Handwritten Digit Recognition System using a Convolutional Neural Network in Python and Java

MINOR PROJECT REPORT

By

A.L. RAHUL (RA2211003011009)
ARIHANT DEBNATH (RA2211003011033)
ARYA RAY (RA2211003011059)

Under the guidance of

Mr.N.A.S.Vinoth

In partial fulfillment for the Course

of

21CSC203P – ADVANCED PROGRAMMING PRACTICE

in Dept. of Computing Technologies

FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR

NOVEMBER 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this minor project report for the course **21CSC203P – ADVANCED PROGRAMMING PRACTICE** entitled in **"Handwritten Digit Recognition System using a Convolutional Neural Network"** is the bonafide work of **A.L. RAHUL (RA2211003011009), ARIHANT DEBNATH (RA2211003011033), ARYA RAY (RA2211003011059)** who carried out the work under my supervision.

[illegible] 14/11/2023

SIGNATURE

Mr.N.A.S.Vinoth

Assistant Professor

Department of Computing Technologies

SRMIST

Kattankulathur

Department of Computing Technology
Kattankulathur
603 203
* SRM IST *

M.Pushpalatha

SIGNATURE

Dr. Pushpalatha M

Head of The Department

Department of Computing Technologies

SRMIST

Kattankulathur

ABSTRACT

This project addresses the challenge of automated recognition of handwritten digit sequences through a comprehensive two-step approach involving initial digit segmentation and subsequent digit classification using a recognition module. The practical implications span various domains such as mail sorting, bank check processing, and form data entry. The central focus is on developing a robust algorithm capable of efficiently identifying handwritten digits from diverse sources like scanners and digital devices. Handwritten character recognition is one of the practically important issues in pattern recognition applications. The main purpose of this project is to build an automatic handwritten digit recognition method for the recognition of handwritten digit strings. The applications of digit recognition include postal mail sorting, bank check processing, form data entry, etc. The heart of the problem lies within the ability to develop an efficient algorithm that can recognize handwritten digits and which is submitted by users by way of a scanner, tablet, and other digital devices.

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. Muthamizhchelvan**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to **Chairperson**, School of Computing **Dr. Revathi Venkataraman** and **Dr. M. Pushpalatha, Head of The Department,** Department of Computing Technologies for imparting confidence to complete my course project.

We wish to express my sincere thanks to **Course Audit Professors Dr. Vadivu. G, Professor,** Department of Data Science and Business Systems and **Dr. Sasikala. E Professor,** Department of Computing Technologies and **Course Coordinator Mrs. G. Malarselvi,** Department of Computing Technologies for their constant encouragement and support.

We are thankful to our Course project Faculty **Mr.N.A.S. Vinoth, Assistant Professor,** Department of Computing Technologies, for his assistance, timely suggestion and guidance throughout the duration of this course project.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

TABLE OF CONTENTS

1. INTRODUCTION

Machine learning and deep learning plays an important role in computer technology and artificial intelligence. With the use of deep learning and machine learning, human effort can be reduced in recognizing, learning, predictions and many more areas.

This project presents recognizing the handwritten digits (0 to 9) from the famous MNIST dataset, comparing classifiers like KNN, PSVM, NN and convolution neural network on basis of performance, accuracy, time, sensitivity, positive productivity, and specificity with using different parameters with the classifiers.

To make machines more intelligent, the developers are diving into machine learning and deep learning techniques. A human learns to perform a task by practicing and repeating it again and again so that it memorizes how to perform the tasks. Then the neurons in his brain automatically trigger and they can quickly perform the task they have learned. Deep learning is also very similar to this.It uses different types of neural network architectures for different types of problems. **For example** – object recognition, image and sound classification, object detection, image segmentation, etc.

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 flavors. 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.

1.1 Motivation

Multitasking can be described as the ability to do more than one task at a time. We cannot stop ourselves, with life being as competitive and as hectic as it is sometimes. Hence multitasking on a regular basis is vital and The lack of measures in the area of multitasking in computers has given the motivation to develop a project in the field of multiprogramming systems and to develop a multitasking system which is able to run multiple tasks at a time. This “multitasking” is achieved through the CPU scheduling algorithms and hence this system would demonstrate the necessity and practical utility of the CPU scheduler.

1.2 Objective

Develop a robust Convolutional Neural Network (CNN) based Handwritten Digit Recognition System using Python and Java. The goal is to create a model capable of accurately identifying and classifying handwritten digits from images. This project aims to explore and apply CNN concepts, addressing challenges such as data quality, hyperparameter tuning, and overfitting to achieve a high level of accuracy and generalization in digit recognition. The system's success will not only contribute to understanding CNNs but also lay the foundation for potential extensions to recognize letters and individual handwriting styles.

1.3 Problem Statement

The goal of this project is to create a model that will be able to recognize and determine the handwritten digits from its image by using the concepts of Convolution Neural Network. Though the goal is to create a model which can recognize the digits, it can be extended to letters and an individual’s handwriting. The major goal of the proposed system is understanding Convolutional Neural Network, and applying it to the handwritten recognition system.

1.4 Challenges

1. **Data Quality and Quantity:**

   Gathering a sufficiently large and diverse dataset of handwritten digits can be a challenge. The quality of the data, including variations in writing styles, noise, and different image conditions, can significantly impact the performance of your model. Ensuring a balanced dataset with representative samples for each digit is essential for the CNN to generalize well.

2. **Hyperparameter Tuning:**

   CNNs have various hyperparameters, such as learning rates, batch sizes, and the architecture itself (number of layers, filter sizes, etc.). Tuning these hyperparameters effectively is crucial for achieving optimal model performance. It can be time-consuming and computationally intensive to explore the vast hyperparameter space, and finding the right combination might require experimentation and fine-tuning.

3. **Overfitting**:

   Overfitting is a common challenge in machine learning projects, especially with neural networks. Your model might perform exceptionally well on the training data but fail to generalize to new, unseen data. Implementing techniques to prevent overfitting, such as dropout layers, regularization, or using more advanced architectures like residual networks, is essential to ensure your model's robustness and accuracy on real-world handwritten digits.

2. LITERATURE SURVEY

S.No	Techniques	Authors	Year	Performance	Quality Measurement	Paper
1	KNN	Nurul Ilmi; W Tjokorda Agung Budi; R Kurniawan Nur	2016	feature extraction method and classification	89.81%	link
2	SVM	Eva Tuba and Nebojsa Bacanin	2015	Classification is facilitated by carefully tuned 45 support vector machines (SVM) using One Against One strategy.	99.05%	link
3	DNN	Mahmoud M. Abu Ghosh; Ashraf Y. Maghari	2017	Comparative Study of Different Neural Networks	98.08%	link
4	CNN	Elizabeth Rani. G Sakthimohan. M Abhigna Reddy. G Selvalakshmi. D Thomalika keerthi Raja Sekar. R	2022	use of CNN greatly improves the HDR results	99.30%	link
5	Random Forest	G. Ramesh, M. Tejas, Rakesh Thakur & H. N. Champa	2022	Usage of random forest for classification	99.63%	link

3. REQUIREMENTS

3.1 Software requirements:

These are the software configurations used:

Operating system	**:**	Windows 10.
IDE	:	Jupyter Notebook, Vs Code
Python	:	Python is an interpreted, high-level, general purpose programming language created by Guido Van Rossum
Jupyter Notebook	**:**	Jupyter is a free, open-source, interactive web tool known as a computational notebook, which researchers can use to combine software code, computational output, explanatory text and multimedia resources in a single document.

3.2 Hardware requirements:

These are the Hardware interfaces used Processor: Intel Pentium 4 or equivalent

RAM: Minimum of 256 MB or higher HDD: 10 GB or higher

Monitor: 15” or 17” color monitor

Mouse: Scroll or optical mouse

Keyboard: Standard 110 keys keyboard

4. ARCHITECTURE AND DESIGN

CNN ARCHITECTURE

CNNs are a class of Deep Neural Networks that can recognize and classify particular features from images and are widely used for analyzing visual images. Their applications range from image and video recognition, image classification, medical image analysis, computer vision and natural language processing.

The term 'Convolution" in CNN denotes the mathematical function of convolution which is a special kind of linear operation wherein two functions are multiplied to produce a third function which expresses how the shape of one function is modified by the other.

Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernels), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1.

4.1 Basic Architecture -

There are two main parts to a CNN architecture

- A convolution tool that separates and identifies the various features of the image for analysis in a process called as Feature Extraction
- A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.

CNN Layers:

The multiple occurrences of these layers shows how deep our network is, and this formation is known as the deep neural network.

- **Input**: raw pixel values are provided as input.
- **Convolutional layer**: Input layers translate the results of the neuron layer. There is a need to specify the filter to be used. Each filter can only be a 5*5 window that slides over input data and gets pixels with maximum intensities.
- **Rectified linear unit [ReLU] layer:** provided activation function on the data taken as an image. In the case of back propagation, ReLU function is used which prevents the values of pixels from changing.
- **Pooling layer:** Performs a down-sampling operation in volume along the dimensions (width, height).
- **Fully connected layer:** score class is focused, and a maximum score of the input digits is found.

As we go deeper and deeper in the layers, the complexity is increased a lot. But it might be worth going as accuracy may increase but unfortunately, time consumption also increases.

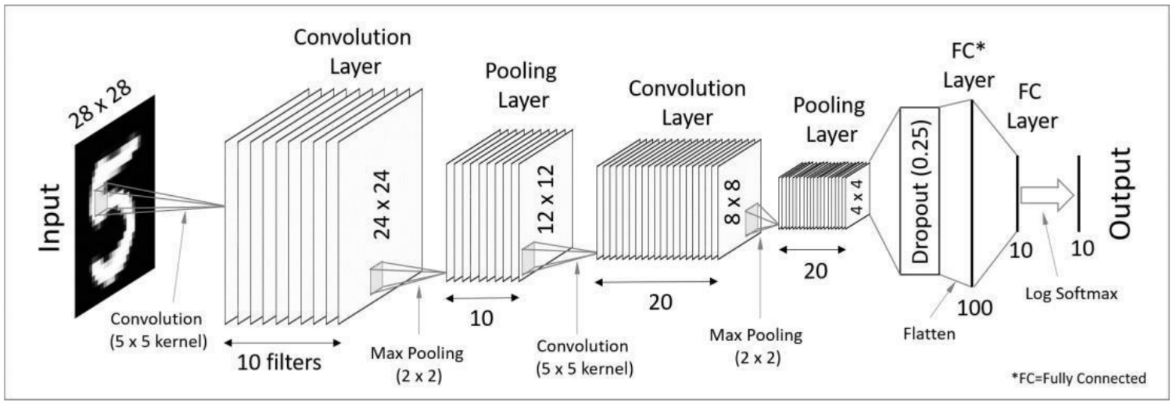


Figure 10 CNN Architecture For Handwritten Digit Recognition

4.1.1 Convolutional Layer

This layer is the first layer that is used to extract the various features from the input images. In this layer,the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM).

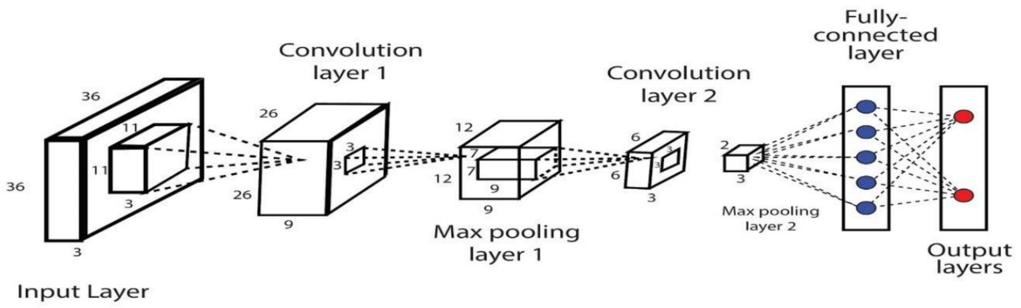


Figure 11 Convolutional Layer

The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image.

4.1.2 Pooling Layer

In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon the method used, there are several types of Pooling operations.

In Max Pooling, the largest element is taken from the feature map. Average Pooling calculates the average of the elements in a predefined size Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer.

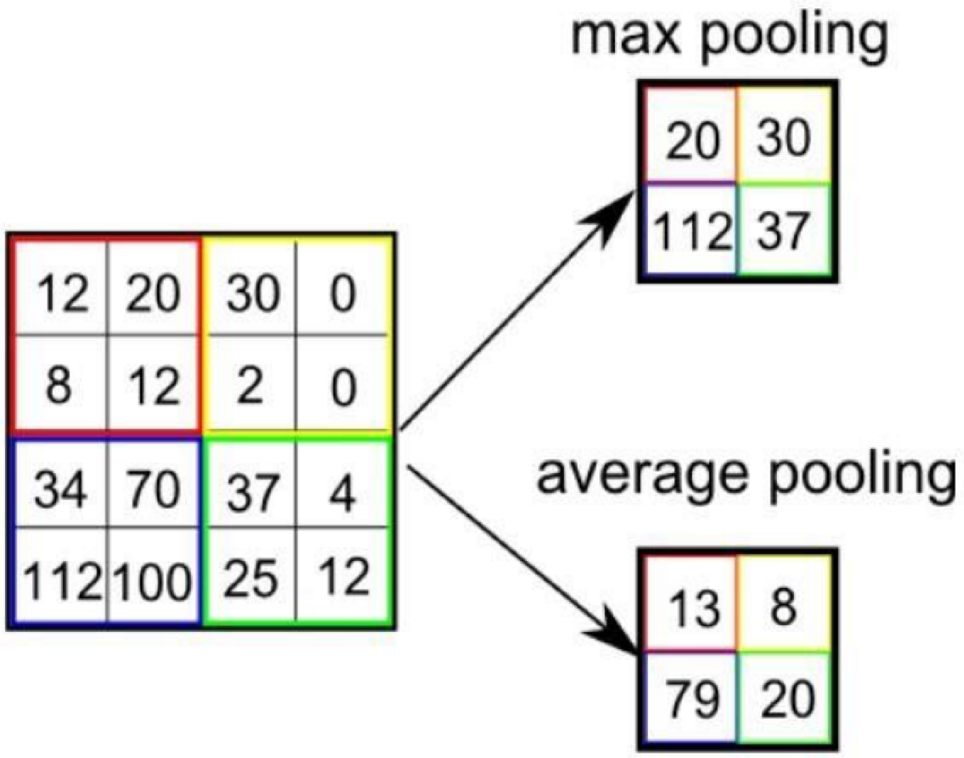


Figure 12 Pooling Layer

4.1.3 Fully Connected Layer

The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.

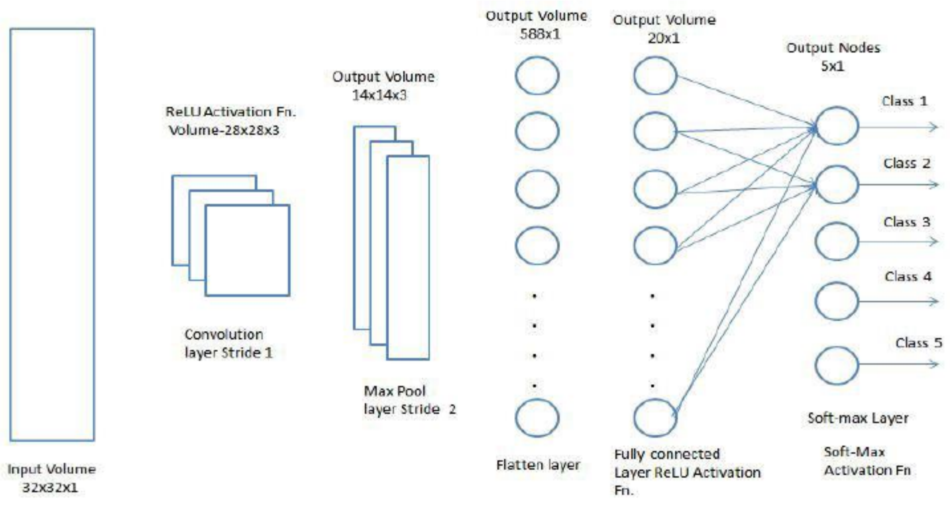
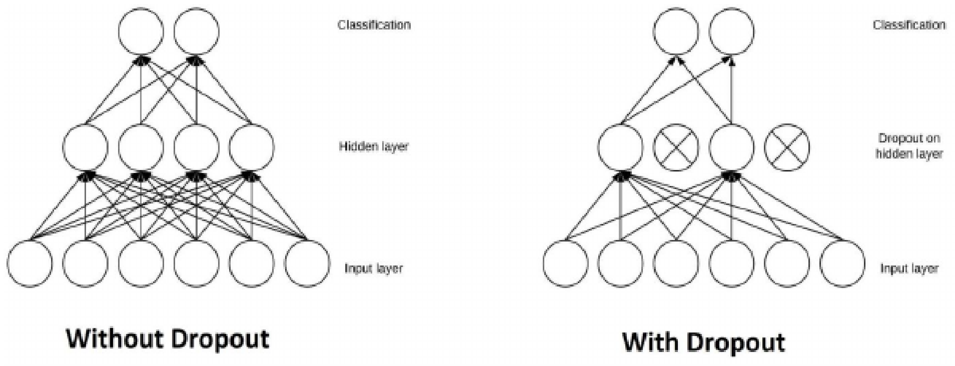


Figure 13 Fully Connected Layer

In this, the input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place.

4.1.4 Dropout

Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on the training data causing a negative impact in the model's performance when used on new data.



5. METHODOLOGY

5.1 Basic steps in constructing a Machine Learning model:

5.1.1 - Data Collection

- The quantity & quality of your data dictate how accurate our model is.
- The outcome of this step is generally a representation of data (Guo simplifies to specifying a table) which we will use for training
- Using pre-collected data, by way of datasets from Kaggle, UCI, etc., still fits into this step

5.1.2 - Data Preparation

- Wrangle data and prepare it for training
- Clean that which may require it (remove duplicates, correct errors, deal with missing values, normalization, data type conversions, etc.)
- Randomize data, which erases the effects of the particular order in which w e collected and/or otherwise prepared our data
- Visualize data to help detect relevant relationships between variables or class imbalances (bias alert!), or perform other exploratory analysis
- Split into training and evaluation sets

5.1.3 - Choose a Model

- Different algorithms are for different tasks; choose the right one

5.1.4 - Train the Model

- The goal of training is to answer a question or make a prediction correctly as often as possible
- Linear regression example: algorithm would need to learn values for m (or W) and b (x is input, y is output)
- Each iteration of process is a training step

5.1.5 - Evaluate the Model

- Uses some metric or combination of metrics to "measure" objective performance of model
- Test the model against previously unseen data
- This unseen data is meant to be somewhat representative of model performance in the real world, but still helps tune the model (as opposed to test data, which does not)
- Good train/eval split? 80/20, 70/30, or similar, depending on domain, data availability, dataset particulars, etc.

5.1.6 - Parameter Tuning

- This step refers to hyperparameter tuning, which is an "artform" as opposed to a science
- Tune model parameters for improved performance
- Simple model hyperparameters may include: number of training steps, learning rate, initialization values and distribution, etc.

5.1.7 - Make Predictions

- Using further (test set) data which have, until this point, been withheld from the model (and for which class labels are known), are used to test the model; a better approximation of how the model will perform in the real world

5.2 Methodologies for Handwritten Digit Recognition System

We used MNIST as a primary dataset to train the model, and it consists of 70,000 handwritten raster images from 250 different sources out of which 60,000 are used for training, and the rest are used for training validation. Our proposed method mainly separated into stages, preprocessing, Model Construction, Training & Validation, Model Evaluation & Prediction. Since the loading dataset is necessary for any process, all the steps come after it.

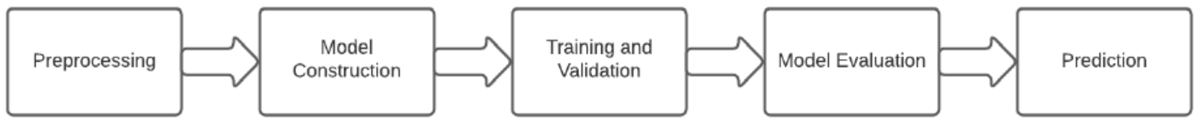


Figure 3 Steps in System development

5.2.1 Import the libraries:

Libraries required are Keras, Tensor flow, Numpy, Pillow, Tkinkter.

1. Keras:

Keras is a powerful and easy-to-use free open source Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in just a few lines of code. It uses libraries such as Python, C#, C++ or standalone machine learning toolkits. Theano and TensorFlow are very powerful libraries but difficult to understand for creating neural networks. Keras is based on a minimal structure that provides a clean and easy way to create deep learning models based on TensorFlow or Theano. Keras is designed to quickly define deep learning models. Well, Keras is an optimal choice for deep learning applications.

2. TensorFlow:

TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow. TensorFlow tutorial is designed for both beginners and professionals. Our tutorial provides all the basic and advanced concepts of machine learning and deep learning concepts such as deep neural network, image processing and sentiment analysis.TensorFlow is one of the famous deep learning frameworks, developed by Google Team.

3. Numpy:

NumPy is a Python library used for working with arrays. It also has functions for working in the domain of linear algebra, Fourier transform, and matrices. Numpy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays.

4. Pillow:

Pillow is a free and open source library for the Python programming language that allows you to easily create & manipulate digital images. Pillow is built on top of PIL (Python Image Library). PIL is one of the important modules for image processing in Python. However, the PIL module has not been supported since 2011 and doesn’t support python 3.

5. Tkinter:

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and an easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

5.3 Loading The Data Set:

5.3.1 MNIST Data Set:

Modified National Institute of Standards and Technology (MNIST) is a large set of computer vision dataset which is extensively used for training and testing different systems. It was created from the two special datasets of National Institute of Standards and Technology (NIST) which holds binary images of handwritten digits. The training set contains handwritten digits from 250 people, among them 50% training dataset was employees from the Census Bureau and the rest of it was from high school students. However, it is often attributed as the first datasets among other datasets to prove the effectiveness of the neural networks.

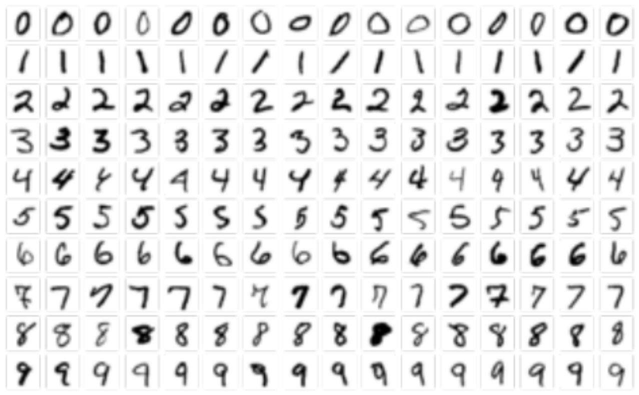


Figure 4 MNIST Data Set

The database contains 60,000 images used for training as well as few of them can be used for cross-validation purposes and 10,000 images used for testing. All the digits are grayscale and positioned in a fixed size where the intensity lies at the center of the image with 28×28 pixels. Since all the images are 28×28 pixels, it forms an array which can be flattened into 28*28=784 dimensional vector. Each component of the vector is a binary value which describes the intensity of the pixel.

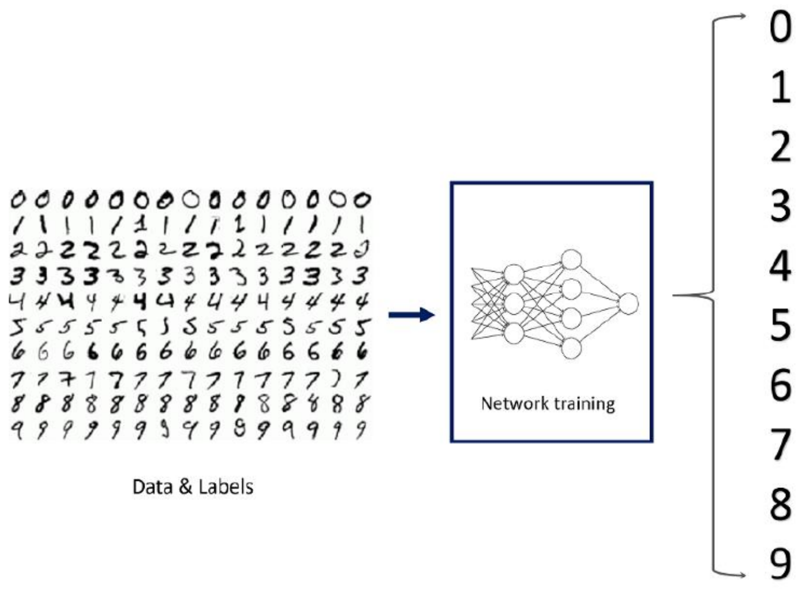


Figure 5 MNIST Examole

5.6 Model Construction

Now, comes the fun part where we finally get to use the meticulously prepared data for model building. Depending on the data type (qualitative or quantitative) of the target variable (commonly referred to as the Y variable) we are either going to be building a classification (if Y is qualitative) or regression (if Y is quantitative) model.

5.6.1 Learning Algorithms

Machine learning algorithms could be broadly categorized to one of three types:

1. Supervised learning

In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances.

It is a machine learning task that establishes the mathematical relationship between input X and output Y variables. Such X, Y pair constitutes the labeled data that are used for model building in an effort to learn how to predict the output from the input. Supervised learning problems can be
further grouped into regression and classification problems.

- Classification: A classification problem is when the output variable is a category, such as “red”or “blue” or “disease” and “no disease”.
- Regression: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

2. Unsupervised learning

is a machine learning task that makes use of only the input X variables.Such X variables are unlabeled data that the learning algorithm uses in modeling the inherent structure of the data. Unsupervised learning problems can be further grouped into clustering and association problems.

- Clustering: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

3. Reinforcement learning

Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation.

Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement the agent decides what to do to perform the given task.

In the absence of a training dataset, it is bound to learn from its experience. It is a machine learning task that decides on the next course of action and it does this by learning through trial and error in an effort to maximize the reward.

- Input: The input should be an initial state from which the model will start
- Output: There are many possible output as there are variety of solution to a particular problem
- Training: The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output. The model continues to learn. The best solution is decided based on the maximum reward.

6. IMPLEMENTATION

1. Python Implementation :

train_digit_recognizer.py

```
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()

from keras.layers import Dropout, BatchNormalization

(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(BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(Dropout(0.25))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(Dropout(0.25))

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='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=15, batch_size=64)

test_loss, test_acc = model.evaluate(test_images, test_labels)

print(test_acc)

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

final_gui.py

```
from tkinter import *

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

model = load_model('mnist.h5')
image_folder = "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
   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()
   y = root.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
       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)
   cv2.destroyAllWindows()

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()
```

2. Java Implementation

MNIST.java

```java
package org.tensorflow.keras.datasets;

import org.tensorflow.data.Dataset;
import org.tensorflow.keras.utils.DataUtils;
import org.tensorflow.keras.utils.Keras;
import org.tensorflow.op.Ops;
import org.tensorflow.types.TFloat32;

import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.nio.FloatBuffer;
import java.util.Arrays;
import java.util.zip.GZIPInputStream;

/**
 * Code based on example found at:
 *
https://github.com/karllessard/models/tree/master/samples/languages/java/mni
st/src/main/java/org/tensorflow/model/sample/mnist
 * <p>
 * Utility for downloading and using MNIST data with a local keras
installation.
 */
public class MNIST {
  private static final int IMAGE_MAGIC = 2051;
  private static final int LABELS_MAGIC = 2049;
  private static final int OUTPUT_CLASSES = 10;

  private static final String TRAIN_IMAGES = "train-images-idx3-ubyte.gz";
  private static final String TRAIN_LABELS = "train-labels-idx1-ubyte.gz";
  private static final String TEST_IMAGES = "t10k-images-idx3-ubyte.gz";
  private static final String TEST_LABELS = "t10k-labels-idx1-ubyte.gz";

  private static final String ORIGIN_BASE =
"http://yann.lecun.com/exdb/mnist/";

  private static final String LOCAL_PREFIX = "datasets/mnist/";

  public final Dataset train;
  public final Dataset test;

  private MNIST(Dataset train, Dataset test) {
    this.train = train;
    this.test = test;
  }

  /**
   * Download MNIST dataset files to the local .keras/ directory.
   *
   * @throws IOException when the download fails
   */
  public static void download() throws IOException {
    DataUtils.getFile(LOCAL_PREFIX + TRAIN_IMAGES, ORIGIN_BASE +
TRAIN_IMAGES);
    DataUtils.getFile(LOCAL_PREFIX + TRAIN_LABELS, ORIGIN_BASE +
TRAIN_LABELS);
    DataUtils.getFile(LOCAL_PREFIX + TEST_IMAGES, ORIGIN_BASE +
TEST_IMAGES);
    DataUtils.getFile(LOCAL_PREFIX + TEST_LABELS, ORIGIN_BASE +
TEST_LABELS);
  }

  public static MNIST loadData(Ops tf) throws IOException {
    // Download MNIST files if they don't exist.
    MNIST.download();

    // Construct local data file paths
    String trainImagesPath = Keras.path(LOCAL_PREFIX,
TRAIN_IMAGES).toString();
    String trainLabelsPath = Keras.path(LOCAL_PREFIX,
TRAIN_LABELS).toString();

    String testImagesPath = Keras.path(LOCAL_PREFIX,
TEST_IMAGES).toString();
    String testLabelsPath = Keras.path(LOCAL_PREFIX,
TEST_LABELS).toString();

    // Read data files into arrays
    float[][][] trainImages = readImages2D(trainImagesPath);
    float[][] trainLabels = readLabelsOneHot(trainLabelsPath);

    float[][][] testImages = readImages2D(testImagesPath);
    float[][] testLabels = readLabelsOneHot(testLabelsPath);


    Dataset train = Dataset.fromTensorSlices(tf,
        Arrays.asList(
            tf.constant(trainImages),
            tf.constant(trainLabels)),
        Arrays.asList(TFloat32.DTYPE, TFloat32.DTYPE));

    Dataset test = Dataset.fromTensorSlices(tf,
        Arrays.asList(
            tf.constant(testImages),
            tf.constant(testLabels)),
        Arrays.asList(TFloat32.DTYPE, TFloat32.DTYPE));

    return new MNIST(train, test);
  }

  /**
   * Reads MNIST image files into an array, given an image datafile path.
   *
   * @param imagesPath MNIST image datafile path
   * @return an array of shape (# examples, # classes) containing the label
data
   * @throws IOException when the file reading fails.
   */
  static float[][][] readImages2D(String imagesPath) throws IOException {
    try (DataInputStream inputStream = new DataInputStream(new
GZIPInputStream(new FileInputStream(imagesPath)))) {

      if (inputStream.readInt() != IMAGE_MAGIC) {
        throw new IllegalArgumentException("Invalid Image Data File");
      }

      int numImages = inputStream.readInt();
      int rows = inputStream.readInt();

      return readImageBuffer2D(inputStream, numImages, rows);
    }
  }
  /**
   * Reads MNIST label files into an array, given a label datafile path.
   *
   * @param labelsPath MNIST label datafile path
   * @return an array of shape (# examples, # classes) containing the label
data
   * @throws IOException when the file reading fails.
   */
  static float[][] readLabelsOneHot(String labelsPath) throws IOException {
    try (DataInputStream inputStream = new DataInputStream(new
GZIPInputStream(new FileInputStream(labelsPath)))) {
      if (inputStream.readInt() != LABELS_MAGIC) {
        throw new IllegalArgumentException("Invalid Label Data File");
      }
      int numLabels = inputStream.readInt();
      return readLabelBuffer(inputStream, numLabels);
    }
  }

  private static byte[][] readBatchedBytes(DataInputStream inputStream, int
batches, int bytesPerBatch)
      throws IOException {
    byte[][] entries = new byte[batches][bytesPerBatch];
    for (int i = 0; i < batches; i++) {
      inputStream.readFully(entries[i]);
    }
    return entries;
  }

  private static float[][][] readImageBuffer2D(DataInputStream inputStream,
int numImages, int imageWidth)
      throws IOException {
    float[][][] unsignedEntries = new
float[numImages][imageWidth][imageWidth];
    for (int i = 0; i < unsignedEntries.length; i++) {
      byte[][] entries = readBatchedBytes(inputStream, imageWidth,
imageWidth);
      for (int j = 0; j < unsignedEntries[0].length; j++) {
        for (int k = 0; k < unsignedEntries[0][0].length; k++) {
          unsignedEntries[i][j][k] = (float) (entries[j][k] & 0xFF) /
255.0f;
        }
      }
    }
    return unsignedEntries;
  }

  private static float[][] readLabelBuffer(DataInputStream inputStream, int
numLabels) throws IOException {
    byte[][] entries = readBatchedBytes(inputStream, numLabels, 1);

    float[][] labels = new float[numLabels][OUTPUT_CLASSES];
    for (int i = 0; i < entries.length; i++) {
      labelToOneHotVector(entries[i][0] & 0xFF, labels[i], false);
    }

    return labels;
  }
  private static void labelToOneHotVector(int label, float[] oneHot, boolean
fill) {
    if (label >= oneHot.length) {
      throw new IllegalArgumentException("Invalid Index for One-Hot
Vector");
    }

    if (fill)
      Arrays.fill(oneHot, 0);
    oneHot[label] = 1.0f;
  }
}
```

MNISTBasicGraphClassifier.java

```
package org.tensorflow.keras.examples.mnist;

import org.tensorflow.Graph;
import org.tensorflow.Operand;
import org.tensorflow.Output;
import org.tensorflow.Session;
import org.tensorflow.data.Dataset;
import org.tensorflow.data.DatasetIterator;
import org.tensorflow.keras.datasets.MNIST;
import org.tensorflow.op.Op;
import org.tensorflow.op.Ops;
import org.tensorflow.op.core.Assign;
import org.tensorflow.op.core.Variable;
import org.tensorflow.tools.Shape;
import org.tensorflow.training.optimizers.GradientDescent;
import org.tensorflow.training.optimizers.Optimizer;
import org.tensorflow.types.TFloat32;
import org.tensorflow.types.TInt64;

import java.io.IOException;
import java.util.List;

import static org.tensorflow.tools.ndarray.NdArrays.vectorOf;

/**
 * An example showing a simple feed-forward classifier for MNIST
 * using tf.data and core TensorFlow (in Graph Mode).
 */
public class MNISTBasicGraphClassifier implements Runnable {
```

```java
  private static final int INPUT_SIZE = 28 * 28;

  private static final float LEARNING_RATE = 0.2f;
  private static final int FEATURES = 10;
  private static final int BATCH_SIZE = 100;
  private static final int EPOCHS = 10;

  static class EpochLogs {
    float loss = 0;
    float accuracy = 0;
    int batches = 0;

    void batchUpdate(float batchLoss, float batchAccuracy) {
      ++batches;
      loss += batchLoss;
      accuracy += batchAccuracy;
    }
  }

  public static void main(String[] args) {
    new MNISTBasicGraphClassifier().run();
  }

  public Operand<TFloat32> predict(Ops tf, Operand<TFloat32> images,
Variable<TFloat32> weights,
                                   Variable<TFloat32> biases) {
    return tf.nn.softmax(tf.math.add(tf.linalg.matMul(images, weights),
biases));
  }

  public Operand<TFloat32> crossEntropyLoss(Ops tf, Operand<TFloat32>
predicted, Operand<TFloat32> actual) {
    return tf.math.mean(tf.math.neg(tf.reduceSum(tf.math.mul(actual,
tf.math.log(predicted)), tf.constant(1))),
        tf.constant(0));
  }

  public Operand<TFloat32> accuracy(Ops tf, Operand<TFloat32> predicted,
Operand<TFloat32> actual) {
    Operand<TInt64> yHat = tf.math.argMax(predicted, tf.constant(1));
    Operand<TInt64> yTrue = tf.math.argMax(actual, tf.constant(1));
    Operand<TFloat32> accuracy =
tf.math.mean(tf.dtypes.cast(tf.math.equal(yHat, yTrue), TFloat32.DTYPE),
        tf.constant(0));
    return accuracy;
  }
  public void run() {
    try (Graph graph = new Graph()) {
      Ops tf = Ops.create(graph);

      // Load datasets
      MNIST mnist;
      try {
        mnist = MNIST.loadData(tf);
      } catch (IOException e) {
        System.out.println("Could not load dataset.");
        return;
      }

      Dataset train = mnist.train.batch(BATCH_SIZE);
      Dataset test = mnist.test.batch(BATCH_SIZE);

      // Extract iterators and train / test components
      DatasetIterator trainIterator = train.makeInitializeableIterator();
      Op initTrainIterator = trainIterator.makeInitializer(train);

      List<Output<?>> components = trainIterator.getNext();
      Operand<TFloat32> trainImages =
components.get(0).expect(TFloat32.DTYPE);
      Operand<TFloat32> trainLabels =
components.get(1).expect(TFloat32.DTYPE);

      DatasetIterator testIterator = test.makeInitializeableIterator();
      Op initTestIterator = testIterator.makeInitializer(test);

      List<Output<?>> testComponents = testIterator.getNext();
      Operand<TFloat32> testImages =
testComponents.get(0).expect(TFloat32.DTYPE);
      Operand<TFloat32> testLabels =
testComponents.get(1).expect(TFloat32.DTYPE);


      // Flatten image tensors
      trainImages = tf.reshape(trainImages,
          tf.constant(vectorOf(-1, INPUT_SIZE)));
      testImages = tf.reshape(testImages,
          tf.constant(vectorOf(-1, INPUT_SIZE)));

      // Declare, initialize weights
      Variable<TFloat32> weights = tf.variable(Shape.of(INPUT_SIZE,
FEATURES), TFloat32.DTYPE);
      Assign<TFloat32> weightsInit = tf.assign(weights,
          tf.zeros(tf.constant(vectorOf(INPUT_SIZE, FEATURES)),
TFloat32.DTYPE));

      Variable<TFloat32> biases = tf.variable(Shape.of(FEATURES),
          TFloat32.DTYPE);
      Assign<TFloat32> biasesInit = tf.assign(biases,
          tf.zeros(tf.constant(vectorOf(FEATURES)), TFloat32.DTYPE));

      // SETUP: Training
      Operand<TFloat32> trainPrediction = predict(tf, trainImages, weights,
biases);
      Operand<TFloat32> trainAccuracy = accuracy(tf, trainPrediction,
trainLabels);

      Operand<TFloat32> trainLoss = crossEntropyLoss(tf, trainPrediction,
trainLabels);

      Optimizer gradientDescent = new GradientDescent(graph, LEARNING_RATE);
      Op optimizerTargets = gradientDescent.minimize(trainLoss);

      // SETUP: Testing
      Operand<TFloat32> testPrediction = predict(tf, testImages, weights,
biases);
      Operand<TFloat32> testAccuracy = accuracy(tf, testPrediction,
testLabels);
      Operand<TFloat32> testLoss = crossEntropyLoss(tf, testPrediction,
testLabels);

      try (Session session = new Session(graph)) {

        // Initialize weights and biases
        session.runner()
            .addTarget(weightsInit)
            .addTarget(biasesInit)
            .run();

        // Run training loop
        for (int i = 0; i < EPOCHS; i++) {
          // reset iterator object
          session.run(initTrainIterator);

          EpochLogs epochLogs = new EpochLogs();

          session.runner()
```

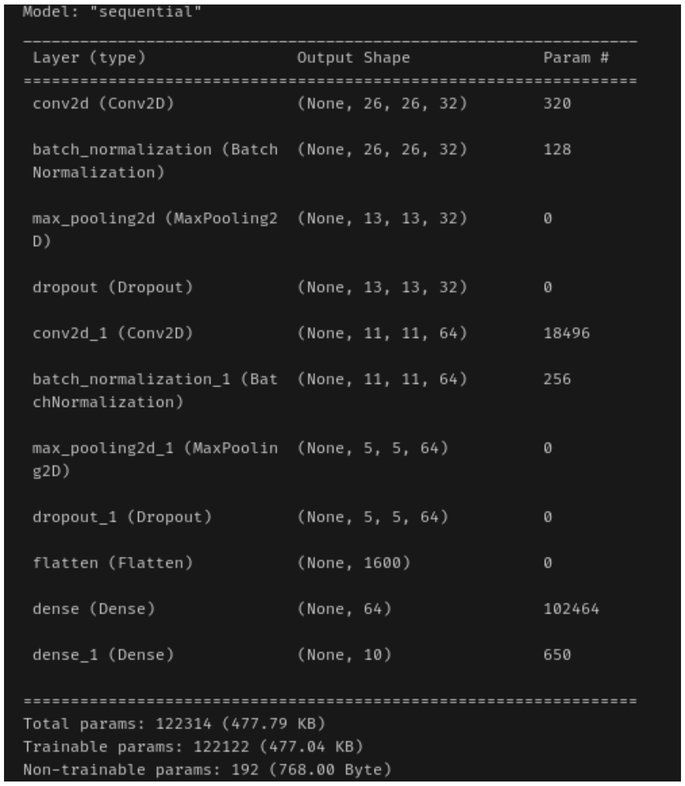
```
              .addTarget(optimizerTargets)
              .fetch(trainLoss)
              .fetch(trainAccuracy)
              .repeat()
              .limit(500)
              .forEach(outputs ->
                  epochLogs.batchUpdate(outputs.popFloat(),
outputs.popFloat())
              );

          System.out.println("Epoch Accuracy " + i + ": " +
epochLogs.accuracy / epochLogs.batches);
        }

        // Evaluate on test set
        session.run(initTestIterator);

        EpochLogs testLogs = new EpochLogs();

        session.runner()
            .fetch(testLoss)
            .fetch(testAccuracy)
            .repeat()
            .forEach(outputs -> testLogs.accuracy += outputs.popFloat());

        System.out.println("Test Accuracy " + ": " + testLogs.accuracy);
      }
    }
  }
}
```

Output of MNISTBasicgraphClassifier.java:

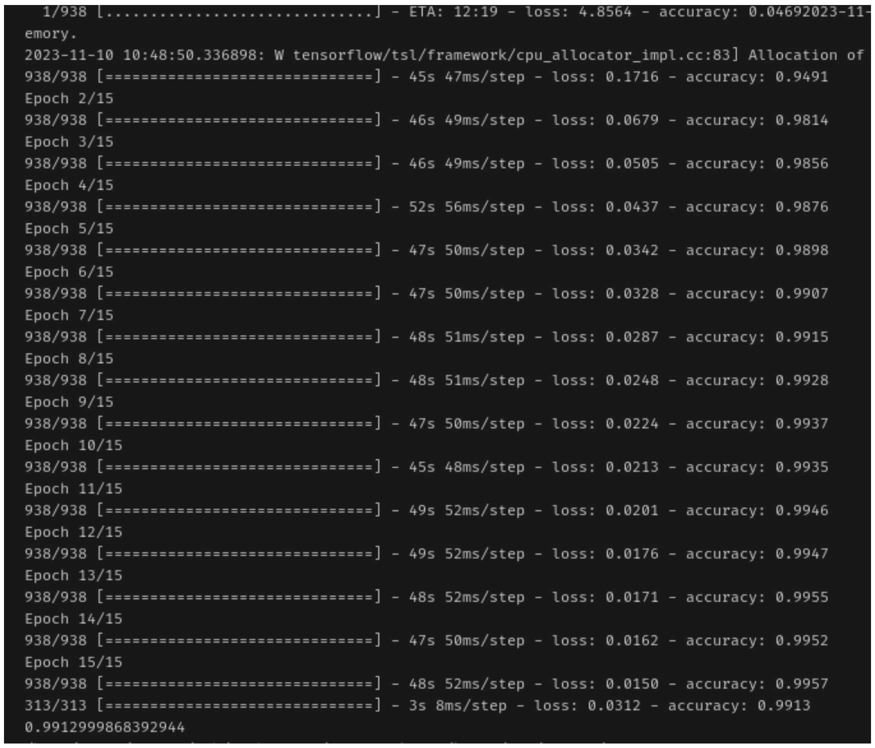
```
Epoch Accuracy 1: 0.9077837
Epoch Accuracy 2: 0.9139839
Epoch Accuracy 3: 0.91738385
Epoch Accuracy 4: 0.91953397
Epoch Accuracy 5: 0.921067
Epoch Accuracy 6: 0.922234
Epoch Accuracy 7: 0.92283416
Epoch Accuracy 8: 0.9237675
Epoch Accuracy 9: 0.92438453
Test Accuracy : 0.92010015
```

7. RESULTS AND DISCUSSION

a. Running train_digitrecognizer.py file - Epochs Running:

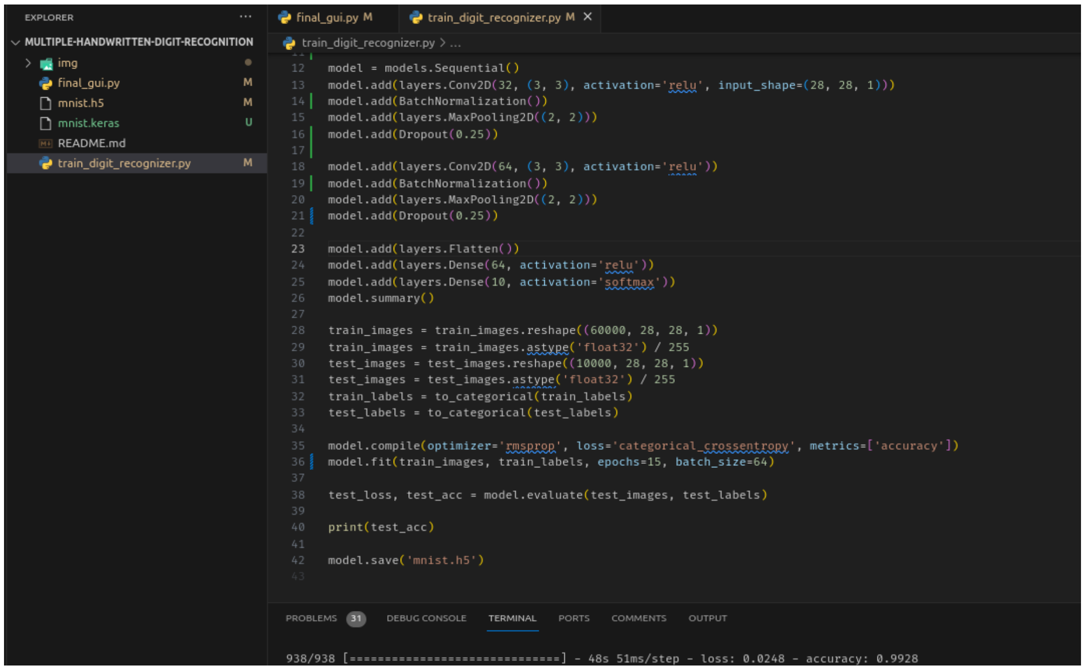


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 batch_normalization (Batch  (None, 26, 26, 32)        128
 Normalization)

 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 D)

 dropout (Dropout)           (None, 13, 13, 32)        0

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 batch_normalization_1 (Bat  (None, 11, 11, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 dropout_1 (Dropout)         (None, 5, 5, 64)          0

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 64)                102464

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 122314 (477.79 KB)
Trainable params: 122122 (477.04 KB)
Non-trainable params: 192 (768.00 Byte)
```



```
  1/938 [..............................] - ETA: 12:19 - loss: 4.8564 - accuracy: 0.04692023-11-
emory.
2023-11-10 10:48:50.336898: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
938/938 [==============================] - 45s 47ms/step - loss: 0.1716 - accuracy: 0.9491
Epoch 2/15
938/938 [==============================] - 46s 49ms/step - loss: 0.0679 - accuracy: 0.9814
Epoch 3/15
938/938 [==============================] - 46s 49ms/step - loss: 0.0505 - accuracy: 0.9856
Epoch 4/15
938/938 [==============================] - 52s 56ms/step - loss: 0.0437 - accuracy: 0.9876
Epoch 5/15
938/938 [==============================] - 47s 50ms/step - loss: 0.0342 - accuracy: 0.9898
Epoch 6/15
938/938 [==============================] - 47s 50ms/step - loss: 0.0328 - accuracy: 0.9907
Epoch 7/15
938/938 [==============================] - 48s 51ms/step - loss: 0.0287 - accuracy: 0.9915
Epoch 8/15
938/938 [==============================] - 48s 51ms/step - loss: 0.0248 - accuracy: 0.9928
Epoch 9/15
938/938 [==============================] - 47s 50ms/step - loss: 0.0224 - accuracy: 0.9937
Epoch 10/15
938/938 [==============================] - 45s 48ms/step - loss: 0.0213 - accuracy: 0.9935
Epoch 11/15
938/938 [==============================] - 49s 52ms/step - loss: 0.0201 - accuracy: 0.9946
Epoch 12/15
938/938 [==============================] - 49s 52ms/step - loss: 0.0176 - accuracy: 0.9947
Epoch 13/15
938/938 [==============================] - 48s 52ms/step - loss: 0.0171 - accuracy: 0.9955
Epoch 14/15
938/938 [==============================] - 47s 50ms/step - loss: 0.0162 - accuracy: 0.9952
Epoch 15/15
938/938 [==============================] - 48s 52ms/step - loss: 0.0150 - accuracy: 0.9957
313/313 [==============================] - 3s 8ms/step - loss: 0.0312 - accuracy: 0.9913
0.9912999868392944
```

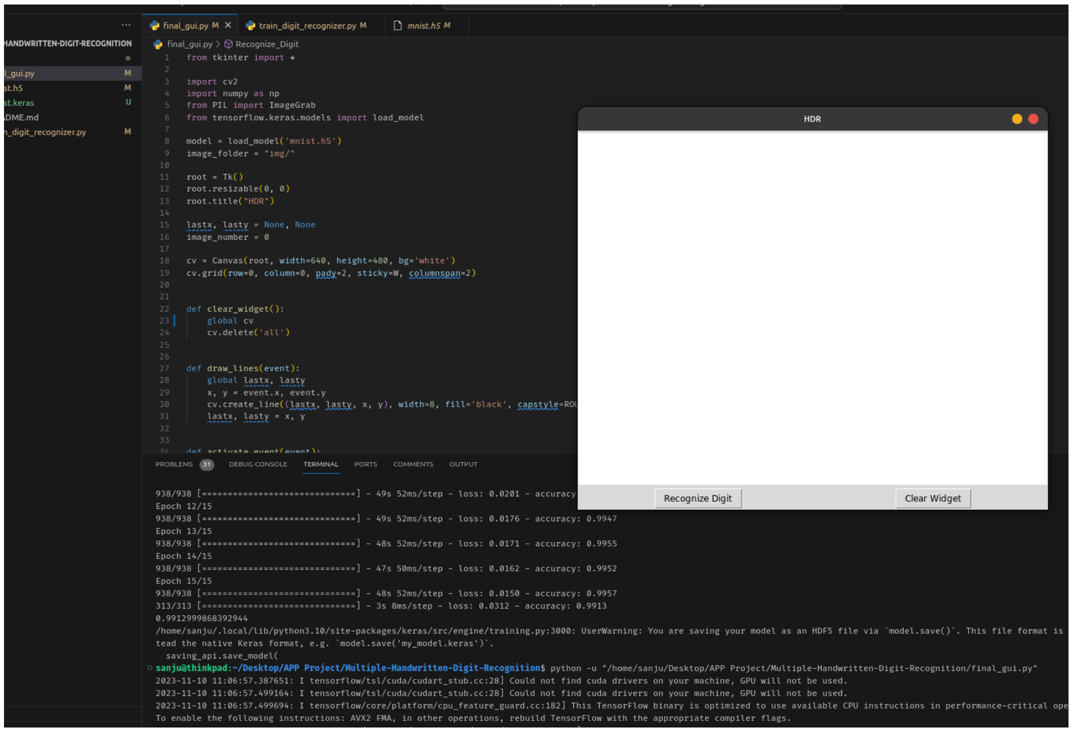
b. Saving the model mnist.h5 file



```python
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(Dropout(0.25))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(Dropout(0.25))

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='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=15, batch_size=64)

test_loss, test_acc = model.evaluate(test_images, test_labels)

print(test_acc)

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

```
938/938 [==============================] - 48s 51ms/step - loss: 0.0248 - accuracy: 0.9928
```

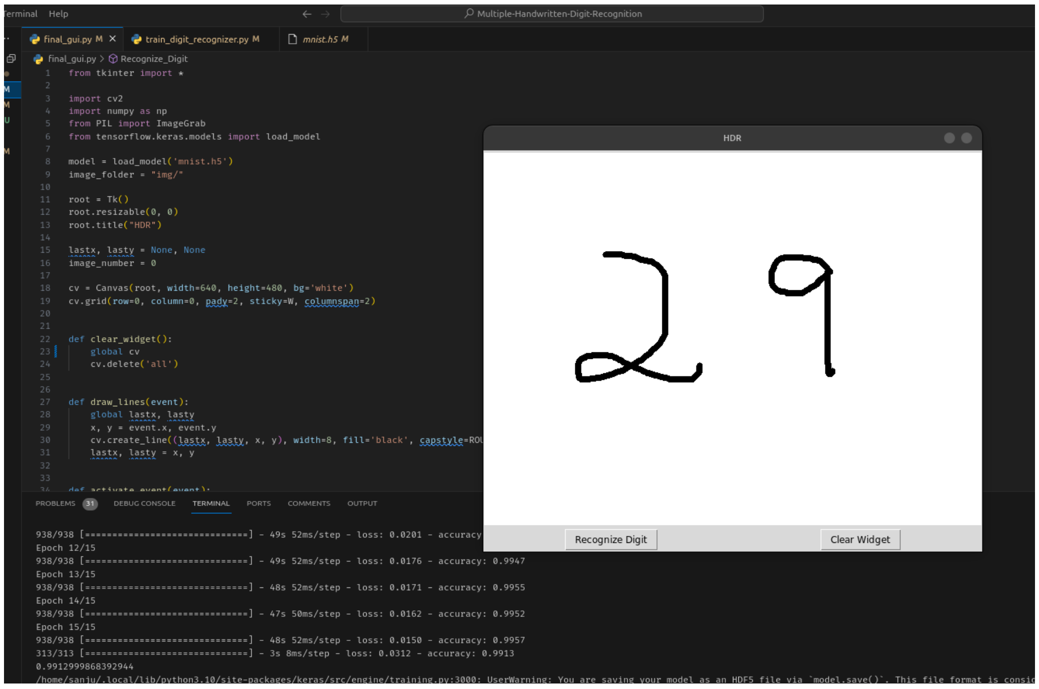
c. Running the final_gui.py file



```python
from tkinter import *

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

model = load_model('mnist.h5')
image_folder = "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=ROU
    lastx, lasty = x, y


def activate_event(event):
```

```
938/938 [==============================] - 49s 52ms/step - loss: 0.0201 - accuracy
Epoch 12/15
938/938 [==============================] - 49s 52ms/step - loss: 0.0176 - accuracy: 0.9947
Epoch 13/15
938/938 [==============================] - 48s 52ms/step - loss: 0.0171 - accuracy: 0.9955
Epoch 14/15
938/938 [==============================] - 47s 50ms/step - loss: 0.0162 - accuracy: 0.9952
Epoch 15/15
938/938 [==============================] - 48s 52ms/step - loss: 0.0150 - accuracy: 0.9957
313/313 [==============================] - 3s 8ms/step - loss: 0.0312 - accuracy: 0.9913
0.9912999868392944
/home/sanju/.local/lib/python3.10/site-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is
tead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
sanju@thinkpad:~/Desktop/APP Project/Multiple-Handwritten-Digit-Recognition$ python -u "/home/sanju/Desktop/APP Project/Multiple-Handwritten-Digit-Recognition/final_gui.py"
2023-11-10 11:06:57.387651: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda drivers on your machine, GPU will not be used.
2023-11-10 11:06:57.499164: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda drivers on your machine, GPU will not be used.
2023-11-10 11:06:57.499694: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical ope
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

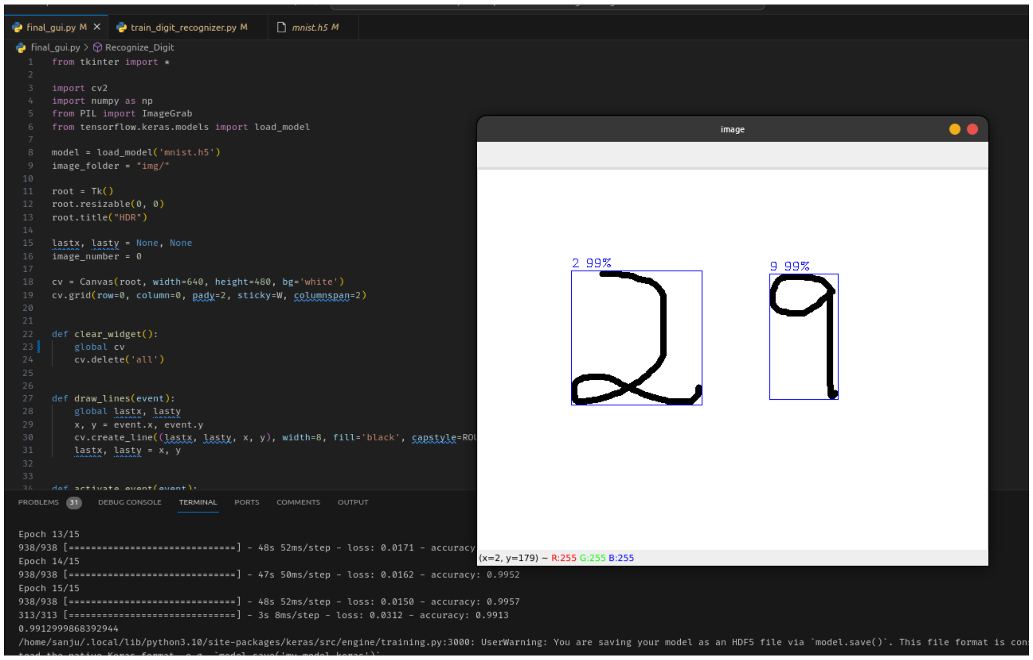
d. Draw numbers with your mouse



```
from tkinter import *

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

model = load_model('mnist.h5')
image_folder = "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=ROU
    lastx, lasty = x, y


def activate_event(event):
```

```
938/938 [==============================] - 49s 52ms/step - loss: 0.0201 - accuracy
Epoch 12/15
938/938 [==============================] - 49s 52ms/step - loss: 0.0176 - accuracy: 0.9947
Epoch 13/15
938/938 [==============================] - 48s 52ms/step - loss: 0.0171 - accuracy: 0.9955
Epoch 14/15
938/938 [==============================] - 47s 50ms/step - loss: 0.0162 - accuracy: 0.9952
Epoch 15/15
938/938 [==============================] - 48s 52ms/step - loss: 0.0150 - accuracy: 0.9957
313/313 [==============================] - 3s 8ms/step - loss: 0.0312 - accuracy: 0.9913
0.9912999868392944
/home/sanju/.local/lib/python3.10/site-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is consid
```

e. The program takes the screenshot and feeds it into the model



```
from tkinter import *

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

model = load_model('mnist.h5')
image_folder = "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=ROU
    lastx, lasty = x, y


def activate_event(event):
```

```
Epoch 13/15
938/938 [==============================] - 48s 52ms/step - loss: 0.0171 - accuracy
Epoch 14/15
938/938 [==============================] - 47s 50ms/step - loss: 0.0162 - accuracy: 0.9952
Epoch 15/15
938/938 [==============================] - 48s 52ms/step - loss: 0.0150 - accuracy: 0.9957
313/313 [==============================] - 3s 8ms/step - loss: 0.0312 - accuracy: 0.9913
0.9912999868392944
/home/sanju/.local/lib/python3.10/site-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is con
```

8.CONCLUSION AND FUTURE WORK

8.1 Conclusion:

Our project HANDWRITTEN DIGIT RECOGNITION deals with identifying the digits. The main purpose of this project is to build an automatic handwritten digit recognition method for the recognition of handwritten digit strings.

In this project, different machine learning methods, which are SVM (Support Vector Machine), ANN(Artificial Neural Networks), and CNN (Convolutional Neural Networks) architectures are used to achieve high performance on the digit string recognition problem.

8.2 Future Work :

- **Handwriting Styles Recognition:** Extend the system to recognize various handwriting styles, enabling personalized recognition and text analysis.
- Alphanumeric Character Recognition: Expand its scope to recognize letters and alphanumeric characters for comprehensive text processing.
- Offline Handwriting Recognition: Develop offline capabilities for recognizing handwriting without the need for a network connection.
- Deep Learning Advancements: Incorporate state-of-the-art deep learning techniques and pre-trained models for improved accuracy and efficiency.
- Mobile and Web Integration: Create mobile and web applications for on-the-go digit recognition and seamless integration into digital platforms.
- Real-Time Recognition: Implement real-time recognition for applications like form processing, making it more interactive and efficient.
- Improved User Interface: Enhance the user interface for better user experience, providing real-time feedback and interactive features.
- Multilingual Support: Enable recognition of multiple languages, making it versatile for global users.
- Accessibility Features: Develop features for visually impaired users, promoting accessibility and inclusivity.
- Cloud Integration: Explore cloud-based solutions for scalability, improved performance, and data management.

10. REFERENCES

1. https://www.tensorflow.org/learn
2. https://www.geeksforgeeks.org/python-tkinter-tutorial/
3. https://medium.com/the-andela-way/applying-machine-learning-to-recognize handwritten-
characters-babcd4b8d705
4. https://nanonets.com/blog/handwritten-character-recognition/
5. https://www.geeksforgeeks.org/python-tkinter-tutorial/
6. https://medium.com/the-andela-way/applying-machine-learning-to-recognize handwritten-
characters-babcd4b8d705
7. https://nanonets.com/blog/handwritten-character-recognition/
8. https://www.tensorflow.org/learn
9. R. Alhajj and A. Elnagar, "Multiagents to separating handwritten connected digits," in IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, vol. 35, no. 5, pp. 593-602, Sept. 2005. https://doi.org/10.1109/TSMCA.2005.843389
10.https://towardsdatascience.com/useful-plots-to-diagnose-your-neural-network-521907fa2f45
11.https://towardsdatascience.com/the-best-machine-learning-algorithm-for-handwritten-digits-recognition-2c6089ad8f09
12. https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python
13. https://www.analyticsvidhya.com/blog/2021/01/building-a-cnn-model-with-95-accuracy/
14.https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/
15. https://www.kdnuggets.com/2018/11/top-python-deep-learning-libraries.html
16. https://analyticsindiamag.com/7-types-classification-algorithms/
17.https://towardsdatascience.com/image-recognition-with-machine-learning-on-python-image-processing-3abe6b158e9a