REAL TIME SUDOKU SOLVER

Report submitted in fulfillment of the requirements
for the Exploratory Project of

Second Year B.Tech.

By

Komal Garg (20095054) and Shashi Kant (20095106)

Under the guidance of

Prof. V. N. Mishra

Department of Electronics Engineering

INDIAN INSTITUTE OF TECHNOLOGY (BHU) VARANASI
Varanasi 221005, India
May 2022

DECLARATION

We certify that

1. The work contained in this report is original and has been done by us and the general supervision of our supervisor.
2. The work has not been submitted for any project.
3. Whenever we have used materials (data, theoretical analysis, results) from other sources, we have given due credit to them by citing them in the text of the thesis and giving their details in the references.
4. Whenever we have quoted written materials from other sources, we have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

CERTIFICATE

This is to certify that this project report "**To make a real time sudoku solver using OpenCV and Python**.**"** is submitted by **Komal Garg** and **Shashi Kant** who carried out the project work under the supervision of **Prof. V. N. Mishra.**

We approve this project for submission of the Exploratory Project, IIT(BHU) Varanasi.

Signature of Supervisor
Prof. V. N. Mishra
Department of Electronics Engineering
IIT(BHU) Varanasi

<u>ACKNOWLEDGEMENTS</u>

We would like to express our sincere gratitude to Prof. V. N. Mishra Sir to let us do this project under his supervision. Also, we would like to thank our Parents, Teachers, and Friends for supporting us in all our endeavors.

Place: IIT (BHU) Varanasi
Date: 01-05-2022 KOMAL GARG and SHASHIKANT

ABSTRACT

Many people try solving Sudoku puzzles everyday. These puzzles are usually found in newspapers, magazines and so on. Whenever a person is unable to solve a puzzle or is running short on time to solve the puzzle, it will be very convenient to show the solved puzzle as an augmented reality**.**

In this project, we propose an optimal way of recognizing a Sudoku puzzle using computer vision and Deep Learning, and solve the puzzle using constraint programming and backtracking algorithm to display the solved puzzle as augmented reality.

All the work was created in Python utilizing PyCharm IDE, OpenCV, Keras and TensorFlow Libraries.

TABLE OF CONTENTS

1. INTRODUCTION

The Sudoku Puzzle

Sudoku is a logic-based combinatorial puzzle. When introduced in Japan, it receives its current name Sudoku, meaning" single number" in Japanese. It is now popular all over the world. The game consists of a 9x9 grid divided into 9 square sub-grids (of size 3x3). Some of the squares in the grid contain a number from 1 to 9. The player is presented with a partially filled grid and their aim is to fill the rest of the squares with numbers from 1 to 9. The rules are fairly simple: each row, column and block must contain each of these numbers exactly once.

The capabilities of the project include being able to detect and recognize such a puzzle by getting feed from the rear camera of a device. It should be able to solve the game in real time and fill the empty squares with the correct digits, displaying them over the feed from the camera on the screen in the corresponding positions.

Sudoku puzzle

2. BACKGROUND

2.1 Image Transformations

A substantial part of computer vision and computer graphics concerns with image
processing or, in particular, image transformations. There are different kinds of transformations in geometry.
An affine transformation is a transformation that preserves points, straight lines, planes, parallelism and ratios of distances between points lying on a straight line, while it does not necessarily preserve angles between lines or distances between points. When an image is shot, different objects in it have different perspective distortion depending on their location in the scene relative to the position of the camera. This happens when objects from the real world space which have 3D coordinates are projected onto the 2D image space. Parallel projections only preserve straight lines. Parallel lines in objects, then, do not necessarily appear parallel in the images. In order to convert between the two spaces, we need to transform (warp) the images. In other words, we needed to find a suitable mapping between points from one plane to the other and apply it to the picture to obtain an image with the desired object properties, essentially looking at the scene from another point of view. Such mapping is called a homography. It has 8 degrees of freedom. To estimate homography, we need at least 4 points
of reference.

2.2 Knuth's Algorithm X

We can represent every sudoku in the form of a matrix as follows:

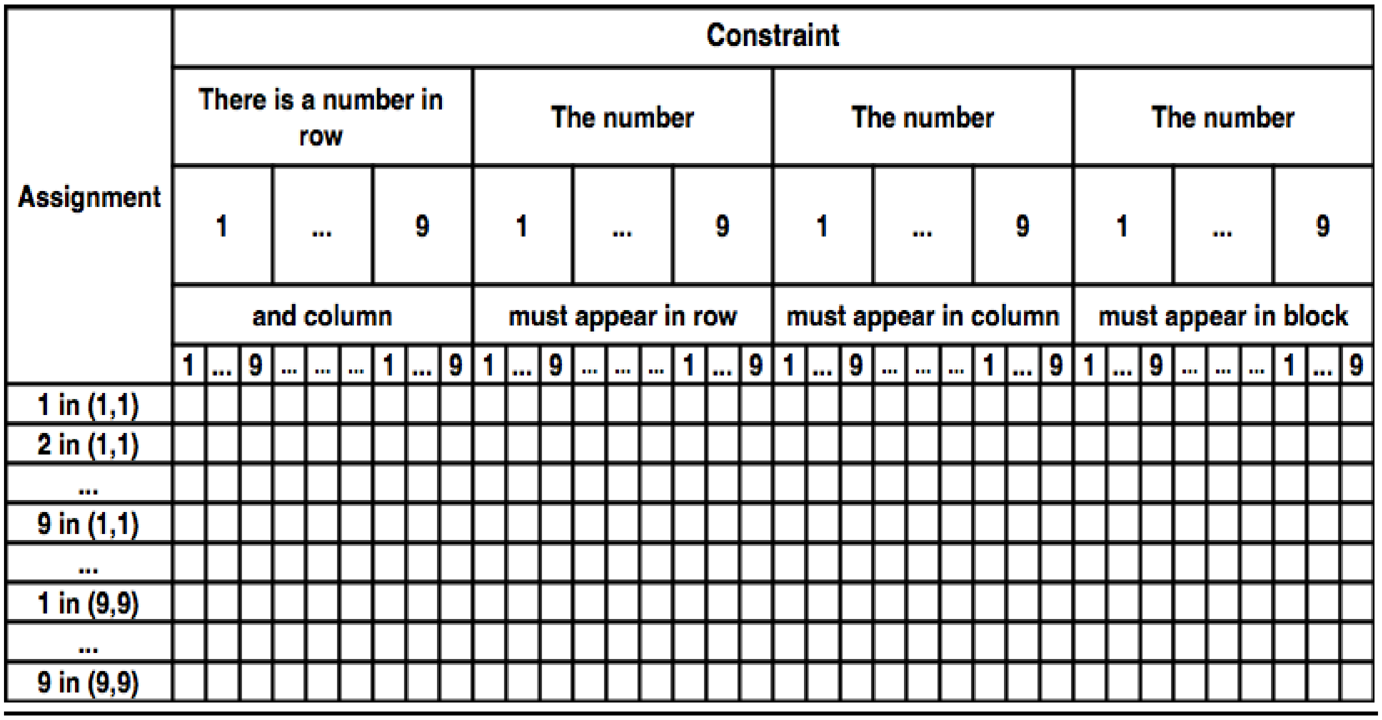
- Rows 1-729 indicate whether the chosen assignment of values includes each number in each position.

Columns represent the rules of the puzzle as constraints:

- Constraints 1-81: We must place some number in each cell.
- Constraints 82-162: We must place each number somewhere in each row.
- Constraints 163-243: We must place each number somewhere in each column.
- Constraints 244-324: We must place each number somewhere in each block.

Thus, a cell in this matrix contains 1 if the (value; position) pair from the corresponding row satisfies the constraint in the corresponding column. The constraint matrix contains 236196 cells in total, but any candidate solution only fills 2916 of them with 1's.

Given a matrix of 0’s and 1’s (such as the one described above) the exact cover problem concerns with finding an assignment of values such that there is exactly one 1 in each column. In order for a solution of a sudoku puzzle to be correct, this is precisely what the case must be.



Assignment	Constraint																																			
	There is a number in row									The number									The number									The number								
	1			...			9			1			...			9			1			...			9			1			...			9		
	and column									must appear in row									must appear in column									must appear in block								
	1	...	9	...	...	...	1	...	9	1	...	9	...	...	...	1	...	9	1	...	9	...	...	...	1	...	9	1	...	9	...	...	...	1	...	9
1 in (1,1)																																				
2 in (1,1)																																				
...																																				
9 in (1,1)																																				
...																																				
1 in (9,9)																																				
...																																				
9 in (9,9)																																				

Constraint matrix

Algorithm X:

Donald Knuth has created an algorithm that finds a solution to the exact cover problem defined by such matrix. He decides to call it algorithm X “for lack of a better name” as he states in his paper.

Algorithm X works in the following way:

1. If there are no unsatisfied constraints remaining, the solution set is complete. Otherwise, pick an unsatisfied constraint (a column that does not contain a 1 in any of the rows in the solution set).
2. Pick a row that satisfies that constraint (one that has 1 at its intersection with the chosen column). If no such row exists, the solution cannot be continued. Backtrack to the previous time a row was chosen and select the next one instead.
3. Add that row to the solution set.
4. Delete all rows that satisfy any of the constraints satisfied by the chosen row:that is, all rows that have 1 in the same column as any of the cells containing 1 in the chosen row.
5. Return to Step 1.

2.3. Machine Learning

2.3.1. Feature Engineering

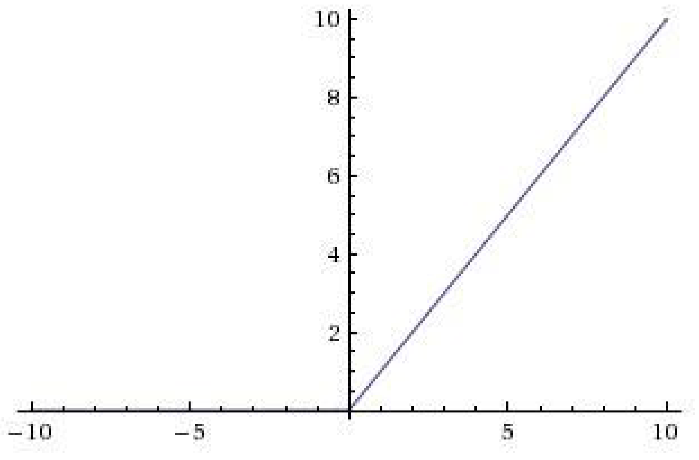
The essence of machine learning lies in performing metadata analysis. The very first step in doing machine learning is feature engineering. We can't use the raw information, which is, in our case images, as an input to the classifier. We need to process it first and extract the appropriate features from it. We use the combined set of features which is called a feature vector as the input. To identify objects in images, we can compile a vector from some of the object descriptors explained in the previous section. However, since we need to recognize digits, because of the properties of their shape, there is a more appropriate approach. That is, removing color from the images and scaling them so that they all are of equal size and the number of pixels in each is small enough to make processing possible. The values of the pixels, then, form a feature
vector.

2.3.2. Convolutional neural network (CNN)

CNNs are an expansion of multilayer perceptron, which might learn filters that require to be processed by the machine learning models, as tested prior in [Yann LeCun et al. 1989] utilizing back-propagation. Convolutional networks are primarily applied on visual symbolism. Since the preparation cycle includes finding out about examples from more modest examples, A convolutional neural organization comprises of particular covered up layers additionally to the information and yield layers.
These particular layers usually incorporate convolutional layer with filters which will be learned, rectified long measure layer for utilization of enactment work, pooling layer for down-testing and misfortune layer for determination of punishment for wrong output. We use Keras with TensorFlow as backend to handle the CNN model and supply a examination between the particular CNN usage framed by picking distinctive hyperparameters related with each layer.

2.3.3. Activation Functions

We have used SoftMax activation function for the output layer and ReLU activation function for other layers.
ReLU: The Rectified Linear Unit is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value x it returns that value back. So, it can be written as:
f(x)=max(0,x)
Graphically it looks like this:



Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector.
The most common use of the softmax function in applied machine learning is in its use as an activation function in a neural network model. Specifically, the network is configured to output N values, one for each class in the classification task, and the softmax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one. Each value in the output of the softmax function is interpreted as the probability of membership for each class.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

2.3.4. Optimizers

Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum.

$$For\ each\ Parameter\ w^j$$

$$(j\ subscript\ dropped\ for\ clarity)$$

$$\nu_t = \beta_1 * \nu_{t-1} - (1-\beta_1) * g_t$$
$$s_t = \beta_2 * s_{t-1} - (1-\beta_2) * g_t^2$$

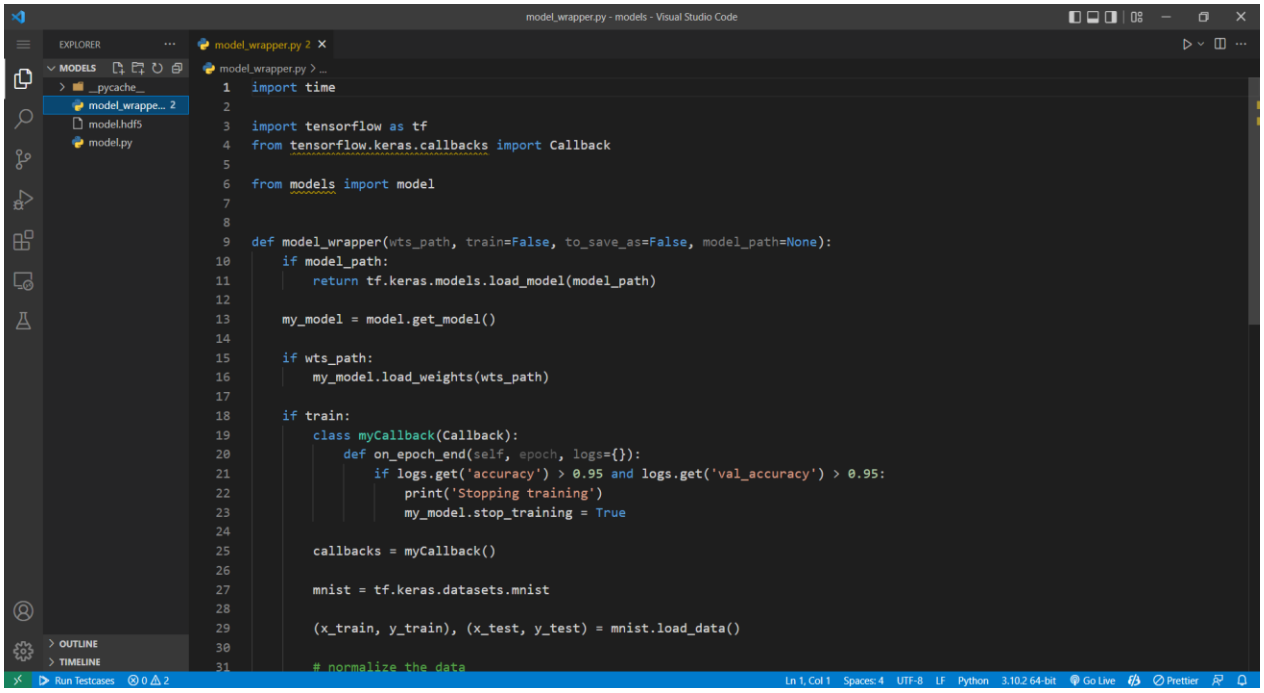
$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

η : *Initial Learning rate*
g_t : *Gradient at time t along* ω^j
ν_t : *Exponential Average of gradients along* ω_j
s_t : *Exponential Average of squares of gradients along* ω_j
β_1, β_2 : *Hyperparameters*

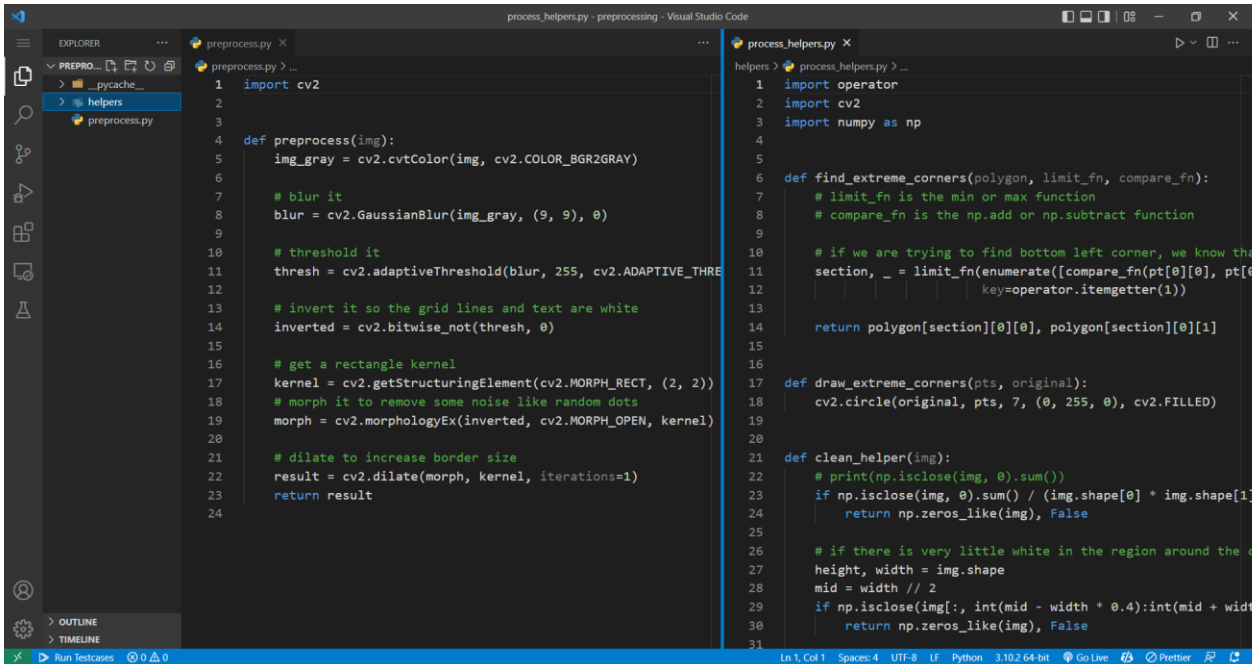
3. Flow of code

First, we train our model (Model Name: **model. hdf5**) by using MNIST dataset to recognize the digits (1-9) effectively,


model_wrapper.py - models - Visual Studio Code
EXPLORER
MODELS
pycache
model_wrappe... 2
model.hdf5
model.py
OUTLINE
TIMELINE
Run Testcases
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.10.2 64-bit Go Live Prettier

```python
import time

import tensorflow as tf
from tensorflow.keras.callbacks import Callback

from models import model


def model_wrapper(wts_path, train=False, to_save_as=False, model_path=None):
    if model_path:
        return tf.keras.models.load_model(model_path)

    my_model = model.get_model()

    if wts_path:
        my_model.load_weights(wts_path)

    if train:
        class myCallback(Callback):
            def on_epoch_end(self, epoch, logs={}):
                if logs.get('accuracy') > 0.95 and logs.get('val_accuracy') > 0.95:
                    print('Stopping training')
                    my_model.stop_training = True

        callbacks = myCallback()

        mnist = tf.keras.datasets.mnist

        (x_train, y_train), (x_test, y_test) = mnist.load_data()

        # normalize the data
```

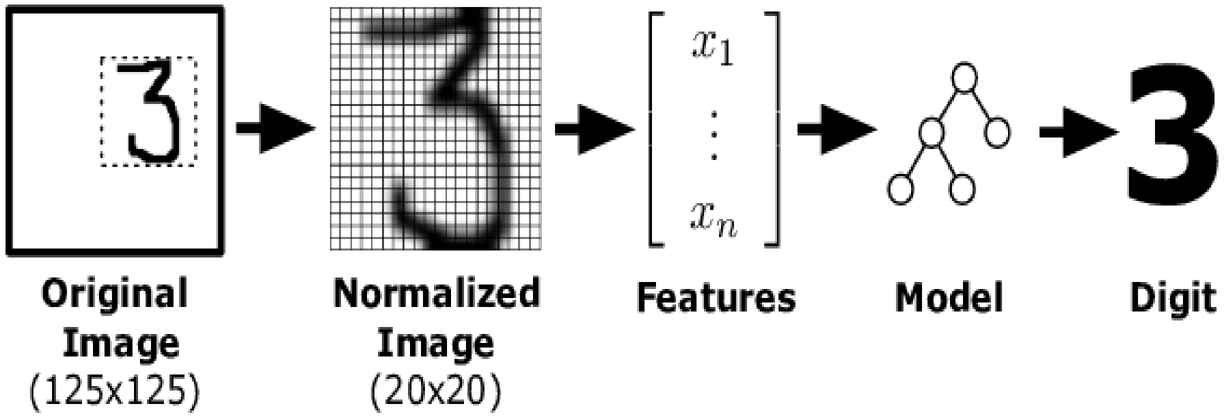
We use OpenCV (python library) to get an image, via **cv2.VideoCapture(0)** and convert it into grayscale via cv2.COLOR_BGR2GRAY, apply threshold (blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2), after that find the largest contour (9X9 matrix), then we find small contour (3X3 matrix), after that we find small square(grid).



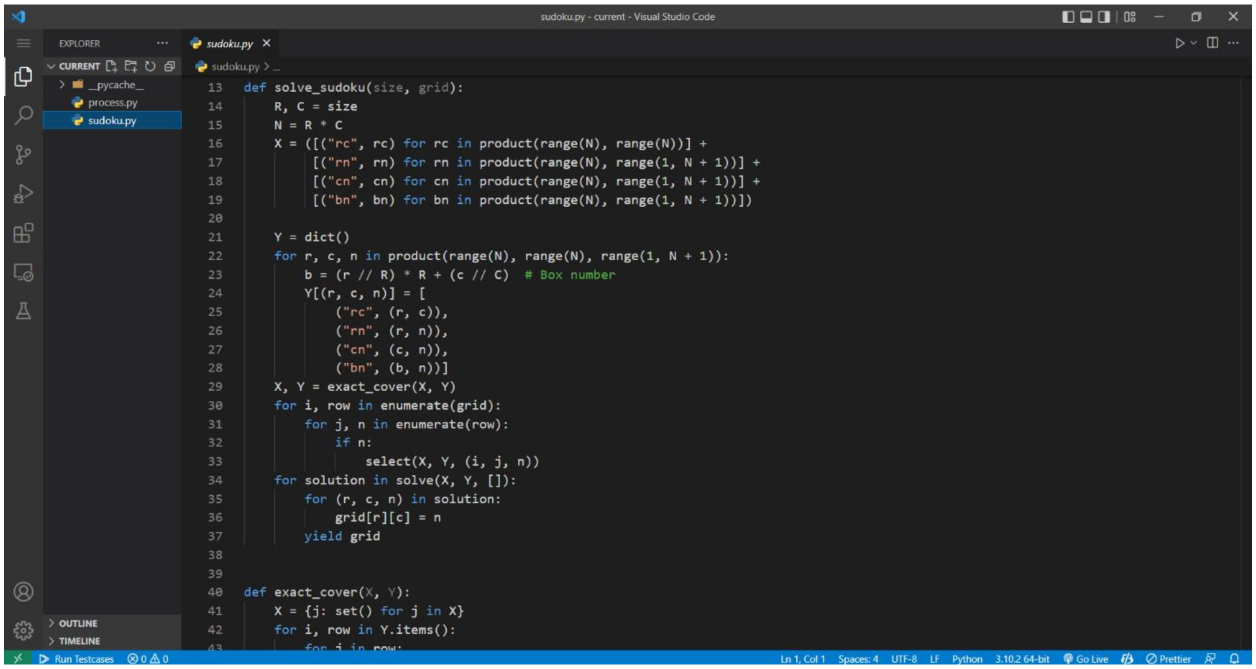
```python
import cv2


def preprocess(img):
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # blur it
    blur = cv2.GaussianBlur(img_gray, (9, 9), 0)

    # threshold it
    thresh = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRE

    # invert it so the grid lines and text are white
    inverted = cv2.bitwise_not(thresh, 0)

    # get a rectangle kernel
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))
    # morph it to remove some noise like random dots
    morph = cv2.morphologyEx(inverted, cv2.MORPH_OPEN, kernel)

    # dilate to increase border size
    result = cv2.dilate(morph, kernel, iterations=1)
    return result

```

```python
import operator
import cv2
import numpy as np


def find_extreme_corners(polygon, limit_fn, compare_fn):
    # limit_fn is the min or max function
    # compare_fn is the np.add or np.subtract function

    # if we are trying to find bottom left corner, we know tha
    section, _ = limit_fn(enumerate([compare_fn(pt[0][0], pt[0
                          key=operator.itemgetter(1))

    return polygon[section][0][0], polygon[section][0][1]


def draw_extreme_corners(pts, original):
    cv2.circle(original, pts, 7, (0, 255, 0), cv2.FILLED)


def clean_helper(img):
    # print(np.isclose(img, 0).sum())
    if np.isclose(img, 0).sum() / (img.shape[0] * img.shape[1]
        return np.zeros_like(img), False

    # if there is very little white in the region around the c
    height, width = img.shape
    mid = width // 2
    if np.isclose(img[:, int(mid - width * 0.4):int(mid + widt
        return np.zeros_like(img), False

```

Once we get data from grid, we use our trained model to recognize the digit written in grid.



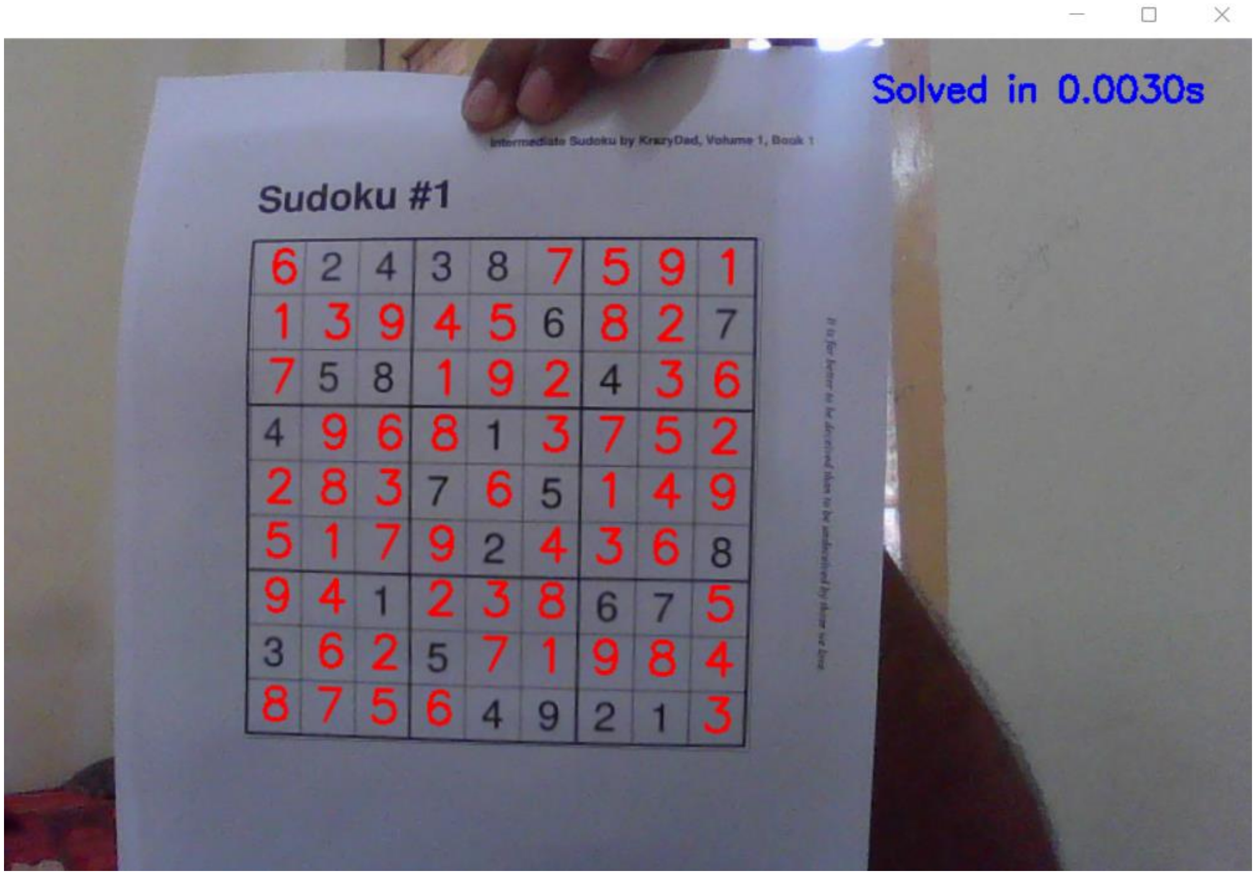
After then we apply sudoku solver Algorithm (Knuth's Algorithm X).


sudoku.py - current - Visual Studio Code
EXPLORER
CURRENT
__pycache__
process.py
sudoku.py
OUTLINE
TIMELINE
Run Testcases
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.10.2 64-bit Go Live Prettier

```python
def solve_sudoku(size, grid):
    R, C = size
    N = R * C
    X = ([("rc", rc) for rc in product(range(N), range(N))] +
         [("rn", rn) for rn in product(range(N), range(1, N + 1))] +
         [("cn", cn) for cn in product(range(N), range(1, N + 1))] +
         [("bn", bn) for bn in product(range(N), range(1, N + 1))])

    Y = dict()
    for r, c, n in product(range(N), range(N), range(1, N + 1)):
        b = (r // R) * R + (c // C)  # Box number
        Y[(r, c, n)] = [
            ("rc", (r, c)),
            ("rn", (r, n)),
            ("cn", (c, n)),
            ("bn", (b, n))]
    X, Y = exact_cover(X, Y)
    for i, row in enumerate(grid):
        for j, n in enumerate(row):
            if n:
                select(X, Y, (i, j, n))
    for solution in solve(X, Y, []):
        for (r, c, n) in solution:
            grid[r][c] = n
        yield grid


def exact_cover(X, Y):
    X = {j: set() for j in X}
    for i, row in Y.items():
        for j in row:
```

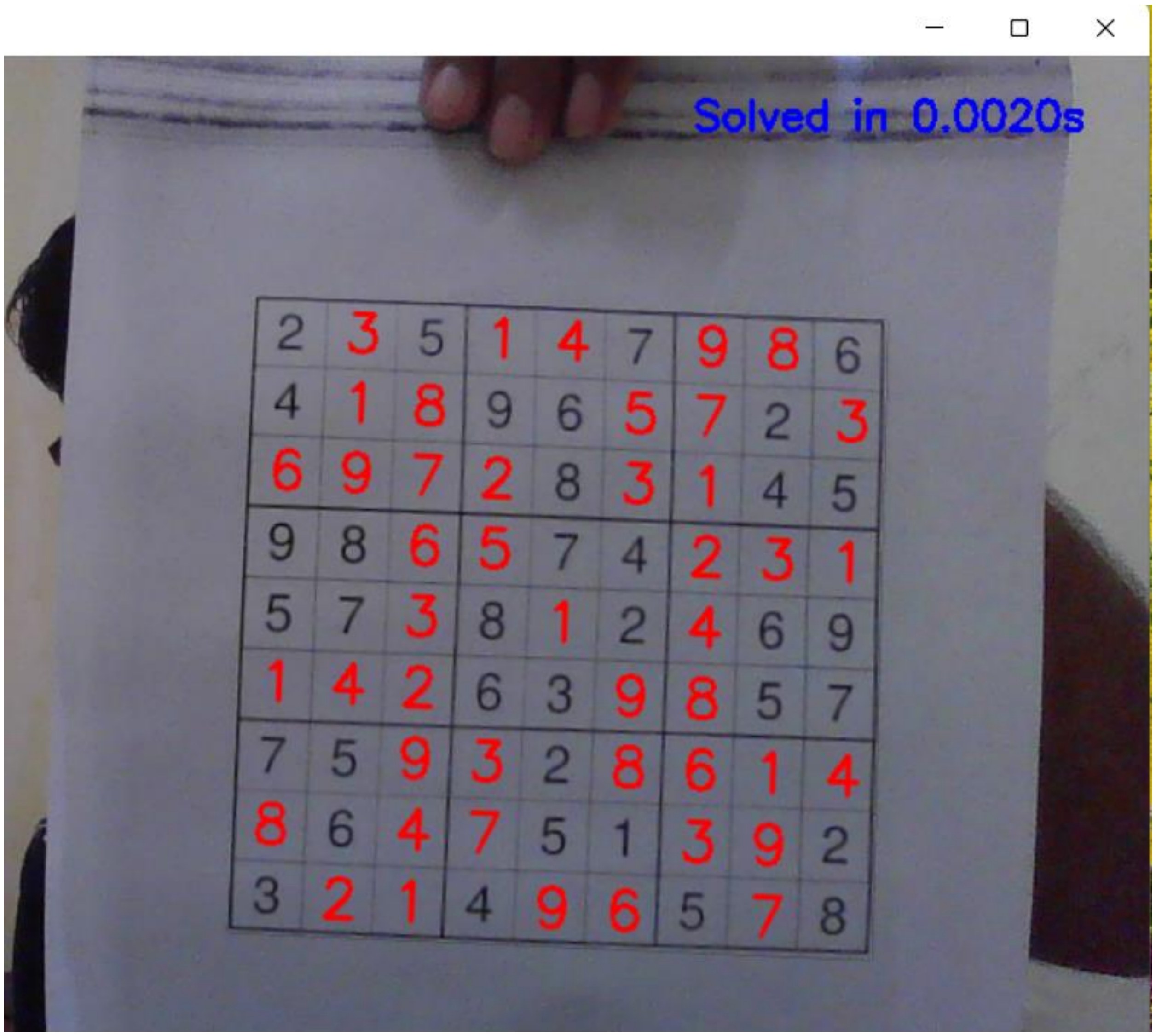
There is some Screenshot taken while solving the Sudoku via code:

1.



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

2.



Solved in 0.0020s

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

4. Procedure

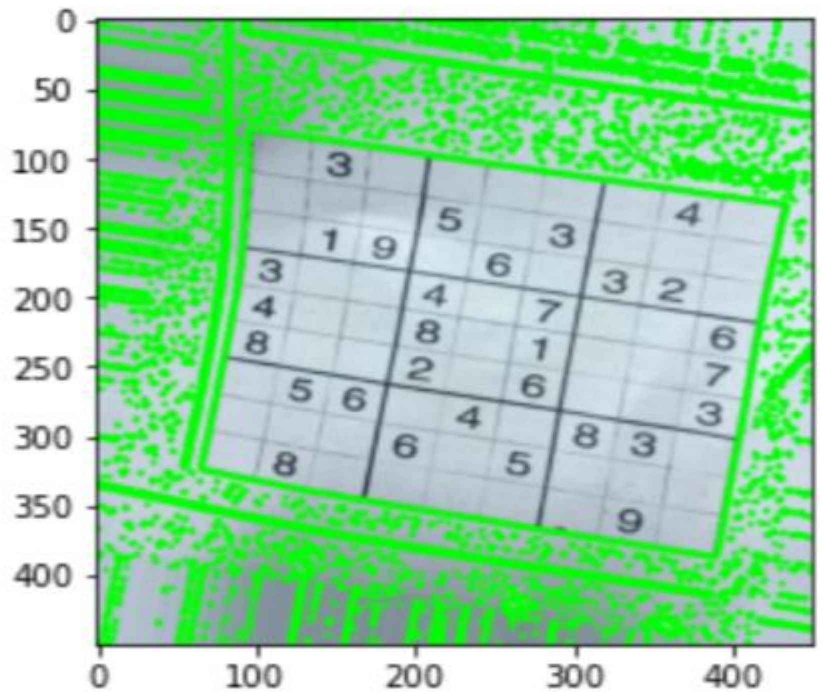
4.1. Training the model

1. In the first step, we load the MNIST dataset. We also import all the libraries required such as OpenCV, NumPy, TensorFlow, Keras etc.

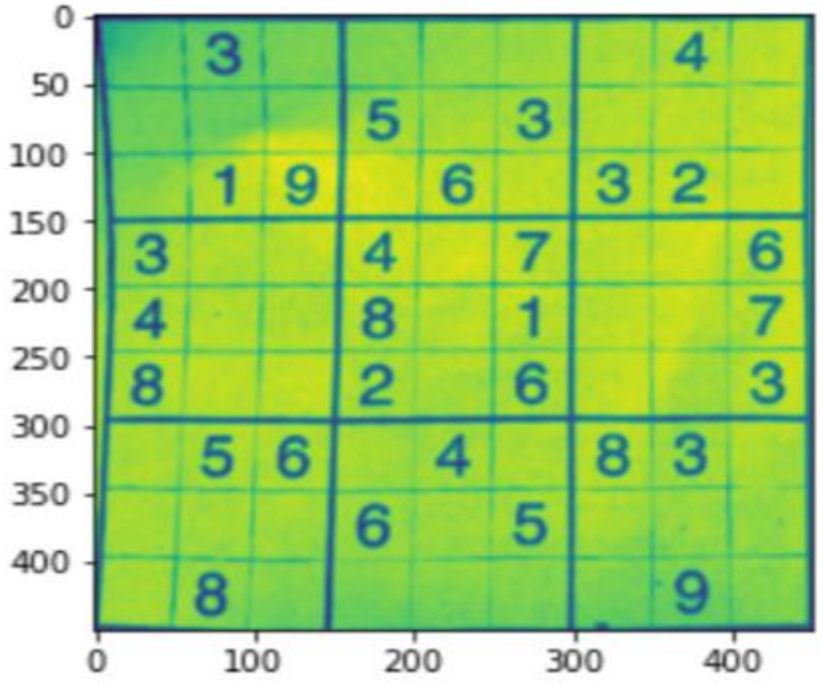
2. Now, we add various layers such as Conv2D, MaxPooling2D, Flatten, Dropout and Dense to our Sequential CNN model. In the last Dense layer, we use the 'SoftMax' activation function to output a vector that gives the probability of each of the 10 classes (i.e., digits from 0-9). Here we use the *'adam'* optimizer and 'categorical_crossentropy*'* as our loss function.
3. This step is the main step where we fit our images in the training set and the test set to our Sequential model that we built using keras library. We have trained the model for 10 epochs (iterations). However, we can train for greater number of epochs to attain higher accuracy lest there occurs over-fitting. We save our model so that we can easily load it for predictions.

4.2. Detecting the Contour

Now, we have a model that can perceive digits in the picture. Nonetheless, that digit recognizer doesn't do us much good in the event that we cannot find the sudoku puzzle board in a picture. We are going to detect contour. We sill detect the biggest contour of the image.



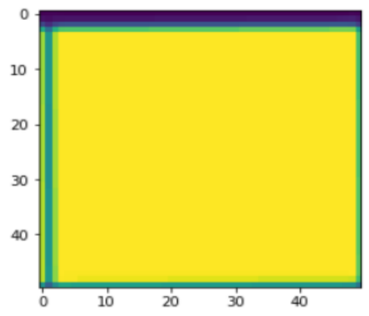
Grayscale image:



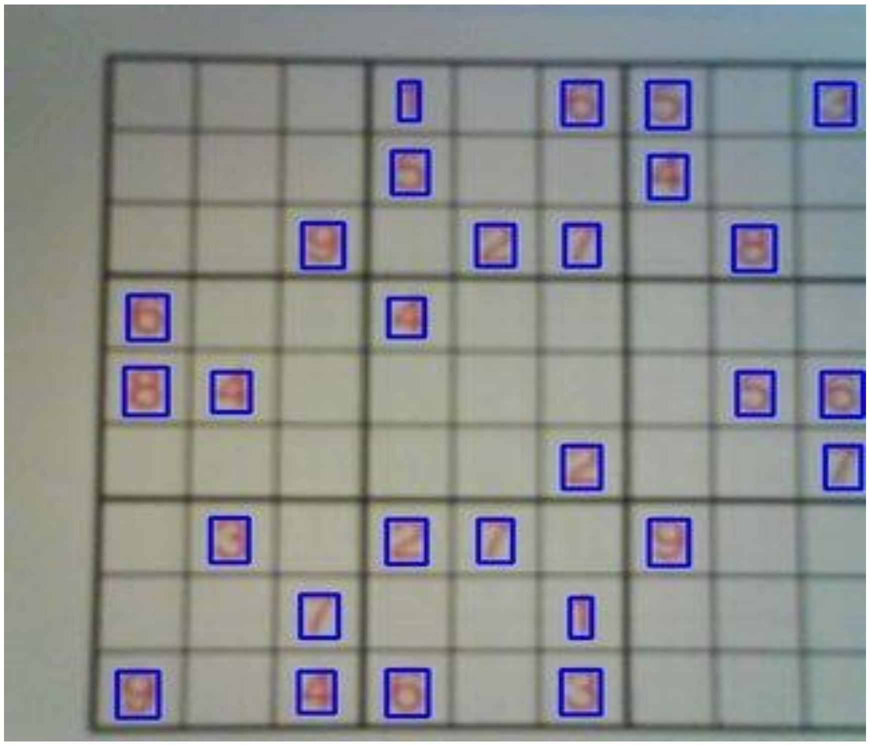
4.3. Separating squares and cropping digits

In this section, we are going to split the Square and crop the digits

- First split the Sudoku into 81 cells with digits or empty spaces
- Cropping the cells
- Using the model to classify the digits in the cells so that the empty cells are classified as zero
- Finally, detect the output in an array of 81 digits.



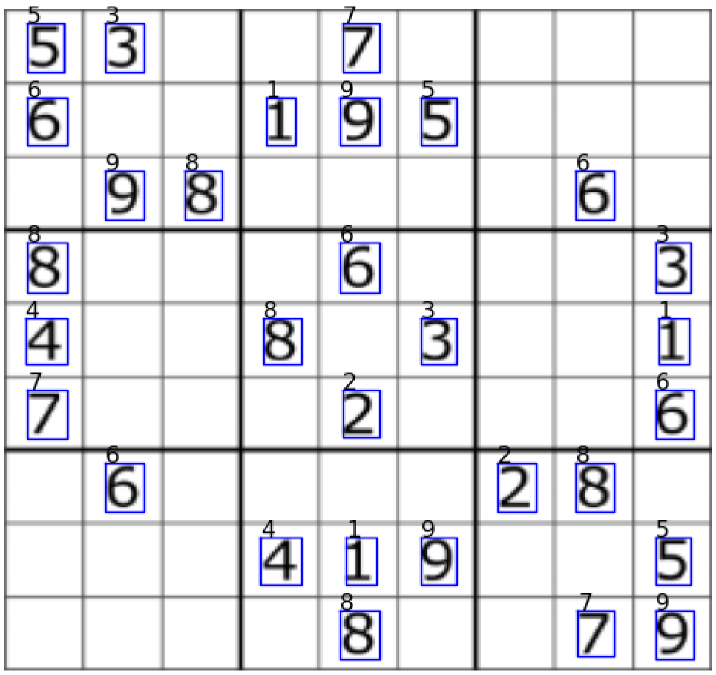
Grid



Cropping Digit from Grid

4.4. Digits Recognition

We use our trained Model to recognize the digit in grid

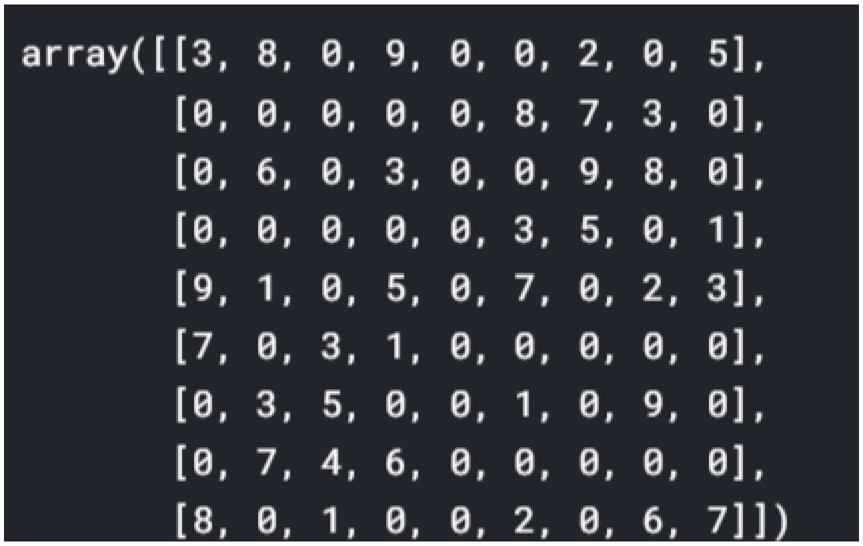


Digit Recognition

4.5. Solving the puzzle

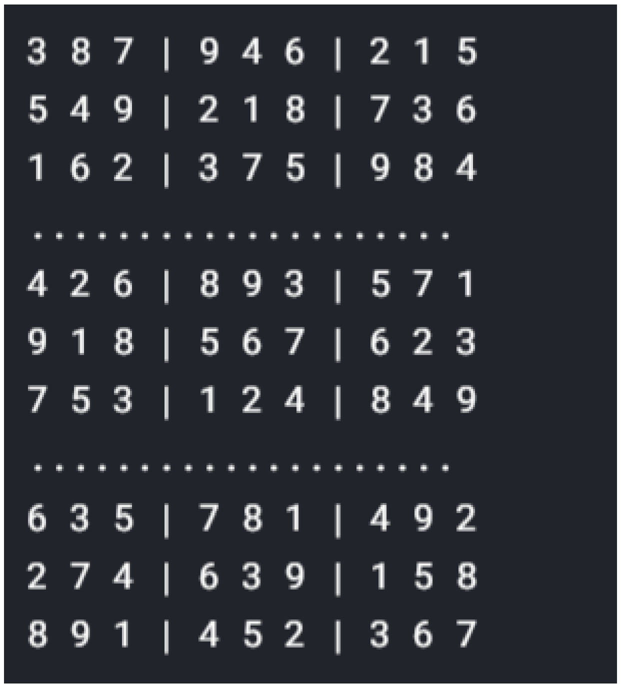
In this section, we are going to perform two operations:

- Reshaping the array into a 9 x 9 matrix



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

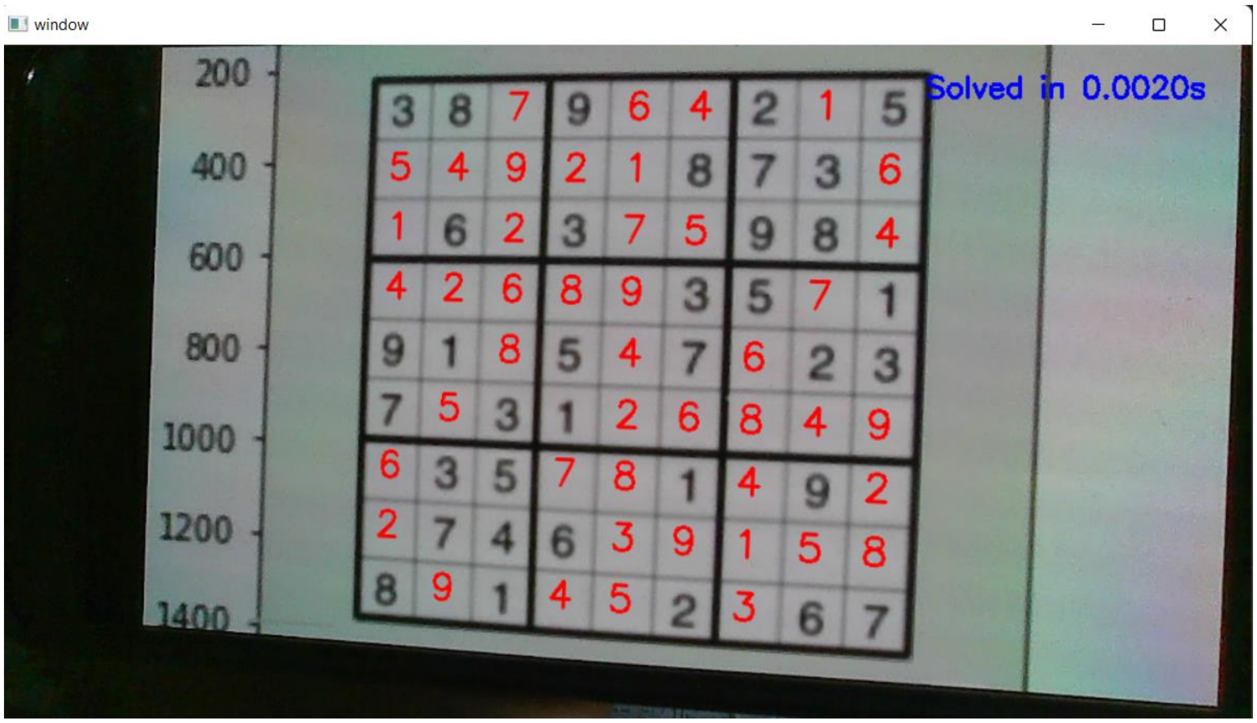
- Solving the matrix using Knuth's Algorithm X



```
3 8 7 | 9 4 6 | 2 1 5
5 4 9 | 2 1 8 | 7 3 6
1 6 2 | 3 7 5 | 9 8 4
.....................
4 2 6 | 8 9 3 | 5 7 1
9 1 8 | 5 6 7 | 6 2 3
7 5 3 | 1 2 4 | 8 4 9
.....................
6 3 5 | 7 8 1 | 4 9 2
2 7 4 | 6 3 9 | 1 5 8
8 9 1 | 4 5 2 | 3 6 7
```

5.Result

After Applying algo to solve the sudoku we get the final output:



Solved in 0.0020s

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

6. Limitations

- Our model is 85% accurate i.e., sometime our model predicts the wrong digit thus, sudoku doesn't solve.
- Cannot solve puzzles that don't have a distinguishable four-point outer border.
- Since, our project work on real time so, lightning is important while webcam is ON.

7. Summary and conclusion

A full-fledged system for showing the solution of a Sudoku puzzle as Augmented Reality was proposed in the paper. Using image processing, object localization, image classification, constraint programming and algorithm X.

8. References

- https://www.kaggle.com/competitions/digit-recognizer/data?select=train.csv (MNIST dataset to train our model)
- https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html (opencv VideoCapture)
- https://docs.opencv.org/4.x/db/deb/tutorial_display_image.html (opencv wokring on images)
- https://en.wikipedia.org/wiki/Knuth%27s_Algorithm_X (Knuth’s Algorithm X, for solving Sudoku)
- https://www.analyticsvidhya.com/blog/2021/10/a-beginners-guide-to-feature-engineering-everything-you-need-to-know/ (feature engineering)
- https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/ (convolutional neural network, CNN)
- https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/ (activation function)
- https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/ (optimizers in machine learning)