computer script fonts and used them to train the CNN model parameters before the digits extracted from Sudoku cards served as prediction targets for accuracy assessment. As for the method of collecting image pictures, various computer fonts, including Arial, Calibri, Time New Roman, etc., are used to capture image from 0 to 9. However, when using computer fonts to capture image from 0 to 9, there will be problems of image skew and offset. Among them, Figure 23 is the uncorrected self-made training sample, Figure 24 is the digital image training set after offset correction. Our experiments revealed that the reinforcement of the computer script font samples increased the prediction accuracy of the CNN to over 96%.

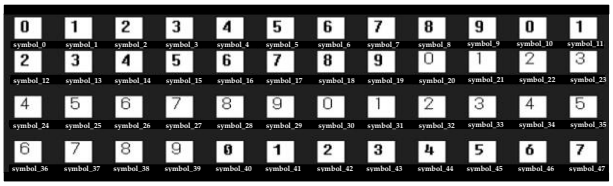

Figure 23. The uncorrected self-made training sample

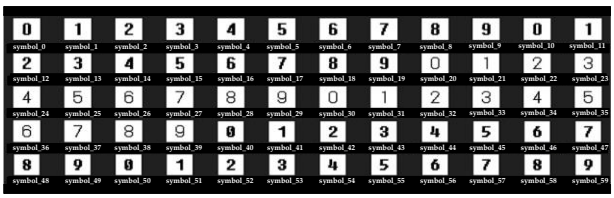

Figure 24. The self-made training sample after offset correction

3.4 Recognition of Digits on Sudoku Game Cards

A camera was used to capture Sudoku card images, and HT was employed to cover the vertical lines, horizontal lines, and outer frames in the images. Using the image processing algorithm proposed in this paper, we captured the numbers in the images, calculated their location coordinates in the image, and stored them in a directory folder. The MNIST dataset was used to train a CNN in order to predict the numbers captured from the Sudoku cards.

Using the CNN structure in Figure 16 and training with the 4,580 images in the self-made dataset, we predicted the numbers in the images stored in the directory folder. As shown in Figure 25, the prediction results achieved an accuracy of 96%, which indicates satisfactory recognition performance.

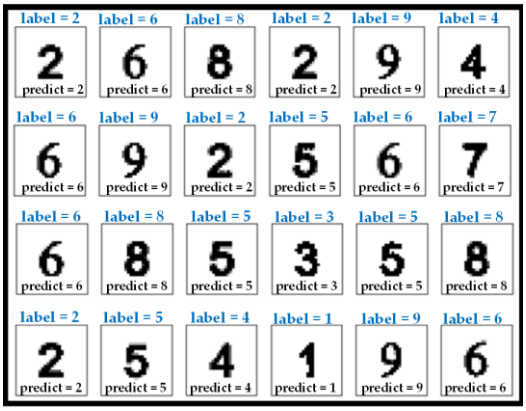

Figure 25. Prediction performance with Sudoku card

4 Conclusion

Number puzzle games are the most natural means of introducing children to the world of mathematics. Challenging number puzzle games can be fascinating for children and encourages them to keep playing and wanting more. Hidden behind number puzzle games are generally logical reasoning and induction, which are two essential abilities in learning mathematics. Using computers to solve number puzzle games or fun magic is a matter of course. With Sudoku as an example, computerizing Sudoku games provides a unique solution tool; it attempts to provide humanistic solution methods, completely simulates the thinking process of the human brain during problem solving, and explains the reasons behind each step along the way. When children feel that a hard Sudoku game is impossible to solve, their first thought is often to give up. If suitable hints or directions could be given, their confidence in mathematics following success at the game could be regained. Traditional number puzzle games are usually presented using game cards or physical grids, such as Sudoku cards or Klotski sliding block puzzles. The means of

computerizing game cards, physical grids, and sliding block puzzles was thus the motivation behind this study. The contents of this paper include image capture by a camera, HT, number capture, the use of CNN for digit recognition, and a Sudoku game algorithm. In the future, children can watch the problem-solving process on a computer while manipulating sliding block puzzle pieces and work their way toward success. We adopted a Python work platform, incorporated OpenCV to read the image data of Sudoku cards, used HT to remove the excess straight lines and outer frames from the image, captured the numbers on Sudoku cards using an image processing algorithm, and stored them in a file directory. Using the Tensorflow and Keras embedded in the Python platform, we established a neural net-work model, read the images of the numbers in the file directory for recognition, and compared the recognition performances of an MLP model and a CNN model trained using the MNIST dataset. To enhance the recognition accuracy, we collected 4,580 images of digits in computer script fonts and used them to train the CNN model parameters before the digits extracted from Sudoku cards served as prediction targets for accuracy assessment. The accuracy reached 96%, thereby presenting good performance.

References

[1] J. F. Crook, A pencil-and-paper algorithm for solving Sudoku puzzles, *Notices of the American Mathematical Society*, Vol. 56, No 4, pp. 460-468, April, 2009.

[2] O. Eva, B. Desmond, B. Dunka, A Hybrid Backtracking and Pencil and Paper Sudoku Solver, *International Journal of Computer Applications*, Vol. 181, No. 47, pp. 39-43, April, 2019.

[3] P. V. C. Hough, *Method and means for recognizing complex patterns*, United States: No. US3069654A, December, 1962.

[4] R. O. Duda, P. E. Hart, Use of the Hough Transform to Detect Lines and Curves in Pictures, *Communications of the ACM*, Vol. 15, No. 1, pp. 11-15, January, 1972.

[5] V. J. Schneider, Real time circle detection by simplified Hough transform on smartphones, *Proc. SPIE 11736, Real-Time Image Processing and Deep Learning 2021, 117360F*, The International Society for Optics and Photonics, Bellingham, Washington, 2021.

[6] F. Han, Y. Guo, L. Wang, A New Ellipse Detector Based on Hough Transform, *2009 Second International Conference on Information and Computing Science*, Manchester, UK, 2009, pp. 301-305.

[7] L. Xu, E. Oja, P. Kultanen, A new curve detection method: Randomized hough transform (RHT), *Pattern Recognition Letters*, Vol. 11, No. 5, pp. 331-338, May, 1990.

[8] R. Cucchiara, F. Filicori, The Vector-Gradient Hough Transform, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20, No. 7, pp. 746-750, July, 1998.

[9] S. Luo, J. Liu, Research on Car License Plate Recognition Based on Improved YOLOv5m and LPRNet, *IEEE Access*, Vol. 10, pp. 93692-93700, September, 2022, doi: 10.1109/ACCESS.2022.3203388.

[10] Z. Akhtar, R. Ali, Automatic Number Plate Recognition Using Random Forest Classifier, *SN Computer Science*, Vol. 1, No. 3, pp. 1-9, May, 2020.

[11] S. Suzuki, K. Abe, Topological structural analysis of digitized binary images by border following, *Computer Vision, Graphics, and Image Processing*, Vol. 30, No. 1, pp. 32-46, April, 1985.

[12] M. Sun, M. Shi, H. Han, Contour Extraction and Vectorization Algorithm for Paper-Cut Pattern, *2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science & Engineering (ACIT-CSII-BCD)*, Las Vegas, Nevada, USA, 2016, pp. 342-347.

[13] Jrunw, *Illustrated Top 10 Classics for Getting Started with Machine Learning*, Retrieved from: https://reurl.cc/95dKLa, December, 2021.

[14] M. Minsky, S. Papert, *Perceptrons: an introduction to computational geometry*, The MIT Press, 1969.

[15] D. Rumelhart, G. Hinton, R. Williams, Learning representations by back-propagating errors, *Nature*, Vol. 323, No. 6088, pp. 533-536, October, 1986.

[16] R. Hecht-Nielsen, Theory of the backpropagation neural network, *International 1989 Joint Conference on Neural Networks*, DC, Washington, USA, 1989, pp. 593-605.

[17] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, Backpropagation Applied to Handwritten Zip Code Recognition, *Neural Computation*, Vol. 1, No. 4, pp. 541-551, December, 1989.

[18] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, Handwritten digit recognition with a back-propagation network, *Advances in Neural Information Processing Systems 2*, Denver, CO, USA, 1989, pp. 396-404.

[19] R. Salakhutdinov, A. Mnih, G. Hinton, Restricted Boltzmann machines for collaborative filtering, *Proceedings of the 24th international conference on Machine learning – ICML*, Corvalis, Oregon, USA, 2007, pp. 791-798.

[20] G. E. Hinton, S. Osindero, Y.-W. Teh, A Fast Learning Algorithm for Deep Belief Nets, *Neural Computation*, Vol. 18, No. 7, pp. 1527-1554, July, 2006.

[21] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278-2324, November, 1998.

[22] H. Y. Lee, *Convolutional Neural (CNN)*, Retrieved from: https: // hackmd. Io /@allen 108108/rkn-oVGA4, December, 2021.

[23] S. Weidman, *Deep Learning from Scratch, Building with python First Principles*, O'Reilly, 2019.

[24] I. Naseer, S. Akram, T. Masood, A. Jaffar, M. A. Khan, A. Mosavi, Performance Analysis of State-of-the-Art CNN Architectures for LUNA16, *Sensors*, Vol. 22, No. 12, Article No. 4426, June, 2022, doi: 10.3390/s22124426.

Biographies

Pu-Sheng Tsai was born in Taiwan, R.O.C., in 1962. He received the M.S. degree in automatic control from the Feng Chia University, Taichung, Taiwan, R.O.C., in 1985 and the Ph.D. degree in electrical engineering from the National Taiwan University, in 1998. He is currently an Associate Professor in the Department of Electronic Engineering at Ming Chuan University, Taoyuan, Taiwan. His main research interests are artificial intelligence, virtual reality somatosensory interaction, internet of things, Embedded Micro-controller application and design.

Ter-Feng Wu was born in Taiwan in 1962. He received the B.S. degree in Department of Industrial Education from National Taiwan Normal University, Taipei, Taiwan, in 1986. He received the M.S. degree in Department of Control Engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1993. He received the Ph.D. degree in Department Electrical Engineering from National Taiwan University, Taipei, Taiwan, in 2006. He is currently a professor and will be a dean of College of Electrical Engineering & Computer Science, National Ilan University, Yilan, Taiwan, since February 1, 2023. His research interests include intelligent control, neural network, fuzzy CMAC, green energy, unmanned aerial vehicles (UAVs) and mobile robot, etc.

Jen-Yang Chen was born in Hsin-Chu, Taiwan, R.O.C., in May 1960. He received the Ph.D. in electrical engineering from Tamkang University, in 2000. He is currently a full professor in the Department of Electronic Engineering at Ming Chuan University. His research interests include intelligent control, soft computing, science education, robotic control, and adaptive control.

Jen-Feng Huang was born in Taiwan, R.O.C., in 1974. He received the M.S. degree in electronic engineering from China University of Science and Technology, Taipei, Taiwan, R.O.C., in 2021. He currently works in Jian Quan Clinc Medical Test Center as a senior quality control & test engineer. His main research interests are Embedded Micro-controller application, mechanical vision, medical testing and artificial intelligence.