Sudoku Puzzle Recognition using OpenCV Library

Taufikurrahman Anwar
Informatics Engineering, Bandung Institute of Technology
13517074@std.stei.itb.ac.id

***Abstract*—Sudoku is a well-known and loved puzzle by all ages, often appearing in daily newspapers. 'Sudoku' is a popular Japanese puzzle game that trains our logical mind. The word Sudoku means 'the digits must remain single'. While this puzzle is really amusing to solve, humans can only do so much compared to the processing speed of a computer. Using the OpenCV Library, we can detect a Sudoku Puzzle, take that puzzle, recognize the numbers on the puzzle, and recreate it as a digital copy of the puzzle.**

Keywords—Sudoku, OpenCV, Detect, Recognize, Python.

I. Background

Sudoku is a puzzle game created around the 19th century, is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid (also called "boxes", "blocks", or "regions") contain all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution.

Sudoku was only published in the newspaper due to the difficulty to create these puzzles, until Wayne Gould, a retired Judge from Hong Kong created a computer program to mass produce these puzzles for the world to enjoy. To this day, there are still plenty of newspapers that publish sudoku puzzles in their newspapers, but we want the convenience of having those puzzles saved as a digital copy to be solved later without having to bring the newspaper and a pen/pencil everywhere we go.

Even though there's already plenty of media that provides these sudoku puzzles digitally, this project was made for learning on how Image Processing and Computer Vision can be made useful on everyday things. Hence this project will mostly talk about the Computer Vision and Image Processing methods and technology to discern a sudoku puzzle from a picture of a newspaper page and turn it into a digital version of the same puzzle.

II. Python

Python is a general purpose programming language. Python's main design point is to emphasize code readability and simplicity that helps to maintain small and large scale projects.

Python comes with plenty of useful libraries and modules, even if the library that we need is not included on the main python package, it is very likely that it is available on the internet and can be very easily installed to the machine.

In this project we will be using Python as the main programming language we use. The main reason for using python is that there are a lot of available libraries that can be used for our purposes. Python is also very easy to read and lately has been a very popular language for anyone learning programming to use.

III. OpenCV

OpenCV is the abbreviation for Open Source Computer Vision is a library of programming functions mainly used for real time computer vision.

OpenCV-Python is a wrapper from the original C/C++ OpenCV library, so it's just as fast as it's on C/C++ and it's also easy to use and read since we are programming in Python.

This library will be the main focus of this project, it will be used to read the image from our machine, prepare our image for processing, extract the sudoku grid from the newspaper picture, and to extract the number from each cell on the puzzle grid. In this project the OpenCV used is OpenCV version 4.5.2.52

IV. Tensorflow

TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

Included in this machine learning library is a module named Keras. Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation.

This library will be used for developing a model for recognizing the numbers on the puzzle and figuring out what numbers are actually written on the paper. The Tensorflow used on this project is Tensorflow 2.5.0.

V. MNIST Database

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.

The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset.

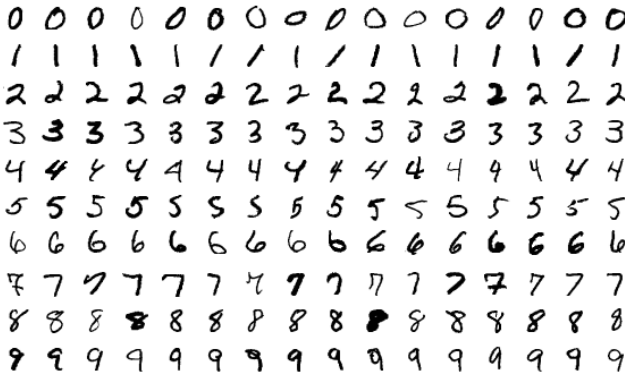

Fig. 1 MNIST Database sample image

The MNIST database is used on this project to train our Machine Learning model to recognize the numbers shown on the puzzle.

VI. Methods

In general, there are a few problems we need to solve to make this project work.

A. Image Preprocessing

The first problem we have to tackle is to prepare our image so that we can focus on the important things. There are a few things we need to do to prepare our image, from denoising, converting to grayscale image, and converting to binary image.

a). Image Denoising

To do this task, we need to Blur the image using gaussian blur to reduce noise obtained in the thresholding algorithm (adaptive thresholding). When we blur an image, we make the color transition from one side of an edge in the image to another smooth rather than sudden. The effect is to average out rapid changes in pixel intensity. The blur, or smoothing, of an image removes "outlier" pixels that may be noise in the image.

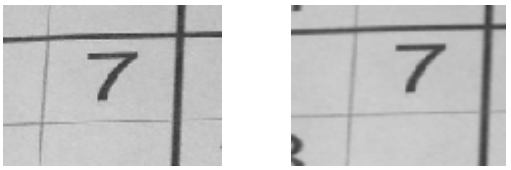

Fig. 1 Sample image before and after denoising

As we can see the image on the right has slightly more smooth and connected lines which will help recognizing the line as one connected line.

b). Converting to Grayscale Image

To do this task, all we have to do is to use the Convert Color function included in the OpenCV library. As we only have interest in the puzzle which is only about the position and the numbers on them, we have no interest in the color of the image, of course we can still process it as a colored picture, but it will take more time and memory to do.

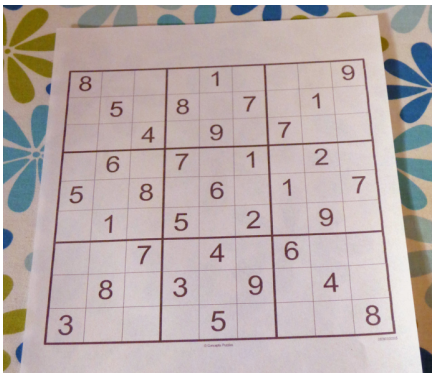

Fig. 2 Original image

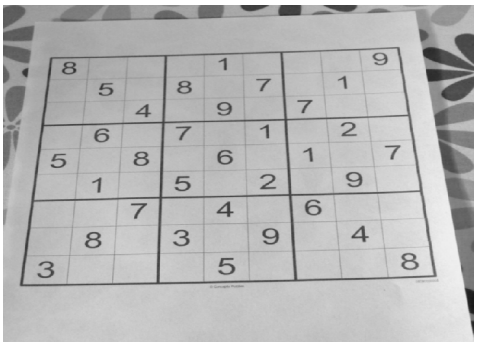

Fig. 3 Original image after converted to grayscale

c). Converting to Binary Image

Here, the matter is straight-forward. For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0, otherwise it is set to a maximum value which practically converts our image from grayscale to Binary as it only has 2

possible values for each pixel. The function cv.threshold is used to apply the thresholding.

On this project we will use the Adaptive Gaussian Thresholding as our Thresholding algorithm

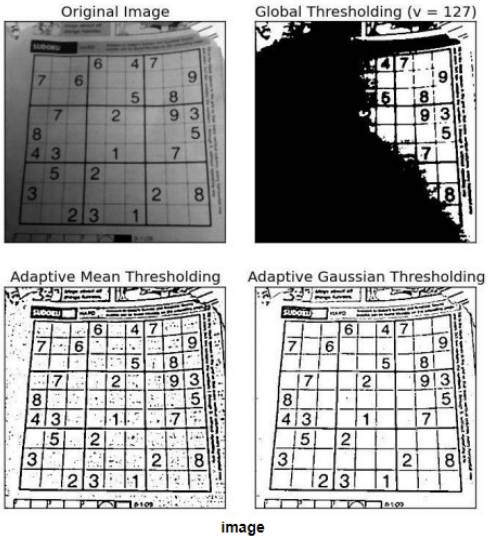

Fig. 4 Sample image with a few of thresholding algorithm

B. Grid Extraction

We will tackle this problem by using the function `findContours()` and figuring out which out of all the found contours is our puzzle grid. The function `findContours()` will give us every shape that's on the screen and we will find external contours and then sort by area in descending order. Thus, the largest polygon is stored in contours[0].

Fig. 5 Result given by the `findContours()` function is marked with the blue lines

After knowing where the grid is, we can crop out all the other elements on the picture as we have no interest in those whatsoever. So, the next thing we will do is to crop out all other elements beside the grid and warp the grid to be a straight rectangle filling the whole image.

In order to crop the image, we need to know the dimensions of the sudoku. Although sudoku is a square and has equal dimensions, in order to ensure that we don't crop any part of the image we will calculate the height and width. We also need to construct dimensions for the cropped image. Since the index starts from 0, we start from the points are (0, 0) , (width-1,0) ,(width-1, height-1), and (0, height-1). We will then get the grid and warp the image.

```
#Calculate width
width_A = np.sqrt(((bottom_right[0] - bottom_left[0]) ** 2)
+ ((bottom_right[1] - bottom_left[1]) ** 2))

width_B = np.sqrt(((top_right[0] - top_left[0]) ** 2)
+ ((top_right[1] - top_left[1]) ** 2))
width = max(int(width_A), int(width_B))

#Calculate height
height_A = np.sqrt(((top_right[0] - bottom_right[0]) ** 2)
+ ((top_right[1] - bottom_right[1]) ** 2))

height_B = np.sqrt(((top_left[0] - bottom_left[0]) ** 2)
+ ((top_left[1] - bottom_left[1]) ** 2))
height = max(int(height_A), int(height_B))

#Calculate image dimensions
ordered_corners = (top_left, top_right, bottom_right, bottom_left
dimensions = np.array([
    [0, 0],
    [width - 1, 0],
    [width - 1, height - 1],
    [0, height - 1]
    ], dtype="float32")
ordered_corners = np.array(ordered_corners, dtype="float32")

#Warp image according to the grid
grid = cv.getPerspectiveTransform(ordered_corners, dimensions)
grid = cv.warpPerspective(img, grid, (width, height))
grid = cv.GaussianBlur(grid,(5,5),0)
```

Fig. 6 Code used to measure the grid and warp the image

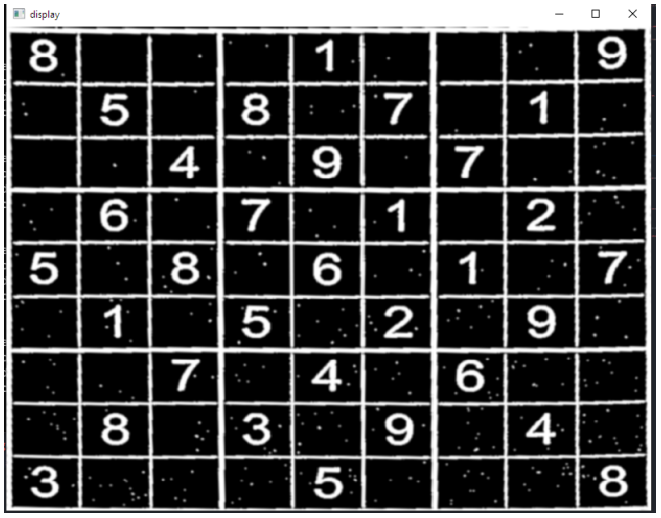

Fig. 7 Sample Image after cropped and warped

C. Cell Extraction

As we all know, the playing grid consists of smaller boxes which house the numbers we need. We will need to differentiate for each of the cells to properly place the number in the correct order.

Since we know that the playing grid consists of 9x9 smaller boxes, we can estimate the size of the smaller box as the size of the large rectangle divided by 9. This is really helpful as our picture only consists of the puzzle grid and even though it does not always form a square we can always know that the cells would have equal width and height all along the board.

Thus we can know the position of the box corresponding to it's index by the offset to the image origin point.

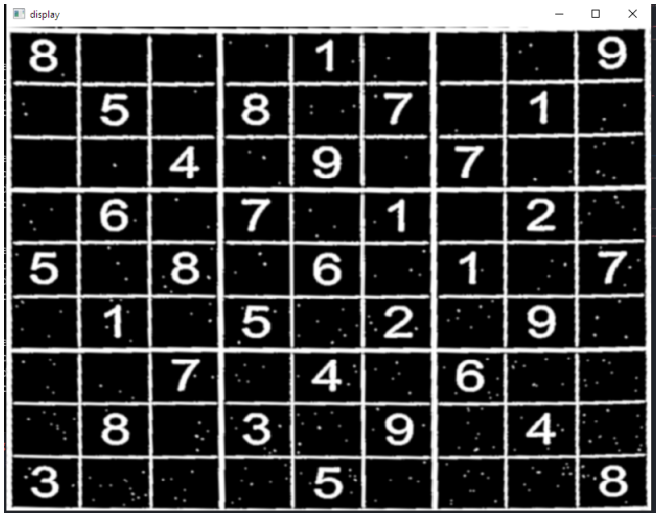

Fig. 8 Each of the cells have uniform height and width

Fig. 9 A few cell after the grid divided into 81 equal parts

D. Character Recognition

For the Character recognition we have two major tasks to do. First, we will have to prepare and train the machine learning model to accurately predict the number seen on the puzzle. Second, we will have to prepare our numbers to match the model so it can be properly processed and predicted.

a). For the machine learning model, we use the MNIST database as our dataset,

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Reshape to be samples*pixels*width*height
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')

# One hot Cpde
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
num_classes = y_test.shape[1]

# convert from integers to floats
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
# normalize to range [0, 1]
X_train = (X_train / 255.0)
X_test = (X_test / 255.0)

return X_test, y_test, X_train, y_train
```

Fig. 5 Codes used to load the MNIST database

Then we need to train the model and then write the model to a file so we won't have to do the training for each time we run the code.

```
def evaluate_model(X_train, y_Train, n_folds=5):

    accuracy, data = list(), list()

    # prepare 5-cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)

    for x_train, x_test in kfold.split(X_train):
        # create model
        model = create_model()
        # select rows for train and test
        trainX, trainY, testX, testY = X_train[x_train], y_Train[x_train], X_train[x_tes
        # fit model
        data_fit = model.fit(trainX, trainY, validation_data=(testX, testY), epochs=10,
        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)
        # stores accuracy
        accuracy.append(acc)
        data.append(data_fit)
    return accuracy, data


def run():
    X_test, y_test, X_train, y_train = input_data()

    model = create_model()
    model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=

    test(X_train, model)

    # save model and architecture to single file
    model.save("model.h5")
    print("Saved model to disk")
```

Fig. 5 Codes used to train and write the machine learning model

Based on the test data provided by the MNIST database, our model is at a very high accuracy level, scoring higher than 0.98 on every test batch.

```
- val_accuracy: 0.9858

- val_accuracy: 0.9870

- val_accuracy: 0.9907

val_accuracy: 0.9899

- val_accuracy: 0.9898

- val_accuracy: 0.9921

- val_accuracy: 0.9911

- val_accuracy: 0.9926

- val_accuracy: 0.9921

- val_accuracy: 0.9918
```

Fig. 6 Trained model accuracy values

b). To match our machine learning model, which accepts 28x28 pixels binary images, we will have to resize each of our cell images to fit the model to be later processed.

While cropping the image into a 28x28 pixels image is easy enough, we want to only crop the unnecessary part of the image and not the number itself, to do that we need to know the location of our number inside the cell as it is our Region of Interest and if no number found inside the image we can skip those image as we can assume those cells to be empty.

```python
# threshold the image
gray = cv.threshold(image, thresh, 255, cv.THRESH_BINARY)[1]

# Find contours
cnts = cv.findContours(gray, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
# print(i,j,len(cnts[0]))
cnts = cnts[0] if len(cnts) == 2 else cnts[1]

for c in cnts:
    x, y, w, h = cv.boundingRect(c)

    if (x < 3 or y < 3 or h < 3 or w < 3):
        continue
    ROI = gray[y:y + h, x:x + w]
    ROI = scale_and_centre(ROI, 120)

    # Writing the cleaned cells
    tmp_sudoku[i][j] = image_to_array.predict(ROI)
    display_image(str(tmp_sudoku[i][j]),ROI)
```

Fig. 5 Codes used to determine the Region of Interest (ROI)

Fig. 5 Sample case when the grid lines is also detected on the image

By using the code shown before, we can determine which of the found contours meets our interest, in this case we simply take the contour which has the most area as it is most likely to be the number we were looking for compared to the gridlines or even the noise dots.

Fig. 5 Few sample cell image after cropped and resized

E. Writing the grid into a 2D array

The last thing we need to do is to write the numbers we get to a 2 Dimensional Array to represent the sudoku puzzle and print it out to the screen as our output.

To do this we simply iterate through our initial grid which is filled with zeros as a default value, and if on a specific index we know there is a number from the character recognition before, we will write what the model predicted what the number would be.

```python
def predict(img):
    model = load_model("model.h5")

    image = img.copy()
    image = cv.resize(image, (28, 28))
    image = image.astype('float32')
    image = image.reshape(1, 28, 28, 1)
    image /= 255

    prediction = model.predict(image.reshape(1, 28, 28, 1), batch_size = 1)
    prediction = prediction.argmax()


    return prediction
```

Fig. 5 Codes used to load the model and predict the selected number

After we iterate through the whole grid all that is left to do is to print out the result on the terminal.

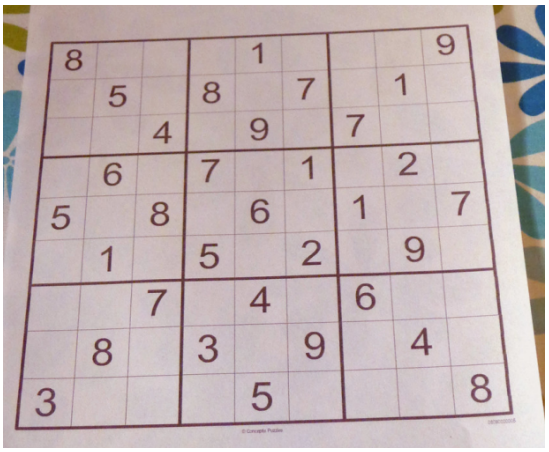

Fig. 5 Original image

```
[8, 0, 0, 0, 7, 0, 0, 0, 9]
[0, 5, 0, 8, 0, 7, 0, 7, 0]
[0, 0, 4, 0, 9, 0, 7, 0, 0]
[0, 6, 0, 7, 0, 7, 0, 2, 0]
[5, 0, 8, 0, 6, 0, 7, 0, 7]
[0, 7, 0, 5, 0, 2, 0, 8, 0]
[0, 0, 2, 0, 4, 0, 6, 0, 0]
[0, 8, 0, 3, 0, 9, 0, 4, 0]
[3, 0, 0, 0, 5, 0, 0, 0, 8]
```

Fig. 5 The output of the system on the sample image

VII. Analysis

In this part we will show how the image recognition project performs on a few different sample pictures.

Fig. 5 Sample image 1

```
[9, 3, 5, 0, 0, 2, 2, 0, 0]
[0, 0, 0, 0, 9, 3, 7, 4, 7]
[0, 4, 0, 0, 0, 0, 0, 0, 0]
[4, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 7, 5, 9, 2, 0, 0]
[7, 0, 2, 4, 8, 0, 3, 7, 8]
[5, 0, 0, 4, 7, 5, 6, 2, 0]
[0, 2, 7, 3, 2, 0, 0, 0, 0]
[0, 0, 4, 7, 1, 2, 7, 5, 7]
```

Fig. 5 System's output for Sample image 1

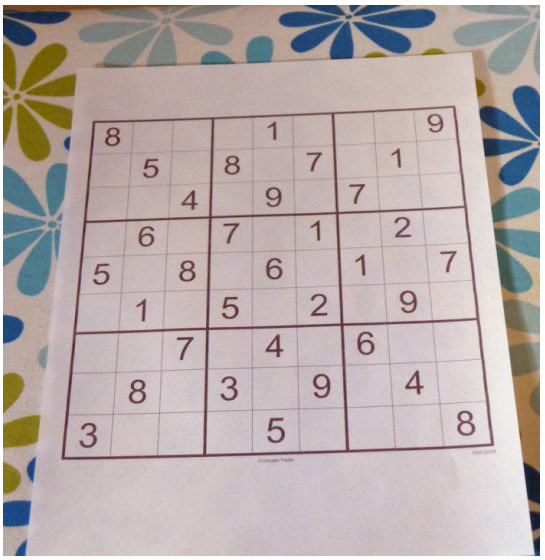

Fig. 5 Sample image 2

```
[8, 0, 0, 0, 7, 0, 0, 0, 9]
[0, 5, 0, 8, 0, 7, 0, 7, 0]
[0, 0, 4, 0, 9, 0, 7, 0, 0]
[0, 6, 0, 7, 0, 7, 0, 2, 0]
[5, 0, 8, 0, 6, 0, 7, 0, 7]
[0, 7, 0, 5, 0, 2, 0, 8, 0]
[0, 0, 2, 0, 4, 0, 6, 0, 0]
[0, 8, 0, 3, 0, 9, 0, 4, 0]
[3, 0, 0, 0, 5, 0, 0, 0, 8]
```

Fig. 5 System's output for Sample image 2

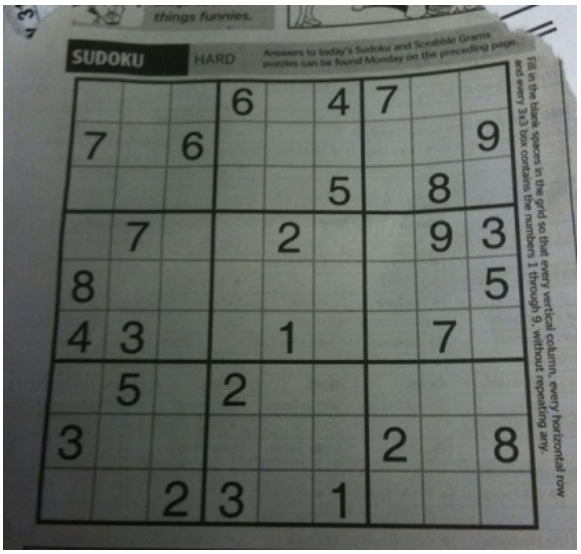

Fig. 5 Sample image 3

```
[0, 0, 0, 8, 0, 0, 7, 0, 0]
[7, 0, 8, 0, 0, 0, 2, 0, 8]
[0, 0, 0, 0, 0, 5, 0, 8, 0]
[0, 7, 0, 0, 2, 0, 0, 8, 3]
[8, 0, 0, 8, 0, 0, 0, 0, 5]
[4, 3, 0, 0, 7, 0, 0, 7, 0]
[0, 5, 0, 2, 0, 0, 0, 0, 0]
[3, 0, 0, 0, 0, 0, 2, 0, 8]
[0, 0, 2, 3, 0, 7, 0, 0, 0]
```

Fig. 5 System's output for Sample image 3

As we can see, the system is pretty accurate on determining which cell has a number on them and separating the sudoku grid from the rest of the image, but it is still very inaccurate on determining the numbers inside each cell..

VIII. Conclusion

Based on implementation, and testing. We can conclude that the system works quite well on determining where everything is, but the flaws lie on the machine learning model used on this project as it struggles to differentiate between '1' and '7', '5' and '6', '8' and '9'.

The flaw probably comes from the dataset used, as it is a dataset of human handwritings and not perfectly suited to recognize uniform and robotic fonts used on newspapers.

On the other hand, OpenCV and TensorFlow are really easy to use and to understand but the skill ceiling is still way above our current skill point, as perhaps there are still a lot of improvements to do on the computer vision side as well, as we still face a lot of noise on the images, the image is not properly cropped, and the likes.

Overall, while our purpose in the making of this project is for our personal learning, this project is really entertaining and challenging. This project makes us think more on how can computer vision be more useful for our daily lives and how amazing the current technology actually is.

References

[1] "Welcome to OpenCV-Python Tutorials's documentation! — OpenCV-Python Tutorials 1 documentation", Opencv-python-tutroals.readthedocs.io, 2020. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/. [Accessed: 23-Mar- 2020].

[2] "Image Processing with Python Blurring images", datacarpentry.org, 2021. [Online]. Available: https://datacarpentry.org/image-processing/06-blurring/ [Accessed: 24-Mar- 2021].

[3] Abbott, P. (Ed.). "In and Out: In-Flight Puzzle." Mathematica J. 9, 528-531, 2005.

[4] Boyer, C. "Sudoku's French Ancestors." Pour la Science. June 2006.