IBM Innovation Center for Education

PHEMESOFT Software That Matters
TRUSTED DELIVERY PARTNER for IBM ICE

SVKM'S NMIMS

Final Project Report

on

HANDWRITTEN DIGIT RECOGNITION

Team members:

Riya Nimje (B229 , 70021118032)
Jahanvi Saraf (B239 , 70021118043)
Shreya Paliwal (A231, 7001118039)

Guided by:
Prof. Dhanajay Joshi

Table of Contents

EXECUTIVE SUMMARY

The capacity of computers to detect human handwritten digits is known as handwritten digit recognition. Because handwritten digits are not flawless and can be generated with a variety of tastes, it is a difficult assignment for the machine. The solution to this problem is handwritten digit recognition, which uses an image of a digit to recognise the digit present in the image.

Our project uses a Neural Network to recognise handwritten digits using the MNIST dataset. It basically recognises handwritten digits from scanned images.

We've taken it a step further with our handwritten digit recognition system, which not only recognises scanned images of handwritten numbers but also allows users to write digits on the screen for detection using an integrated GUI.

Approaches we have used

We will approach this project by using a three-layered Neural Network.

- The input layer: It distributes the features of our examples to the next layer for calculation of activations of the next layer.
- The hidden layer: They are made of hidden units called activations providing nonlinear ties for the network. A number of hidden layers can vary according to our requirements.
- The output layer: The nodes here are called output units. It provides us with the final prediction of the Neural Network on the basis of which final predictions can be made.

On a handwritten digit recognition system, we successfully constructed a Python deep learning project. We created and trained a Convolutional Neural Network (CNN) that is exceptionally good at image classification. Later, we'll create the GUI, which will allow us to draw a digit on the canvas, classify it, and display the results.

1.BACKGROUND

1.1.Aim

The topic of transcribed digit acknowledgment has been an open one in the field of example order for a long time. The primary goal of this project is to provide useful and reliable information. Processes for acknowledging numerical data that has been transcribed examining several existing organization models This is a project that considers the Convolutional Neural Networks display (CCN). The results show that the CNN classifier beat the Neural Network in terms of computing efficiency without sacrificing execution. The Convolutional neural network from Machine Learning can be used to recognise handwritten digits. The core structure of my project development is based on the MNIST (Modified National Institute of Standards and Technologies) database and CNN compilation. So, in order to run the model, we'll need libraries like 'NumPy', 'Pandas,' 'TensorFlow', and 'Keras'. These are the fundamental pillars that my major project is built upon. We're going to use the MNIST dataset to create a handwritten digit recognition software. Finally, we'll create a user interface that allows you to draw a digit and instantly recognise it.

1.2.Technologies

Jupyter Notebook

Python (3.3 or greater/2.7)

1.3.Hardware Architecture

RAM : 2-4 GB

Minimum Operating System required : Windows 7

1.4.Software Architecture

Jupyter Notebook with appropriate Python Version

2.SYSTEM REQUIREMENTS

2.1.Functional requirements

- Mnist Dataset
- Numpy
- Tensorflow
- Keras
- Pillow

2.2.Non-Functional Requirements

- Easy to use
- Easy to understand
- Flexibility
- Easy interface
- User-centric – easy communication yet attractive

2.3.User requirements

- Laptop
- Jupyter Notebook

2.4.Environmental requirements

- Proper Internet Connectivity

3.DESIGN AND ARCHITECTURE

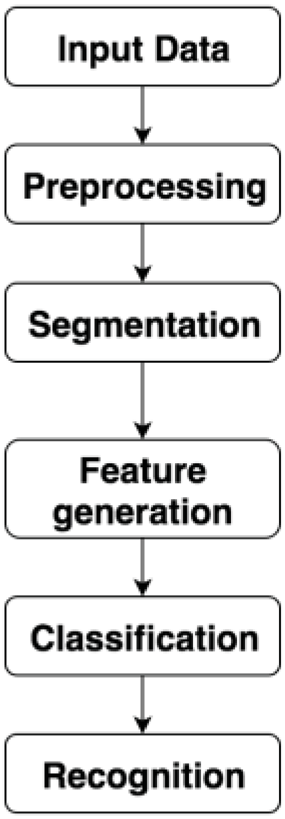


Fig. Steps of the typical handwritten digit recognition system

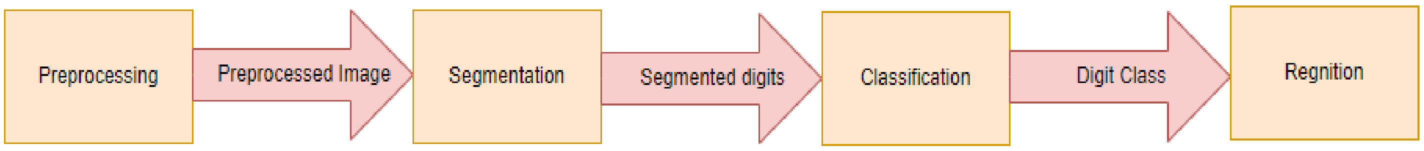


Fig. Modules of Handwritten digit recognition

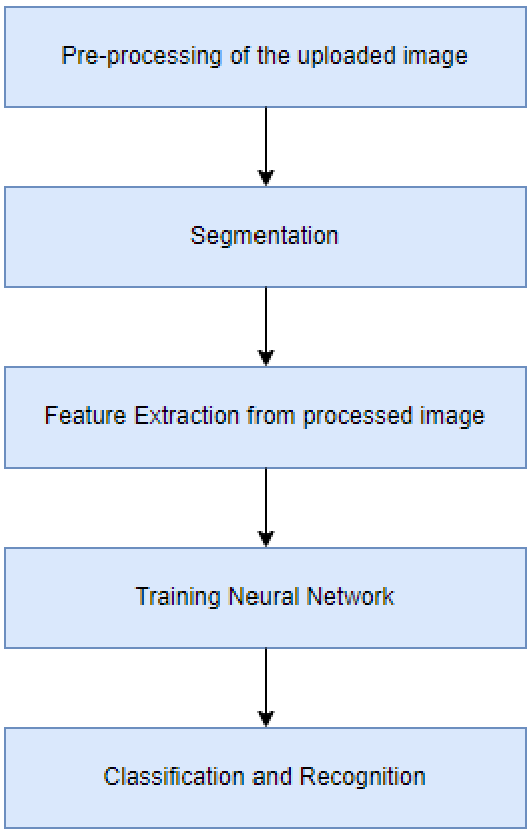

Pre-processing of the uploaded image
Segmentation
Feature Extraction from processed image
Training Neural Network
Classification and Recognition

Fig. Block diagram of proposed model

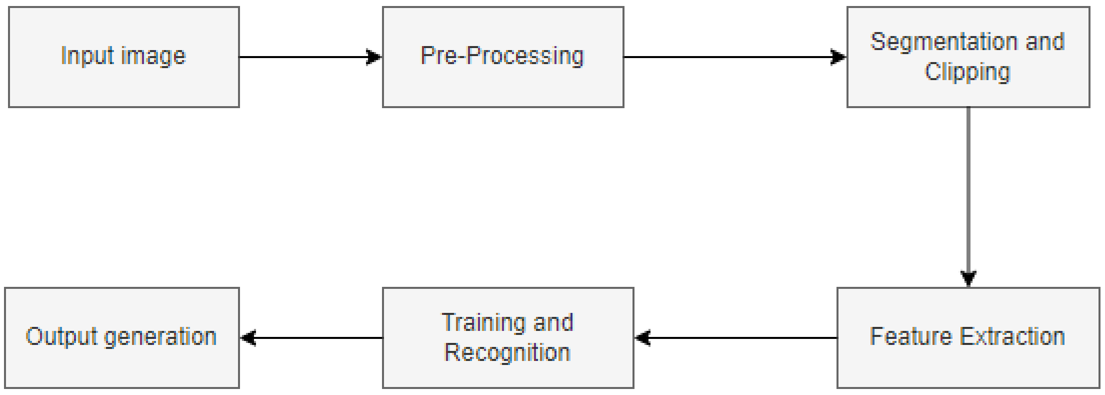


Fig. Architecture of proposed model

4. IMPLEMENTATION

4.1. Import the libraries and load the dataset..

To begin, we will import all of the modules that we will require for training our model. MNIST is one of the datasets already included in the Keras package. As a result, we can quickly import the data and begin working with it. The mnist.load data() method returns both the training and testing data, as well as their labels.

4.2. Preprocess the data

Because the image input cannot be immediately fed into the model, we must do some operations and process the data before it can be fed into our neural network.The training data has a dimension of (60000,28,28). We rearrange the matrix to shape the CNN model, which will require one extra dimension (60000,28,28,1).

4.3. Create the model

Now, in our project, we'll build our CNN model. Convolutional and pooling layers are the most common layers in a CNN model. CNN works effectively for picture classification challenges because it performs better with data that is represented as grid structures. The dropout layer is utilized to disable part of the neurons, which reduces the model's over fit during training. After that, we'll use the Adadelta optimizer to compile the model.

4.4. Train the model

The model. Keras' fit() function will begin the model's training. The training data, validation data, epochs, and batch size are all taken into account.The model must be trained, which takes time. The weights and model definition are saved in the 'mnist.h5' file after training.

4.5. Evaluate the model

Our dataset contains 10,000 photos that will be used to assess how well our model performs. Because the testing data was not used in the training of the data, it represents new information for our model. We got around 99 percent accuracy with the MNIST dataset.

4.6. Create GUI to predict digits

Now for the GUI, we've developed a new file in which we've created an interactive window in which we can draw digits on canvas and recognise them using a button. The

Python standard library includes the Tkinter library. We created the predict digit() function, which takes an image as input and predicts the digit using the trained model.

Then we develop the App class, which is in charge of creating the app's user interface. By capturing the mouse event, we establish a canvas on which we may draw, and by pressing a button, we call the predict digit() function and display the results.

5.TESTING

5.1.Test Plan Objectives

The main objective of handwritten digit recognition system testing is accuracy and a high recognition rate. To accomplish this, the system includes a variety of phases such as digitization, preprocessing, feature extraction, and classification, among others. Each step has its own relevance in the system.

5.2.Data Set

This is probably one of the most popular datasets among machine learning and deep learning enthusiasts. The MNIST dataset 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.

5.3.Testing Plan

The Test Plan is designed to prescribe the scope, approach and resources of the project for Handwritten Digit Recognition . The plan identifies the digits to be tested, the features to be tested, the types of testing to be performed, the personnel responsible for testing, the resources required to complete testing, and the risks associated with the plan.

5.4.Test Strategy

To test this Handwritten Digit Recognition model just draw digits as input in GUI and check the output with their accuracy and test it whether it is user expected or not.

5.5.System Test

To run this Handwritten digit Recognition model just check if the Jupyter Notebook exists or not for the appropriate python version and also check the proper internet connectivity.

5.6.Performance Test

Handwritten Digit recognition model trained using Mnist dataset gives the accuracy of 99% on testing images.

5.7.Basic Test

In this Handwritten Digit Recognition model here first of all check the basic requirements like appropriate python version and proper network connectivity. Further give digits as input in the model to check the output with their accuracy.

5.8.Testing Environment

Minimum requirements required for testing includes:

a) Hardware: minimum RAM of 4-8 GB

b) Operating System: Windows

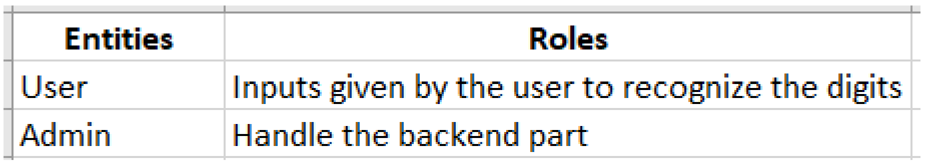
c) Internet connection: internet connection required.

d) Python 3.3 or greater, or Python 2.7

5.9.User Acceptance Test

In this Handwritten Digit Recognition model here we built a GUI where the user gives input as digits and our model gives exact output as the user expected.

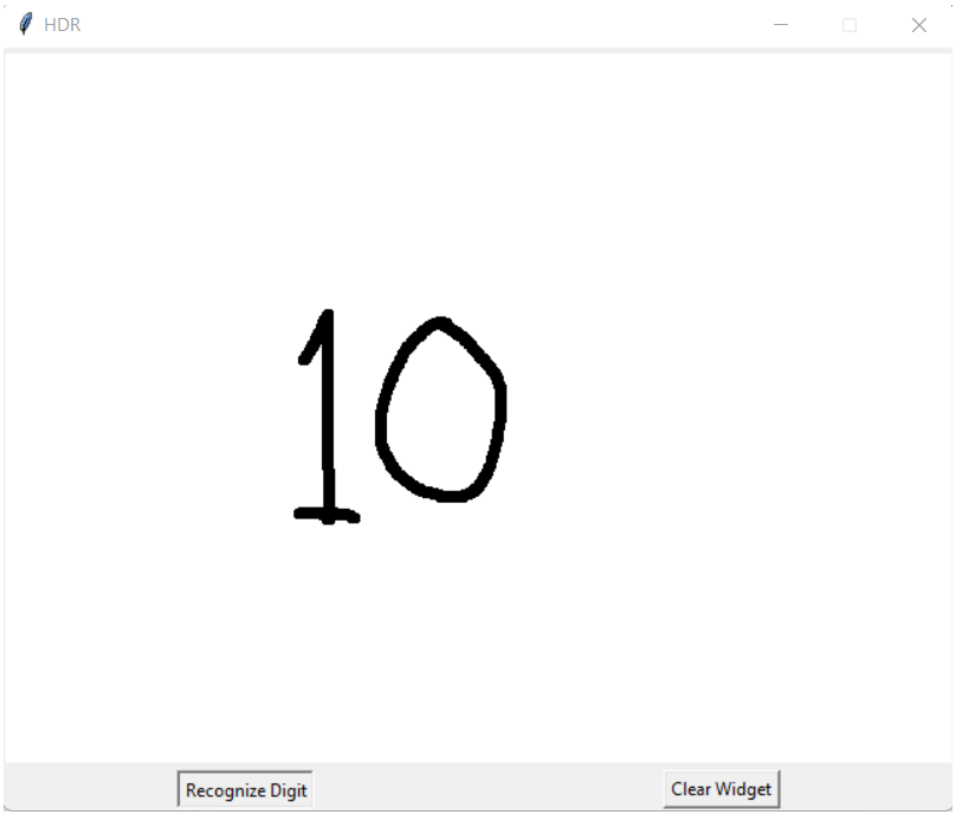
5.10.System



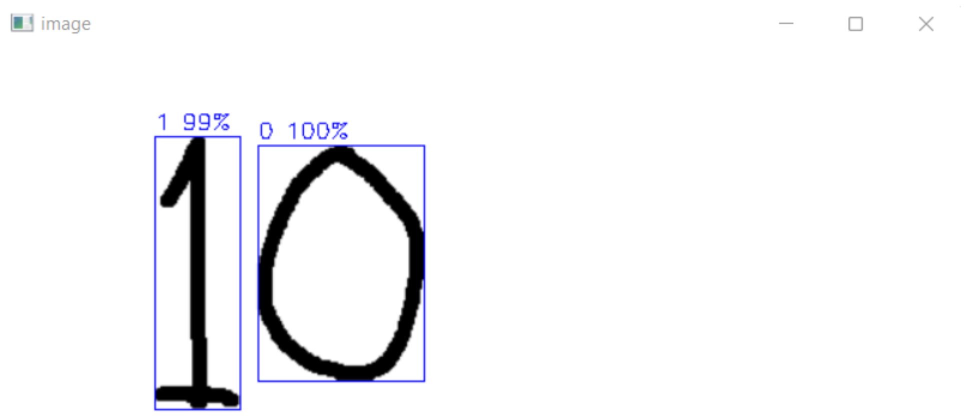
Entities	Roles
User	Inputs given by the user to recognize the digits
Admin	Handle the backend part

6.GRAPHICAL USER INTERFACE (GUI) LAYOUT

Input screen -



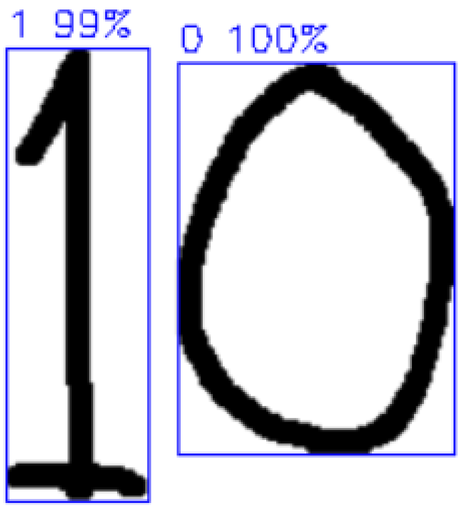
Output Screen -



7. CUSTOMER TESTING

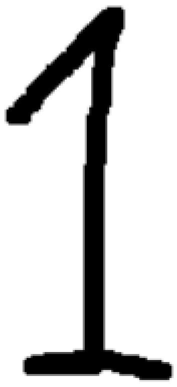
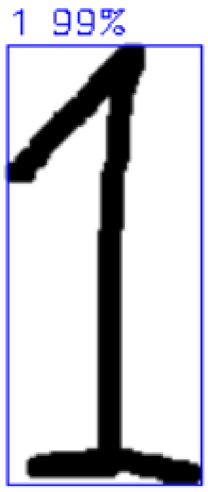
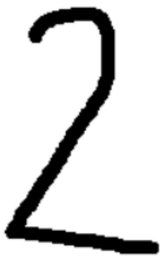
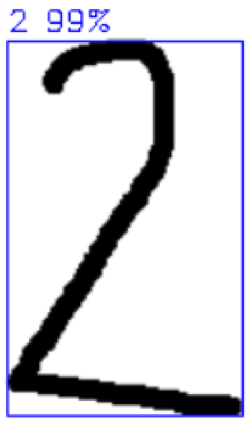
Input by customer -

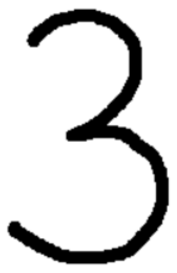
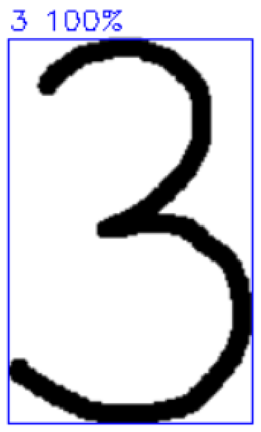
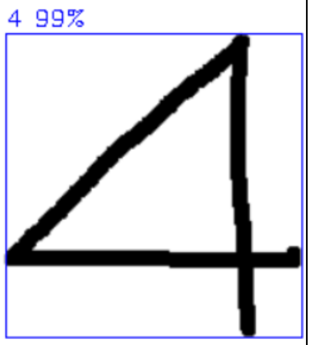
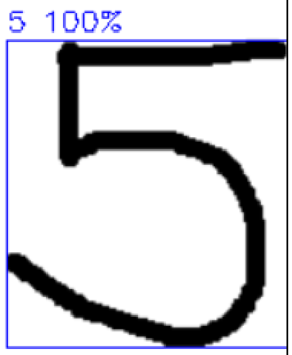
Output by model -

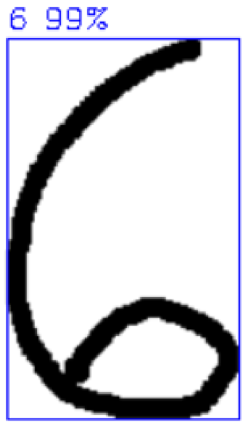
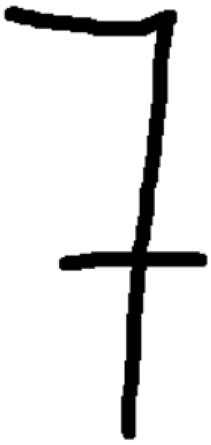
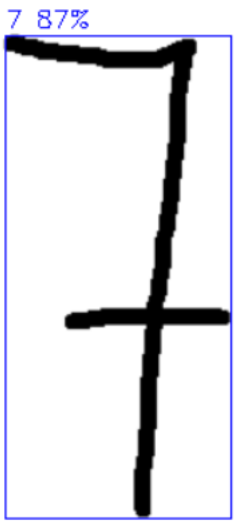
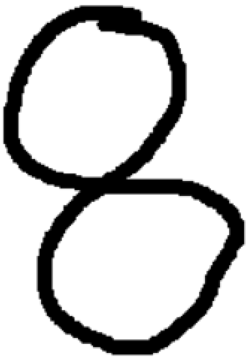
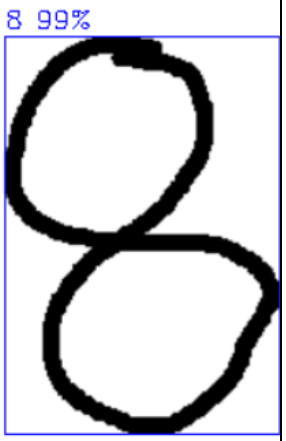


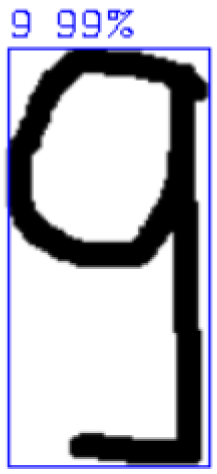
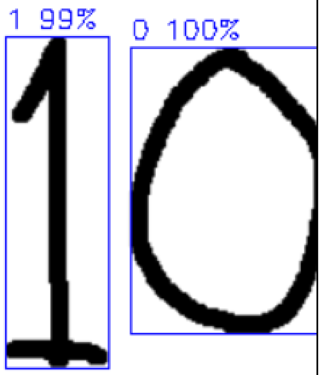
8.EVALUATION

8.1.Table

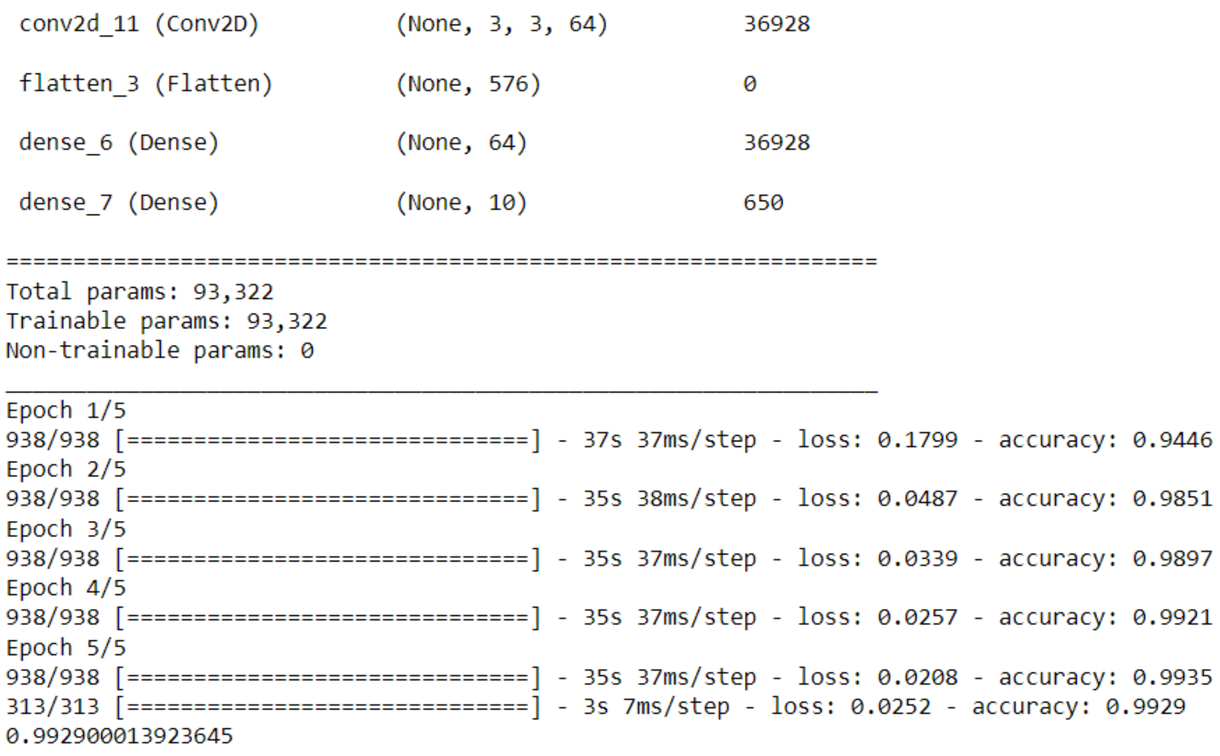
USER INPUT	EXPECTED OUTCOME	MODEL OUTCOME
1 	1	
2 	2	

3 	3	
4	4	
5	5	

6	6	
7 	7	
8 	8	

9	9	
10	10	

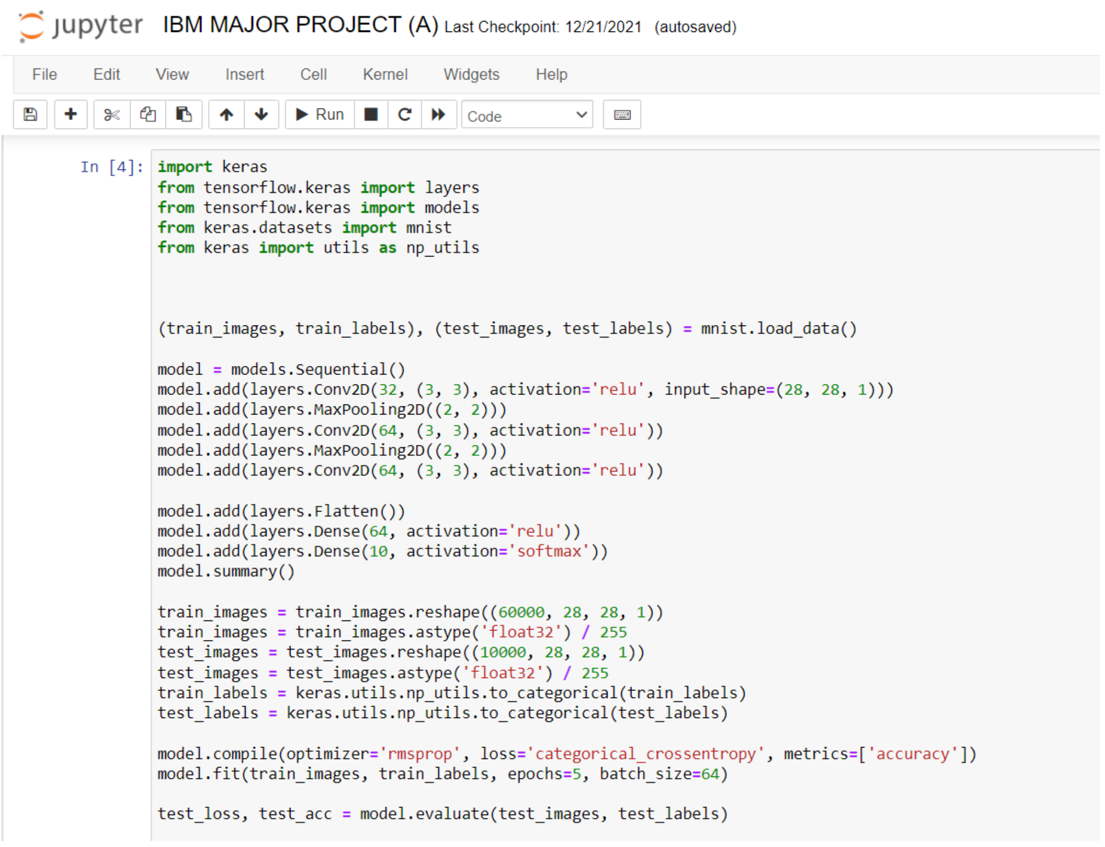
8.2.Performance



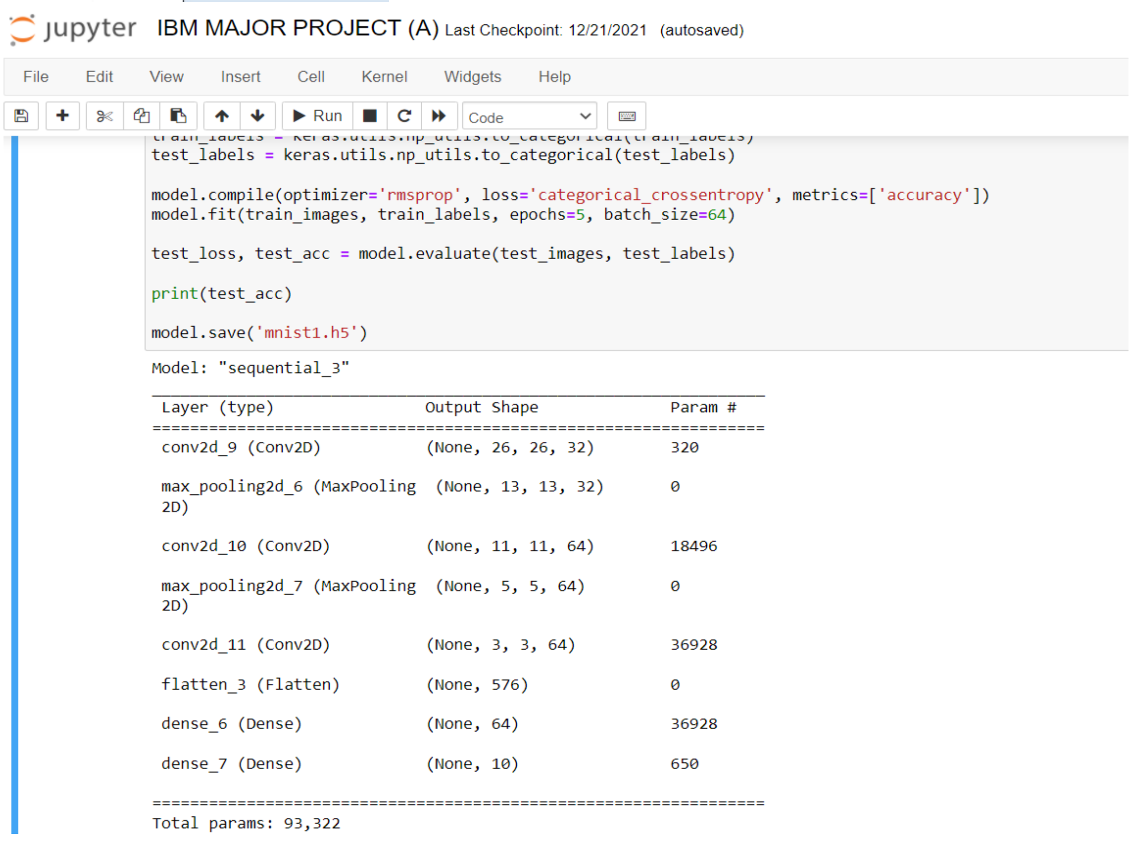
```
 conv2d_11 (Conv2D)          (None, 3, 3, 64)          36928

 flatten_3 (Flatten)         (None, 576)               0

 dense_6 (Dense)             (None, 64)                36928

 dense_7 (Dense)             (None, 10)                650

=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
938/938 [==============================] - 37s 37ms/step - loss: 0.1799 - accuracy: 0.9446
Epoch 2/5
938/938 [==============================] - 35s 38ms/step - loss: 0.0487 - accuracy: 0.9851
Epoch 3/5
938/938 [==============================] - 35s 37ms/step - loss: 0.0339 - accuracy: 0.9897
Epoch 4/5
938/938 [==============================] - 35s 37ms/step - loss: 0.0257 - accuracy: 0.9921
Epoch 5/5
938/938 [==============================] - 35s 37ms/step - loss: 0.0208 - accuracy: 0.9935
313/313 [==============================] - 3s 7ms/step - loss: 0.0252 - accuracy: 0.9929
0.992900013923645
```

After implementation we got the accuracy of 99.2%

9.SNAPSHOTS OF THE PROJECT

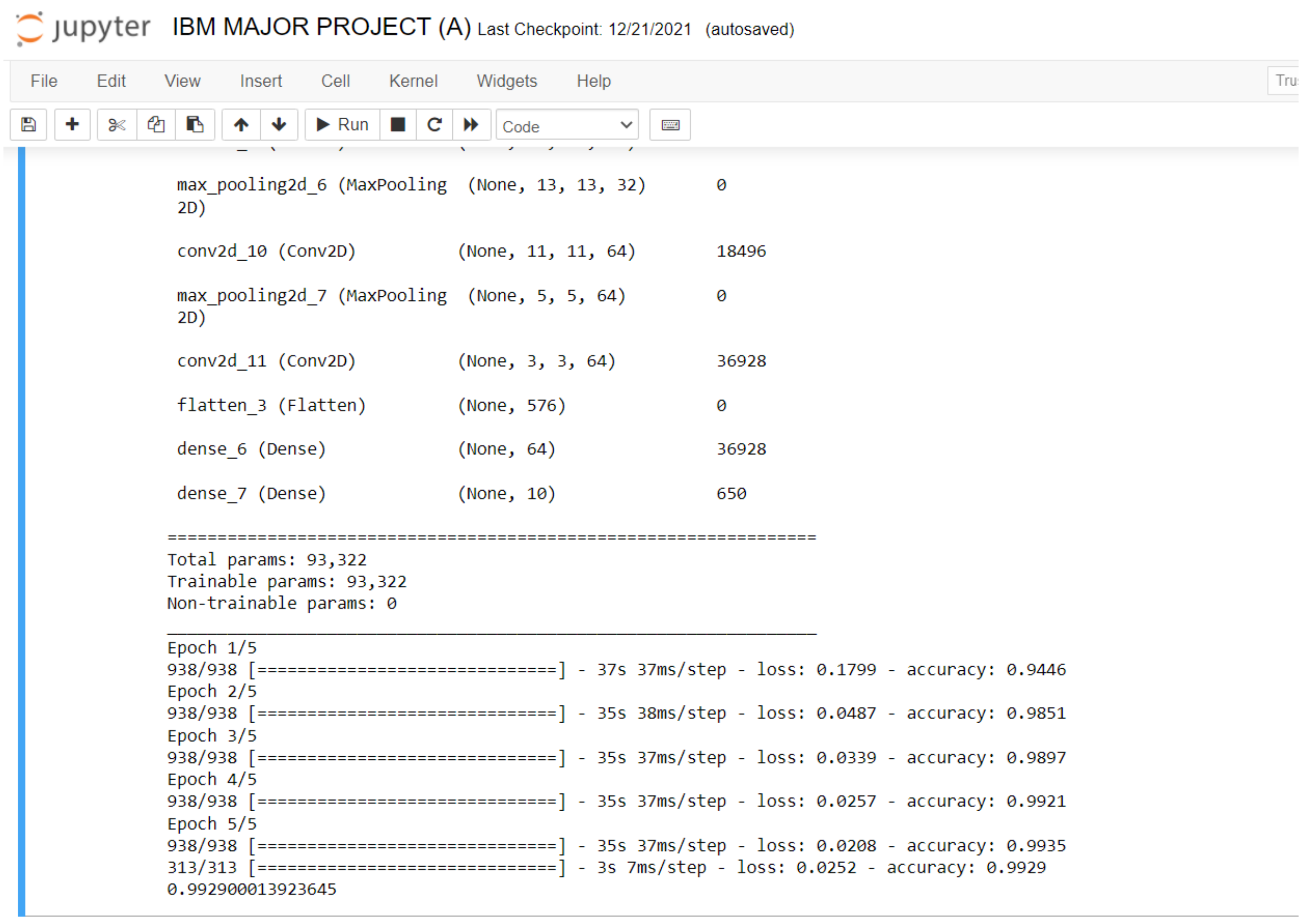

jupyter IBM MAJOR PROJECT (A) Last Checkpoint: 12/21/2021 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

```
In [4]: import keras
        from tensorflow.keras import layers
        from tensorflow.keras import models
        from keras.datasets import mnist
        from keras import utils as np_utils



        (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 = keras.utils.np_utils.to_categorical(train_labels)
        test_labels = keras.utils.np_utils.to_categorical(test_labels)

        model.compile(optimizer='rmsprop', 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)
```

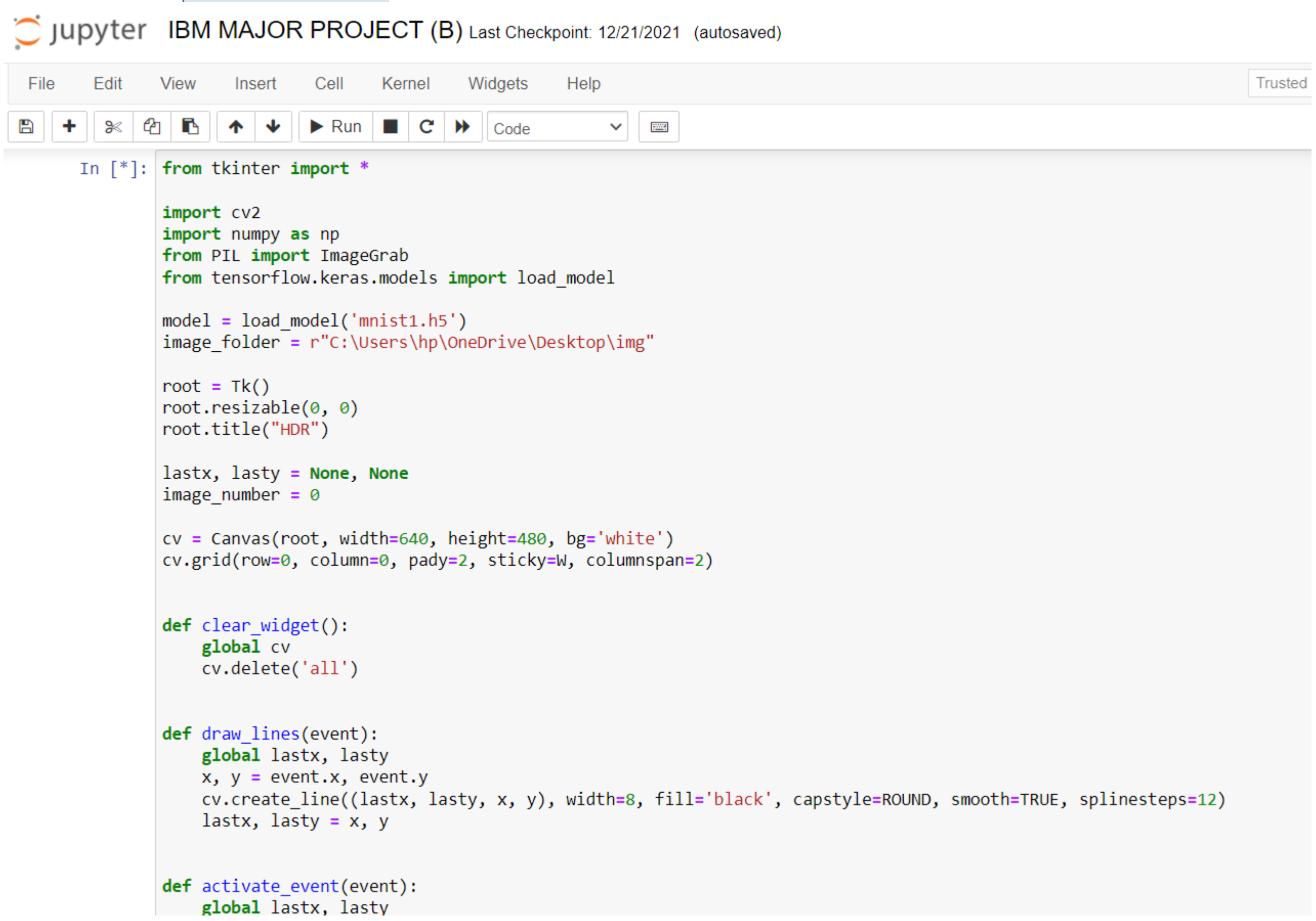

jupyter
IBM MAJOR PROJECT (A) Last Checkpoint: 12/21/2021 (autosaved)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Code

```
train_labels = keras.utils.np_utils.to_categorical(train_labels)
test_labels = keras.utils.np_utils.to_categorical(test_labels)

model.compile(optimizer='rmsprop', 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('mnist1.h5')
```

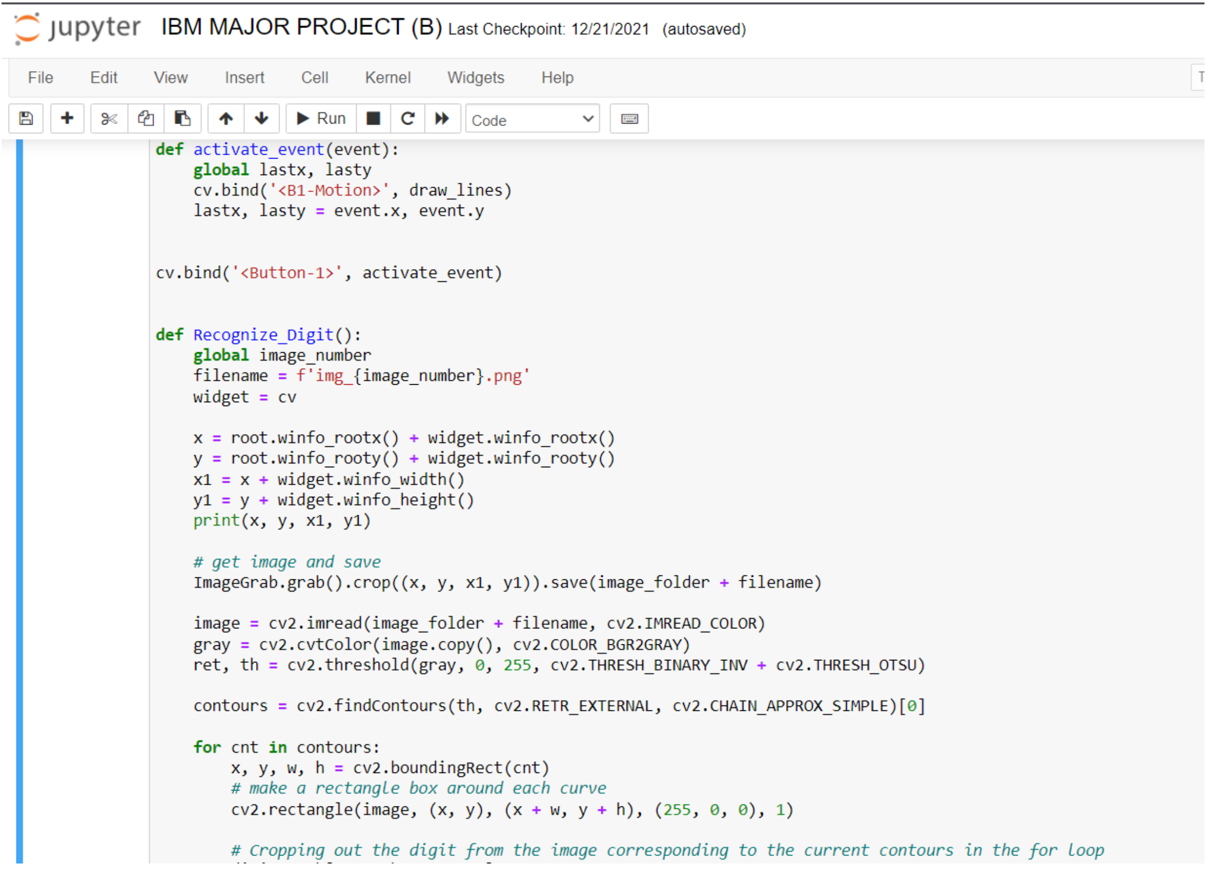
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_6 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 conv2d_10 (Conv2D)          (None, 11, 11, 64)        18496

 max_pooling2d_7 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_11 (Conv2D)          (None, 3, 3, 64)          36928

 flatten_3 (Flatten)         (None, 576)               0

 dense_6 (Dense)             (None, 64)                36928

 dense_7 (Dense)             (None, 10)                650

=================================================================
Total params: 93,322
```


jupyter
IBM MAJOR PROJECT (A) Last Checkpoint: 12/21/2021 (autosaved)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Tru
Run
Code

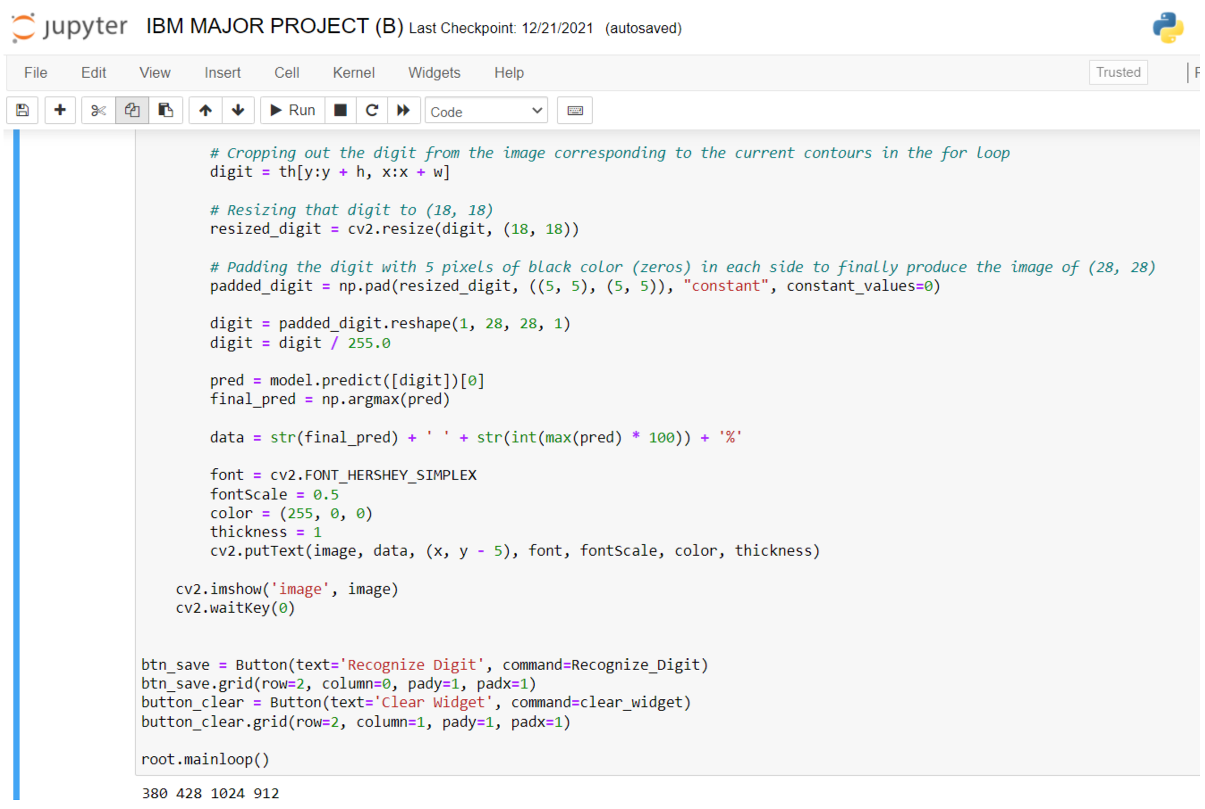
```
 max_pooling2d_6 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 conv2d_10 (Conv2D)          (None, 11, 11, 64)        18496

 max_pooling2d_7 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_11 (Conv2D)          (None, 3, 3, 64)          36928

 flatten_3 (Flatten)         (None, 576)               0

 dense_6 (Dense)             (None, 64)                36928

 dense_7 (Dense)             (None, 10)                650

=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
938/938 [==============================] - 37s 37ms/step - loss: 0.1799 - accuracy: 0.9446
Epoch 2/5
938/938 [==============================] - 35s 38ms/step - loss: 0.0487 - accuracy: 0.9851
Epoch 3/5
938/938 [==============================] - 35s 37ms/step - loss: 0.0339 - accuracy: 0.9897
Epoch 4/5
938/938 [==============================] - 35s 37ms/step - loss: 0.0257 - accuracy: 0.9921
Epoch 5/5
938/938 [==============================] - 35s 37ms/step - loss: 0.0208 - accuracy: 0.9935
313/313 [==============================] - 3s 7ms/step - loss: 0.0252 - accuracy: 0.9929
0.992900013923645
```


jupyter IBM MAJOR PROJECT (B) Last Checkpoint: 12/21/2021 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Run
Code

```
In [*]: from tkinter import *

        import cv2
        import numpy as np
        from PIL import ImageGrab
        from tensorflow.keras.models import load_model

        model = load_model('mnist1.h5')
        image_folder = r"C:\Users\hp\OneDrive\Desktop\img"

        root = Tk()
        root.resizable(0, 0)
        root.title("HDR")

        lastx, lasty = None, None
        image_number = 0

        cv = Canvas(root, width=640, height=480, bg='white')
        cv.grid(row=0, column=0, pady=2, sticky=W, columnspan=2)


        def clear_widget():
            global cv
            cv.delete('all')


        def draw_lines(event):
            global lastx, lasty
            x, y = event.x, event.y
            cv.create_line((lastx, lasty, x, y), width=8, fill='black', capstyle=ROUND, smooth=TRUE, splinesteps=12)
            lastx, lasty = x, y


        def activate_event(event):
            global lastx, lasty
```

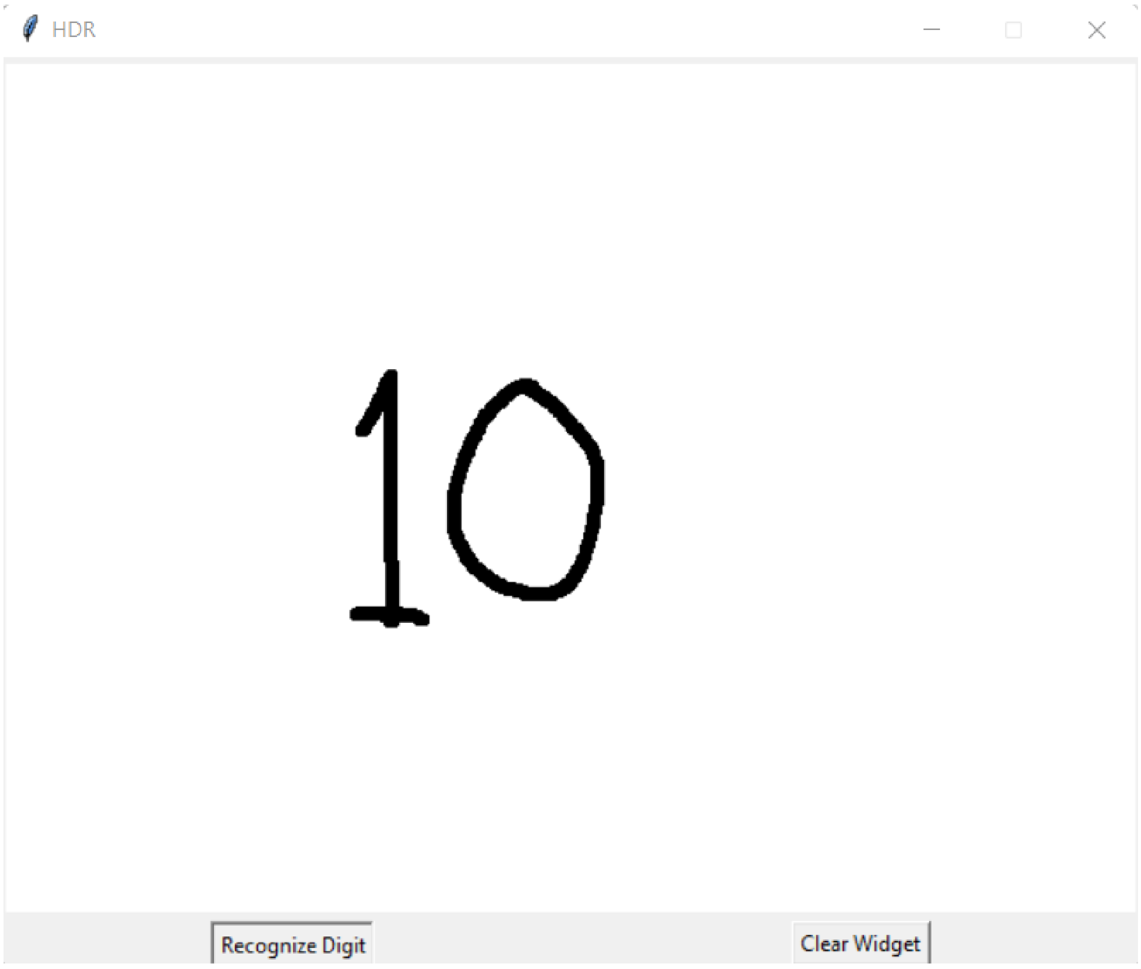

jupyter IBM MAJOR PROJECT (B) Last Checkpoint: 12/21/2021 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run Code

```
def activate_event(event):
    global lastx, lasty
    cv.bind('<B1-Motion>', draw_lines)
    lastx, lasty = event.x, event.y


cv.bind('<Button-1>', activate_event)


def Recognize_Digit():
    global image_number
    filename = f'img_{image_number}.png'
    widget = cv

    x = root.winfo_rootx() + widget.winfo_rootx()
    y = root.winfo_rooty() + widget.winfo_rooty()
    x1 = x + widget.winfo_width()
    y1 = y + widget.winfo_height()
    print(x, y, x1, y1)

    # get image and save
    ImageGrab.grab().crop((x, y, x1, y1)).save(image_folder + filename)

    image = cv2.imread(image_folder + filename, cv2.IMREAD_COLOR)
    gray = cv2.cvtColor(image.copy(), cv2.COLOR_BGR2GRAY)
    ret, th = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

    contours = cv2.findContours(th, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]

    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)
        # make a rectangle box around each curve
        cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 1)

        # Cropping out the digit from the image corresponding to the current contours in the for loop
```


jupyter IBM MAJOR PROJECT (B) Last Checkpoint: 12/21/2021 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Run Code

```
        # Cropping out the digit from the image corresponding to the current contours in the for loop
        digit = th[y:y + h, x:x + w]

        # Resizing that digit to (18, 18)
        resized_digit = cv2.resize(digit, (18, 18))

        # Padding the digit with 5 pixels of black color (zeros) in each side to finally produce the image of (28, 28)
        padded_digit = np.pad(resized_digit, ((5, 5), (5, 5)), "constant", constant_values=0)

        digit = padded_digit.reshape(1, 28, 28, 1)
        digit = digit / 255.0

        pred = model.predict([digit])[0]
        final_pred = np.argmax(pred)

        data = str(final_pred) + ' ' + str(int(max(pred) * 100)) + '%'

        font = cv2.FONT_HERSHEY_SIMPLEX
        fontScale = 0.5
        color = (255, 0, 0)
        thickness = 1
        cv2.putText(image, data, (x, y - 5), font, fontScale, color, thickness)

    cv2.imshow('image', image)
    cv2.waitKey(0)


btn_save = Button(text='Recognize Digit', command=Recognize_Digit)
btn_save.grid(row=2, column=0, pady=1, padx=1)
button_clear = Button(text='Clear Widget', command=clear_widget)
button_clear.grid(row=2, column=1, pady=1, padx=1)

root.mainloop()
```

```
380 428 1024 912
```


HDR
Recognize Digit
Clear Widget

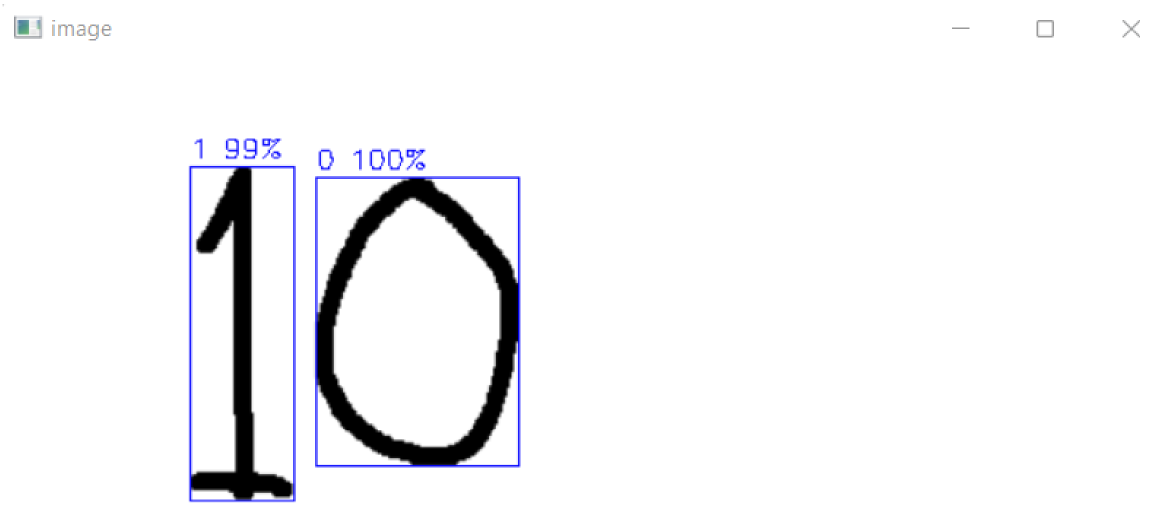


image
1 99%
0 100%

10.CONCLUSIONS

We have built and trained the Convolutional neural network for handwritten digit recognition tasks. We have also built the GUI where we draw a digit on the canvas then we predict the digit and show the results.We have created and trained a Convolutional Neural Network (CNN) that is exceptionally good at picture classification. Later, we have created the GUI, which will allow us to draw a digit on the canvas, classify it, and display the results. With the rmsprop optimizer, we were able to achieve a recognition rate of 99.89% for the MNIST database.

11.FURTHER DEVELOPMENT OR RESEARCH

The future steps would be to examine the results of all the variants more closely in order to discover new rules. We will be able to improve the performance of these versions by extracting and implementing them. Furthermore, it would be beneficial if we could make some changes to the reference set as well as the rules to make our programme more flexible and capable of recognising both typed and handwritten digits.

Furthermore, the matrices that indicate the first maximum overlap of each test image with the reference images, as well as the number of pixels left out from each, could be very useful in the future. These matrices could be combined with various clustering methods to create a programme capable of quickly recognising handwritten digits.

Finally, in order to build more rules, we considered employing linear or high level regression in the versions we developed. Because regression is best suited for binary classification and not for classifying a digit out of ten, this technique might be used to determine whether digit is the most suitable, the first maximum or second maximum, allowing us to produce more rules and therefore increase efficiency.

Although a new approach for cutting or segmenting digit strings can be offered, there are several limitations to this method that must be addressed. As a result, there is scope for future work such as:

→ Different classification models might be used at the same time to increase segmentation performance.

→ To lower the algorithm's complexity, it's advisable to reduce the amount of hypotheses. This will allow the method to run faster.

→ Better filters must be utilized to eliminate the unnecessary segmentation hypotheses in order to save computation time.

12.REFERENCES

- https://data-flair.training/blogs/python-deep-learning-project-handwritten-digit-recognition/
- https://www.analyticsvidhya.com/blog/2021/11/newbies-deep-learning-project-to-recognize-handwritten-digit/
- https://www.geeksforgeeks.org/handwritten-digit-recognition-using-neural-network/