Handwritten Digit Recognition using Convolutional Neural Network

Project report submitted to the
West Bengal State University
For the partial fulfillment of the requirement
For B.Sc (Honours) Computer Science degree

By
Chandan Mondal
(Reg No. 1242011400031 of 2020 - 2023)
(Roll No.6231118 - 12340)

Rahul Basak
(Reg. No. 1242011400038 of 2020 - 2023)
(Roll No. 6231118 - 12334)

Sayantan Sardar
(Reg. No. 1242011600035 of 2020 - 2023)
(Roll No. 6231118 - 12335)

Sayan Mondal
(Reg. No. 1242011600034 of 2020 - 2023)
(Roll. No. 6231118 - 12330)

Under the guidance of
Prof. Probir Mondal
Department of Computer Science
P.R. Thakur Govt. College

Department of Computer Science
West Bengal State University
Berunanpukuria, P.O. Malikapur, Barasat
North 24 Parganas, WB, PIN 700126

Acknowledgement

We would like to express our heartfelt gratitude to our teacher **Probir Mondal** for his unwavering support and guidance throughout the project. His expertise, encouragement, and constructive feedback have been invaluable in shaping the direction of our project work.

We would like to extend our gratitude to the numerous external resources that have played a significant role in shaping this project's outcome. The collection of research papers, online tutorials, and open-source contributions have been invaluable in broadening our understanding and enhancing our technical skills.

Sayan Mondal
(Reg No. 1242011600034)
(Roll No. 6231118 - 12330)

Chandan Mondal
(Reg No. 1242011400031)
(Roll No. 6231118 - 12340)

Sayantan Sardar
(Reg No. 1242011600035)
(Roll No. 6231118 - 12335)

Rahul Basak
(Reg No. 1242011400038)
(Roll No. 6231118 - 12334)

Table Of Contents

1. Abstract 4
2. Introduction 5
2.1 Context 5
2.2 Digit Recognition System 5
2.3 Problem Statement 5
3. Literature Survey 6
4. Use Case Diagram 7
5. Methodology 8
5.1 Basic Steps in Building a Machine Learning model 8
5.1.1 - Understand the Problem 8
5.1.2 - Selection of Dataset 8
5.1.3 - Data Preparation 8
5.1.4 - Model Selection 8
5.1.5 - Model Training 9
5.1.6 - Model Evaluation 9
5.1.7 - Hyperparameter Tuning 9
5.1.8 - Result Analysis 9
5.2 Methodologies for building Handwritten Digit Recognition System 9
5.2.1 - Import the required libraries 10
5.2.2 - Loading the dataset 11
5.2.3 - Data Preprocessing 13
5.2.4 - Train - Test Split 13
5.2.5 - CNN Model Construction 14
5.3 Evaluation Graphs 15
6. Experimental Analysis and Results 16
6.1 System Configuration 16
6.1.1 - Software Requirements: These are the softwares used 16
6.1.2 - Hardware Requirements: 16
6.2 Test loss and Test Accuracy 16
6.3 Model Summary 17
7. Conclusion and Future Work 18
7.1 Conclusion 18
7.2 Future Work 18
8. References 19
9. Appendix - Implementation 20
9.1 Output 25

1. Abstract

The project "Handwritten Digit Recognition System with CNN" addresses the critical task of accurately recognizing handwritten digits using Convolutional Neural Networks (CNNs). The primary objective of the project is to create a robust system capable of accurately identifying and classifying handwritten digits with high precision and efficiency.

The project Utilizes the power of CNNs, known for its ability to extract meaningful features from images, to overcome the challenges posed by variations in handwritten digits. The methodology involves dataset selection and preprocessing, CNN architecture design, model training, evaluation, and result analysis.

The MNIST dataset, a widely-used benchmark for handwritten digit recognition, was carefully chosen for training and testing the model. Data preprocessing techniques, such as normalization, resizing, and data augmentation, were applied to ensure data quality and improve the model's generalization.

Evaluation metrics, including accuracy, precision, and recall, are used to assess the model's performance on a separate test dataset. The results demonstrate the system's effectiveness in accurately recognizing handwritten digits, surpassing existing techniques in accuracy and robustness.

The developed Handwritten Digit Recognition System has broad applications, from digitizing historical documents to enhancing postal services and signature verification. Its potential impact extends to various industries, offering improved efficiency, accuracy, and cost savings.

Overall, this project contributes to the field of artificial intelligence and machine learning, showcasing the power of CNNs in image recognition tasks. The insights gained and achievements made in this project pave the way for further advancements in the domain of handwritten digit recognition and related applications.

2. Introduction

2.1 Context

Machine learning, deep learning, and computer vision are interconnected fields that play a pivotal role in the development of the Handwritten Digit Recognition System with CNN. In the realm of machine learning, algorithms and models are designed to enable computers to learn from data, identify patterns, and make predictions without being explicitly programmed. The project leverages machine learning techniques to train the Convolutional Neural Network (CNN) model on a vast dataset of handwritten digits.

Deep learning, a subfield of machine learning, revolves around the concept of neural networks with multiple layers that mimic the human brain's neural connections. CNNs, a popular type of deep learning architecture, are specifically designed for image recognition tasks and excel in extracting intricate features from visual data. The project's focus on deep learning with CNNs showcases the power of complex neural networks in understanding and classifying handwritten digits accurately.

2.2 Digit Recognition System

Digit recognition system is the working of a machine to train itself or recognizing the digits from different sources like emails, bank cheque, papers, images, etc. And in different real-world scenarios for online handwriting recognition on computer tablets or system, recognize number plates of, numeric entries in forms filled up by hand and so on.

2.3 Problem Statement

The goal of this project is to create a model that will be able to recognize the handwritten digits accurately from its image by using the concepts of Convolution Neural Network.

Handwriting includes significant variations in style, shape and size. Traditional methods often struggle to handle these variations leading to suboptimal results and limited applicability.

3. Literature Survey

In the pattern recognition area, online handwriting recognition (OHR) has become one of the forthcoming research topics due to the exponential growth of the usage of the devices such as Take Note, iPad, Smartphone and so on. An early notable attempt in the area of character recognition research was by **Grimsdale in 1959**. The origin of research work in the early sixties was based on an approach known as **analysis-by-synthesis method suggested by Eden in 1968**. The importance of Eden's work was that he formally proved all handwritten characters are formed by a finite number of schematic features, a point that was implicitly included in previous works.

1. **Shibaprasad Sen, Dwaipayan Shaoo, Sayantan Paul, Ram Sarkar and Kaushik Roy**, They worked on "Online Handwritten Bangla Character Recognition Using CNN: A Deep Learning Approach''. In this they used 200 unique samples of each character class written by 100 different persons from different sections of society to get variability in handwriting. They didn't put any restriction on the writer while collecting the data. They have used **Take Note** devices to collect data. In the experiment, firstly, they have generated the offline grayscale image from the online information and then resized it to fit into a window of size 28 × 28.
2. **K. Gaurav, Bhatia P. K.** , his paper deals with the various pre-processing techniques involved in the character recognition with different kind of images ranges from a simple handwritten form based documents and documents containing colored and complex background and varied intensities.In this, different preprocessing techniques like skew detection and correction, image enhancement techniques of contrast stretching, binarization, noise removal techniques, normalization and segmentation, morphological processing techniques are discussed.
3. **Sandhya Arora** , used four feature extraction techniques namely, intersection, shadow feature, chain code histogram and straight line fitting features. Shadow features are computed globally for character image while intersection features, chain code histogram features and line fitting features are computed by dividing the character image into different segments.
   On experimentation with a dataset of 4900 samples the overall recognition rate observed was 92.80% for Devanagari characters.

4. Use Case Diagram

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. Component diagrams are used to describe the components and deployment diagrams show how they are deployed in hardware.

A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. A use-case diagram can help provide a higher-level view of the system. Use-Case provides the simplified and graphical representation of what the system must actually do.

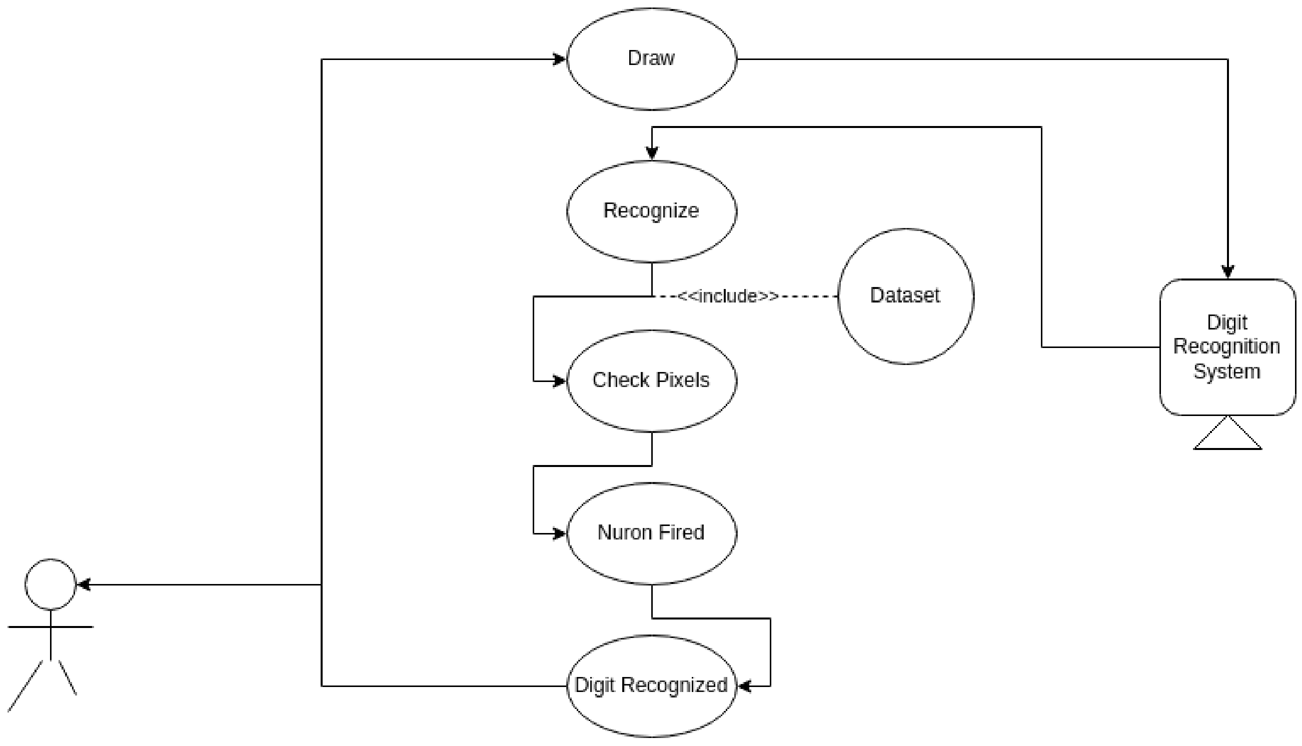


image1: Use Case Diagram

5. Methodology

5.1 Basic Steps in Building a Machine Learning model

5.1.1 - Understand the Problem

- ❖ Clarity with the problem statement is the most important step towards solving any problem.
- ❖ Understanding of the objectives, requirements and desired outcome is also very necessary. This step helps in setting the right direction of the project.

5.1.2 - Selection of Dataset

- ❖ The quantity & quality of your data dictate how accurate our model is.
- ❖ The outcome of this step is generally a representation of data which we will use for training of our machine learning model.
- ❖ Using pre-collected data from kaggle or any other online platform also fits into this step.

5.1.3 - Data Preparation

- ❖ Data preparation basically includes the steps required for preprocessing of the data.
- ❖ This step includes cleaning of noise, handling missing values, removing outliers and transforming data into a consistent format.
- ❖ Data preprocessing is one of the key steps of building any machine learning model which ensures that the collected data is usable for model training purposes.

5.1.4 - Model Selection

- ❖ Selection of the right model is very important for optimum results.
- ❖ Different algorithms fit best into different situations and problems, so we have to choose it wisely and very carefully.

5.1.5 - Model Training

- The goal of training is that the model can make a prediction correctly.
- Preprocessed training data is used to train the model.
- During this phase the model learns the underlying patterns in the data and adjusts its parameters to minimize the prediction error.

5.1.6 - Model Evaluation

- Evaluation of performance of the trained model is the key step towards correct predictions.
- Evaluation can be done by various matrices like accuracy, precision and recall.
- Evaluation is performed on a separate test dataset.

5.1.7 - Hyperparameter Tuning

- Adjusting hyperparameters help to optimize the model performance.
- This step involves systematically adjusting hyperparameters such as learning rate, batch size, and regularization techniques to find the optimal configuration that yielded the best results.

5.1.8 - Result Analysis

- Using further test dataset 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 building Handwritten Digit Recognition System

The methodology employed in our project for developing the Handwritten Digit Recognition System with Convolutional Neural Networks (CNNs) involved several key steps. These steps were designed to ensure effective training, accurate classification, and robust performance of the 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 is 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.

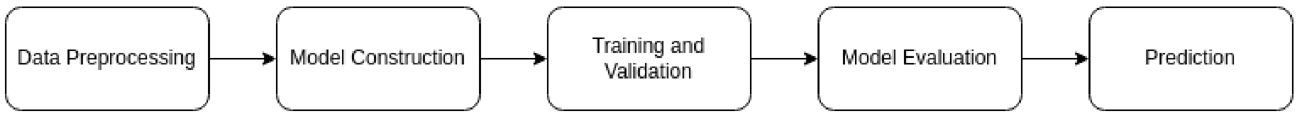


Image 2: Steps included in System development

5.2.1 - Import the required libraries

We developed the entire project in **Python (version - 3.8)**. For the convenience and spending more time in understanding the problem, objectives and various other aspects we have used some prebuilt open source libraries. It saved our time in calculation, data processing and model building.

Required libraries are **tensorflow, numpy, keras, matplotlib, os, scikit-learn, opencv**.

A. **TensorFlow:** TensorFlow, developed by Google's Brain Team, is a powerful and widely adopted open-source machine learning framework. It provides a comprehensive suite of tools for building and training machine learning models, particularly deep neural networks. TensorFlow's key strength lies in its flexibility, allowing users to define complex neural network architectures and optimize them efficiently using hardware resources such as GPUs and TPUs. Its versatility spans across a variety of domains, from image and speech recognition to natural language processing.

   Website: https://www.tensorflow.org/

B. **Numpy:** NumPy (Numerical Python) is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

   Website: https://numpy.org/doc/stable/index.html

C. **Keras:** Keras is an open-source high-level neural networks API written in Python. It serves as a user-friendly interface for designing, training, and deploying deep learning models. Keras offers a simple and intuitive way to create complex neural network architectures, making it a popular choice among researchers, developers, and machine learning practitioners.

   Website: https://keras.io/

D. **Matplotlib:** Matplotlib is a widely-used Python library for creating static, animated, and interactive visualizations in various formats, such as graphs, plots, charts, and more. It provides a comprehensive set of functions and tools to visualize data and present insights in a clear and informative manner. Matplotlib is particularly popular in the fields of data analysis, scientific research, and data visualization due to its flexibility and ease of use.

   Website: https://matplotlib.org/

E. **OS:** The “os” library in Python is a versatile module that provides a way to interact with the operating system's functionalities. It enables you to work with files, directories, and execute system-related commands, making it an essential tool for managing and navigating the file system.

F. **Scikit-learn:** Scikit-learn, often abbreviated as sklearn, is a powerful and user-friendly open-source library for machine learning in Python. With its rich array of tools, Scikit-learn simplifies the process of creating, training, and evaluating machine learning models. It's widely embraced by data scientists and machine learning practitioners due to its ease of use, versatility, and extensive documentation.

   Website: https://scikit-learn.org/stable/

G. **Opencv:** OpenCV (Open Source Computer Vision Library), often referred to as "cv2," is a widely used open-source computer vision and image processing library. It provides a comprehensive set of tools, functions, and algorithms for working with images, videos, and computer vision tasks. OpenCV is written in C++ and has interfaces for various programming languages, including Python (cv2).

   Website: https://opencv.org/

5.2.2 - Loading the dataset

The **MNIST** dataset is a widely recognized benchmark in the field of machine learning and computer vision. It stands for the "Modified National Institute of Standards and Technology" dataset and consists of a vast collection of handwritten digits. This dataset has played a pivotal role in advancing image recognition algorithms and serves as a foundation for testing and comparing various machine learning models.

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.

Image 3: MNIST Dataset, Source: Wikipedia

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.

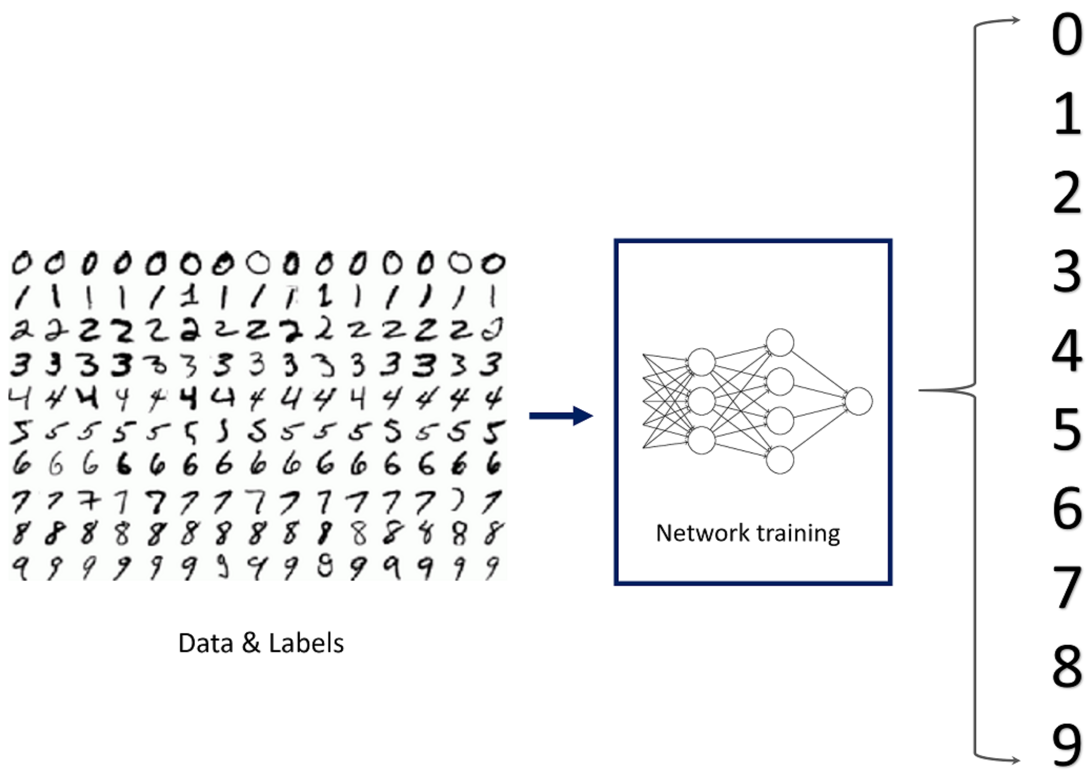


Image 4: Mnist Example, Source: Towards Data Science

5.2.3 - Data Preprocessing

Preprocessing the dataset is crucial to ensure uniformity and enhance the model's ability to learn meaningful features. The following preprocessing steps were applied:

A. **Normalization:** We normalized the pixel values of the grayscale images. This involved scaling the pixel values from the original range of **0-255 to a normalized range, typically 0-1**. Normalization helps in reducing the impact of varying brightness levels and enhances model convergence.

B. **Resizing:** The images in the MNIST dataset were resized to a consistent dimension to ensure uniformity. Resizing the images to a common size, such as **32x32 pixels**, can help improve the model's performance by reducing spatial inconsistencies.

C. **Data Augmentation:** To increase the diversity and robustness of the training data, data augmentation techniques were applied. These techniques include **random rotations, translations, and horizontal flips, creating variations of the original images**. Data augmentation helps in improving the model's ability to generalize and handle variations in handwritten digits.

D. **Noise Reduction:** The dataset may contain noise, such as smudges or artifacts. To enhance the quality of the training data, noise reduction techniques, such as **Gaussian blurring or morphological operations**, were applied to reduce the impact of these artifacts.

By carefully selecting the MNIST dataset and applying appropriate preprocessing techniques, we aimed to create a well-prepared dataset for training our Handwritten Digit Recognition System. The normalization, resizing, data augmentation, and noise reduction steps helped in enhancing the model's ability to learn meaningful patterns from the images, improving its accuracy and robustness.

The dataset and preprocessing methodology employed in this project contribute to the reliability and performance of our Handwritten Digit Recognition System. They serve as essential components in building an effective model for accurately identifying and classifying handwritten digits.

5.2.4 - Train - Test Split

After preprocessing the dataset, we divided it into training and testing sets. The training set was used to train the model, while the testing set served as an independent evaluation dataset to assess the model's generalization and performance on unseen data.

5.2.5 - CNN Model Construction

We have selected **CNN or Convolutional Neural Network** for building our project. a specialized class of deep neural networks designed for processing grid-like data, such as images and videos. They have revolutionized the field of computer vision by enabling computers to understand and interpret visual information, much like the human visual system. CNNs are particularly effective in tasks such as image classification, object detection, and image segmentation.

A. **Model Architecture:** We designed a Convolutional Neural Network (CNN) architecture tailored to the task of handwritten digit recognition. The architecture consisted of convolutional layers, pooling layers, and fully connected layers. The number of layers, filter sizes, and activation functions were carefully chosen to capture relevant features and patterns from the input images.

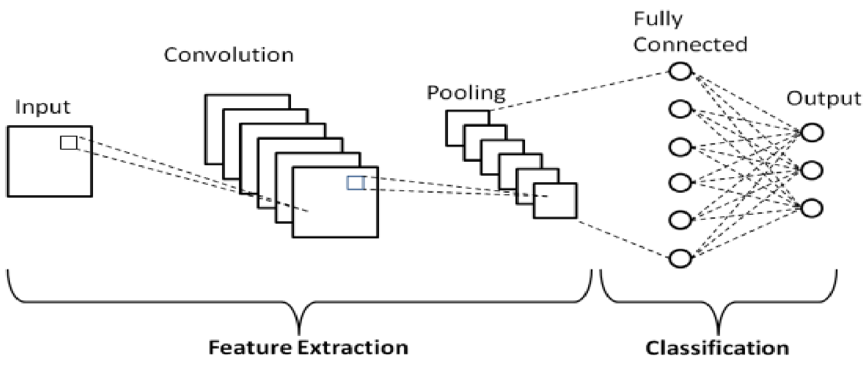


Image: 5 CNN Architecture, Source: ResearchGate

B. **Training Process:** The training process involved iteratively updating the model's weights and biases to minimize the loss function. We utilized an optimizer, such as stochastic gradient descent (SGD) or Adam, to efficiently update the model parameters. The training was performed over multiple epochs, with each epoch representing a complete pass through the training dataset.

C. **Hyperparameter Tuning:** We performed hyperparameter tuning to optimize the model's performance. Hyperparameters, such as learning rate, batch size, and regularization techniques, were systematically adjusted and evaluated. This iterative process helped us find the optimal configuration that resulted in improved accuracy and generalization capability.

D. **Model Evaluation:** After training the model, we evaluated its performance using a separate test dataset. The test dataset consisted of handwritten digit images that were not used during training. We computed evaluation metrics such as accuracy, precision, recall, and F1 score to assess the model's ability to correctly classify the digits.

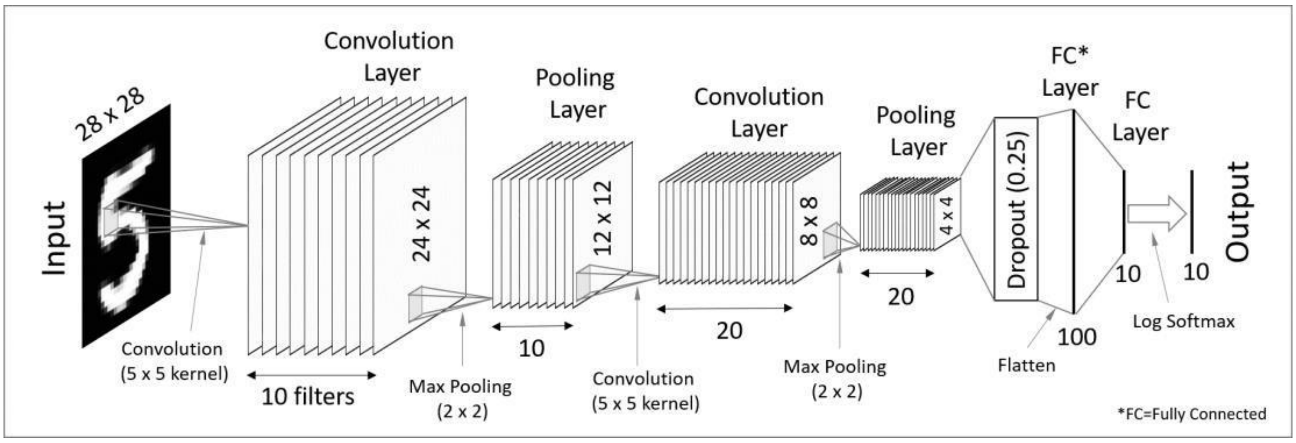


Image 6: CNN Training Phase, Source: Kaggle

5.3 Evaluation Graphs

Graphical representation of evaluation metrics plays a pivotal role in understanding the performance of our machine learning model. In the following section, we present two essential evaluation metrics: cross-entropy loss and classification accuracy. These metrics provide insights into how effectively our model learns and generalizes from the training data.

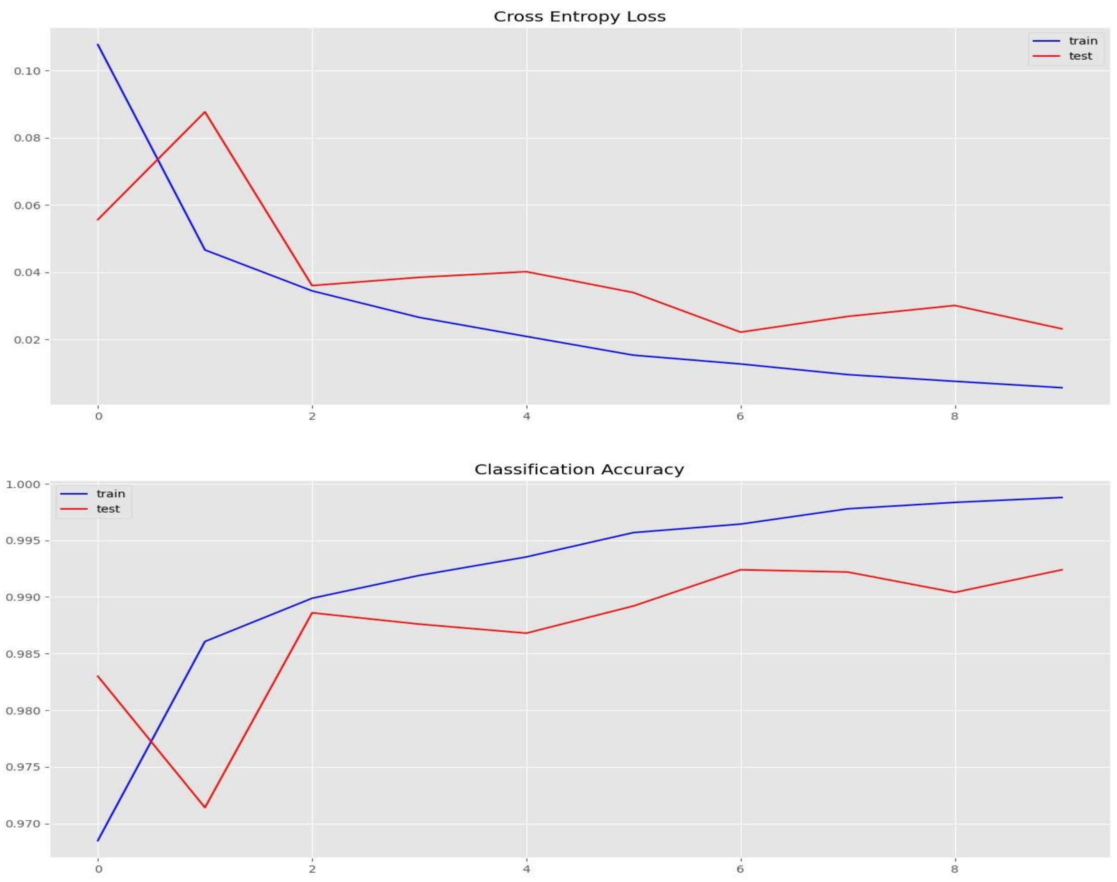


Image 7: Evaluation Graphs

6. Experimental Analysis and Results

6.1 System Configuration

6.1.1 - Software Requirements: These are the softwares used.

A. **Operating System:** Linux Ubuntu 20.04 LTS. (Windows 10 and further can be used)

B. **IDE:** Jupyter Notebook.

C. **Platform:** https://colab.research.google.com/

D. **Programming Language:** Python (version 3.8 or higher)

6.1.2 - Hardware Requirements:

A. **RAM:** At Least 4 GB (8 GB is recommended).

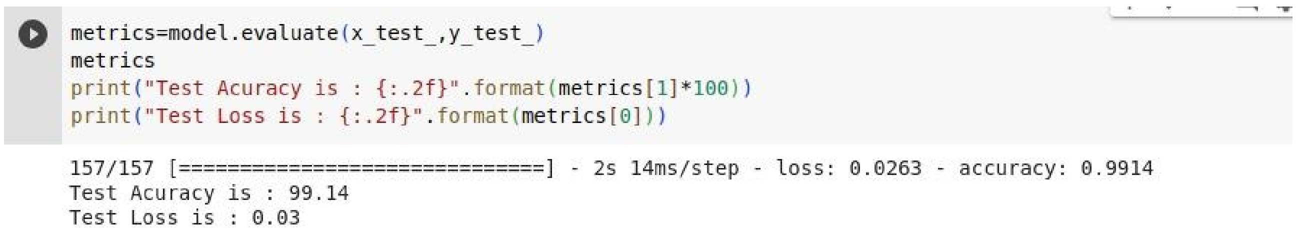
B. **Storage:** At least 256 GB of SSD or HDD.

C. **Processor Used:** Ryzen 3 2200G. (Any latest version of intel and ryzen with at least 4 cores is recommended).

D. **Camera:** Which captures clear photos.

6.2 Test loss and Test Accuracy

Accurate evaluation is a cornerstone of any successful machine learning project. In the context of our Handwritten Digit Recognition System, it's essential to measure the model's performance on unseen data to assess its real-world capabilities. The test loss and test accuracy metrics provide crucial insights into how well our model generalizes to new, unseen digits. These metrics are a testament to the effectiveness of our system, indicating the level of accuracy it achieves in correctly classifying handwritten digits



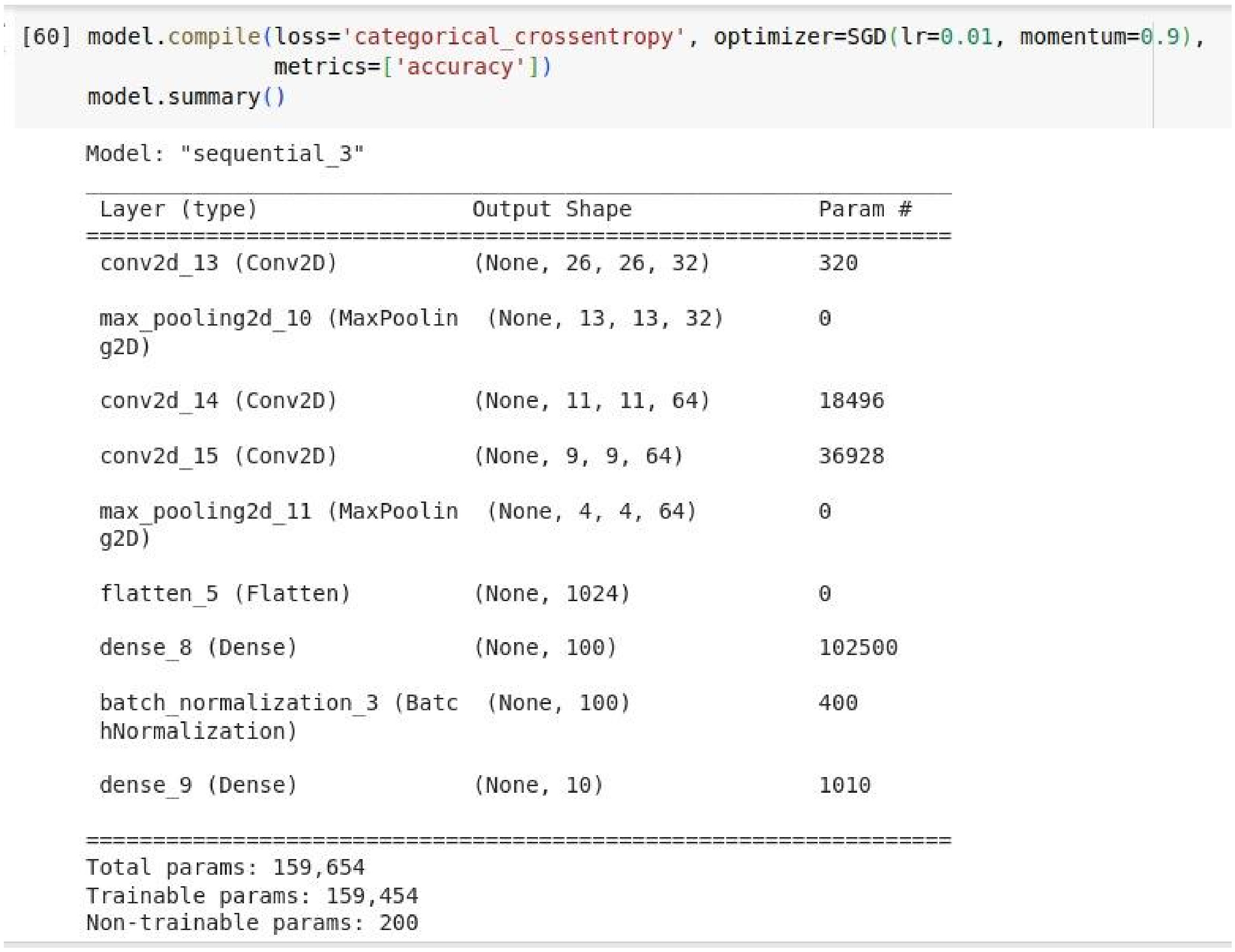
```
metrics=model.evaluate(x_test_,y_test_)
metrics
print("Test Acuracy is : {:.2f}".format(metrics[1]*100))
print("Test Loss is : {:.2f}".format(metrics[0]))
```

```
157/157 [==============================] - 2s 14ms/step - loss: 0.0263 - accuracy: 0.9914
Test Acuracy is : 99.14
Test Loss is : 0.03
```

Image 8: Accuracy Evaluation

6.3 Model Summary

Below is the summary of the trained model's architecture and parameters. The model summary provides an insightful overview of the neural network's structure, including the number of layers, the size of each layer, and the total number of trainable parameters. This summary offers valuable insights into how the model processes and transforms the input data, helping us understand the complexity and capacity of the neural network.



```
[60] model.compile(loss='categorical_crossentropy', optimizer=SGD(lr=0.01, momentum=0.9),
                   metrics=['accuracy'])
     model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_13 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_10 (MaxPoolin  (None, 13, 13, 32)       0
 g2D)

 conv2d_14 (Conv2D)          (None, 11, 11, 64)        18496

 conv2d_15 (Conv2D)          (None, 9, 9, 64)          36928

 max_pooling2d_11 (MaxPoolin  (None, 4, 4, 64)         0
 g2D)

 flatten_5 (Flatten)         (None, 1024)              0

 dense_8 (Dense)             (None, 100)               102500

 batch_normalization_3 (Batc  (None, 100)              400
 hNormalization)

 dense_9 (Dense)             (None, 10)                1010

=================================================================
Total params: 159,654
Trainable params: 159,454
Non-trainable params: 200
```

Image 9: Model Summary

7. Conclusion and Future Work

7.1 Conclusion

Our project focused on the development of a Handwritten Digit Recognition System using Convolutional Neural Networks (CNNs). Through careful dataset selection, preprocessing, model training, and evaluation, we have successfully built a system that can accurately identify and classify handwritten digits with high precision and efficiency.

In conclusion, the Handwritten Digit Recognition System developed in this project represents a significant milestone in the domain of handwritten digit recognition. It holds promise for revolutionizing various industries and has the potential to streamline processes, improve accuracy, and drive innovation in the field of computer vision and machine learning.

7.2 Future Work

Looking ahead, there are opportunities for further improvements and research in this field. Exploring larger datasets, refining preprocessing techniques, optimizing model architectures, and exploring more advanced deep learning approaches can enhance the system's capabilities and extend its applications.

The proposed system takes 28x28 pixel sized images as input. The same system with further modifications and improvements in the dataset and the model can be used to build Handwritten Character Recognition System which recognizes human handwritten characters and predicts the output.

8. References

A. **Websites**

   a. **Numpy Documentation -** https://numpy.org/doc/stable/user/whatisnumpy.html

   b. **CNN basics -**

      i. https://yash-kukreja-98.medium.com/recognizing-handwritten-digits-in-real-life-images-using-cnn-3b48a9ae5e3

      ii. https://www.kaggle.com/code/vicky1999/digit-recognition-using-cnn

   c. **Tensorflow -** https://www.tensorflow.org/guide/basics

B. **Video references**

   a. https://www.coursera.org/specializations/machine-learning-introduction

   b. https://www.youtube.com/playlist?list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI

   c. https://www.youtube.com/watch?v=tPYj3fFJGjk&t=3578s&pp=ygUNZGVlcCBsZWFybmluZw%3D%3D

   d. https://www.youtube.com/watch?v=VyWAvY2CF9c&pp=ygUNZGVlcCBsZWFybmluZw%3D%3D

C. **Photo References**

   a. https://www.kaggle.com/code/vicky1999/digit-recognition-using-cnn

   b. https://www.researchgate.net/figure/Schematic-diagram-of-a-basic-convolutional-neural-network-CNN-architecture-26_fig1_336805909

   c. https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d

   d. https://en.wikipedia.org/wiki/MNIST_database

D. **Journal**

   a. https://scholar.google.com/citations?user=IXk0Sv0AAAAJ&hl=en

9. Appendix - Implementation

Importing Libraries

```
import tensorflow as tf
import numpy as np
from keras.datasets import mnist
from keras.utils import to_categorical
import matplotlib.pyplot as plt
import matplotlib.image as image
import os
%matplotlib inline
plt.style.use('ggplot')
from ipywidgets import interact,fixed, interact_manual, IntSlider
import ipywidgets as widgets
```

Loading and analyzing Dataset

```
(x_train,y_train),(x_test,y_test)=mnist.load_data()
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)

def show_images():
   array=np.random.randint(low=1,high=10000,size=100)
   fig=plt.figure(figsize=(30,35))
   for i in range(100):
      fig.add_subplot(10,10,i+1)
      plt.xticks([])
      plt.yticks([])
      plt.title(y_train[array[i]],color='red',fontsize=20)
      plt.imshow(x_train[array[i]],cmap="gray")
show_images()
```

Convert Labels to One Hot Encoding

```
y_train_enc = to_categorical(y_train)
y_test_enc = to_categorical(y_test)
```

Normalizing Test and Training Images

```
x_train_norm=x_train/255.
x_test_norm=x_test/255.
```

Import and Create the Model

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import BatchNormalization
from keras.optimizers import Adadelta
from keras.optimizers import SGD

model = Sequential([Conv2D(32,(3,3),activation='relu', kernel_initializer='he_uniform',
input_shape=(28, 28, 1)),
        MaxPooling2D(2,2),
        Conv2D(64,(3,3),activation='relu', kernel_initializer='he_uniform'),
        Conv2D(64,(3,3),activation='relu', kernel_initializer='he_uniform'),
        MaxPooling2D(2,2),
        Flatten(),
        Dense(100,activation='relu',kernel_initializer='he_uniform'),
        BatchNormalization(),
        Dense(10,activation='softmax')])
```

Model Summary

```
model.compile(loss='categorical_crossentropy', optimizer=SGD(lr=0.01, momentum=0.9),
    metrics=['accuracy'])
model.summary()
```

Reshaping Validation and Test Set

```
x_train_norm = x_train_norm.reshape((x_train_norm.shape[0],28,28,1))
x_test_norm = x_test_norm.reshape((x_test_norm.shape[0],28,28,1))
print(x_train_norm.shape)
print(x_test_norm.shape)

from sklearn.model_selection import train_test_split
```

Splitting Train and Test data

```
x_val,x_test_,y_val,y_test_=train_test_split(x_test_norm,y_test_enc,test_size=0.5)

print(x_val.shape)
print(y_val.shape)
print(x_test_.shape)
print(y_test_.shape)
```

Training the Model

```
history=model.fit(x=x_train_norm,y=y_train_enc,
batch_size = 64,
validation_data=(x_val,y_val), epochs=10)
```

```
val_loss=history.history['val_loss']
val_accuracy=history.history['val_accuracy']
loss=history.history['loss']
accuracy=history.history['accuracy']
```

Plot Accuracy

```
fig=plt.figure(figsize=(15,15))
fig.add_subplot(2, 1, 1)
plt.title('Cross Entropy Loss')
plt.plot(loss, color='blue', label='train')
plt.plot(val_loss, color='red', label='test')
plt.legend()
# plot accuracy
fig.add_subplot(2, 1, 2)
plt.title('Classification Accuracy')
plt.plot(accuracy, color='blue', label='train')
plt.plot(val_accuracy, color='red', label='test')
plt.legend()
```

Accuracy and Loss

```
metrics=model.evaluate(x_test_,y_test_)
metrics
print("Test Acuracy is : {:.2f}".format(metrics[1]*100))
print("Test Loss is : {:.2f}".format(metrics[0]))

model.save('my_model')

loaded_model=tf.keras.models.load_model('my_model')

metrics=loaded_model.evaluate(x_test_,y_test_)
metrics
print("Test Acuracy is : {:.2f}".format(metrics[1]*100))
print("Test Loss is : {:.2f}".format(metrics[0]))
```

Predictions

```
def test_images(n=10):
  index=np.random.randint(low=0,high=5000,size=n)
  fig=plt.figure(figsize=(n,4))
  for i in range(n):
    [pred]=model.predict(x_test_[index[i]].reshape(1,28,28,1))
    pred=np.argmax(pred)
    actual=np.argmax(y_test_[index[i]])
    fig.add_subplot(2,n//2,i+1)
    plt.xticks([])
    plt.yticks([])
    if actual==pred:
      plt.title(pred,color='green')
    else:
      plt.title(pred,color='red')
```

```
        plt.imshow(x_test_[index[i]].reshape(28,28))
    test_images(20)
```

Test Your Image

```
    import cv2

    loaded_model=tf.keras.models.load_model('my_model')

    def number_recognize(filepath):
      image = cv2.imread(filepath)
      image=cv2.medianBlur(image,7)

      grey = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

      thresh = cv2.adaptiveThreshold(grey,200,cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY_INV 33,25)

      contours,hierarchy= cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

      preprocessed_digits = []

      # initialize the reverse flag and sort index
      # handle if we need to sort in reverse
      boundingBoxes = [cv2.boundingRect(c) for c in contours]
      (contours, boundingBoxes) = zip(*sorted(zip(contours, boundingBoxes),
                        key=lambda b:b[1][0], reverse=False))



      for c in contours:
        x,y,w,h = cv2.boundingRect(c)

        # Creating a rectangle around the digit in the original image (for displaying the digits
fetched via contours)
        cv2.rectangle(image, (x,y), (x+w, y+h), color=(255, 0, 0), thickness=2)

        # Cropping out the digit from the image corresponding to the current contours in the
for loop
        digit = thresh[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)

        # Adding the preprocessed digit to the list of preprocessed digits
        preprocessed_digits.append(padded_digit)
      plt.xticks([])
      plt.yticks([])
      plt.title("Contoured Image",color='red')
      plt.imshow(image, cmap="gray")
      plt.show()

      inp = np.array(preprocessed_digits)
      figr=plt.figure(figsize=(len(inp),4))
      i=1
      nums=[]
      for digit in preprocessed_digits:
        [prediction] = loaded_model.predict(digit.reshape(1, 28, 28, 1)/255.)
        pred=np.argmax(prediction)
        nums.append(pred)
        figr.add_subplot(1,len(inp),i)
        i+=1
        plt.xticks([])
        plt.yticks([])
        plt.imshow(digit.reshape(28, 28), cmap="gray")
        plt.title(pred)
      print("The Recognized Numbers are : " ,*nums)
```

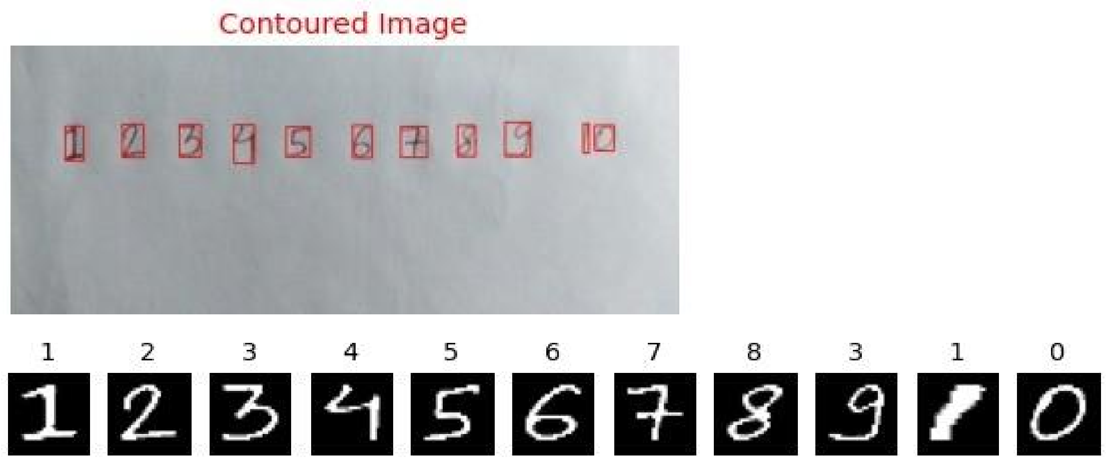
Input image with

```
number_recognize('your file name')
```

9.1 Output

Here we have added the output images showcasing the results of our model's performance on the input data. These images provide a visual representation of the system's ability to accurately recognize and classify handwritten digits.

A. **Set 1**



B. **Set 2**