Review 2

Conundrum Unraveller using CNN/KNN

Tanishq Padwal 18BCE2237

Nikhil Philip Chako 18BBCE2185

23rd October, 2020

Introduction

Abstract :

In the last decade, solving the Sudoku puzzle has become every one’s passion. The simplicity of puzzle’s structure and the low requirement of mathematical skills caused people to have enormous interest in accepting challenges to solve the puzzle.

Many puzzles such as Kakuro, Sudoku, Killer Sudoku etc are solved by people of various age groups on a daily basis. These puzzles come in a variety of difficulty levels and sizes that at times are not solvable or require help from different sources to be completed.

In this project, we have presented a way to solve these complicated puzzles using various Image recognition and preprocessing techniques and algorithms. The purpose is to implement an efficient algorithm and then use image recognition to take input of question and solve the puzzle using CNN/KNN which are machine learning algorithms.

We have finished 95% implementation of the code and the last 5% is some bug that we are trying to overcome. We have achieved satisfactory results which are accurate.

Problem Statement :

Many puzzles such as Kakuro, Sudoku, Killer Sudoku etc are solved by people of various age groups on a daily basis. These puzzles come in a variety of difficulty levels and sizes that at times are not solvable or require help from different sources to be completed.

Thus to make solving such puzzles easier in absence of an internet connection, we will be implementing a puzzle solver using machine learning algorithms such as CNN/KNN and image processing techniques to extract puzzle from image using an user friendly Python based GUI.

Tools and algorithms used :

Python 3.8.2

- Keras-Applications==1.0.8
- Keras-Preprocessing==1.1.0
- numpy==1.17.2
- opencv-contrib-python==4.1.1.26
- Pillow==6.2.0
- scikit-learn==0.21.3
- scipy==1.3.1
- sklearn==0.0
- tensorboard==1.14.0
- tensorflow==1.15.2
- tensorflow-estimator==1.14.0

Image preprocessing techniques used :

Gaussian Blurring :

Blurring using a Gaussian function. This is to reduce noise and detail.

Adaptive Gaussian Thresholding :

Adaptive thresholding with a Gaussian Function to account for different illuminations in different parts of the image.

Inverting :

Inverting to make the digits and lines white while making the background black.

Dilation :

Dilation with a plus shaped 3X3 Kernel to fill out any cracks in the board lines and thicken the board lines.

Flood Filling :

Since the board will probably be the largest blob a.k.a connected component with the largest area, floodfilling from different seed points and finding all connected components followed by finding the largest floodfilled area region will give the board

Hough Line Transform :

Hough Line Transform to find all the lines in the detected outerbox.

Merging related lines. The lines found by the Hough Transform that are close to each other are fused together.

Thresholding and Inverting the grid :

The cropped image from the previous step is adaptive thresholded and inverted.

Slicing :

Slicing the grid into 81 slices to get images of each cell of the Sudoku board.

Blackfilling and centering the number :

Any white patches other than the number are removed by floodfilling with black from the outer layer points as seeds. Then the approximate bounding box of the number is found and centered in the image.

Recognition :

Convolutional Neural Network :

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

K Nearest Neighbours :

A supervised machine learning algorithm (as opposed to an unsupervised machine learning algorithm) is one that relies on labeled input data to learn a function that produces an appropriate output when given new unlabeled data.

Dataset Used :

MNIST handwritten digits dataset which has around 70,000 28X28 images which includes numbers from 0-9.

Source :

http://yann.lecun.com/exdb/mnist/

CNN has around 98 percent accuracy on the test set.

KNN uses K=3 and around 97 percent accuracy on the test set.

Input / Output :

Input :

The GUI Homepage that opens up as soon as you run the application.You need to select an image of a Sudoku Puzzle through the GUI Home Page.Once you press Next, a number of stages of image processing take place which are displayed by the GUI leading up to recognitionFor recognition, a CNN or KNN can be used. This option can be toggled. Once recognized, the board is displayed and you can rectify any wrongly recognized entries in the board.Finally click on reveal solution to display the solution.

Output :

After pressing on reveal solution, the stated processing takes place which then returns the solution and missing values from the original Sudoku puzzle which was used.All solution values displayed are on the GUI.

Code :

SudokuSolver.py :

```
from operator import attrgetter

class Solver:

    def checkvalidpuzzle(self, arr):

        subsquarestartingpoints = [[0, 0], [0, 3], [0, 6], [3, 0], [3, 3], [3, 6], [6, 0], [6, 3], [6, 6]]

        # Checking row validity of every row

        for row in range(9):

            has = set()

            for col in range(9):

                if arr[row][col] == 0:

                    continue

                if arr[row][col] in has:

                    return False

                has.add(arr[row][col])

        # Checking column validity of every column

        for col in range(9):

            has = set()

            for row in range(9):

                if arr[row][col] == 0:

                    continue

                if arr[row][col] in has:
                    return False
                has.add(arr[row][col])
        # Checking box validity
        for pointrow, pointcol in subsquarestartingpoints:
            has = set()
            for row in range(3):
                for col in range(3):
                    if arr[pointrow+row][pointcol+col] == 0:
                        continue
                    if arr[pointrow+row][pointcol+col] in has:
                        return False
                    has.add(arr[pointrow+row][pointcol+col])
        return True

    def print_board(self, arr):
        for i in range(9):
            for j in range(9):
                if arr[i][j]==0:
                    print("_", end=" ")
                else:
                    print(arr[i][j], end=" ")
            print("")
```

Training KNN model :

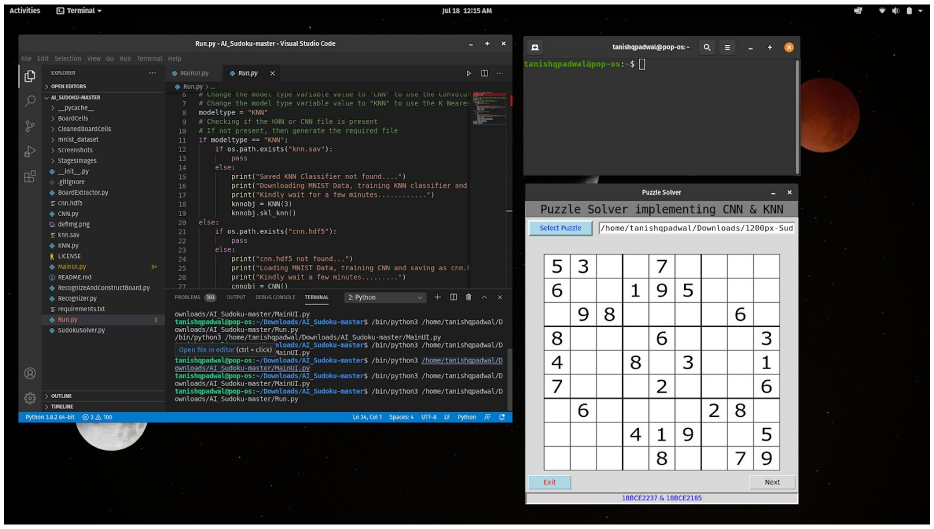
```
import numpy as np

from sklearn import datasets

from sklearn.metrics import classification_report

from sklearn.neighbors import KNeighborsClassifier

import pickle

k = 3

class KNN:

    def __init__(self, k):

        self.mnist = datasets.fetch_openml('mnist_784',
data_home='mnist_dataset/')

        self.data, self.target = self.mnist.data, self.mnist.target

        self.indx = np.random.choice(len(self.target), 70000,
replace=False)

        self.classifier = KNeighborsClassifier(n_neighbors=k)

    def mk_dataset(self, size):

        train_img = [self.data[i] for i in self.indx[:size]]

        train_img = np.array(train_img)

        train_target = [self.target[i] for i in self.indx[:size]]

        train_target = np.array(train_target)

        return train_img, train_target

    def skl_knn(self):

        fifty_x, fifty_y = self.mk_dataset(50000)

        test_img = [self.data[i] for i in self.indx[60000:70000]]
        test_img1 = np.array(test_img)

        test_target = [self.target[i] for i in self.indx[60000:70000]]

        test_target1 = np.array(test_target)

        self.classifier.fit(fifty_x, fifty_y)


        y_pred = self.classifier.predict(test_img1)

        pickle.dump(self.classifier, open('knn.sav', 'wb'))

        print(classification_report(test_target1, y_pred))

        print("KNN Classifier model saved as knn.sav!")
```

Choose between KNN or CNN :

```
from MainUI import MainUI

from CNN import CNN

from KNN import KNN

import os

modeltype = "KNN"

if modeltype == "KNN":

    if os.path.exists("knn.sav"):

        pass

    else:

        print("Saved KNN Classifier not found....")

        print("Downloading MNIST Data, training KNN classifier and saving
as knn.sav......")
```

```
        print("Kindly wait for a few minutes...........")

        knnobj = KNN(3)

        knnobj.skl_knn()

else:

    if os.path.exists("cnn.hdf5"):

        pass

    else:

        print("cnn.hdf5 not found...")

        print("Loading MNIST Data, training CNN and saving as
cnn.hdf5.....")

        print("Kindly wait a few minutes.........")

        cnnobj = CNN()

        cnnobj.build_and_compile_model()

        cnnobj.train_and_evaluate_model()

        cnnobj.save_model()

MainUIobj = MainUI(modeltype)

MainUIobj.mainloop()

MainUIobj.cleanup()
```

Screenshots :

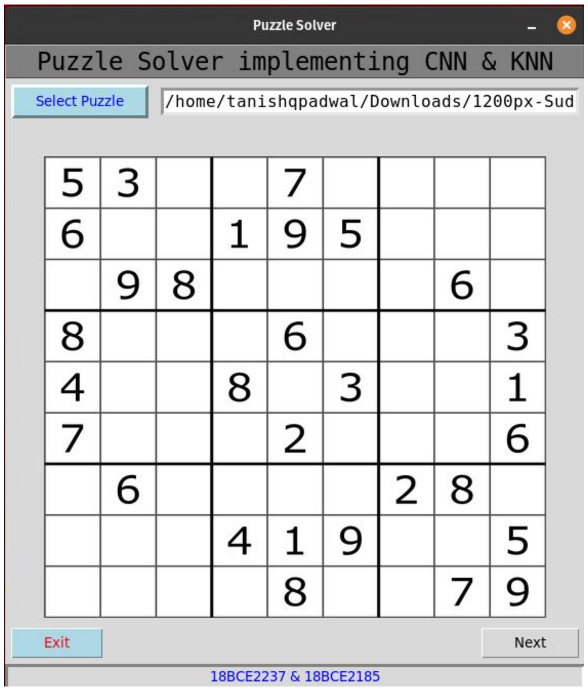

Run.py - AI_Sudoku-master - Visual Studio Code
MainUI.py
Run.py
Puzzle Solver
Puzzle Solver implementing CNN & KNN
Select Puzzle
Exit
Next
18BCE2237 & 18BCE2185

```
# Change the model type variable value to "CNN" to use the Convolu
# Change the model type variable value to "KNN" to use the K Neares
modeltype = "KNN"
# Checking if the KNN or CNN file is present
# If not present, then generate the required file
if modeltype == "KNN":
    if os.path.exists("knn.sav"):
        pass
    else:
        print("Saved KNN Classifier not found....")
        print("Downloading MNIST Data, training KNN classifier and
        print("Kindly wait for a few minutes...........")
        knnobj = KNN(3)
        knnobj.skl_knn()
else:
    if os.path.exists("cnn.hdf5"):
        pass
    else:
        print("cnn.hdf5 not found...")
        print("Loading MNIST Data, training CNN and saving as cnn.h
        print("Kindly wait a few minutes.........")
        cnnobj = CNN()
```

```
ownloads/AI_Sudoku-master/MainUI.py
tanishqpadwal@pop-os:~/Downloads/AI_Sudoku-master$ /bin/python3 /home/tanishqpadwal/D
ownloads/AI_Sudoku-master/Run.py
/bin/python3 /home/tanishqpadwal/Downloads/AI_Sudoku-master/MainUI.py
tanishqpadwal@pop-os:~/Downloads/AI_Sudoku-master$ /bin/python3 /home/tanishqpadwal/D
ownloads/AI_Sudoku-master/MainUI.py
tanishqpadwal@pop-os:~/Downloads/AI_Sudoku-master$ /bin/python3 /home/tanishqpadwal/D
ownloads/AI_Sudoku-master/MainUI.py
tanishqpadwal@pop-os:~/Downloads/AI_Sudoku-master$ /bin/python3 /home/tanishqpadwal/D
ownloads/AI_Sudoku-master/MainUI.py
tanishqpadwal@pop-os:~/Downloads/AI_Sudoku-master$ /bin/python3 /home/tanishqpadwal/D
ownloads/AI_Sudoku-master/Run.py
```

```
tanishqpadwal@pop-os:~$
```

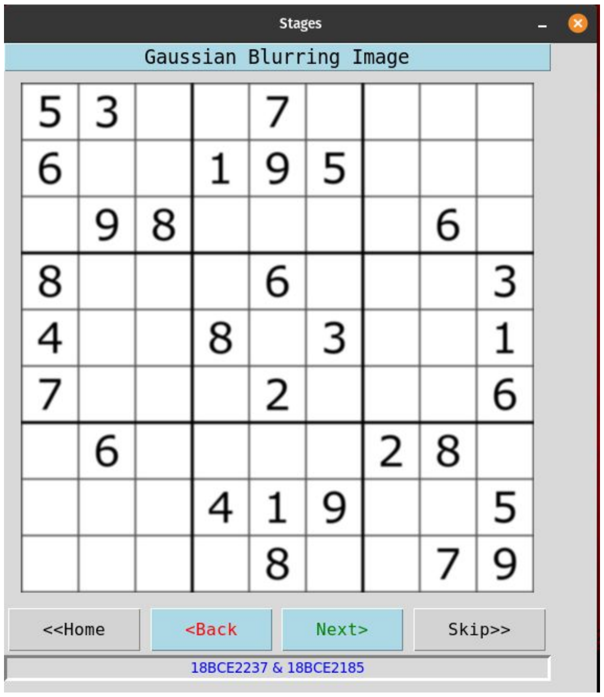
Step 0 :



/home/tanishqpadwal/Downloads/1200px-Sud

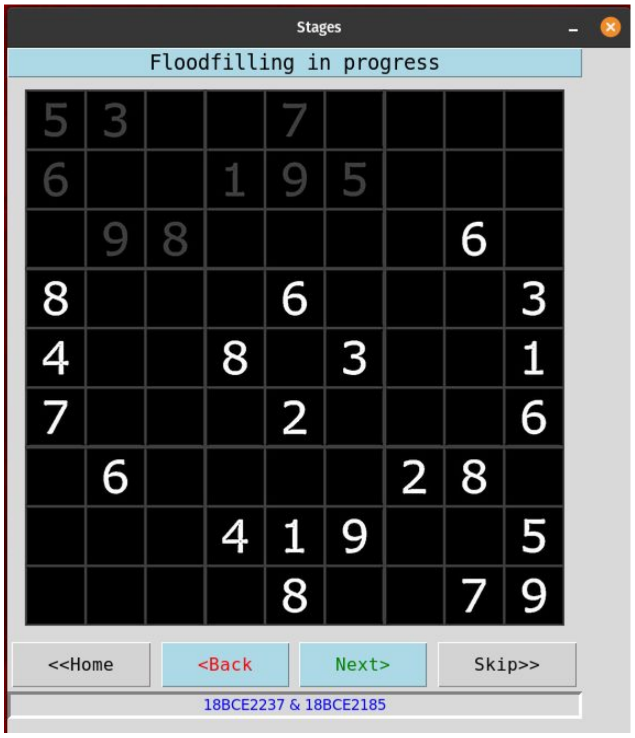
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Step 1 : Gaussian Blurring



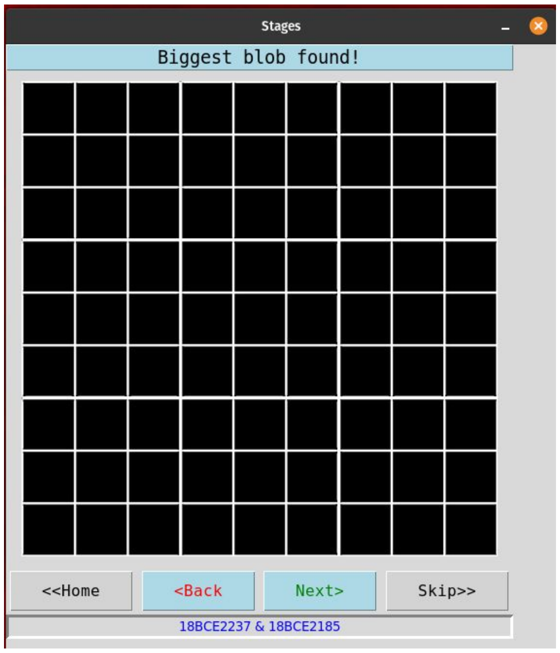
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Step 2 : Floodfilling

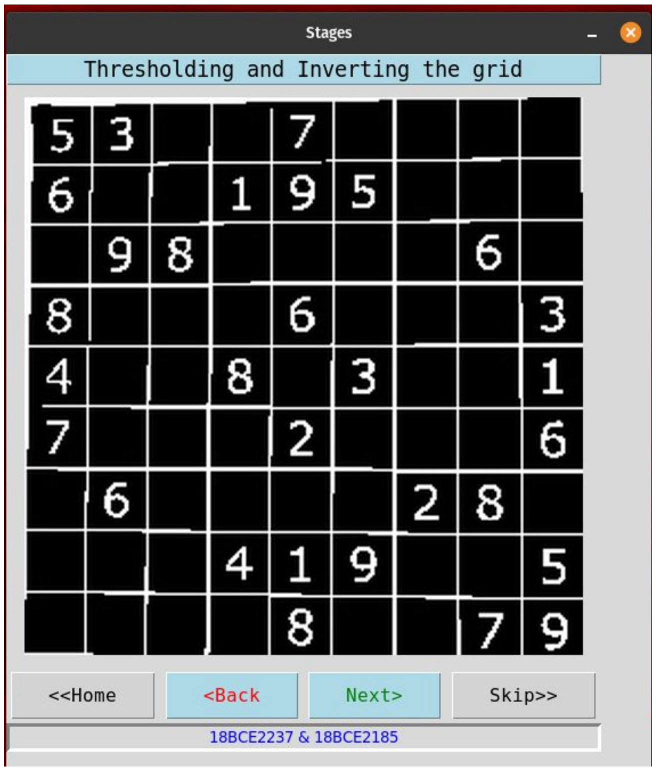


5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

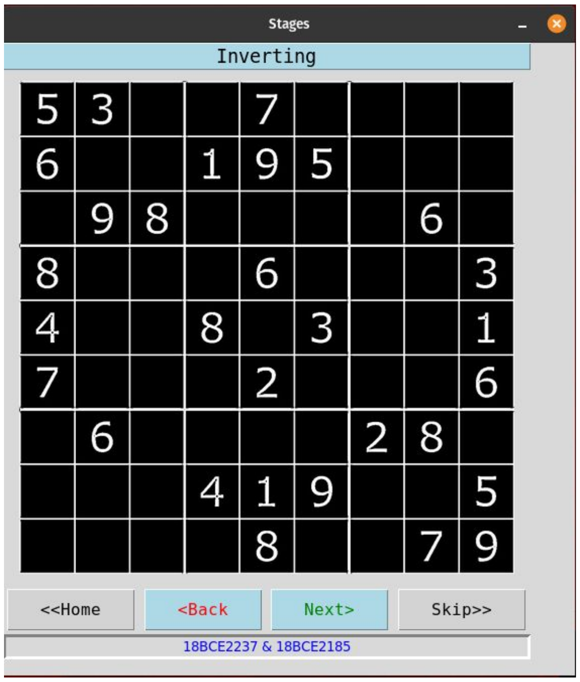
Step 3 : Finding biggest blob



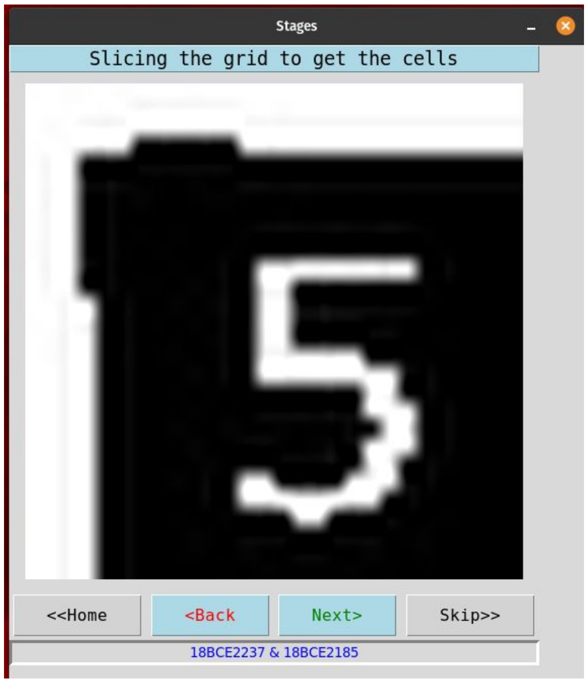
Step 4 : Thresholding and Inverting



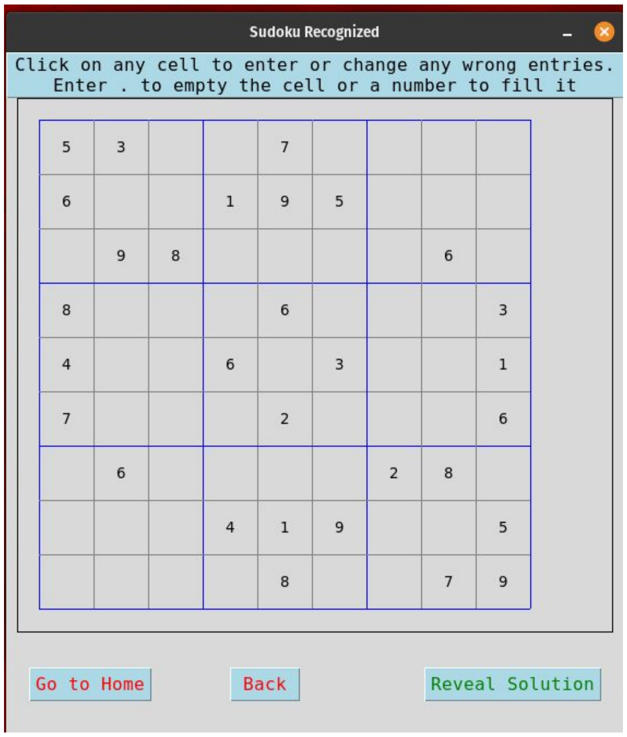
Step 5 : Inverting



Step 6 : Slicing



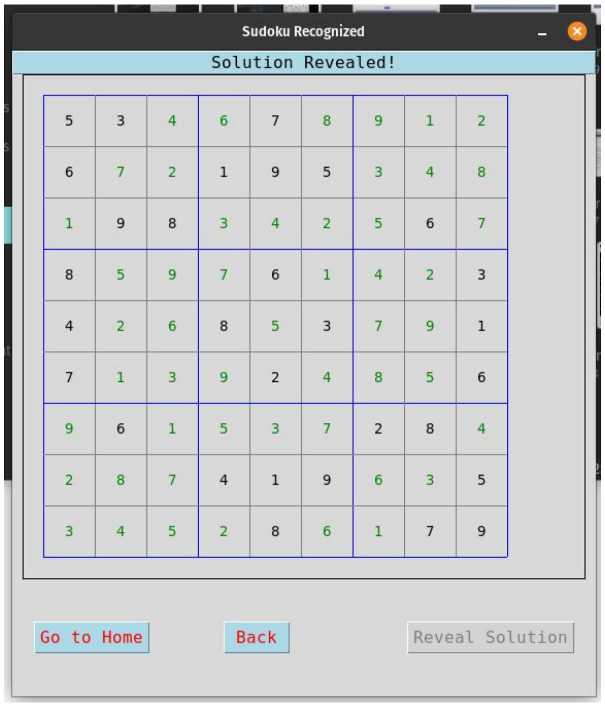
Step 7 : Check if board is correct



Click on any cell to enter or change any wrong entries.
Enter . to empty the cell or a number to fill it

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

Step 8 : Final Result



Solution Revealed!

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