

**DEPARTEMENT OF COMPUTER SCIENCE**

**Course code: INT 404**

**AI PROJECT**

**Topic: Sudoku Problem Solver**

**SUBMITTED TO : MS. POOJA RANA**

**SUBMITTED BY : ADITYA TIWARI**

**SECTION : K18AP**

**REG NO. : 11807407**

**ROLL NO. : 68**

**CONTENT**

**1. ABSTRACT**

**2. INTRODUCTION**

**3. LITERATURE REVIEW**

**4. PROPOSED METHODOLOGY**

**5. CODE**

**6.RESULT AND DISCUSSION**

**7. CONCLUSION**

**8.BIBLIOGRAPHY**

**ABSTRACT**

The handwritten digit recognition is the ability of computers to recognize human handwritten digits. It is a hard task for the machine because handwritten digits are not perfect and can be made with many different flavours. The handwritten digit recognition is the solution to this problem which uses the image of a digit and recognizes the digit present in the image.

**INTRODUCTION**

Handwritten digit recognition is gaining a huge demand in the branch of computer vision. We are going to implement a better and accurate approach to perceive and foresee manually written digits from 0 to 9. A class of multilayer sustain forward system called Convolutional network is taken into consideration. A Convolutional network has a benefit over other Artificial Neural networks in extracting and utilizing the features data, enhancing the knowledge of 2D shapes with higher degree of accuracy and unvarying to translation, scaling and other distortions. The LeNet engineering was initially presented by LeCun et al in their paper. The creator’s execution of LeNet was primarily focused on digit and character recognition.LeNet engineering is clear and simple making it easy for implementation of CNN’s. We are going to take the MNIST dataset for training and recognition. The primary aim of this dataset is to classify the handwritten digits 0-9 . We have a total of 70,000 images for training and testing. Each digit is represented as a 28 by 28 grey scale pixel intensities for better results. The digits are passed into input layers of LeNet and then into the hidden layers which contain two sets of convolutional, activation and pooling layers. Then finally it is mapped onto the fully connected layer and given a SoftMax classifier to classify the digits. We are going to implement this network using keras deep learning inbuilt python library.

**LITERATURE REVIEW**

In order to implement a handwritten digit recognition app using the MNIST dataset, we will be using a special type of deep neural network also In the end, we are going to build a GUI in which you can draw the digit and recognize it straight away.

### Prerequisites

### The interesting Python project requires you to have basic knowledge of Python programming, deep learning with Keras library and the Tkinter library for building GUI.

### The MNIST dataset

This is probably one of the most popular datasets among machine learning and deep learning enthusiasts. The [MNIST dataset](http://yann.lecun.com/exdb/mnist/) contains 60,000 training images of handwritten digits from zero to nine and 10,000 images for testing. So, the MNIST dataset has 10 different classes. The handwritten digits images are represented as a 28×28 matrix where each cell contains grayscale pixel value.

Below are the steps to implement the handwritten digit recognition project:

### 1. Import the libraries and load the dataset

First, we are going to import all the modules that we are going to need for training our model. The Keras library already contains some datasets and MNIST is one of them. So, we can easily import the dataset and start working with it. The **mnist.load\_data()** method returns us the training data, its labels and the testing data and its labels.

1. import keras
2. from keras.datasets import mnist
3. from keras.models import Sequential
4. from keras.layers import Dense, Dropout, Flatten
5. from keras.layers import Conv2D, MaxPooling2D
6. from keras import backend as K
7. # the data, split between train and test sets
8. (x\_train, y\_train), (x\_test, y\_test) = mnist.**load\_data**()
9. **print**(x\_train.shape, y\_train.shape)

### 2. Pre-process the data

The image data cannot be fed directly into the model, so we need to**perform some operations and process the data** to make it ready for our neural network. The dimension of the training data is (60000,28,28). The CNN model will require one more dimension so we reshape the matrix to shape (60000,28,28,1).

1. x\_train = x\_train.**reshape**(x\_train.shape[0], 28, 28, 1)
2. x\_test = x\_test.**reshape**(x\_test.shape[0], 28, 28, 1)
3. input\_shape = (28, 28, 1)
4. # convert class vectors to binary class matrices
5. y\_train = keras.utils.**to\_categorical**(y\_train, num\_classes)
6. y\_test = keras.utils.**to\_categorical**(y\_test, num\_classes)
7. x\_train = x\_train.**astype**('float32')
8. x\_test = x\_test.**astype**('float32')
9. x\_train /= 255
10. x\_test /= 255
11. **print**('x\_train shape:', x\_train.shape)
12. **print**(x\_train.shape[0], 'train samples')
13. **print**(x\_test.shape[0], 'test samples')

### 3. Create the model

Now we will **create our CNN model** in Python data science project. A CNN model generally consists of convolutional and pooling layers. It works better for data that are represented as grid structures; this is the reason why CNN works well for image classification problems. The dropout layer is used to deactivate some of the neurons and while training, it reduces offer fitting of the model. We will then compile the model with the Ad delta optimizer.

1. batch\_size = 128
2. num\_classes = 10
3. epochs = 10
4. model = **Sequential**()
5. model.**add**(**Conv2D**(32, kernel\_size=(3, 3),activation='relu',input\_shape=input\_shape))
6. model.**add**(**Conv2D**(64, (3, 3), activation='relu'))
7. model.**add**(**MaxPooling2D**(pool\_size=(2, 2)))
8. model.**add**(**Dropout**(0.25))
9. model.**add**(**Flatten**())
10. model.**add**(**Dense**(256, activation='relu'))
11. model.**add**(**Dropout**(0.5))
12. model.**add**(**Dense**(num\_classes, activation='softmax'))
13. model.**compile**(loss=keras.losses.categorical\_crossentropy,optimizer=keras.optimizers.**Adadelta**(),metrics=['accuracy'])

### 4. Train the model

The**model.fit() function** of Keras will start the training of the model. It **takes the training data, validation data, epochs, and batch size.**

It takes some time to train the model. After training, we save the weights and model definition in the ‘mnist.h5’ file.

1. hist = model.**fit**(x\_train, y\_train,batch\_size=batch\_size,epochs=epochs,verbose=1,validation\_data=(x\_test, y\_test))
2. **print**("The model has successfully trained")
3. model.**save**('mnist.h5')
4. **print**("Saving the model as mnist.h5")

### 5. Evaluate the model

We have 10,000 images in our dataset which will be used to**evaluate how good our model works**. The testing data was not involved in the training of the data therefore, it is new data for our model. The MNIST dataset is well balanced so we can get around 99% accuracy.

1. score = model.**evaluate**(x\_test, y\_test, verbose=0)
2. **print**('Test loss:', score[0])
3. **print**('Test accuracy:', score[1])

### 6. Create GUI to predict digits

Now for the GUI, we have created a new file in which we **build an interactive window to draw digits on canvas** and with a button, we can recognize the digit. The Tkinter library comes in the Python standard library. We have created a function **predict\_digit()** that takes the image as input and then uses the trained model to predict the digit.

Then we **create the App class** which is responsible for building the GUI for our app. We create a canvas where we can draw by capturing the mouse event and with a button, we trigger the predict\_digit() function and display the results.

Here’s the full code for our gui\_digit\_recognizer.py file:

1. from keras.models import load\_model
2. from tkinter import \*
3. import tkinter as tk
4. import win32gui
5. from PIL import ImageGrab, Image
6. import numpy as np
7. model = **load\_model**('mnist.h5')
8. def **predict\_digit**(img):
9. #resize image to 28x28 pixels
10. img = img.**resize**((28,28))
11. #convert rgb to grayscale
12. img = img.**convert**('L')
13. img = np.**array**(img)
14. #reshaping to support our model input and normalizing
15. img = img.**reshape**(1,28,28,1)
16. img = img/255.0
17. #predicting the class
18. res = model.**predict**([img])[0]
19. return np.**argmax**(res), **max**(res)
20. class **App**(tk.Tk):
21. def **\_\_init\_\_**(self):
22. tk.Tk.**\_\_init\_\_**(self)
23. self.x = self.y = 0
24. # Creating elements
25. self.canvas = tk.**Canvas**(self, width=300, height=300, bg = "white", cursor="cross")
26. self.label = tk.**Label**(self, text="Thinking..", font=("Helvetica", 48))
27. self.classify\_btn = tk.**Button**(self, text = "Recognise", command = self.classify\_handwriting)
28. self.button\_clear = tk.**Button**(self, text = "Clear", command = self.clear\_all)
29. # Grid structure
30. self.canvas.**grid**(row=0, column=0, pady=2, sticky=W, )
31. self.label.**grid**(row=0, column=1,pady=2, padx=2)
32. self.classify\_btn.**grid**(row=1, column=1, pady=2, padx=2)
33. self.button\_clear.**grid**(row=1, column=0, pady=2)
34. #self.canvas.bind("<Motion>", self.start\_pos)
35. self.canvas.**bind**("<B1-Motion>", self.draw\_lines)
36. def **clear\_all**(self):
37. self.canvas.**delete**("all")
38. def **classify\_handwriting**(self):
39. HWND = self.canvas.**winfo\_id**() # get the handle of the canvas
40. rect = win32gui.**GetWindowRect**(HWND) # get the coordinate of the canvas
41. im = ImageGrab.**grab**(rect)
42. digit, acc = **predict\_digit**(im)
43. self.label.**configure**(text= **str**(digit)+', '+ **str**(**int**(acc\*100))+'%')
44. def **draw\_lines**(self, event):
45. self.x = event.x
46. self.y = event.y
47. r=8
48. self.canvas.**create\_oval**(self.x-r, self.y-r, self.x + r, self.y + r, fill='black')
49. app = **App**()
50. **mainloop**()

**PROPOSED METHODOLOGY**

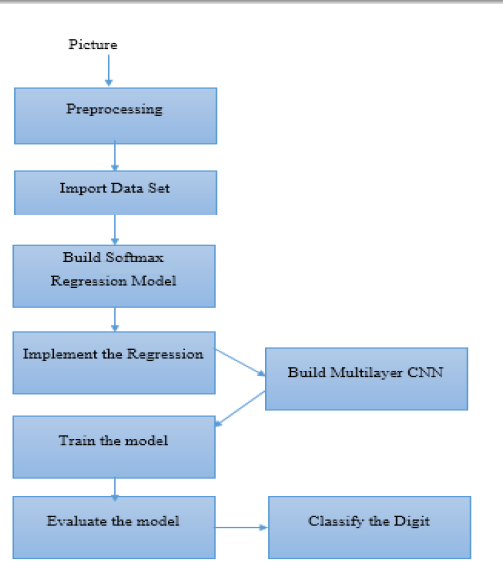
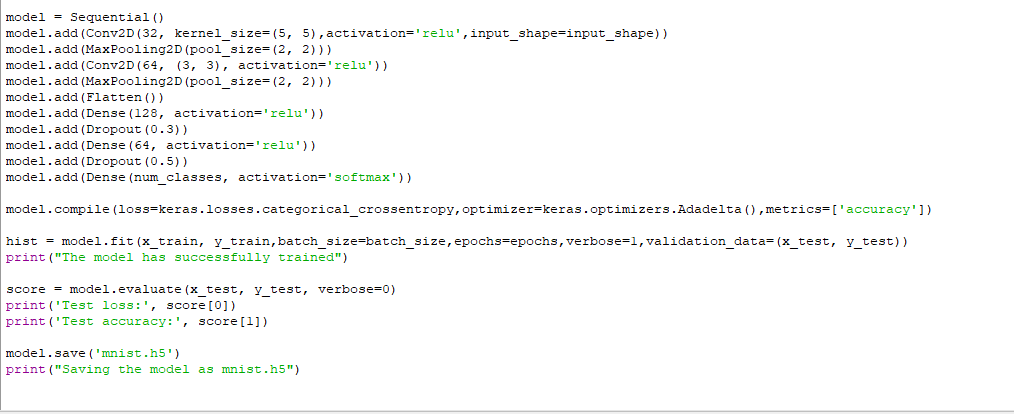
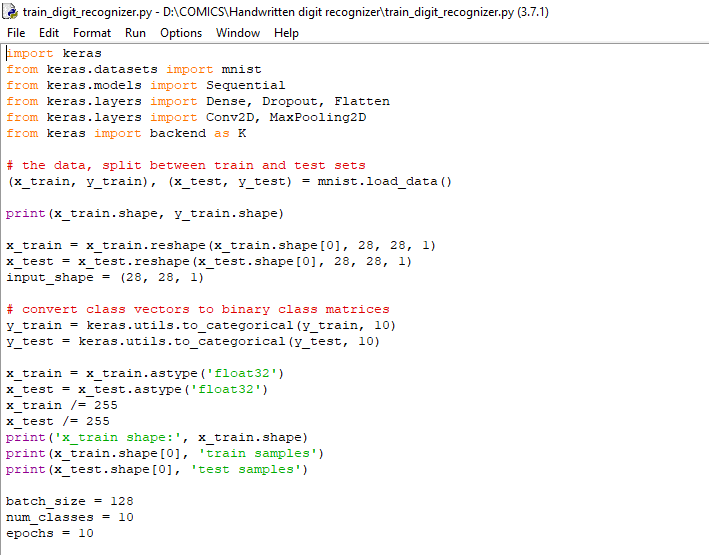
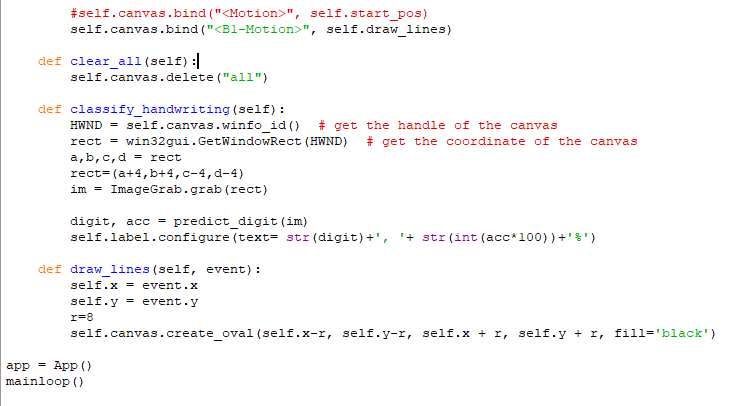
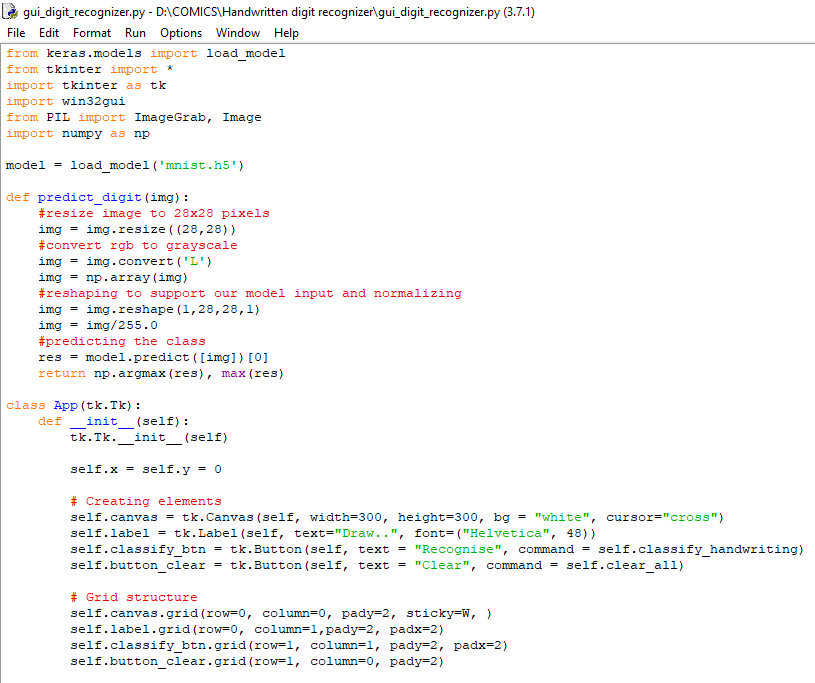


Fig. 1 Production system and control strategy

CODE

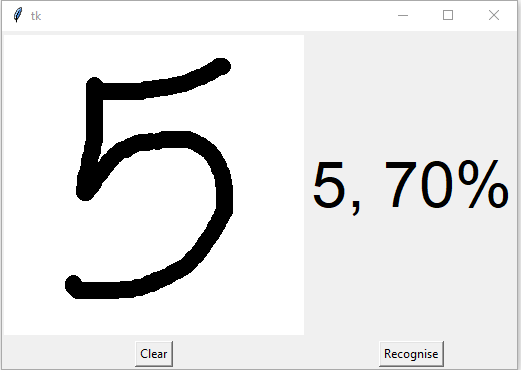
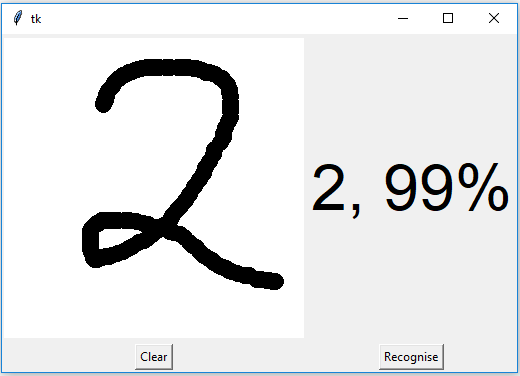
To train digit recognizer

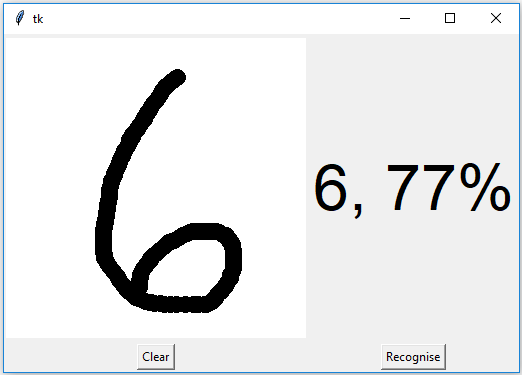


For the GUI : 

**RESULT AND DISCUSSION**

Here, we taken some random numbers like 2, 5, 6 etc to test and we get the recognized number with the accuracy to which extent the numbers match with those patterns matching with the data present in the data set.





**CONCLUSION:**

## **Summary**

In this project, we have successfully built a Python deep learning project on handwritten digit recognition app. We have built and trained the Convolutional neural network which is very effective for image classification purposes. Later, we build the GUI where we draw a digit on the canvas then we classify the digit and show the results.

**REFERENCES:**

[1. For reading on digit recognizer](http://1. For reading on digit recognizerwww.researchgate.net › publication › 298808334_Handwritten_Text_R..)

### [www.researchgate.net › publication › 298808334\_Handwritten\_Text\_R..](http://1. For reading on digit recognizerwww.researchgate.net › publication › 298808334_Handwritten_Text_R..)

2. **[Handwritten Digit Recognition using Machine Learning ...](https://globaljournals.org/GJCST_Volume18/3-Handwritten-Digit-Recognition.pdf)**

[globaljournals.org › GJCST\_Volume18 › 3-Handwritten-Digit-Recog...](https://globaljournals.org/GJCST_Volume18/3-Handwritten-Digit-Recognition.pdf)