**Sudoku Solver Robotics Project**

****

### **INTRODUCTION -**

Like many others, you also must have played the sudoku game at some point in your life. It comes with a newspaper like The Times of India, The Hindu, etc., or at the back of our notebook. Everyone knows how to solve the game, but we have to scratch our heads for hours to get to the correct answer. Weprediction model will learn how to build a python script to take a sudoku image and display the solved image

**The rules of the game:**

It’s all about putting digits between 1 and 9 into a square, 9x9 grid, subdivided into nine boxes. But The value of a cell can’t be repeated amongst any of its peers.

**PRE-REQUISITES OF THE PROJECT -**

* ​Basics of Python programming language
* Good understanding of Opencv library and image processing techniques.
* Basics of Deep Learning and Tensorflow-Keras.

### **TECHSTACK AND INSTALLATION -**

* Python - 3.8.6 (Download from [here](https://www.python.org/) )
* Tensorflow Keras - ( pip install TensorFlow or follow from [here](https://www.tensorflow.org/install/pip) )
* PIL - ( pip install pillow )
* Numpy ( pip install NumPy )
* Opencv ( pip install opencv-python )
* Matplotlib ( only for visualization of datasets - pip install matplotlib )

### **SUDOKU EXTRACTION -**

### 

### 

### 

### 

### 

### 

### 

### 

### 

### **Input image (unsolved sudoku image)**

### 

* **Preprocessing of the image-**

### Converts grey scaled version Original image into using a gaussian blur function, adaptive thresholding, and dilation to expose an image’s main features.

def pre\_process\_image(img, skip\_dilate=False):

proc = cv2.GaussianBlur(img.copy(), (9, 9), 0)

proc = cv2.adaptiveThreshold(

proc, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY,11,2)

proc = cv2.bitwise\_not(proc, proc)

*if* not skip\_dilate:

kernel = np.array([[0., 1., 0.], [1., 1., 1.], [0., 1., 0.]], np.uint8)

proc = cv2.dilate(proc, kernel)

*return* proc

### 

### 

### 

### 

### 

### 

### 

Preprocessed image

### 

* **FInding the contour of the largest polygon-**

### We are finding the four extreme corners of the most prominent contour in the image. The image which we pre-processed is used here.

def find\_corners\_of\_largest\_polygon(img):

opencv\_version = cv2.\_\_version\_\_.split('.')[0]

*if* opencv\_version == '3':

p

\_, contours, h = cv2.findContours(

img.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

*else*:

contours, h = cv2.findContours(

img.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE

contours = sorted(contours, key=cv2.contourArea,

reverse=True)

polygon = contours[0]

bottom\_right, \_ = max(

enumerate([pt[0][0] + pt[0][1] *for* pt *in* polygon]), key=operator.itemgetter(1))

top\_left, \_ = min(enumerate([pt[0][0] + pt[0][1]

*for* pt *in* polygon]), key=operator.itemgetter(1))

bottom\_left, \_ = min(enumerate([pt[0][0] - pt[0][1]

*for* pt *in* polygon]), key=operator.itemgetter(1))

top\_right, \_ = max(enumerate([pt[0][0] - pt[0][1] *for* pt *in* polygon]),key=operator.itemgetter(1))

*return* [polygon[top\_left][0], polygon[top\_right][0], polygon[bottom\_right][0], polygon[bottom\_left][0]]

* **Cropping and warping of the image-**

### I am cropping and warping a rectangular section from an image into a square of similar size.

def crop\_and\_warp(img, crop\_rect):

top\_left, top\_right, bottom\_right, bottom\_left = crop\_rect[

0], crop\_rect[1], crop\_rect[2], crop\_rect[3]

src = np.array([top\_left, top\_right, bottom\_right,

bottom\_left], dtype='float32')

side = max([

distance\_between(bottom\_right, top\_right),

distance\_between(top\_left, bottom\_left),

distance\_between(bottom\_right, bottom\_left),

distance\_between(top\_left, top\_right)

])

dst = np.array([[0, 0], [side - 1, 0], [side - 1, side - 1],

[0, side - 1]], dtype='float32')

m = cv2.getPerspectiveTransform(src, dst)

*return* cv2.warpPerspective(img, m, (int(side), int(side)))

* **Infer grid from the square image**

### Infers 81 cell grid from a square image.

def infer\_grid(img):

squares = []

side = img.shape[:1]

side = side[0] / 9

*for* j *in* range(9):

*for* i *in* range(9):

p1 = (i \* side, j \* side)

p2 = ((i + 1) \* side, (j + 1) \* side)

squares.append((p1, p2))

*return* squares

* **Get each digit**

### The next step is to extract digits from their cells and build an array.

def get\_digits(img, squares, size):

digits = []

img = pre\_process\_image(img.copy(), skip\_dilate=True)

*for* square *in* squares:

digits.append(extract\_digit(img, square, size))

*return* digits

### 

### 

### 

### 

### 

### 

### 

### 

### 

### **DIGIT RECOGNITION -**

* **Extract each number present in the image**

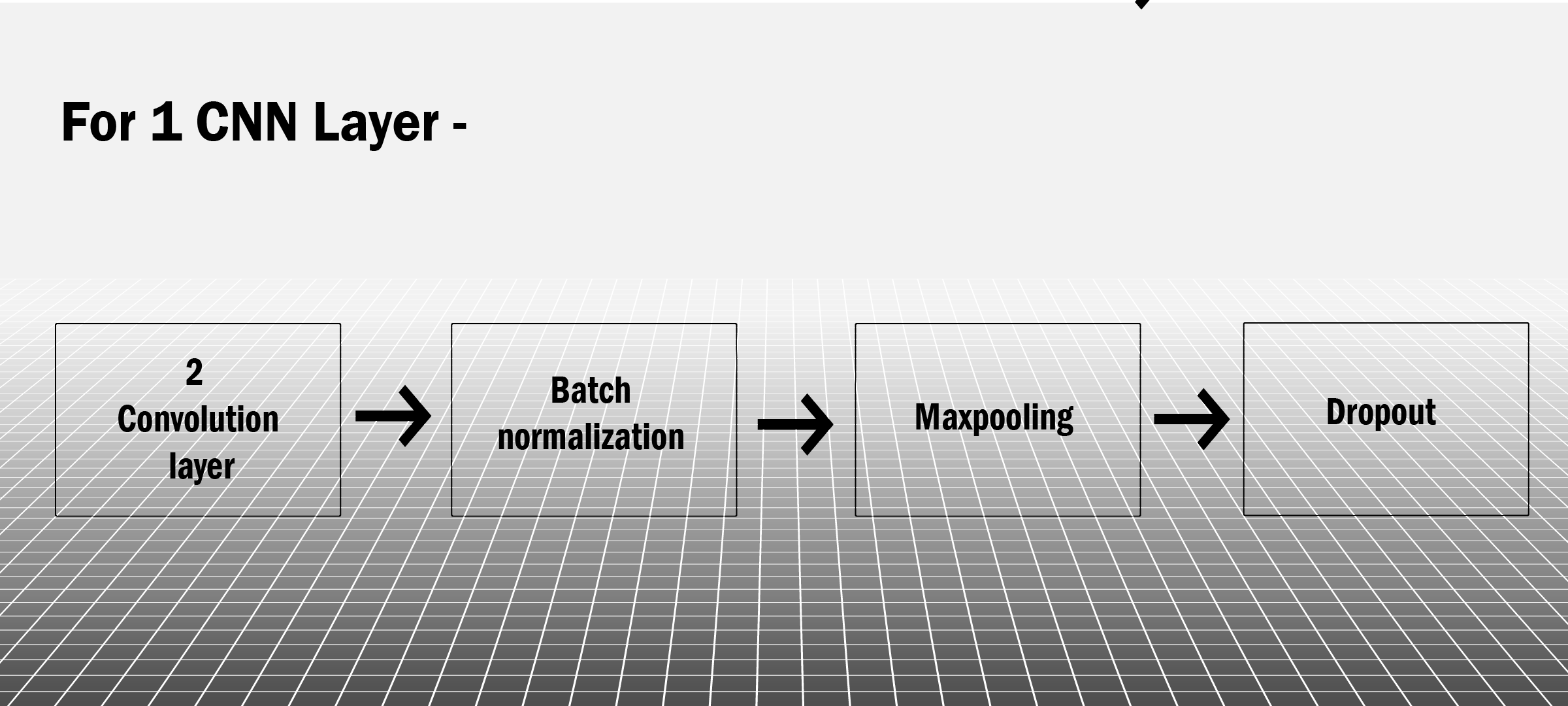
### For digit recognition, we used a neural network over the MNIST dataset containing 60,000 images of digits from 0 to 9.

* **Model creation**

### Next, we will create a triple CNN model to predict the handwritten digit.

### Here, we will be roughly using the below Deep learning pipeline on each CNN.

3 Dense layers follow this. Each containing Batch Normalization and Dropout with relu(Rectified Linear Unit ) as the activation function, and the last layer is the output layer containing ten neurons with softmax. Softmax is taken as the activation function to find the probabilities of the outputs.



model = tf.keras.models.Sequential([

# first CNN

Conv2D(32, kernel\_size=(3, 3), activation='relu', padding='same', input\_shape = (28,28,1)),

Conv2D(32, kernel\_size=(3, 3), activation='relu', padding='same'),

BatchNormalization(),

MaxPool2D(pool\_size=(2, 2)),

Dropout(0.25),

#second CNN

Conv2D(64, kernel\_size=(3, 3), activation='relu', padding='same'),

Conv2D(64, kernel\_size=(3, 3), activation='relu', padding='same'),

BatchNormalization(),

MaxPool2D(pool\_size=(2, 2)),

Dropout(0.25),

# third CNN

Conv2D(128, kernel\_size=(3, 3), activation='relu',padding='same'),

Conv2D(128, kernel\_size=(3, 3), activation='relu',padding='same'),

BatchNormalization(),

MaxPool2D(pool\_size=(2, 2)),

Dropout(0.25),

#

Flatten(),

Dense(512, activation='relu'),

BatchNormalization(),

Dropout(0.5),

Dense(256, activation='relu'),

BatchNormalization(),

Dropout(0.4),

Dense(64, activation='relu'),

BatchNormalization(),

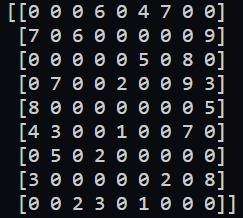
Dropout(0.3),

Dense(10, activation = "softmax")

])

* **Prediction through model**

### Extract each image square from the final image (final processed image) and then give it to the model for prediction and then store the resulting image in the digit array.



### 

### 

### 

### 

### 

### **Unsolved array**

### 

### **COMPUTING THE SOLUTION -**

### We have used the recursive backtracking algorithm on the digit array for the solution computation of Sudoku.

def solve\_board(board):

find = find\_empty(board)

*if* not find:

*return* True

*else*:

(row, col) = find

*for* i *in* range(1, 10):

*if* is\_valid(board, i, (row, col)):

board[row][col] = i

*if*(solve\_board(board)):

*return* True

board[row][col] = 0

*return* False

### 

### 

### 

### 

### 

### 

### **Solved array**

### 

### **RENDERING THE SOLUTION -**

### we divide the warped image into small sub-images, then using PIL, we draw the digit in the square using a solved array. Finally, combine all the squares into the picture (warped image), using inverse homographic transform to transform the warped image’s plane into the original image.

def inverse\_perspective(img, dst\_img, pts):

pts\_source = np.array([[0, 0], [img.shape[1] - 1, 0], [img.shape[1] - 1, img.shape[0] - 1], [0, img.shape[0] - 1]],

dtype='float32')

h, status = cv2.findHomography(pts\_source, pts)

warped = cv2.warpPerspective(img, h, (dst\_img.shape[1], dst\_img.shape[0]))

cv2.fillConvexPoly(dst\_img, np.ceil(pts).astype(int), 0, 16)

dst\_img = dst\_img + warped

*return* dst\_img

### 

### 

### 

**Team Information**

1. Pravesh Pandey (20193065)
2. Johnit Kujur (20194011)
3. Shreyas Kolhatkar (20195132)
4. Parnab Ghosh (20195036)

**Mentors**

1. Bhuvan Jhamb
2. Anuj Tiwari
3. Apoorva Dinesh Singh

**References :**

[Source Code (GitHub Repo)](https://github.com/parnabghosh1004/sudoku-solver)

[Opencv Python Official Documentation](https://docs.opencv.org/master/d6/d00/tutorial_py_root.html)

[Tensorflow Official Documentation for python](https://www.tensorflow.org/api_docs)

### 