# **Handwritten Digit Recognition using CNN and GUI using Tkinter**

# **Introduction**

Everyday, developers are working hard on machines to make them more smart and intelligent by using machine learning and deep learning techniques so that they can perform tasks similar to humans. With the help of these techniques human effort can be reduced in recognizing, learning, predictions and many other areas.

*The ability of computers to recognize human handwritten digits is known as****handwritten digit recognition****from sources such as paper documents, images, touch-screens etc.*

In this project you will discover how to develop a deep learning model to achieve high performance on the handwritten digit recognition task using the **MNIST** dataset and build a **GUI App** based on **Tkinter** where, you can draw the digits (single as well as multiple) and recognize it straight away by draw a bounding box surrounding each digit.

# **Problem Statement**

It is easy for the human to perform a task accurately by practicing it repeatedly and memorizing it for the next time. Human brain can process and analyse images easily. Also, recognize the different elements present in the images.

*The challenge in handwritten digit recognition is mainly caused by the writing style variations of every single individual. So, it is not easy for the machine to recognize the handwritten digits accurately like the humans do. Hence, robust feature extraction is very important to improve the performance of machines.*

# **MNIST Dataset**

The MNIST dataset (Modified National Institute of Standards and Technology) is a large dataset of handwritten digits that is widely used for training and testing in the field of machine learning and deep learning.

*The MNIST dataset contains 60,000 training images and 10,000 testing images of handwritten digits from zero to nine(0 to 9). So, the MNIST dataset has 10 different classes. Each image is represented as a 28×28 matrix where each cell contains grayscale pixel value.*

# **Why deep learning?**

Deep learning is a class of machine learning that uses multiple layers to progressively extract higher level features from the input. Therefore, deep learning reduces the task of developing new feature extractor for every problem. This characteristic of Deep Learning is a major step ahead of traditional Machine Learning.

*Most modern deep learning models are based on artificial neural networks, specifically, Convolutional Neural Networks (CNN) that we will use in this project.*

# **Steps to implement the CNN handwritten digit recognition GUI App:**

1. Import the libraries and load the MNIST dataset
2. Data Preprocess and Normalize
3. Create the model
4. Train the model
5. Evaluate the model
6. Create GUI to predict digits

*The project requires you to have basic knowledge of Python programming, OpenCV, Deep learning with Keras library and the Tkinter library for building GUI.*

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

First, we are going to import libraries that we need to train our model.

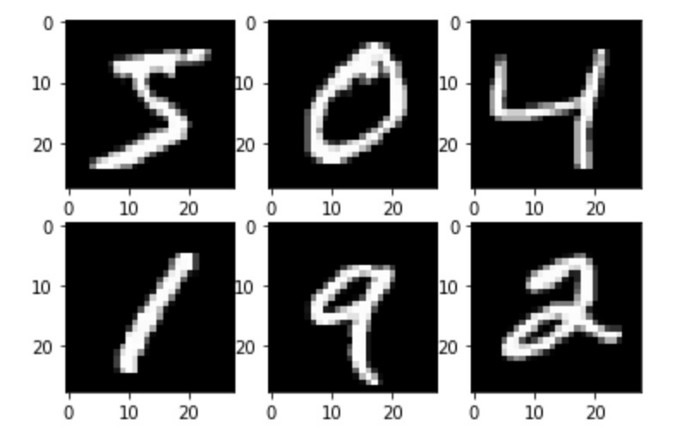


To load a dataset, Keras deep learning library provides a convenience method for loading the MNIST dataset. So we can easily import the dataset by calling the **mnist.load\_data**() function.



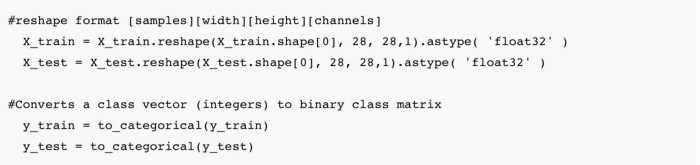
It’s better to visualise the dataset before using it. Here we plot six samples of MNIST training dataset for visualisation.



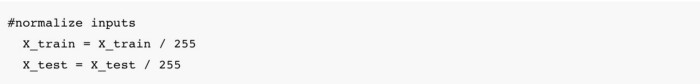


# **2. Data Preprocess and Normalize**

We know that all images (dataset) are represented as a 28×28 matrix containing grayscale pixel values. According to this, the dimension of the training data is (60000,28,28) but the CNN model will require one more dimension so we need to process the data by reshape the matrix to shape (60000,28,28,1).



when using neural network models, It is good to perform some scaling of input values to normalize the pixel values to the range 0 and 1 by dividing each value by the maximum value (Note: The pixel values are gray scale between 0 and 255).



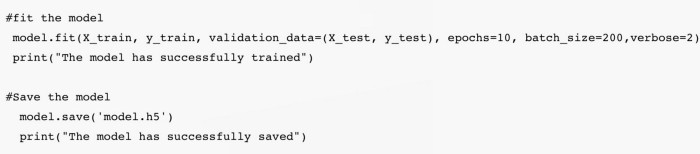
# **3. Create the model**

Now we will create a CNN model with a double convolutional layer of the same size 3×3, max pooling layers and fully connected layers. The dropout layer is used to deactivate some of the neurons to reduce overfitting. Finally, the output layer has 10 neurons for the 10 classes. We will then compile the model with the ADAM / ADADELTA optimizer.



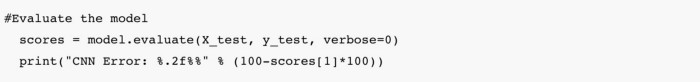
# **4. Train the model**

The **model.fit**() function of Keras takes training data, validation data, epochs, and batch size to train the model. It takes some time to train the model depending upon your working environment (CPU or GPU). After completion, we will save the model in the file named as **‘model.h5’** for later use.



# **5. Evaluate the model**

To evaluate the performance of our model, we have 10,000 testing images of handwritten digits. The MNIST dataset is well balanced so we can get around 99% accuracy (i.e. CNN error < 1%).



# **6. Create GUI to predict digits**

Finally it’s time to build GUI App using Tkinter. We will create a new file to build a GUI.

*The tkinter package is the standard Python interface to the Tk GUI toolkit. It is the fastest and easiest way to create the GUI applications. It provides a variety of common GUI elements – such as buttons, menus and various kinds of entry fields and display areas.*

There are two main methods:

* Tk()
* mainloop()

**Tk**() is the method which we use to create the main window of an application and **mainloop**() is a method on the main window which we execute when we want to run our application.

Let’s start with importing modules that we need for the App.



Here, we load the saved model for recognition of the handwritten digit by providing the model path.



In the block of code, we create the root window, which is the main window of our application with the title “**Handwritten Digit Recognition GUI App**”.

There are different widget classes built into tkinter but we will use few of them to create our application:

* **Canvas** is a widget for drawing graphics. It can be used to create custom widgets so we can draw anything we like inside it. In our application we will use the widget to draw the digit (single as well as multiple).
* A **Button** usually maps directly onto a user action. In our application we use two buttons named as “**Recognize Digit**” and “**Clear Widget**”. when the user clicks on a button, it triggers the functions as assigned.

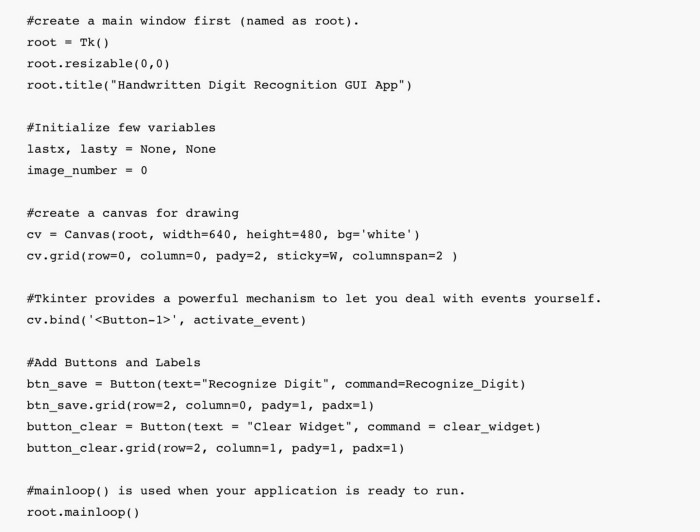
*Tkinter provides a powerful mechanism to let you deal with events and bind handlers to them, using the bind method which we can find on every widget class. Events are uniquely identified by a sequence name in string format.*

In our application we will use “<Button-1>” and “<B1-Motion>” events.

*“<Button-1>” event signals that a left mouse button has been pressed while the mouse cursor is positioned over the widget.*

*“<B1-Motion>” indicates that the mouse was moved while the left button was pressed.*

Now we execute the **mainloop**() method on the main window to run our application. This method will loop forever, waiting for events from the user, until the user exits the program.



In this block of code, we are using three functions:

* clear\_widget()
* activate\_event()
* draw\_lines()

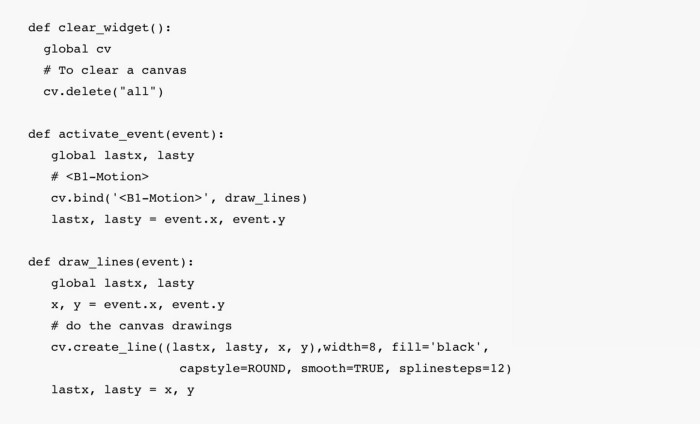
**clear\_widget**() function is used to clear the canvas. Note that items added to the canvas are kept until you remove them. In our application we will use the **delete**() method to clear the previous drawn digits on canvas so we can draw new one.

Events can come from various sources, including key presses and mouse operations by the user. For each widget, you can bind Python functions and methods to events.

In our application, we use the bind method of the Canvas widget to bind a **activate\_event**() function to an event called <Button-1> and inside this callback function we bind another function, which is **draw\_lines**() to an event called <B1-Motion>.

*Note: The mouse is moved, with mouse button 1 being held down and the current position of the mouse pointer is provided in the x and y members of the event object passed to the callback.*

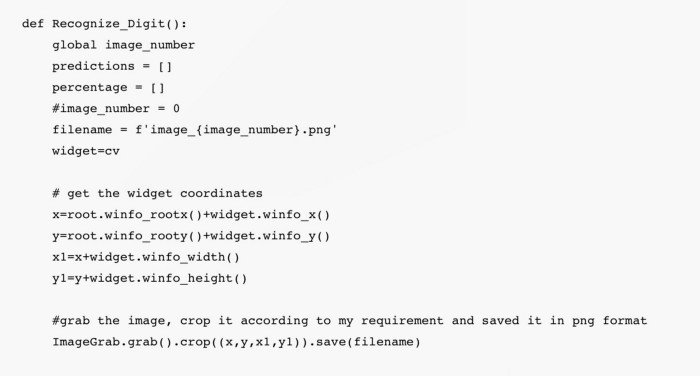
**draw\_lines**() function is used to draw a line on the canvas.



The next block contains single function called **Recognize\_Digit**(). The function is quite big so I will divide it in parts for better understanding.

In this section, we will use **ImageGrab** module to copy the contents of the screen or the clipboard to a **PIL** (Python Imaging Library) image memory. Basically, it takes a snapshot of the screen.

After taking the screen snapshot we will use a **crop** method which takes four coordinates as input and returns a rectangular region from the image (which is a snapshot in the case) and then we will **save** the image under the given filename in png format.

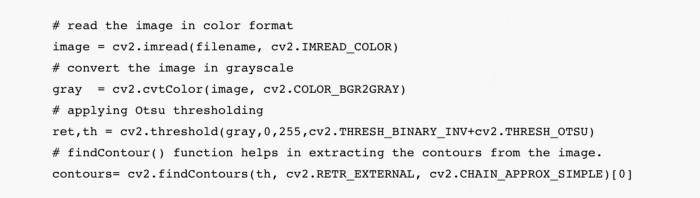


In our application, we will use **OpenCV** (Open source computer vision).

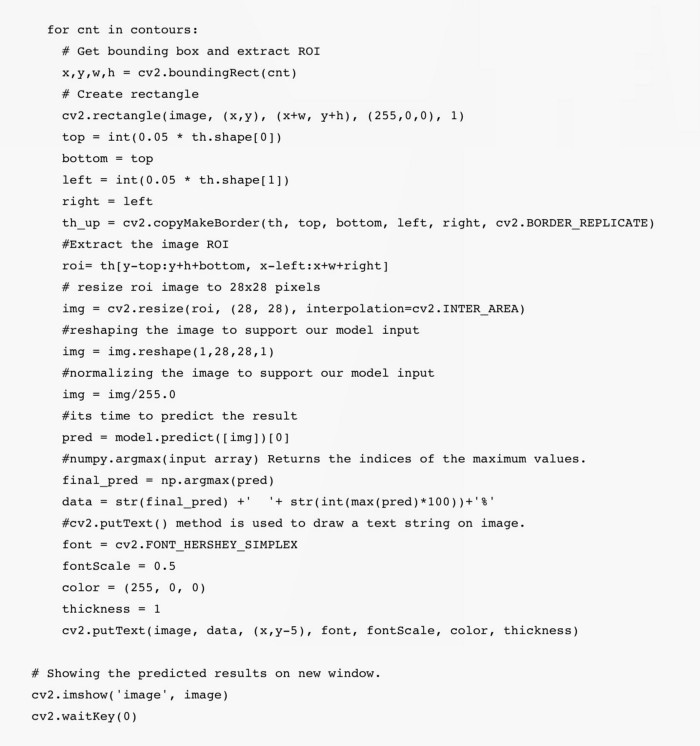
*OpenCV-Python is a library of Python bindings designed to solve computer vision problems.*

In this section, we will use OpenCV to find **contours** of an image that we saved earlier. Contours can be explained simply as a curve joining all the continuous points (along the boundary), having the same color or intensity. It is a useful tool for object detection and recognition.

It is better to use binary images for better accuracy so before finding contours, apply a **threshold**.



This is the last and final section of the function **Recognize\_Digit**(). In this section we are creating **bounding boxes** for contours and extract **ROI**. After extracting Region of interest, we will preprocess (resize, reshape and normalize) the image to support our model input. Now, it’s time to execute the **model.predict**() method to recognize the handwritten digit and draw the bounding box surrounding each digit present in the image with predicted value and percentage.



# **Summary**

In this article, we have successfully built a Handwritten Digit Recognition GUI App in Python based on CNN deep learning.

We have trained the Convolutional neural network which is very effective (CNN error < 1%). At last, we built an application where we can draw a digit (single as well as multiple) on the canvas and display the predicted result (value and percentage) on the top of the bounding box that surrounds each digit present in the image.

**Sample Code:**

**CNN Model Building Code:**

**from tensorflow.keras import layers**

**from tensorflow.keras import models**

**from keras.datasets import mnist**

**from keras.utils import to\_categorical**

**(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()**

**model = models.Sequential()**

**model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)))**

**model.add(layers.MaxPooling2D((2, 2)))**

**model.add(layers.Conv2D(64, (3, 3), activation='relu'))**

**model.add(layers.MaxPooling2D((2, 2)))**

**model.add(layers.Conv2D(64, (3, 3), activation='relu'))**

**model.add(layers.Flatten())**

**model.add(layers.Dense(64, activation='relu'))**

**model.add(layers.Dense(10, activation='softmax'))**

**model.summary()**

**train\_images = train\_images.reshape((60000, 28, 28, 1))**

**train\_images = train\_images.astype('float32') / 255**

**test\_images = test\_images.reshape((10000, 28, 28, 1))**

**test\_images = test\_images.astype('float32') / 255**

**train\_labels = to\_categorical(train\_labels)**

**test\_labels = to\_categorical(test\_labels)**

**model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])**

**model.fit(train\_images, train\_labels, epochs=5, batch\_size=64)**

**test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)**

**print(test\_acc)**

**model.save('mnist.h5')**

**GUI Code:**

**from keras.models import load\_model**

**from tkinter import \***

**import tkinter as tk**

**import win32gui**

**from PIL import ImageGrab, Image**

**import numpy as np**

**model = load\_model('C:/Talent Battle/6 Weeks Project Challenge/Batch 2/mnist.h5')**

**def predict\_digit(img):**

**#resize image to 28x28 pixels**

**img = img.resize((28,28))**

**#convert rgb to grayscale**

**img = img.convert('L')**

**img = np.array(img)**

**#reshaping to support our model input and normalizing**

**img = img.reshape(1,28,28,1)**

**img = img/255.0**

**#predicting the class**

**res = model.predict([img])[0]**

**return np.argmax(res), max(res)**

**class App(tk.Tk):**

**def \_\_init\_\_(self):**

**tk.Tk.\_\_init\_\_(self)**

**self.x = self.y = 0**

**# Creating elements**

**self.canvas = tk.Canvas(self, width=300, height=300, bg = "white", cursor="cross")**

**self.label = tk.Label(self, text="Draw..", font=("Helvetica", 48))**

**self.classify\_btn = tk.Button(self, text = "Recognise", command = self.classify\_handwriting)**

**self.button\_clear = tk.Button(self, text = "Clear", command = self.clear\_all)**

**# Grid structure**

**self.canvas.grid(row=0, column=0, pady=2, sticky=W, )**

**self.label.grid(row=0, column=1,pady=2, padx=2)**

**self.classify\_btn.grid(row=1, column=1, pady=2, padx=2)**

**self.button\_clear.grid(row=1, column=0, pady=2)**

**#self.canvas.bind("<Motion>", self.start\_pos)**

**self.canvas.bind("<B1-Motion>", self.draw\_lines)**

**def clear\_all(self):**

**self.canvas.delete("all")**

**def classify\_handwriting(self):**

**HWND = self.canvas.winfo\_id() # get the handle of the canvas**

**rect = win32gui.GetWindowRect(HWND) # get the coordinate of the canvas**

**a,b,c,d = rect**

**rect=(a+4,b+4,c-4,d-4)**

**im = ImageGrab.grab(rect)**

**digit, acc = predict\_digit(im)**

**self.label.configure(text= str(digit)+', '+ str(int(acc\*100))+'%')**

**def draw\_lines(self, event):**

**self.x = event.x**

**self.y = event.y**

**r=8**

**self.canvas.create\_oval(self.x-r, self.y-r, self.x + r, self.y + r, fill='black')**

**app = App()**

**mainloop()**