

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to our supervisor **Er. Himal Chand Thapa** for his consistent help and support all through the project. His suggestion and direction helped us to adapt improvement of this project. We are thankful to every one of our respected teachers for their motivation and encouragement toward completing this project.

We are also thankful to the Department of Computer Science and Information Technology, Himalaya College of Engineering for providing these opportunities and environments to explore our skills and gain learning experience through this major project.

Additionally, we would like to extend our thanks to the respected teachers, who have provided us with their time and expertise and have made valuable contributions to the project. Their input and feedback have been instrumental in shaping the project and enhancing its quality.

Finally, we would like to acknowledge our friends and family, whose support and encouragement have been a constant source of motivation throughout the project.

Thanking you.

Ashok Kumar Shrestha(20938/075)

Bikesh Gamal(20942/075)

Hemanta Sunuwar(20948/075)

ABSTRACT

Recognizing handwritten digits is a challenging task due to variations in size, thickness, position, and orientation. Individual writing styles and compositional variations also affect the appearance and shape of digits. The recognition process involves identifying and organizing transcribed digits, and it has diverse applications such as automated bank checks, postal addresses, and tax documents. A classification algorithm is being developed to recognize handwritten digits to address this issue.

The implementation of a handwritten digit recognition system using Flutter provides a user-friendly interface for inputting and classifying handwritten digits. Users can draw a digit on the screen using their fingers, and a convolutional neural network (CNN) model predicts the corresponding digit in real-time.

The handwritten digit recognizer system utilizes Python to develop a CNN architecture for handwritten digit recognition, comprising multiple convolutional and pooling layers followed by fully connected layers. The MNIST dataset, consisting of 60,000 training images and 10,000 test images of handwritten digits, is used for training and testing the model. The model's performance is evaluated using accuracy, with the CNN model achieving 99% accuracy on the MNIST dataset.

Keywords: CNN, MNIST, Accuracy, Flutter, Python, TensorFlow, Keras, Classification.

Table of Content

ACKNOWLEDGEMENT	i
ABSTRACT.....	ii
List of Abbreviations	v
List of Figure.....	vi
List of Table.....	vii
Chapter-1: Introduction.....	1
1.1 Introduction	1
1.2 Problem Statement	1
1.3 Objective	1
1.4 Scope and Limitations.....	2
1.4.1 Scope.....	2
1.4.2 Limitations	2
1.5 Report Organization.....	2
Chapter 2: Literature Review	4
2.1 Literature Review.....	4
Chapter 3:Requirement Analysis	5
3.1 Requirement Analysis	5
i. Functional Requirement.....	5
ii. Non-functional Requirement	6
3.2 Feasibility Analysis.....	6
i. Technical feasibility:.....	6
ii. Operational feasibility:	6
iii. Economic feasibility:.....	6
iv. Schedule (Gantt chart showing the project timeline).....	7
Chapter 4: System Design.....	8
4.1 Design	8
4.1.1 Architectural Design	8
4.1.2 Class Diagram	9
4.1.3 Sequence Diagram	10
4.1.4 Activity Diagram	11
4.2 Dataset.....	12

4.3 Algorithm.....	13
Chapter 5:Implementation and Testing.....	18
5.1 System Implementation	18
5.1.1 Implementation Tools	18
5.2 Testing.....	19
Chapter 6: Conclusion and Future Recommendation	20
6.1 Conclusion	20
6.2 Future Recommendation.....	20
REFERENCES	21
APPENDICES	22
Snap shot.....	36

List of Abbreviations

CNN	Convolutional Neural Network
UI	User Interface
SVM	Support Vector Machines
MLP	Multi-Layer Perceptron
ReLU	Rectified Linear Unit

List of Figure

Figure 1.1: Report Organization of Handwritten Digit Recognition System	3
Figure 3.1: Use case diagram of handwritten digit recognition system.....	5
Figure 4.1: Architectural Diagram for Handwritten Digit Recognition	8
Figure 4.2: Class Diagram of Handwritten Digit Recognition	9
Figure 4.3: Sequence Diagram of Handwritten Digit Recognition.....	10
Figure 4.4: Activity diagram of Handwritten Digit Recognition.....	11
Figure 4.5: MNIST Dataset.....	12

List of Table

Table 3.1 Gantt chart for handwritten digit recognition	7
Table 5.1 Testing	19

Chapter-1: Introduction

1.1 Introduction

A handwritten digit recognition system is the recognition of handwritten digits using a convolutional neural network. Although humans find recognizing numbers easy, it is a difficult task for computers. The convolutional neural network is a powerful technique for automatic input classification in deep learning. It is effective in image classification for computer vision. This indicates that it is a reliable technique for end-to-end prediction.

The convolutional neural network automatically extracts useful features from input data. The network consists of three layers, namely the convolutional layer, the pooling layer, and the fully-connected layer. The convolutional layer extracts input features and passes them to the next layer in the form of a features map. The pooling layer reduces data dimensions by applying pooling on the features map to generate new feature maps with reduced dimensions. The fully-connected layer handles the classification task. The activation function calculates probability scores for each class label.

A handwritten digit recognition system offers two options to the user. The first option is to upload an image, and the second option is to draw the digit on the screen. The user can upload an image of a handwritten digit by clicking the image button, while they can draw a digit by clicking the draw button.

1.2 Problem Statement

The problem is to classify handwritten digits. The goal is to draw a digit on a mobile screen and determine which digit it is. Accuracy increases when there is a large number of datasets. There are thousands of handwritten pictures to train models as a part of this problem statement. Due to this reason, this project uses the MNIST dataset to train the model. This project will train the CNN model using TensorFlow to recognize handwritten digits. The challenge is to develop an algorithm or a model that can accurately identify the digits despite variations in writing styles, distortions, noise, and other factors that can affect the appearance of the handwritten digits.

1.3 Objective

The objective of this study is to recognize handwritten digits using a convolutional neural network that will recognize the digit from 0 to 9. Handwritten digit recognition can be used to develop a

computer program or algorithm that can recognize and classify handwritten digits accurately. The main objective is given below:

- To recognize digits from 0-9 drawn by the user and achieve correct results
- To recognize the image of a handwritten digit uploaded by the user.

1.4 Scope and Limitations

1.4.1 Scope

The scope of the handwritten digit recognition system is quite broad such as postal service, security, banking and finance, number plate recognition, education, etc. Handwritten digit recognition can also be extended to recognize digits on bank cheques. A handwritten digit recognition system can also be used to automatically sort mail based on the address including zip code which is represented by digits. It can also be used for signature verifications.

1.4.2 Limitations

The main problem with handwritten digit recognition is it can only recognize a single handwritten digit. People have different handwriting styles and it can be a challenge to build a system that can accurately recognize variations in handwriting. The size and shape of digits can also affect recognition accuracy. If the handwritten digits are poorly written they can be challenging recognition.

1.5 Report Organization

Chapter 1 consists of the introduction of the system, the problem statement of the system, the objectives of this project, and the scope and limitations of this project. Chapter 2 consists of a literature review for this project. Chapter 3 consists of requirement analysis, and feasibility analysis. Under requirement analysis there is a need to study functional requirements and non-functional requirements, under feasibility analysis, there is a need to study technical feasibility, operational feasibility, economical feasibility, and schedule. Chapter 4 consists of design and algorithm, under the design section there is a need to study the class diagram, sequence diagram, and activity diagram. Chapter 5 consists of implementation and testing. Chapter 6 includes the conclusion and future recommendation.

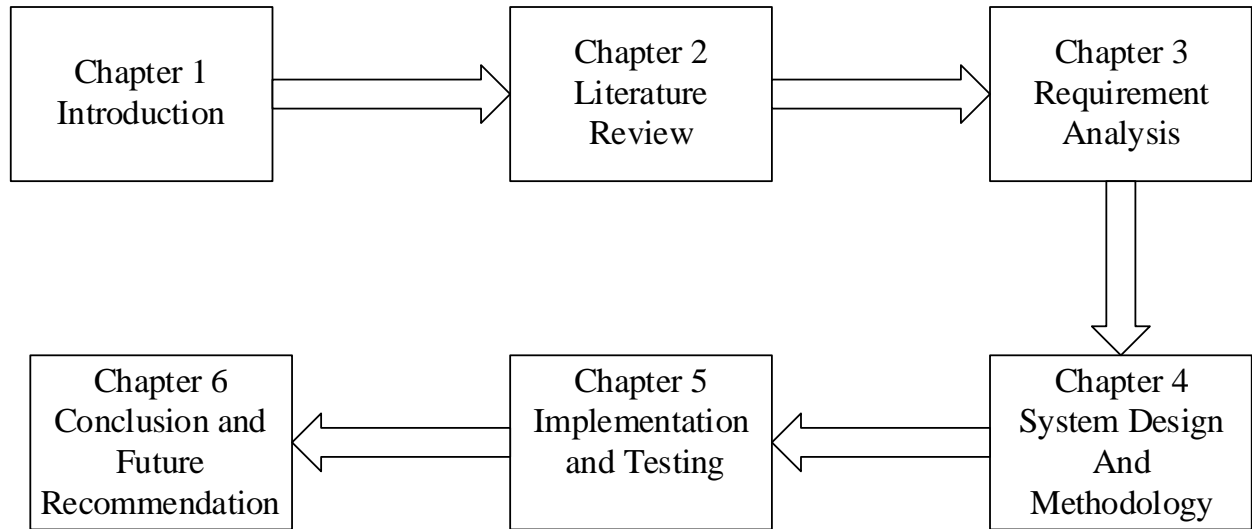


Figure 1.1: Report Organization of Handwritten Digit Recognition System

Chapter 2: Literature Review

2.1 Literature Review

The project on handwritten digit recognition is based on a convolutional neural network and draws inspiration from similar studies. In one such paper [1], the authors used Support Vector Machines (SVM), Multi-Layer Perceptron (MLP), and Convolution Neural Network (CNN) models to perform handwritten digit recognition using MNIST datasets. The goal was to compare the accuracy and execution time of these models to identify the best possible model for digit recognition. Handwriting recognition is an important area of research and development, and deep learning and machine learning algorithms have improved their reliability in classifying objects and performing complex tasks.

Another paper [2] also focused on implementing a classification algorithm for recognizing handwritten digits and compared the performance of SVM, KNN, RFC, and multilayer CNN algorithms. They achieved an accuracy of 98.70% using CNN (Keras + Theano) as compared to SVM, KNN, and RFC algorithms.

A third paper [3] focused on recognizing numbers, characters, and text from handwriting and converting them into visual characters. They compared the performance of various machine learning algorithms in handwriting recognition applications and found that the accuracy of an artificial neural network was 98.66%, a convolutional neural network was 99.45%, K-NN was 97.05%, Naive Bayes was 83.57%, support vector machine was 97.71%, and decision tree was 88.34%. They also developed a handwriting recognition system for numbers.

Chapter 3:Requirement Analysis

3.1 Requirement Analysis

The system "Handwritten Digit Recognition System" has been developed by the team and the requirement for the system is generic.

i. Functional Requirement

The functional requirement of the “Handwritten Digit Recognition System” are:

- Users should be able to draw digits, and the system should predict the respective digit.
- Users should be able to upload an image of a handwritten digit, and the system should predict the respective digit.

Use case diagram

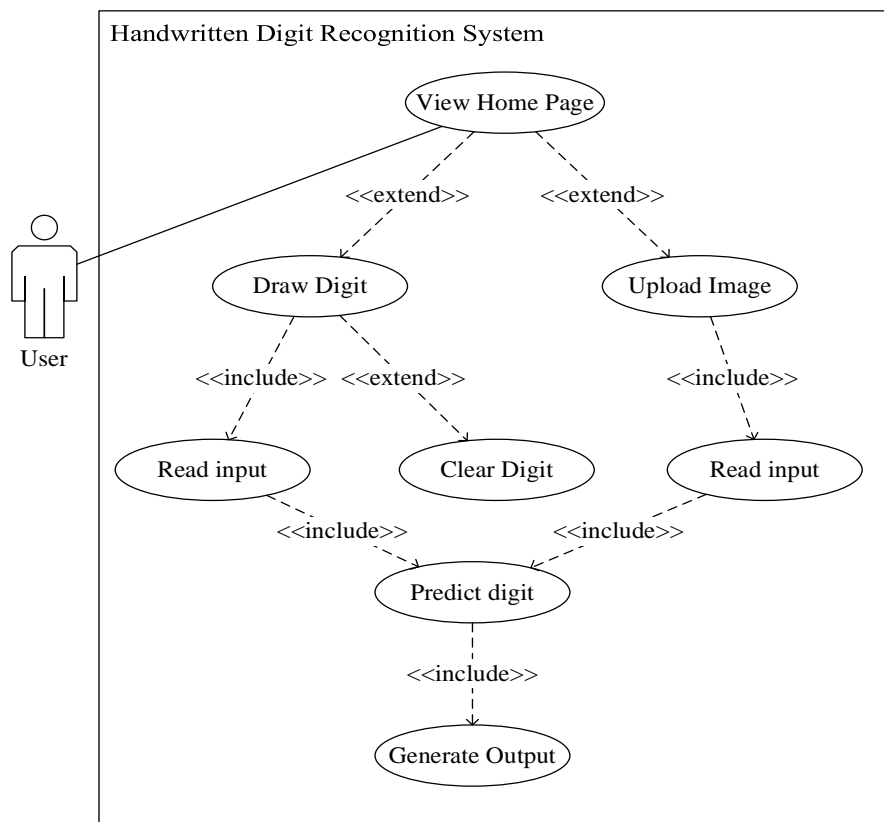


Figure 3.1:Use case diagram of handwritten digit recognition system.

ii. Non-functional Requirement

The non-functional requirements for the 'Handwritten Digits Recognition System' are as follows:

- **Scalability:** The system should be flexible enough to accommodate additional features such as recognition of handwritten characters, number plates, and sorting of postal mail.
- **Usability:** The system should have a user-friendly interface to ensure ease of use and understanding.
- **Interface:** The system's user interface should be intuitive and easy to navigate.
- **Maintainability:** Regular maintenance of the system is essential. Analysis of the system's performance should be conducted to improve its accuracy.
- **Performance:** This system of handwritten digit recognition should be able to provide a prediction of a handwritten digit.

3.2 Feasibility Analysis

i. Technical feasibility:

The system requires a simple user interface. The tools and software product required to develop our system is available on the internet it does not require a special environment to execute it needs jupyter notebook IDE.

ii. Operational feasibility:

The function of the system is possible. The user interface and resulting interface are easy to understand. Hence, this system is operationally feasible.

iii. Economic feasibility:

The system requires a python developer and a flutter developer. All the tasks are done by us. There are no such demerits of this application based on cost. So, the system will be economically feasible.

iv. Schedule (Gantt chart showing the project timeline)

Table 3.1 Gantt chart for handwritten digit recognition

Week/process	Dec (18-24)	Dec 24- Jan 4	Jan (5-17)	Jan (18-31)	Feb 1- Apr 1	Apr (2-27)	Apr 28
Planning							
Analysis							
Design							
Coding							
Testing							
Documentation							
Presentation							

Chapter 4: System Design

4.1 Design

System design is the method of designing the structure, elements, interfaces, and data of a software system to fulfill particular requirements. It involves decomposing the entire system into smaller, more manageable parts, and developing each of these parts to operate together smoothly. This process entails identifying the system's features, defining hardware and software prerequisites, and developing algorithms, interfaces, and data structures.

4.1.1 Architectural Design

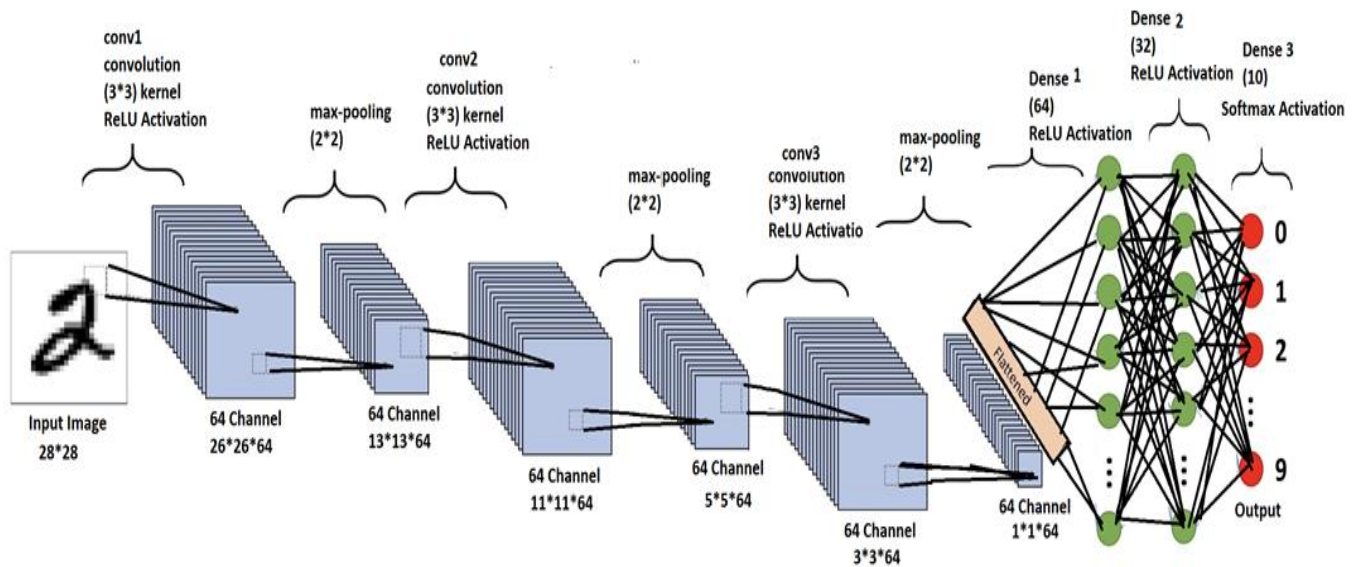


Figure 4.1: Architectural Diagram for Handwritten Digit Recognition

4.1.2 Class Diagram

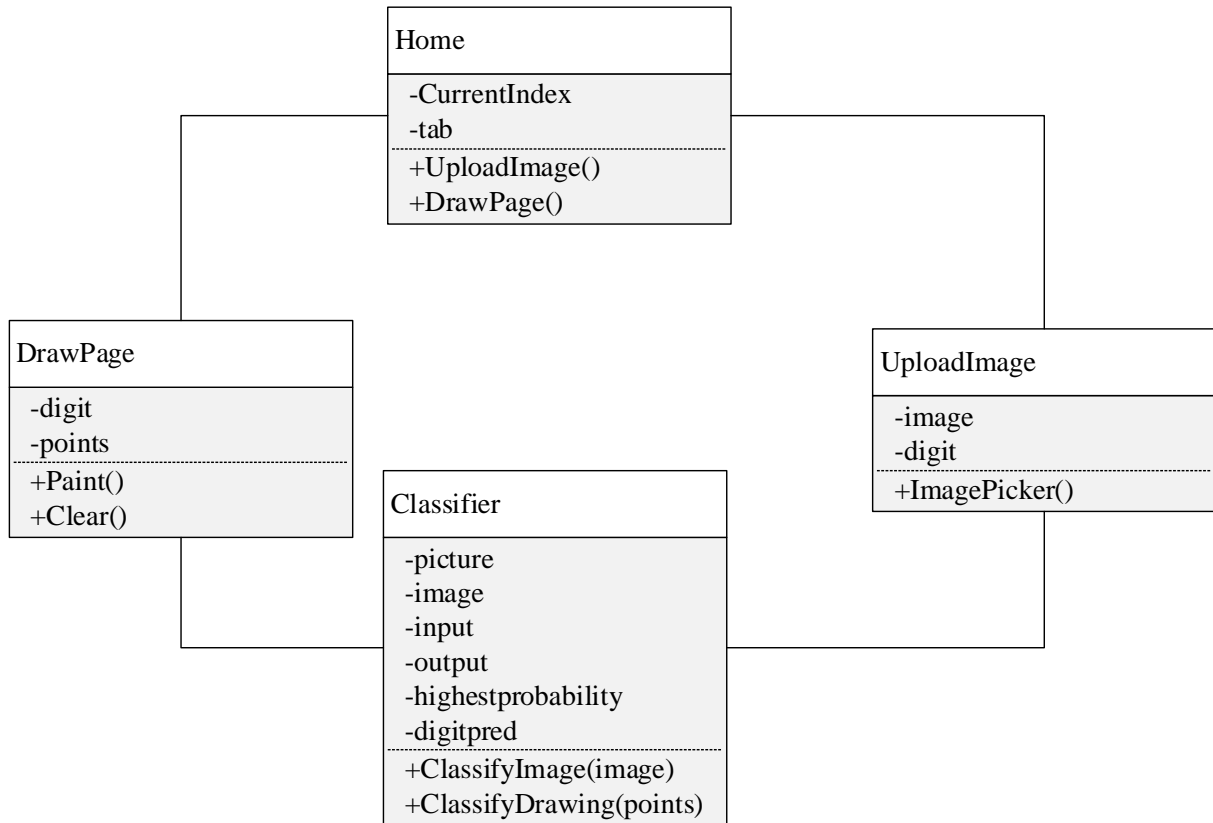


Figure 4.2:Class Diagram of Handwritten Digit Recognition

The Handwritten Digit Recognition System includes a home class that consists UploadImage() method and DrawPage() method. Similarly, there is a class DrawPage that consists of methods Paint() and Clear(). A UploadImage class has one method ImagePicker(). Similarly classifier class consists ClassifyImage() and ClassifyDrawing() methods. There is a general association between the home class and DrawPage class, the home class and UploadImage class, DrawPage class, and Classifier class, and the classifier class and UploadImage class.

4.1.3 Sequence Diagram

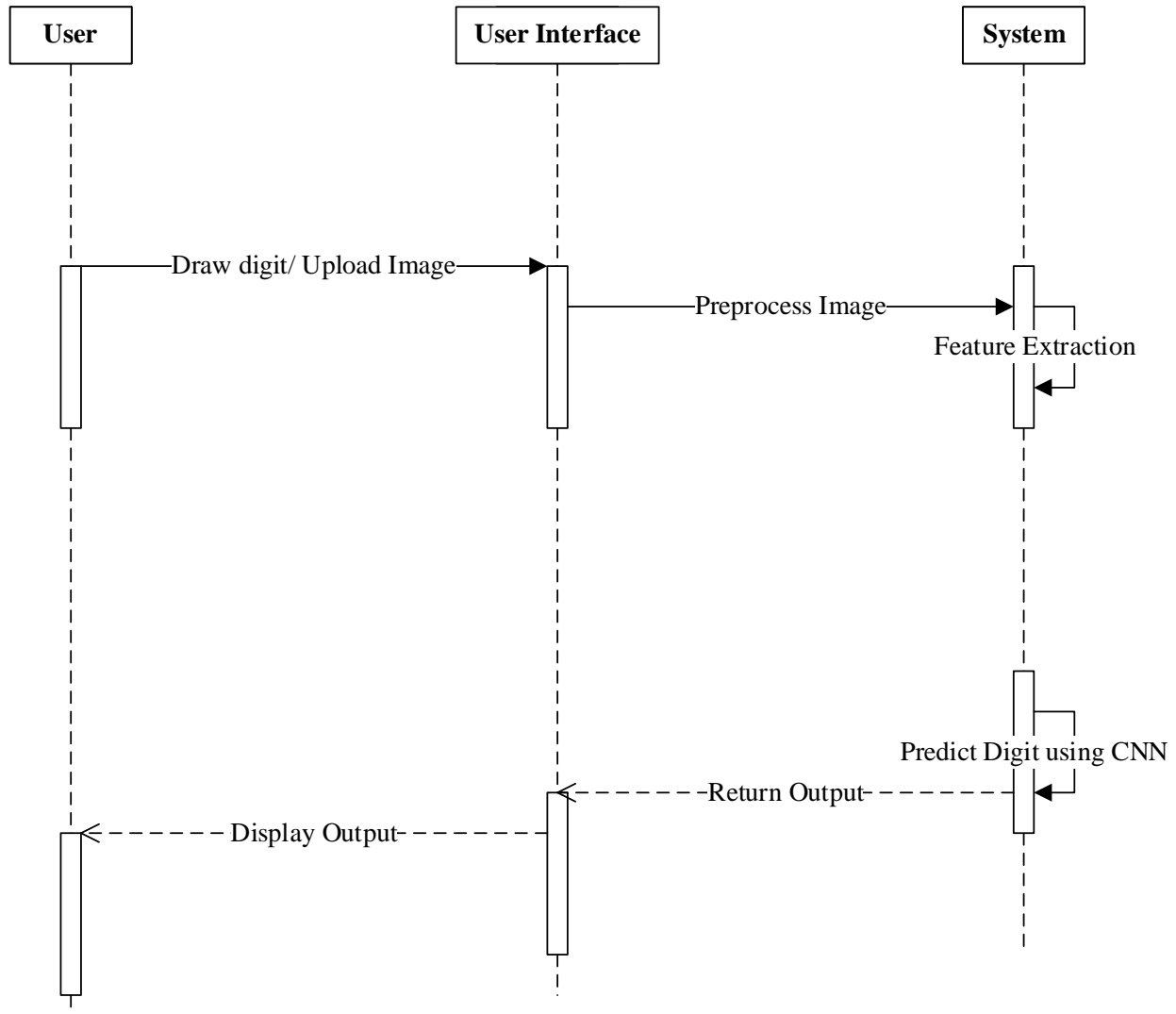


Figure 4.3:Sequence Diagram of Handwritten Digit Recognition

First of all, the user interface displays options to the user, the first option is to upload an image, and the second option is to draw a digit. In the first option, the user needs to upload the image of a handwritten digit so that the system predicts that image, in the second option user can draw a digit on the user interface. And the system will pre-process the image and system performs a feature extraction operation then the system will predict the digit using Convolutional Neural Network and return the output to the user interface. Finally, the user can see the output on the UI.

4.1.4 Activity Diagram

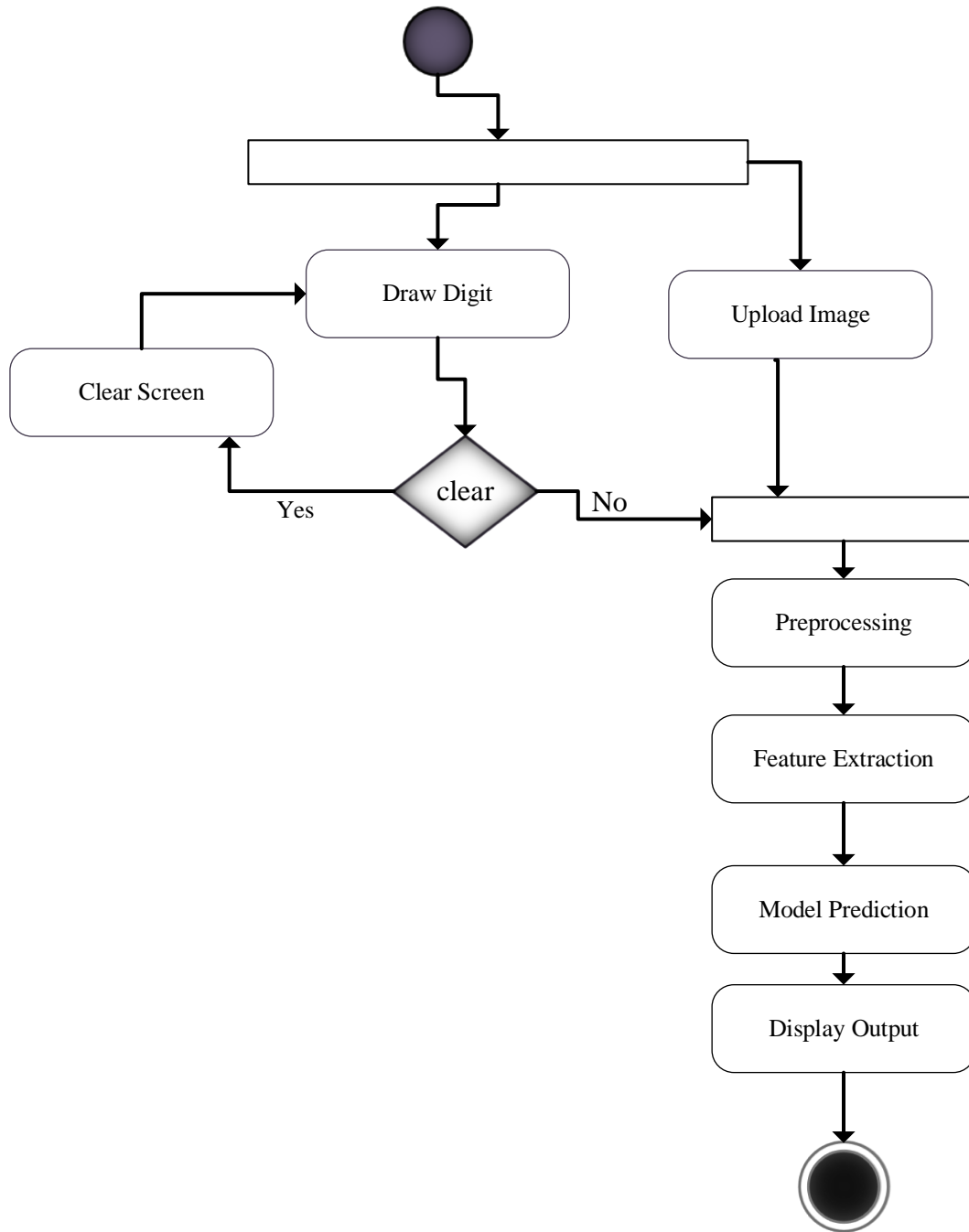


Figure 4.4:Activity diagram of Handwritten Digit Recognition

The user has the ability to draw a digit on the system's user interface. If the user presses the clear button, the system will erase the current drawing on the screen, and the user can then draw a new digit. However, if the clear button is not pressed, the system will pre-process the image of the

handwritten digit, perform feature extraction and model prediction operations. Finally, the output of the prediction will be displayed on the screen.

4.2 Dataset

A dataset is a collection of images or other types of data used for training and testing the network. The dataset is typically divided into two parts: a training set and a test set. For the handwritten digit recognition project there can be used MNIST dataset.



Figure 4.5: MNIST Dataset

This image shows a few examples of handwritten digits from the MNIST dataset. Here each digit is represented as a 28x28 grayscale image. This means that each image consists of 784 pixels (28 x 28), with each pixel represented as a grayscale value ranging from 0 (white) to 255 (black). The grayscale values represent the intensity of the pixel, with higher values indicating darker areas and lower values indicating lighter areas. It consists of a set of 70,000 28x28 grayscale images of handwritten digits from 0 to 9, with 60,000 images used for training and 10,000 for testing. The dataset was created by Yann LeCun, Corinna Cortes, and Christopher J.C. Burges for their paper

"MNIST handwritten digit database" in 1998. It has since become a widely-used benchmark dataset for evaluating image recognition algorithms.

4.3 Algorithm

Step1: Gather and preprocess the dataset. The MNIST dataset is a well-known option that comprises of 60,000 images for training and 10,000 images for testing.

Step 2: Develop the CNN model

2.1 Implement convolution layer

2.2 Implement Activation layer

2.3 Implement pooling layer

2.4 Implement fully-connected layer

Step 3: Train the CNN model using the prepared dataset.

Step 4: Convert the trained model into tflite format.

Step 5: Integrate the tflite model into the flutter app

Step 6: Create a user interface for the system.

The working mechanism of convolution neural network with an example:

1. Convolution layer

The convolution layer is the primary layer that aids in extracting various features from input images. It employs the mathematical operation of convolution, wherein a filter of a particular size $M \times M$ is convolved with the input image. The filter is slid over the input image, and a dot product is computed between the filter and the parts of the input image that align with the size of the filter ($M \times M$). The output is a feature map that provides information about the image, such as the edges and corners. The feature map is subsequently fed to other layers to learn additional features of the

input image. The CNN's convolution layer applies the convolution operation to the input and transmits the result to the next layer. The convolutional layers in CNN are highly beneficial as they maintain the spatial relationship between the pixels.

0	1	5	4	6	7	3	5
9	4	5	2	4	8	6	5
9	7	5	1	3	0	2	4
5	9	7	1	4	5	3	2
1	5	8	6	4	0	1	2
5	4	5	8	9	4	5	7
6	1	5	8	9	2	1	4
5	7	8	5	4	9	3	2
5	8	7	4	6	9	8	0

Op

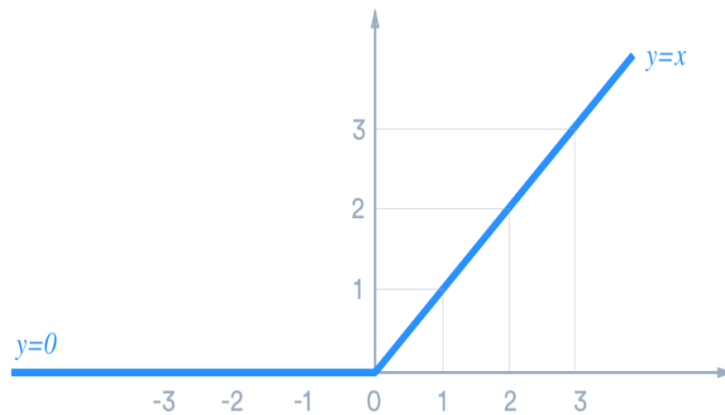
W1	W2	W3
W4	W5	W6
W7	W8	W9

0	0	0	0	0	0
0	V1				0
0					0
0					0
0					0
0	0	0	0	0	0

$$V1=w1*0+w2*1+w3*5+w4*9+w5*4+w6*5+w7*9+w8*7+w9*5$$

2. Activation

ReLU (Rectified Linear Unit) is one of the most commonly used activation functions in CNN models. It adds non-linearity to the network and helps in improving the accuracy of the model. ReLU function is applied to the output of the convolution layer before passing it to the next layer. The ReLU function sets all negative values to zero and keeps all positive values as they are. This helps to remove any negative values that may interfere with the network's learning process.



Before ReLU activation function,

-2	5	4	8	9	-3	2	-1
-5	6	3	4	9	2	5	-7
6	-3	5	-4	-8	2	-1	5
6	8	4	-5	9	8	7	4
-2	3	-1	5	8	-4	5	-2
6	7	8	4	-9	2	1	-3
5	-2	3	-5	4	7	9	6
8	5	-4	1	-3	3	-6	5

After ReLU activation function

0	5	4	8	9	0	2	0
0	6	3	4	9	2	5	0
6	0	5	0	0	2	0	5
6	8	4	0	9	8	7	4
0	3	0	5	8	0	5	0
6	7	8	4	0	2	1	0
5	0	3	0	4	7	9	6
8	5	0	1	0	3	0	5

3. Pooling (reduces the dimension of the matrix)

The Pooling Layer is added after a Convolutional Layer in a CNN. Its primary function is to reduce the size of the feature map generated by the previous layer, thereby reducing the computational complexity of the network. This is achieved by independently operating on each feature map and decreasing the number of connections between layers. Max Pooling selects the largest element from each feature map, while Average Pooling calculates the average of the elements within a given image section. Sum Pooling computes the sum of the elements in the section. The Pooling Layer is typically placed between the Convolutional Layer and the Fully Connected Layer.

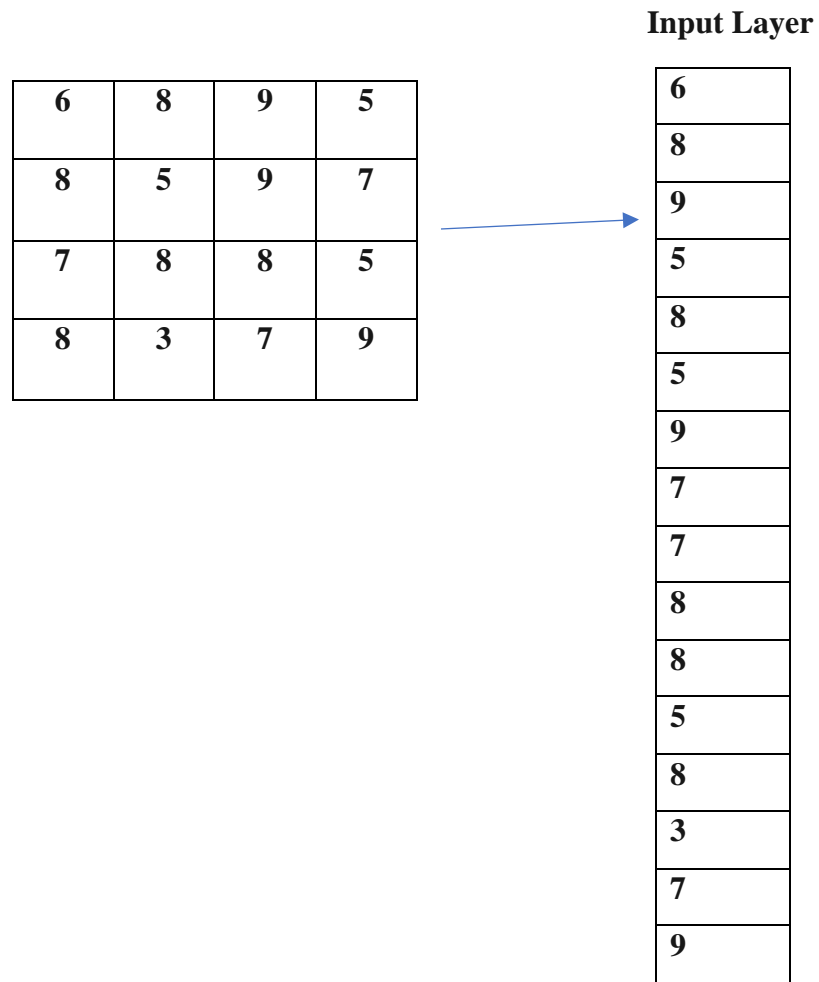
0	5	4	8	9	0	2	0
0	6	3	4	9	2	5	0
6	0	5	0	0	2	0	5
6	8	4	0	9	8	7	4
0	3	0	5	8	0	5	0
6	7	8	4	0	2	1	0
5	0	3	0	4	7	9	6
8	5	0	1	0	3	0	5

2*2 pooling window



6	8	9	5
8	5	9	7
7	8	8	5
8	3	7	9

4. Fully connected



The Fully Connected (FC) layer is an important part of the CNN architecture that connects neurons between two different layers. It is typically placed before the output layer and is composed of weights, biases, and neurons. The input image from previous layers is flattened and passed to the FC layer, where mathematical operations are performed in a series of layers. This stage is where the classification process begins. Two fully connected layers are connected because they can perform better than a single connected layer, and these layers in CNN reduce the need for human supervision.

Chapter 5:Implementation and Testing

5.1 System Implementation

System implementation refers to the process of putting into practice or bringing a software system into operational use. This phase of the software development life cycle (SDLC) involves the actual development and deployment of the software system after the system has been designed, tested, and validated. System implementation is a critical phase of the SDLC as it is the point at which the software system becomes operational and is used by end-users. Successful implementation requires close collaboration between software developers, project managers, quality assurance teams, and end-users to ensure that the system meets the specified requirements and is user-friendly.

5.1.1 Implementation Tools

To create a handwritten digit recognition system using Convolutional Neural Networks (CNN) in Flutter, there are various front-end and back-end tools available. Some of the popular options include:

Frontend Tools


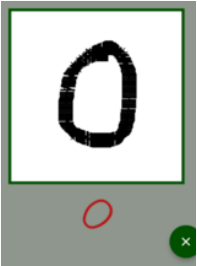






1. Flutter SDK is the primary development tool for creating the user interface and the app's front-end.
2. Dart programming language is used for writing the front-end code of the Flutter app.
3. TensorFlow Lite is a lightweight version of the TensorFlow library that can be used to execute the trained CNN model on the mobile device's front-end.
4. Canvas is a built-in widget in Flutter that enables the user to draw their handwritten digits on the screen.

Back-end Tools

1. Python is the primary programming language used for constructing the CNN model for digit recognition.
2. Keras is a popular deep-learning framework that can be utilized to develop and train the CNN model.

5.2 Testing

Table 5.1 Testing

TID	Test Case	Test Step	Test Data	Expected Output	Result
1	Single Digit Recognition	Browse the drawing page and draw the digit		0	
2	Handwriting Variation	Browse the drawing page and draw the digit		3	
3	Noisy Digit	Browse the drawing page and draw the digit		5	
4	Multi lined Digit	Browse the drawing page and draw the digit		8	

Chapter 6: Conclusion and Future Recommendation

6.1 Conclusion

Building a handwritten digit recognition system using CNN in flutter is a complex task that involves both front-end development and back-end development. Front-end development involves creating a user interface to draw digits for recognition and back-end development involves building a CNN model that can recognize handwritten digits. Overall building a handwritten digit using CNN in flutter requires a combination of machine learning and mobile app development skill.

6.2 Future Recommendation

Handwritten digit recognition can be expanded to recognize the digits written in multiple languages which can help international users. This handwritten digit recognition can be expanded to recognize multiple digits at once and integrated with another technology like a character recognizer to help in identifying the number plates of the vehicle.

REFERENCES

- [1] . Anirudh Mhaske, Atharv Joshi, Dattaram Kajrekar, Ruturaj Jugdar, Prof. Ajita Mahapadi Digit Recognition using MNIST Dataset, International Journal for Research in Applied Science & Engineering Technology (IJRASET), Volume 10 Issue IX Sep 2022.
- [2]. Vijayalaxmi R Rudraswamimath¹, Bhavanishankar K²Handwritten Digit Recognition using CNN, International Journal of Innovative Science and Research Technology, Volume 4, Issue 6, June – 2019.
- [3]. Kübra Gülgün Demirkaya, Ünal Çavuşoğlu Handwritten Digit Recognition with Machine Learning Algorithms, Academic Platform Journal of Engineering and Smart Systems (APJESS) 10(1), 9-18, 2022.
- [4]. Z. Dan, C. Xu, The Recognition of Handwritten Digits Based on BP Neural Networks and the Implementation on Android, In: 3rd International Conference on Intelligent System Design and Engineering Applications, pp. 1498-1509, 2013.
- [5]. Hamid, Norhidayu Abdul, and NilamNur Amir Sjarif,Handwritten recognition using SVM, KNN and neural network, arXiv preprint arXiv:1702.00723 (2017).
- [6]. A. El-Sawy, M. Loey, and H. El-Bakry, “Arabic Handwritten Characters Recognition using Convolutional Neural Network,” WSEAS Transactions on Computer Research, vol. 5, pp. 11–19, 2017.
- [7]. S. S. Mor, S. Solanki, S. Gupta, S. Dhingra, M. Jain, and R. Saxena, “Handwritten Text Recognition: With Deep Learning and Android,” International Journal of Engineering and Advanced Technology, vol. 8, no. 2, pp. 172–178, 2019.
- [8]. E. Dağdeviren, “El Yazısı Rakam Tanıma İçin Destek Vektör Makinelerinin ve Yapay Sinir Ağlarının Karşılaştırması,” 2013.

APPENDICES

```
In [1]: import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import cv2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,Activation,Flatten,Conv2D,MaxPooling2D
```

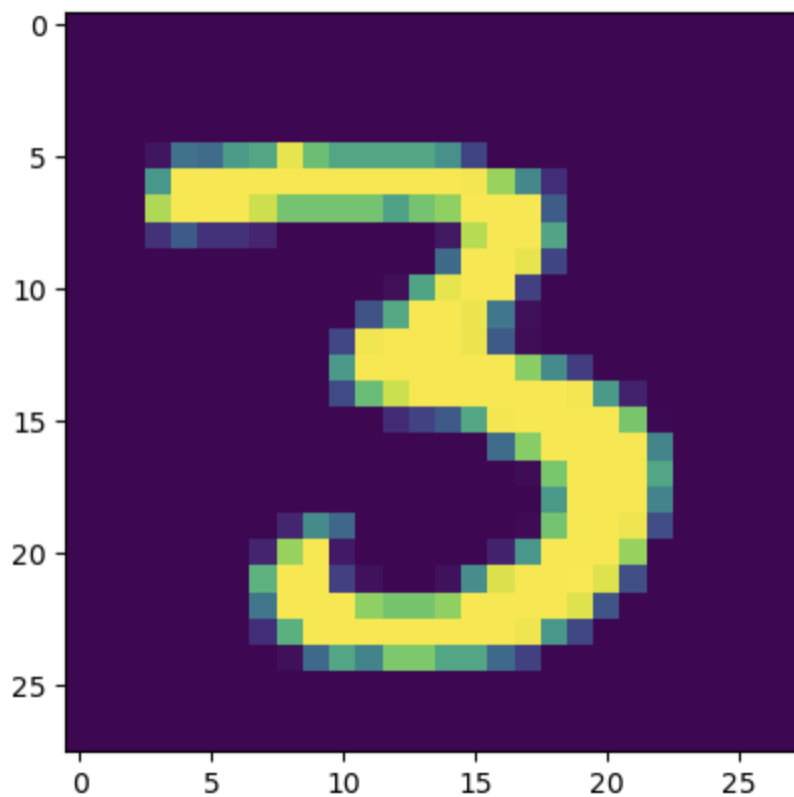
```
In [2]: mnist=tf.keras.datasets.mnist
```

```
In [3]: (x_train,y_train),(x_test,y_test)=mnist.load_data()
```

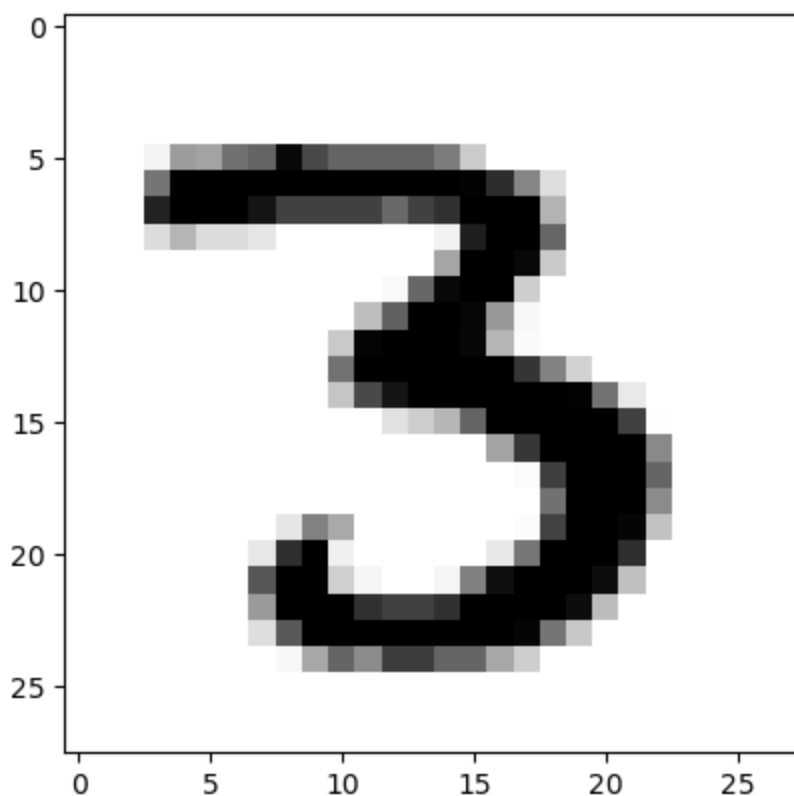
```
In [4]: x_train.shape
```

```
Out[4]: (60000, 28, 28)
```

```
In [5]: plt.imshow(x_train[12])
plt.show()
```



```
In [6]: plt.imshow(x_train[12],cmap=plt.cm.binary)
plt.show()
```



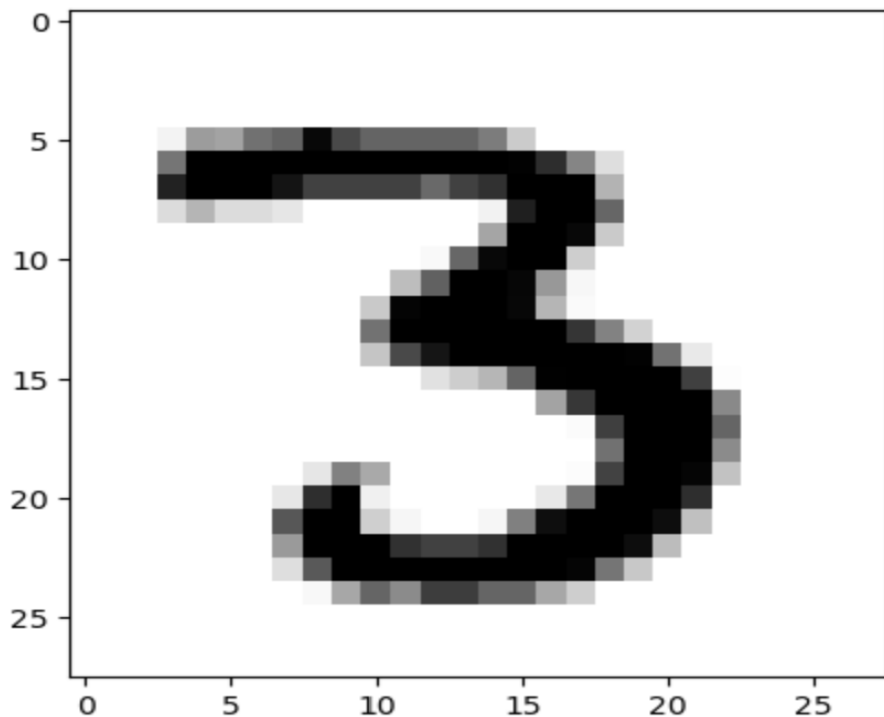
```
In [7]: print(x_train[12])
```

```
[[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 12 99 91 142 155 246 182 155 155 155 155 131 52  0  0
   0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 138 254 254 254 254 254 254 254 254 254 254 254 252 210 122
  33  0  0  0  0  0  0  0  0  0  0]]
```

```
[ 0 0 0 0 0 0 0 0 25 126 86 0 0 0 0 0 0 3
188 254 254 250 61 0 0 0 0 0]
[ 0 0 0 0 0 0 0 24 209 254 15 0 0 0 0 0 23 137
254 254 254 209 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 168 254 254 48 9 0 0 9 127 241 254
254 255 242 63 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 101 254 254 254 205 190 190 205 254 254 254
254 242 67 0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 33 166 254 254 254 254 254 254 254 254 250
138 55 0 0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 0 7 88 154 116 194 194 154 154 88 49
0 0 0 0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0]]
```

```
In [8]: x_train=x_train/255
x_test=x_test/255
plt.imshow(x_train[12],cmap=plt.cm.binary)
```

```
Out[8]: <matplotlib.image.AxesImage at 0x1edfb744490>
```



```
In [10]: img_size=28
x_trainr=np.array(x_train).reshape(-1,img_size,img_size,1)
x_testr=np.array(x_test).reshape(-1,img_size,img_size,1)
print("training smaple dimension:",x_trainr.shape)
print("testing smaple dimension:",x_testr.shape)
```

```
training smaple dimension: (60000, 28, 28, 1)
testing smaple dimension: (10000, 28, 28, 1)
```

```
In [11]: model=Sequential()
model.add(Conv2D(64,(3,3),input_shape=x_trainr.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dense(32))
model.add(Activation('relu'))
model.add(Dense(10))
model.add(Activation('softmax'))
```


In [12]: `model.summary()`

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 64)	640
activation (Activation)	(None, 26, 26, 64)	0
max_pooling2d (MaxPooling2D)	(None, 13, 13, 64)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	36928
activation_1 (Activation)	(None, 11, 11, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	36928
activation_2 (Activation)	(None, 3, 3, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 1, 1, 64)	0
flatten (Flatten)	(None, 64)	0
dense (Dense)	(None, 64)	4160

activation_3 (Activation)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2080
activation_4 (Activation)	(None, 32)	0
dense_2 (Dense)	(None, 10)	330
activation_5 (Activation)	(None, 10)	0

=====
 Total params: 81,066
 Trainable params: 81,066
 Non-trainable params: 0

```
In [13]: model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
In [14]: model.fit(x_trainr,y_train,epochs=15,validation_split=0.3)
```

```
Epoch 1/15
1313/1313 [=====] - 24s 17ms/step - loss: 0.2926 - accuracy: 0.9081 - val_loss: 0.1097 - val_accuracy:
0.9664
Epoch 2/15
1313/1313 [=====] - 23s 17ms/step - loss: 0.0918 - accuracy: 0.9722 - val_loss: 0.0925 - val_accuracy:
0.9724
Epoch 3/15
1313/1313 [=====] - 23s 17ms/step - loss: 0.0644 - accuracy: 0.9796 - val_loss: 0.0643 - val_accuracy:
0.9812
Epoch 4/15
1313/1313 [=====] - 23s 18ms/step - loss: 0.0526 - accuracy: 0.9834 - val_loss: 0.0667 - val_accuracy:
0.9806
Epoch 5/15
1313/1313 [=====] - 23s 18ms/step - loss: 0.0419 - accuracy: 0.9865 - val_loss: 0.0574 - val_accuracy:
0.9838
Epoch 6/15
1313/1313 [=====] - 23s 17ms/step - loss: 0.0345 - accuracy: 0.9890 - val_loss: 0.0573 - val_accuracy:
0.9833

Epoch 7/15
1313/1313 [=====] - 23s 17ms/step - loss: 0.0280 - accuracy: 0.9915 - val_loss: 0.0539 - val_accuracy:
0.9854
Epoch 8/15
1313/1313 [=====] - 23s 17ms/step - loss: 0.0241 - accuracy: 0.9919 - val_loss: 0.0638 - val_accuracy:
0.9821
Epoch 9/15
1313/1313 [=====] - 23s 17ms/step - loss: 0.0234 - accuracy: 0.9924 - val_loss: 0.0549 - val_accuracy:
0.9846
Epoch 10/15
1313/1313 [=====] - 23s 18ms/step - loss: 0.0172 - accuracy: 0.9943 - val_loss: 0.0638 - val_accuracy:
0.9834
Epoch 11/15
1313/1313 [=====] - 23s 18ms/step - loss: 0.0180 - accuracy: 0.9940 - val_loss: 0.0527 - val_accuracy:
0.9862
Epoch 12/15
1313/1313 [=====] - 23s 17ms/step - loss: 0.0151 - accuracy: 0.9952 - val_loss: 0.0657 - val_accuracy:
0.9833
Epoch 13/15
1313/1313 [=====] - 23s 18ms/step - loss: 0.0136 - accuracy: 0.9955 - val_loss: 0.0776 - val_accuracy:
0.9825
Epoch 14/15
1313/1313 [=====] - 23s 18ms/step - loss: 0.0121 - accuracy: 0.9959 - val_loss: 0.0591 - val_accuracy:
0.9870
Epoch 15/15
1313/1313 [=====] - 23s 18ms/step - loss: 0.0116 - accuracy: 0.9963 - val_loss: 0.0579 - val_accuracy:
0.9878
```

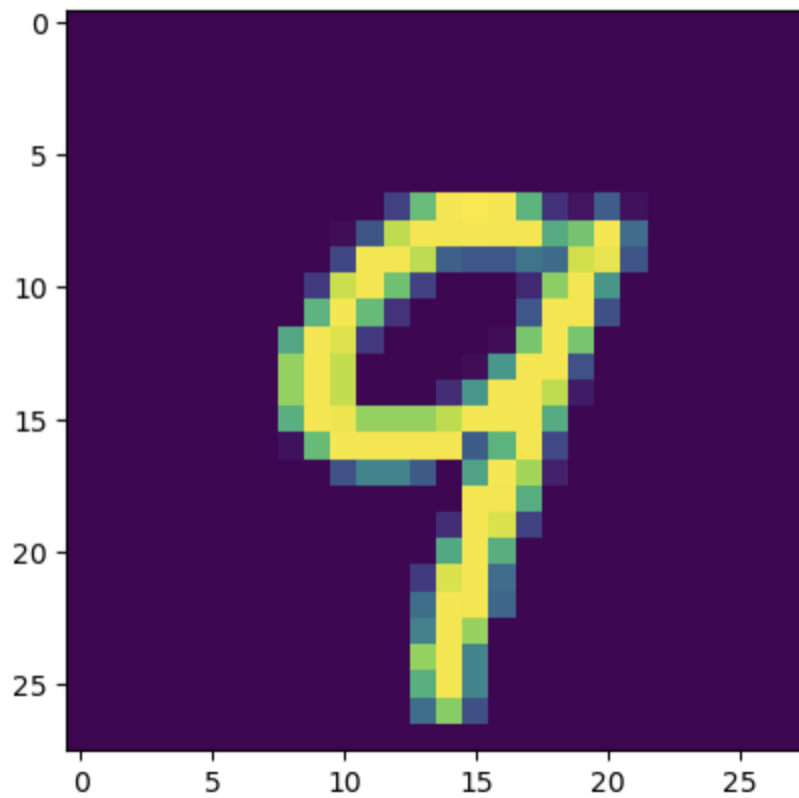
```
In [16]: prediction=model.predict([x_testr])
```

```
313/313 [=====] - 2s 5ms/step
```

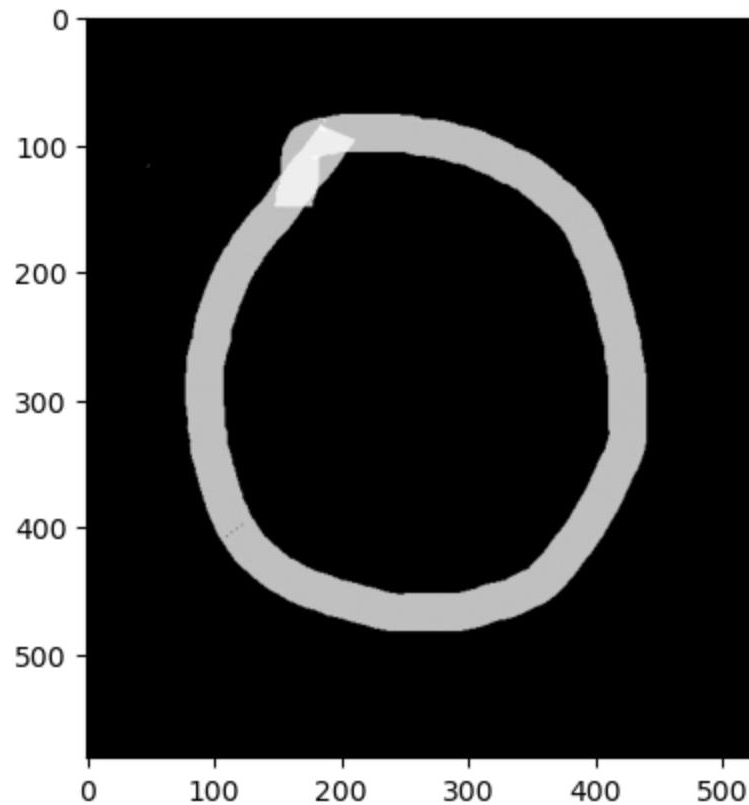
```
In [17]: print(np.argmax(prediction[12]))
```

```
9
```

```
In [18]: plt.imshow(x_test[12])  
plt.show()
```



```
In [19]: img0=cv2.imread('0.png')
plt.imshow(img0)
plt.show()
```



```
In [20]: img0.shape
```

```
Out[20]: (582, 526, 3)
```

```
In [21]: img_g0=cv2.cvtColor(img0,cv2.COLOR_BGR2GRAY)
img_g0.shape
```

```
Out[21]: (582, 526)
```

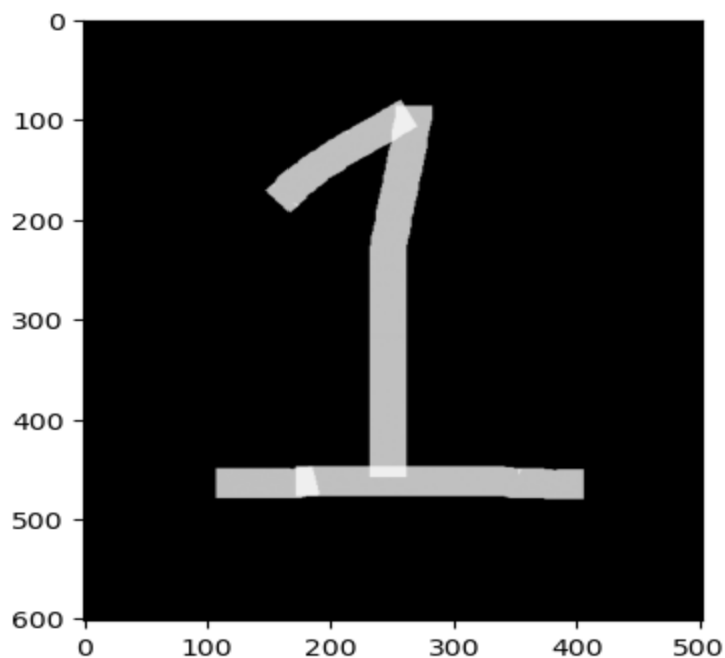
```
In [22]: img_r0=cv2.resize(img_g0,(28,28))
img_r0=img_r0/255
img_r0=np.array(img_r0).reshape(1,img_size,img_size,1)
img_r0.shape
```

```
Out[22]: (1, 28, 28, 1)
```

```
In [23]: prediction0=model.predict(img_r0)
1/1 [=====] - 0s 92ms/step
```

```
In [24]: print(np.argmax(prediction0))
0
```

```
In [25]: img1=cv2.imread('1.png')
plt.imshow(img1)
plt.show()
```



```
In [26]: img1.shape
```

```
Out[26]: (602, 502, 3)
```

```
In [27]: img_g1=cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
img_g1.shape
```

```
Out[27]: (602, 502)
```

```
In [28]: img_r1=cv2.resize(img_g1,(28,28))
img_r1=img_r1/255
img_r1=np.array(img_r1).reshape(1,img_size,img_size,1)
img_r1.shape
```

```
Out[28]: (1, 28, 28, 1)
```

```
In [29]: prediction1=model.predict(img_r1)
```

```
1/1 [=====] - 0s 22ms/step
```

```
In [30]: print(np.argmax(prediction1))
```

```
1
```

```
In [82]: model.save("cnn_model")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _update_step_xla while saving (showing 4 of 4). These functions will not be directly callable after loading.
```

```
INFO:tensorflow:Assets written to: cnn_model\assets
```

```
INFO:tensorflow:Assets written to: cnn_model\assets
```

```
In [83]: saved_model_dir = "cnn_model/"
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
model_lite = converter.convert()
with open('cnn.tflite', 'wb') as f:
    f.write(model_lite)
```

```

class Classifier{

    Classifier();

    ClasssifyImage(XFile image) async {

        var _file = io.File(image.path);

        img.Image? imageTemp = img.decodeImage(_file.readAsBytesSync());

        img.Image resizedImg = img.copyResize(imageTemp!,

            height: 28, width: 28);

        var imgBytes = resizedImg.getBytes();

        var imgAsList = imgBytes.buffer.asUint8List();

        return getpred(imgAsList);}

    ClassifyDrawing(List<Offset?> points) async{

        final picture = toPicture(points);

        final image = await picture.toImage(28, 28);

        ByteData? imgBytes = await image.toByteData();

        var imgAsList = imgBytes?.buffer.asUint8List();

        return getpred(imgAsList!);}

    Future<int> getpred(Uint8List imgAsList) async

    {

        List resultBytes=List.filled(28*28, null, growable: false);

```

```

int index=0;

for(int i=0;i<imgAsList.lengthInBytes;i+=4) {

    final r=imgAsList[i];

    final g=imgAsList[i+1];

    final b=imgAsList[i+2];

    resultBytes[index]=((r+g+b)/3)/255;

    index++;

}

var input=resultBytes.reshape([1,28,28,1]);

var output=List.filled(1*10, null, growable: false).reshape([1,10]);

InterpreterOptions interpreterOptions=InterpreterOptions();

try{

    Interpreter interpreter=await Interpreter.fromAsset(

        'cnn.tflite',

        options: interpreterOptions,

    );

    interpreter.run(input,output);

}catch(e){

    print("Error");

```



```

    }

    double highestprob=0;

    int digitpred=0;

    for(int i=0;i<output[0].length;i++)

    {

        if(output[0][i]>highestprob) {

            highestprob=output[0][i];

            if(highestprob>0.95)

            {

                digitpred=i;

            }

            else{

                digitpred=-1;

            }

        }

    }

    return digitpred;

} }

ui.Picture toPicture(List<Offset?> points){

    final _whitePaint = Paint()

    ..strokeCap = StrokeCap.round

```

```

        ..color = Colors.white

        ..strokeWidth = 16;

    final _bgPaint = Paint()..color = Colors.black;

    final _canvasCullRect = Rect.fromPoints(Offset(0, 0),

        Offset(28, 28));

    final recorder = ui.PictureRecorder();

    final canvas = Canvas(recorder, _canvasCullRect)

        ..scale(28 / 300);

    canvas.drawRect(Rect.fromLTWH(0, 0, 28, 28), _bgPaint)

    for (int i = 0; i < points.length - 1; i++) {

        if (points[i] != null && points[i + 1] != null) {

            canvas.drawLine(points[i]!, points[i + 1]!, _whitePaint);

        }

    }

    return recorder.endRecording();

}

```

Snap shot





