A Project Report on

Developing Real-time Digitization System using Deep Learning for Handwritten Documents

Submitted in partial fulfilment of the requirements for the award of the degree of

Bachelor of Engineering

in

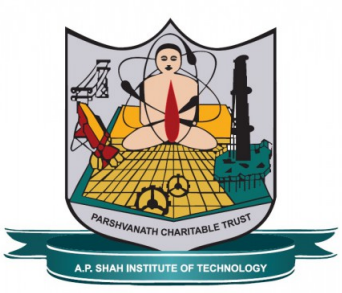
Information Technology

by

Atharv Kailas Joshi(19104036)
Siddhesh Pravin Puranik(19104014)
Niranjan Muniraj Ram(19104025)

Under the Guidance of

Prof. Sonal Balpande



Department of Information Technology
NBA Accredited
A.P. Shah Institute of Technology
G.B.Road,Kasarvadavli, Thane(W), Mumbai-400615
UNIVERSITY OF MUMBAI

Academic Year 2022-2023

Approval Sheet

This Project Report entitled ***"Developing Real-time Digitization System using Deep Learning for Handwritten Documents"*** Submitted by ***"Atharv Kailas Joshi"(19104036), "Siddhesh Pravin Puranik"(19104014), "Niranjan Muniraj Ram"(19104025)***is approved for the partial fulfilment of the requirement for the award of the degree of ***Bachelor of Engineering*** in ***Information Technology*** from ***University of Mumbai***.

Prof.Sonal Balpande
Guide

Dr. Kiran Deshpande
Head Department of Information Technology

Place:A.P.Shah Institute of Technology, Thane
Date:

CERTIFICATE

This is to certify that the project entitled ***"Developing Real-time Digitization System Using Deep Learning for Handwritten Documents"*** submitted by ***"Atharv Kailas Joshi"(19104036), "Siddhesh Pravin Puranik"(19104014), "Niranjan Muniraj Ram"(19104025)*** for the partial fulfilment of the requirement for the award of a degree ***Bachelor of Engineering*** in ***Information Technology***, to the University of Mumbai, is a bonafide work carried out during the academic year 2022-2023.

Prof. Sonal Balpande
Guide

Dr. Kiran Deshpande
Head Department of Information Technology

Dr. Uttam D.Kolekar
Principal

External Examiner(s)

1.

2.

Place:A.P.Shah Institute of Technology, Thane
Date:

Acknowledgement

We have great pleasure in presenting the report on **Developing Real-time Digitization System Using Deep Learning for Handwritten Documents.** We take this opportunity to express our sincere thanks towards our guide **Prof. Sonal Balpande** Department of IT, APSIT thane for providing the technical guidelines and suggestions regarding the line of work. We would like to express our gratitude towards her constant encouragement, support and guidance through the development of the project.

We thank **Dr. Kiran B. Deshpande** Head of Department, IT, APSIT for his encouragement during the progress meeting and for providing guidelines to write this report.

We thank **Prof. Sonal Jain** BE project co-ordinator, Department of IT, APSIT for being encouraging throughout the course and for guidance.

We also thank the entire staff of APSIT for their invaluable help rendered during the course of this work. We wish to express our deep gratitude towards all our colleagues at APSIT for their encouragement.

Atharv Kailas Joshi
19104036

Siddhesh Puranik
19104014

Niranjan Ram
19104025

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, We have adequately cited and referenced the original sources. We also declare that We have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Atharv Kailas Joshi: 19104036)

_____

(Siddhesh Pravin Puranik: 19104014)

_____

(Niranjan Muniraj Ram: 19104025)

Date:

Abstract

The purpose of this research is to automate the existing manual system for digitization of the Devanagari script with the help of an automated approach which reduces time and workload. The automation of digitization enables easy access, manipulation, and longer storage of this data. Unlike Western languages such as English written in the Roman script, Devanagari does not have common or easy-to-use digitization tools. Thus, the proposed approach aims to fill this gap and make the digitization process more efficient for the Devanagari script. This work will focus on two languages in particular: Hindi and Marathi. Both of these use virtually the same script with no major differences in spelling or pronunciation. This work is to serve as a proof of concept for the digitization of the Devanagari script. The goal of this work is to improve the accuracy and efficiency of the Devanagari handwritten text recognition by utilizing the most effective techniques and configuring a Convolutional Neural Network (CNN). By leveraging state-of-the-art approaches and customizing them for the unique characteristics of the Devanagari script, this study significantly enhances the recognition rate and automates the digitization process, allowing for easier access, manipulation, and storage of this important data. This approach uses the SHABD dataset (Complete Hindi characters) which is a cutting-edge and enormous open dataset with 384 unique classes of Devanagari characters. The complete dataset contains 304,150 grayscale images making it an enormous dataset. This work also results in development of a real-time app to convert handwritten text into editable files. This also helps provide a better user experience and accessibility to digitising such an important but overlooked script.

Contents

1 Introduction **1**
- 1.0.1 Problem Definition . . . 2
- 1.0.2 Objectives . . . 2
- 1.0.3 Scope . . . 3

2 Literature Review **4**

3 Project Design **7**
- 3.0.1 Existing System . . . 7
- 3.0.2 Proposed System . . . 7
- 3.0.3 System Diagram . . . 9

4 Project Implementation **11**
- 4.0.1 Technology Stack: . . . 11
- 4.0.2 Interface: . . . 12
- 4.0.3 Code Snippets: . . . 12

5 Testing **18**
- 5.1 Functional Testing . . . 18
  - 5.1.1 Segmentation . . . 18
  - 5.1.2 Character Recognition . . . 19
  - 5.1.3 Various Test Cases: . . . 19

6 Result **20**

7 Conclusions and Future Scope **23**

Bibliography **24**

Appendices **26**
- Appendix-A . . . 26
  - 7.0.1 Libraries Used . . . 26
  - 7.0.2 Default Libraries in Dart: . . . 26
  - 7.0.3 Default Libraries in Flutter: . . . 26
  - 7.0.4 Python Libraries Used: . . . 27

Publication **29**

List of Figures

3.1 Use Case Diagram for the User 8
3.2 Segmentation 9
3.3 Classification 10

4.1 App Interface 12
4.2 Snippet 1 12
4.3 Snippet 2 13
4.4 Snippet 3 13
4.5 Snippet 4 14
4.6 Snippet 5 14
4.7 Snippet 6 15
4.8 Snippet 7 15
4.9 Backend 1 16
4.10 Backend 2 16
4.11 Backend 3 17

5.1 Character segmentation 18
5.2 Snippet 19
5.3 Test Cases 19

6.1 Epoch accuracy 21
6.2 CNN Summary 22

List of Tables

6.1 Computer Specifications of the Members. 20

List of Abbreviations

CNN:	Convolution Neural Network
RNN:	Recurrent Neural Network
DHTR:	Devanagari Handwritten Text Recognition
DHCD:	Devanagari Handwritten Character Dataset
OCR:	Optical Character Recognition
MNIST:	Modified National Institute of Standards and Technology

Chapter 1

Introduction

Devanagari script is the most used script in India[14]. With the growth in the digital and computer world as well as certain good initiatives by the government like "Make in India" and "Digital India", documents also need to be digital. In the Information-age or as some call it the New Media Age one needs to work in the digital constructs. However, it is not always possible, especially in the more rural areas or even with the people whom some might consider old-school. There is an abundance of sources and mediums to digitize the languages that are in Roman script. Indic scripts and especially the Devanagari script lacks behind.
The importance of preserving cultural heritage and historical documents cannot be overstated. However, with the passage of time, these documents become more and more fragile and prone to damage. Digitization not only ensures their preservation but also allows for wider access and dissemination of knowledge. With the increasing need for digital preservation and accessibility, it has become imperative to develop automated systems that can efficiently and accurately digitize scripts that lack common digitization tools. This research is a step towards fulfilling this need for the Devanagari script, and it holds the potential to revolutionize the preservation and accessibility of cultural and historical documents written in this script.
Digitizing old and sometimes obsolete documents written in the Devanagari script can indeed be a challenging task, especially when an automated system for doing so is not available. The growth in technology and the demand for digitization at a rapid rate has led to an increased need for automated systems that can recognize and convert Devanagari script into digital format. This urges one to have an automated system that would help the overall process by making it considerably cheaper and faster. The Indian civil offices use the Devanagari script more than most other scripts. National Informatics Centre (NIC), courtrooms, registrar offices and small-scale vendors maintain various documents which are available in either handwritten or printed form in Devanagari script[1]. Furthermore, some important documents in Maharashtra like the 7/12 extract are exclusively in Marathi in most cases. The data that is stored is huge both in volume and in terms of the physical space it consumes. This is very inefficient and the process of making it digital is laborious.
The complexity of handwritten characters is greater than that of printed characters due to several reasons:

i. Each individual has a diversified writing style causing substantial deviation in character strokes and overall handwriting resulting in a number of variations.

ii. There exist many letters in Devanagari that are similar in both Form and shape.

iii. The inclusion of "matra" and consonant clusters ("jodshabda") in letters adds complexity to the task of classifying handwritten characters. Therefore, using a general classifier to recognize handwritten characters written by multiple writers may not always be feasible.[1]

In addition to the challenges mentioned, there is also the issue of accuracy when it comes to digitizing the Devanagari script. This script has many complex characters and ligatures, which can be difficult for automated systems to recognize and convert accurately. Even small errors in conversion can have a significant impact on the meaning of the text.

To overcome these challenges, researchers and developers are exploring different approaches such as optical character recognition (OCR) and machine learning algorithms to develop more accurate and efficient automated systems for digitizing Devanagari script. These systems can not only save time and resources but can also improve the accuracy and accessibility of important documents.

Moreover, the need for digitization of the Devanagari script is not limited to just government offices and legal documents. There is a growing demand for digital content in regional languages for education, literature, and entertainment purposes. The lack of automated systems for the Devanagari script can limit the availability and accessibility of such content.

Therefore, there is a need for more research and development in the field of automated Devanagari script recognition and conversion. This can not only benefit the government and legal institutions but also promote the growth and development of regional language content in the digital era.

1.0.1 Problem Definition

The Devanagari script is widely used in India, especially in government offices and legal documents. However, digitizing old and handwritten documents in this script can be a challenging and laborious task, and the lack of automated systems for Devanagari script recognition and conversion can limit the accessibility and availability of important information. This creates a need for more research and development in this field to improve the accuracy and efficiency of automated systems for digitizing the Devanagari script. So the problem statement goes as : " The user wants to digitize a piece of document written in the Devanagari script."

1.0.2 Objectives

- To make an app that digitizes handwritten Devanagari in real-time. We plan on implementing that by using cloud services and hosting the model that we have developed on the cloud.

- To convert the text into a digitally editable file format. The CNN model that we will develop using python and keras [11] libraries along with segmentation using different classification techniques will be used to Digitize the file. Tensorflow [10] will be used to implement all of the backend code.

- To enable users to use the app anytime, anywhere. This will be done with the help of a flutter app that will serve as the front-end of the project.

- To bring relevance back to the local languages in this digital age. By having a method to digitize languages like Hindi and Marathi, the use of these languages can be promoted thus bringing relevance to them.

1.0.3 Scope

- This system can be used to digitize handwritten and printed documents written in the Devanagari script.
- It can accurately recognize and convert complex Devanagari characters and ligatures into digital format.
- The system can be used in government offices, legal institutions, and other organizations that deal with Devanagari script documents.
- It can also be used for digitizing historical and cultural documents written in Devanagari script, thereby preserving them for future generations.
- The proposed system can also benefit the growth and development of regional language content in the digital era, promoting education, literature, and entertainment in Devanagari script languages.

Chapter 2

Literature Review

Pande, S.D., et al, in [1], explain how they have used CNN with 4 convolutional, 2 max-pooling, 3 drop out, a flatten and 2 dense layers. This has helped them avoid over-fitting and achieve results more accurately and with less training time. The proposed approach is to implement a DHTR (Devanagari Handwritten Text Recognition) system using the following steps:

1. Segment the input handwritten Devanagari text query image into individual characters.
2. Isolate each character and prepare it for comparison.
3. Assess the resemblance of each isolated character with the characters in the DHTR system's repository.
4. Compare the characters within an anticipated range of resemblance to improve the accuracy of the recognition.

By implementing these steps, the proposed approach aims to accurately recognize handwritten Devanagari text. This approach uses the dataset DHCD[7] and has only 46 classes which do not include the characters that have 'matra' in them. In this work, a proposed system is presented that achieves better performance in handwritten character recognition. The system uses statistical features computed from isolated characters in the input query image, and a character class repository of limited diversity is utilized, which includes very few homogeneous patterns. The proposed approach outperforms other handwritten character recognition systems in terms of accuracy and performance. Additionally, a conflict resolution step is incorporated which reduces ambiguity in the recognition of words. Overall, this work presents a promising approach to improving the accuracy and efficiency of handwritten character recognition systems.

Deore, S.P., Pravin, A. in [2], achieve higher recognition rates with the Devanagari Handwritten Dataset. This work purely focuses on the recognition of the characters and improving the efficiency of doing it. The results of this study demonstrate the effectiveness of incorporating aggressive runtime data augmentation and regularization techniques, such as Dropout and Batch Normalization, to improve the performance of the DHCRS model. The authors were able to achieve the best classification accuracy of 96.55% using a fine-tuned VGG16 architecture on their trivial database. Furthermore, the proposed approach was shown to perform better on two Indic scripts, indicating the potential for broader application and

improved accuracy in character recognition tasks.

K. Dutta, P. Krishnan, M. Mathew and C. V. Jawahar in [3], use a hybrid CNN-RNN architecture to find faster recognition of the Devanagari Handwritten Character Datasets. In this study, the authors not only proposed a new dataset called the IIIT-HW-Dev dataset but also benchmarked it using a CNN-RNN hybrid network. They demonstrated the effectiveness of fine-tuning the network using both synthetic and real handwritten data, resulting in improved word recognition performance. By applying their fine-tuning strategy on the CNN-RNN hybrid network, the authors were able to achieve state-of-the-art results on the RoyDB dataset. This suggests that incorporating additional training data and fine-tuning techniques can significantly enhance the performance of handwriting recognition systems. The authors also explored the use of data augmentation techniques to improve the performance of the recognition system. They found that introducing small variations in the training data, such as rotations and translations, helped to make the model more robust and improved its generalization performance. Additionally, the authors experimented with different CNN architectures, including VGG-16 and ResNet, and found that a modified VGG-16 architecture was the best performing model for their dataset. Overall, this study highlights the importance of data augmentation and fine-tuning techniques in improving the accuracy of handwriting recognition systems, and suggests that hybrid CNN-RNN architectures can be effective for recognizing Devanagari characters.

In [4] by Patil, Jyoti A., and Dr S. R. Patil, an off-line handwritten Devanagari character recognition system with Neural Network has been described. The paper describes a system that can be applied to recognize handwritten names, read documents, and convert any handwritten document into machine-readable text. The system has achieved a high accuracy level of 98%, making it effective in accurately recognizing and digitizing handwritten text. The proposed system uses a feed-forward back-propagation neural network model to recognize Devanagari characters. The input to the network is a preprocessed image of the handwritten character, and the output is the recognized character. The neural network model consists of an input layer, a hidden layer, and an output layer. The input layer has 900 neurons, which correspond to a 30x30 pixel image. The hidden layer has 300 neurons, which act as feature extractors from the input image. The output layer has 36 neurons, each corresponding to a Devanagari character class. The system also implements a feature extraction module, which includes a segmentation module, normalization module, and thinning module. The segmentation module is used to separate individual characters from the input image, while the normalization and thinning modules are used to preprocess the image and extract relevant features. The system's high accuracy in recognizing handwritten Devanagari characters demonstrates its effectiveness and potential for use in various applications, including digitizing handwritten documents and automated recognition of handwritten Devanagari text.

I. Kissos and N. Dershowitz, in [5], discussed the utilization of machine learning techniques to enhance OCR (Optical Character Recognition) accuracy. The approach taken by the researchers in this study involved the combination of features from a language model, OCR model, and document context to improve the accuracy of OCR word recognition. By correcting misspelled OCR words using these combined features, they were able to create a reliable spelling model that can be used for different languages and domains. This method

allowed for greater accuracy in OCR word recognition, which is an essential component of many applications involving text analysis and processing. One of the significant advantages of the approach proposed by Kissos and Dershowitz in [5] is its flexibility and applicability to different languages and domains. By combining features from various models and considering document context, their method can handle a wide range of OCR tasks, including those involving handwritten text and historical documents. This flexibility is particularly valuable in domains where there is a limited amount of training data available, such as in historical or low-resource languages. The ability to leverage features from different models and contextual information also makes the approach suitable for handling OCR errors caused by various factors, such as degradation of text quality due to age or damage to the document.

P. K. Sonawane and S. Shelke presented an approach in [6] that involved fine-tuning a CNN using transfer learning. Their method achieved a validation accuracy of 94.49% and a test accuracy of 95.46%. The training curves demonstrated that the training accuracy exceeded 90% in just three epochs. The results indicated that transfer learning is a more effective option for faster and better training with a limited number of training samples. The following are some key points about their approach:

- They used a pre-trained CNN as a feature extractor and added a fully connected layer on top of it for classification.
- They used the Devanagari Handwritten Character Dataset, which has 46 classes, for training and testing.
- They fine-tuned the pre-trained CNN by training it on the Devanagari dataset for a few epochs and achieved a validation accuracy of 94.49% and a test accuracy of 95.46
- They demonstrated that the training accuracy exceeded 90% in just three epochs, which indicated that transfer learning is a more effective option for faster and better training with a limited number of training samples

Seba Susan* and Jatin Malhotra in [15] have used basic 2 layer CNN to recognise characters in Devanagari script. They used MNIST dataset which has 46 classes and got an accuracy of 97%. They suggested using deep learning techniques to analyze image quadrants by utilizing the hidden state activations obtained from convolutional neural networks trained on five different image quadrants. The scope of their proposed method is limited to character recognition in the Devanagari script using a basic CNN architecture and the MNIST dataset. However, their recommendation to utilize deep learning techniques for analyzing image quadrants could potentially be applied to other Devanagari character recognition systems.

Chapter 3

Project Design

3.0.1 Existing System

- There are several existing systems for handwritten character recognition in the Devanagari script. These systems use a combination of different techniques such as convolutional neural networks (CNN), recurrent neural networks (RNN), transfer learning, and machine learning. Some systems also incorporate data augmentation and regularization techniques such as Dropout and Batch Normalization to improve the performance of the model. Additionally, these systems may use different datasets for training and testing, with varying numbers of classes and character types. Despite differences in approach, all of these systems aim to accurately recognize handwritten Devanagari characters and improve the efficiency and accuracy of handwriting recognition systems.

- There are several potential drawbacks to the existing Devanagari handwriting recognition systems. One major limitation is their performance on characters with "matra," which are diacritical marks used in the Devanagari script to modify the pronunciation of letters. Some systems have limited diversity in their character class repository, which can lead to reduced accuracy in recognizing handwritten characters. Another issue is the reliance on large datasets for training, which can require significant computational resources and may not be feasible in certain contexts. Additionally, some systems may be prone to overfitting, which can lead to reduced generalization performance. Finally, some systems may not perform well in real-world scenarios with variations in handwriting styles, writing tools, and lighting conditions.

3.0.2 Proposed System

The proposed workflow suggests the following steps:

1. User uploads image: The user uploads an image of a document containing the Devanagari script in real time. This image will be used as the input for the conversion model.

2. Pre-processing of data: Before the image can be converted, some pre-processing is necessary to make the image more suitable for conversion. This includes the following steps:

- Identify borders: The first step is to identify the borders of the input image. This is important because it helps the model know where to focus its attention and avoid analyzing irrelevant parts of the image.
- Convert to greyscale: Next, the image is converted to greyscale. This is done to improve visibility and make it easier for the model to identify different parts of the image.
- Normalize image size: The size of the image is normalized to a specific size. This is important because the model needs to work with images of a consistent size to be effective.
- Remove noise: Any noise present in the image is removed. This could be things like specks or scratches that could interfere with the model's ability to accurately analyze the image.
- Correct skew: Finally, any skew or distortion in the image is corrected. This is important because it ensures that the characters in the image are correctly aligned and spaced, which makes them easier for the model to recognize.

3. Segmentation of characters: After the pre-processing is complete, the next step is to segment the characters in the image. This involves identifying individual characters within the words and separating them out so that they can be recognized separately by the model.

4. Character recognition: Once the characters have been segmented, the model can begin the process of recognizing them. This involves analyzing the features of each character and comparing them to a database of known characters to determine which characters are present in the image.

5. Output: Finally, the recognized characters are output as text in digital format, which can then be stored or processed as needed.

It is important to note that the accuracy of the character recognition process is heavily dependent on the quality of the input image and the pre-processing steps. Poor quality images or incorrect pre-processing can lead to inaccuracies or errors in the recognition process. Therefore, it is essential to have a robust pre-processing step that can handle various types of input images and effectively prepare them for the recognition process. Additionally, it is important to have a comprehensive database of known characters and the ability to continuously update it to ensure accurate recognition of even the rarest characters.

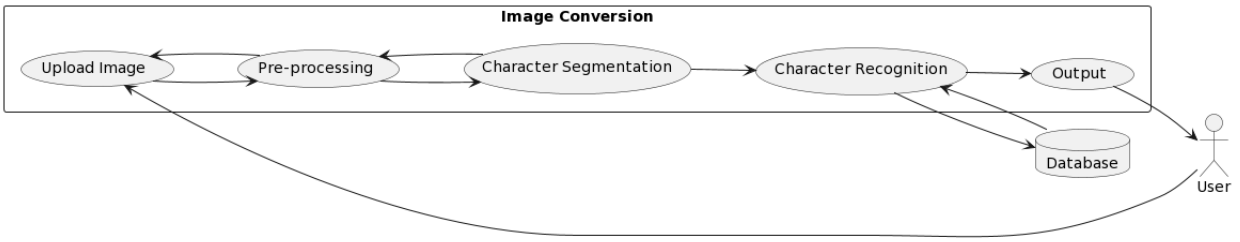


Figure 3.1: Use Case Diagram for the User

3.0.3 System Diagram

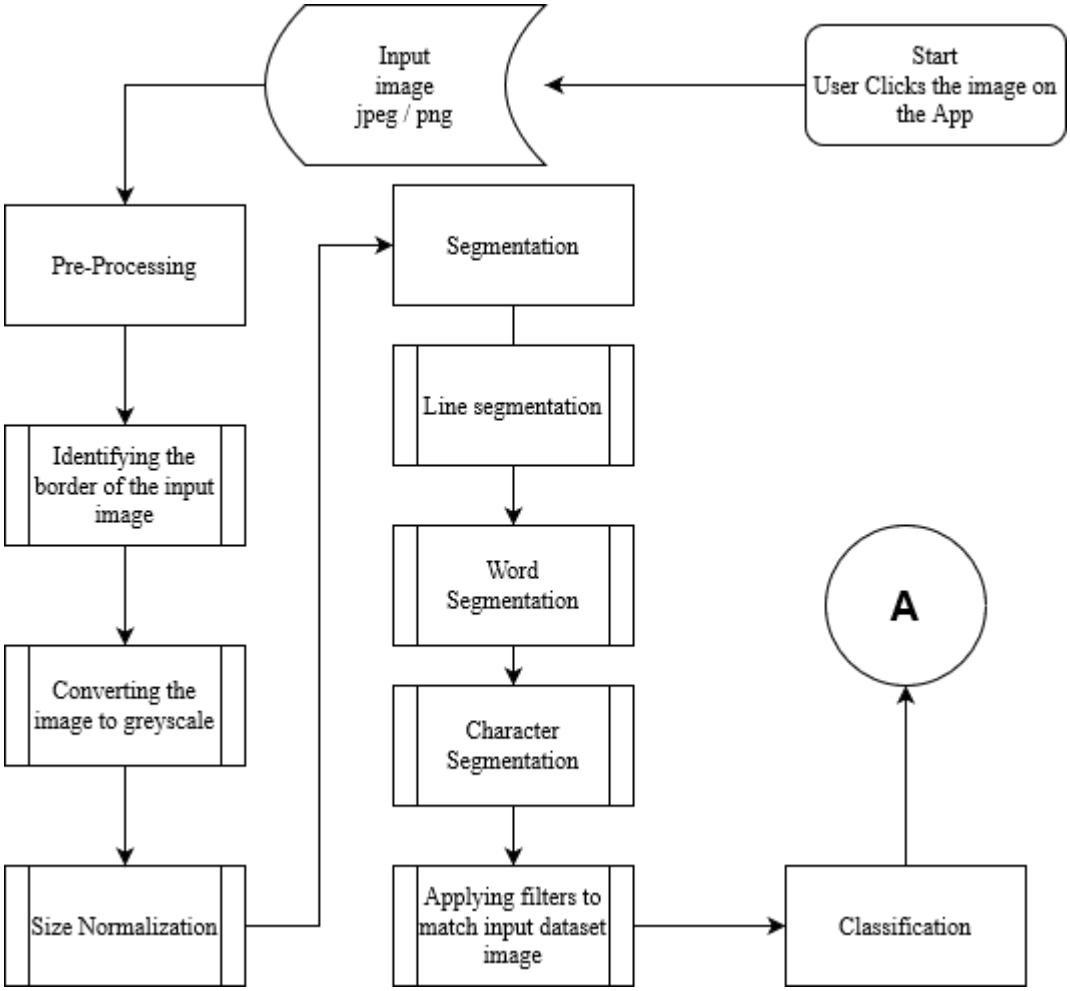


Figure 3.2: Segmentation

This diagram shows how the segmentation process is done in the system.

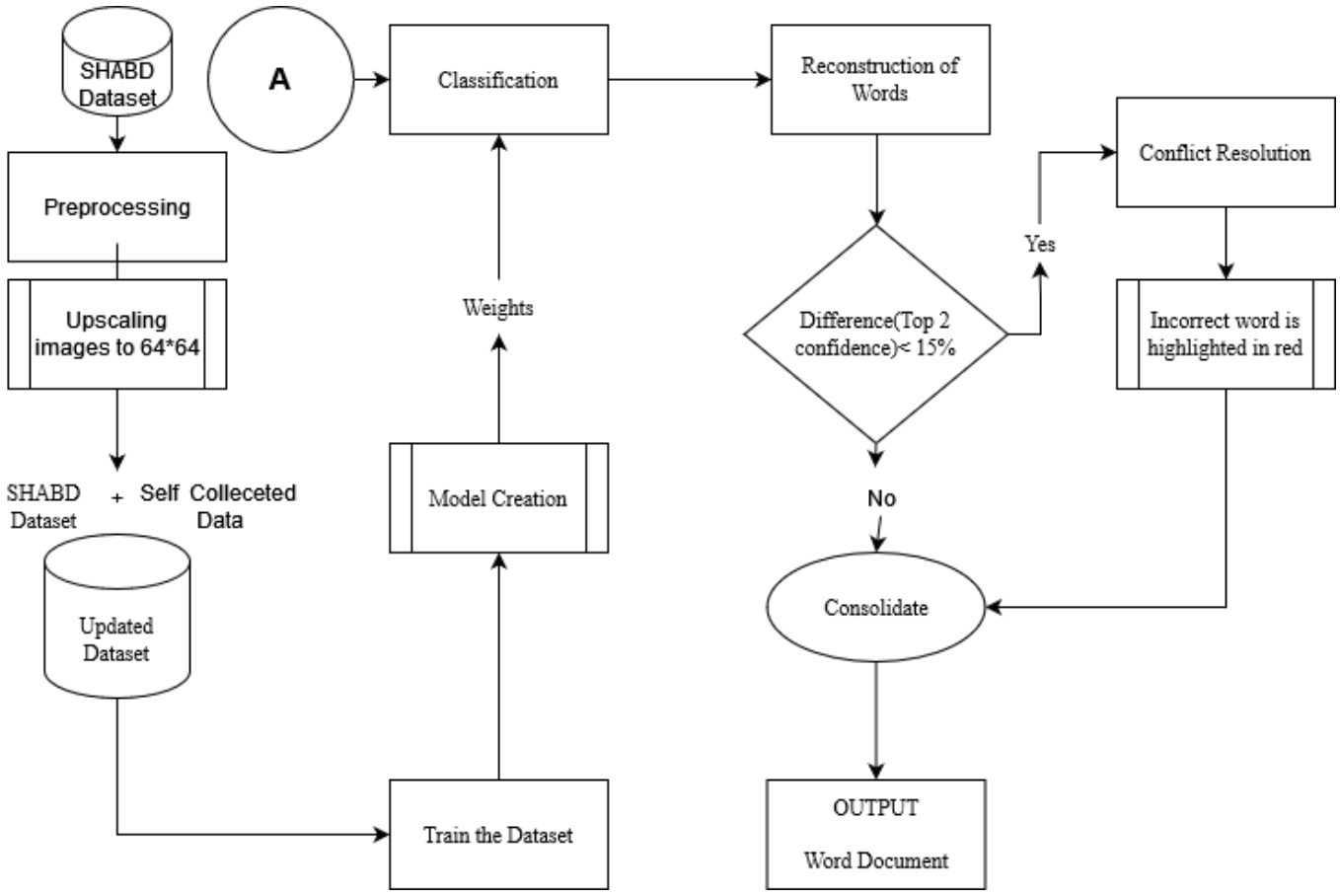


Figure 3.3: Classification

This diagram describes how the actual recognition of characters is done via classification.

Chapter 4

Project Implementation

4.0.1 Technology Stack:

- Frontend :
  - Flutter 3.3.2
  - Dart 2.18.1
- Backend:
  - Python 3.11.7
  - libraries and packages used:
  - keras and tensorflow
  - teserract ocr
  - Cloud Storage : Google cloud

4.0.2 Interface:

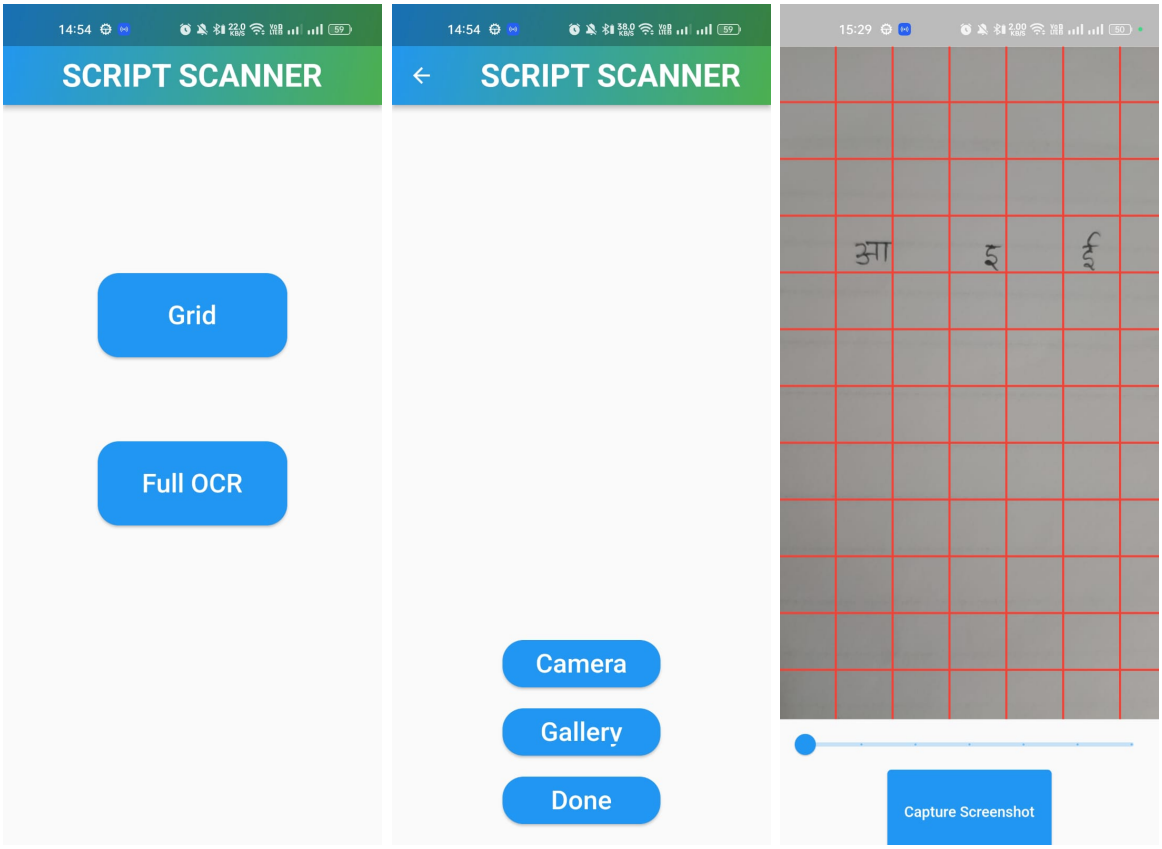


Figure 4.1: App Interface

4.0.3 Code Snippets:

Importing all the dependencies:
All of the required libraries are imported in this step.

```
In [1]:
import os, types
import matplotlib.pyplot as plt
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Convolution2D,MaxPooling2D,Flatten,BatchNormalization,LeakyReLU, Dropout
import tensorflow as tf
from keras.callbacks import ModelCheckpoint
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from keras.callbacks import ModelCheckpoint,EarlyStopping

In [2]:
train_datagen=ImageDataGenerator(rescale=1./255,zoom_range=0.2)

In [3]:
test_datagen=ImageDataGenerator(rescale=1./255)

In [4]:
x_train = train_datagen.flow_from_directory('C:/Users/USER/Desktop/Major/Data HW/MegaData/train', target_size = (64, 64)
                                            batch_size = 256, class_mode = 'categorical')
x_test = test_datagen.flow_from_directory('C:/Users/USER/Desktop/Major/Data HW/MegaData/test', target_size = (64, 64),
                                            batch_size = 256, class_mode = 'categorical')

Found 252850 images belonging to 459 classes.
Found 67447 images belonging to 459 classes.
```

Figure 4.2: Snippet 1

Loading in the data: The data is loaded in the model from our dataset.

```
In [5]: index=list(x_train.class_indices.keys())
        print(index)
```

```
['_', 'fullstop', ''', ':', 'अ', 'अं', 'अः', 'आ', 'इ', 'ई', 'उ', 'ऊ', 'ऋ', 'ए', 'ऐ', 'ओ', 'औ', 'क', 'कं', 'कः', 'का',
'कि', 'की', 'कु', 'कू', 'के', 'कै', 'को', 'कौ', 'क्ष', 'क्षं', 'क्षः', 'क्षा', 'क्षि', 'क्षी', 'क्षु', 'क्षू', 'क्षे', 'क्षै', 'क्षो', 'क्षौ', 'ख',
'खं', 'खः', 'खा', 'खि', 'खी', 'खु', 'खू', 'खे', 'खै', 'खो', 'खौ', 'ग', 'गं', 'गः', 'गा', 'गि', 'गी', 'गु', 'गू', 'गे', 'गै',
'गो', 'गौ', 'घ', 'घं', 'घः', 'घा', 'घि', 'घी', 'घु', 'घू', 'घे', 'घै', 'घो', 'घौ', 'ङ', 'च', 'चं', 'चः', 'चा', 'चि', 'ची', 'चु',
'चू', 'चे', 'चै', 'चो', 'चौ', 'छ', 'छं', 'छः', 'छा', 'छि', 'छी', 'छु', 'छू', 'छे', 'छै', 'छो', 'छौ', 'ज', 'जं', 'जः', 'जा',
'जि', 'जी', 'जु', 'जू', 'जे', 'जै', 'जो', 'जौ', 'ज्ञ', 'ज्ञं', 'ज्ञः', 'ज्ञा', 'ज्ञि', 'ज्ञी', 'ज्ञु', 'ज्ञू', 'ज्ञे', 'ज्ञै', 'ज्ञो', 'ज्ञौ', 'झ',
'झं', 'झः', 'झा', 'झि', 'झी', 'झु', 'झू', 'झे', 'झै', 'झो', 'झौ', 'ञ', 'ञं', 'ञः', 'ञा', 'ञि', 'ञी', 'ञु', 'ञू', 'ञे', 'ञै',
'ञो', 'ञौ', 'ट', 'टं', 'टः', 'टा', 'टि', 'टी', 'टु', 'टू', 'टे', 'टै', 'टो', 'टौ', 'ठ', 'ठं', 'ठः', 'ठा', 'ठि', 'ठी', 'ठु', 'ठू',
'ठे', 'ठै', 'ठो', 'ठौ', 'ड', 'डं', 'डः', 'डा', 'डि', 'डी', 'डु', 'डू', 'डे', 'डै', 'डो', 'डौ', 'ढ', 'ढं', 'ढः', 'ढा', 'ढि', 'ढी', 'ढु',
'ढू', 'ढे', 'ढै', 'ढो', 'ढौ', 'ण', 'णं', 'णः', 'णा', 'णि', 'णी', 'णु', 'णू', 'णे', 'णै', 'णो', 'णौ', 'त', 'तं', 'तः', 'ता',
'ति', 'ती', 'तु', 'तू', 'ते', 'तै', 'तो', 'तौ', 'त्र', 'थ', 'थं', 'थः', 'था', 'थि', 'थी', 'थु', 'थू', 'थे', 'थै', 'थो', 'थौ', 'द',
'दं', 'दः', 'दा', 'दि', 'दी', 'दु', 'दू', 'दे', 'दै', 'दो', 'दौ', 'ध', 'धं', 'धः', 'धा', 'धि', 'धी', 'धु', 'धू', 'धे', 'धै', 'धो',
'धौ', 'न', 'नं', 'नः', 'ना', 'नि', 'नी', 'नु', 'नू', 'ने', 'नै', 'नो', 'नौ', 'प', 'पं', 'पः', 'पा', 'पि', 'पी', 'पु', 'पू', 'पे',
'पै', 'पो', 'पौ', 'फ', 'फं', 'फः', 'फा', 'फि', 'फी', 'फु', 'फू', 'फे', 'फै', 'फो', 'फौ', 'ब', 'बं', 'बः', 'बा', 'बि', 'बी', 'बु',
'बू', 'बे', 'बै', 'बो', 'बौ', 'भ', 'भं', 'भः', 'भा', 'भि', 'भी', 'भु', 'भू', 'भे', 'भै', 'भो', 'भौ', 'म', 'मं', 'मः', 'मा',
'मि', 'मी', 'मु', 'मू', 'मे', 'मै', 'मो', 'मौ', 'य', 'यं', 'यः', 'या', 'यि', 'यी', 'यु', 'यू', 'ये', 'यै', 'यो', 'यौ', 'र', 'रं',
'रः', 'रा', 'रि', 'री', 'रु', 'रू', 'रे', 'रै', 'रो', 'रौ', 'ल', 'लं', 'लः', 'ला', 'लि', 'ली', 'लु', 'लू', 'ले', 'लै', 'लो', 'लौ', 'ळ
', 'ळं', 'ळः', 'ळा', 'ळि', 'ळी', 'ळु', 'ळू', 'ळे', 'ळै', 'ळो', 'ळौ', 'व', 'वं', 'वः', 'वा', 'वि', 'वी', 'वु', 'वू', 'वे', 'वै',
'वो', 'वौ', 'श', 'शं', 'शः', 'शा', 'शि', 'शी', 'शु', 'शू', 'शे', 'शै', 'शो', 'शौ', 'ष', 'षं', 'षः', 'षा', 'षि', 'षी', 'षु', 'षू',
'षे', 'षै', 'षो', 'षौ', 'स', 'सं', 'सः', 'सा', 'सि', 'सी', 'सु', 'सू', 'से', 'सै', 'सो', 'सौ', 'ह', 'हं', 'हः', 'हा', 'हि', 'ही',
'हु', 'हू', 'हे', 'है', 'हो', 'हौ', 'ा', 'ि', 'ी', 'ु', 'ू', 'े', 'ै', 'ो', 'ौ', '।', '०', '१', '२', '३', '४', '५', '६',
७', '८', '९']
```

Figure 4.3: Snippet 2

Making the CNN model:
This is the CNN model that we have created to recognize the characters in our project.

```
In [6]: model=Sequential()
        model.add(Convolution2D(32,(3,3),input_shape=(64,64,3),activation=LeakyReLU(),padding='same'))
        model.add(Convolution2D(32,(3,3),activation=LeakyReLU(),padding='same'))
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(Dropout(0.25))
        model.add(Convolution2D(64,(3,3),activation=LeakyReLU(),padding='same'))
        model.add(Convolution2D(64,(3,3),activation=LeakyReLU(),padding='same'))
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(Dropout(0.25))
        model.add(Convolution2D(128,(3,3),activation='relu',padding='same'))
        model.add(Convolution2D(128,(3,3),activation='relu',padding='same'))
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(Dropout(0.25))
        model.add(Convolution2D(256,(2,2),activation='relu',padding='same'))
        model.add(Convolution2D(256,(2,2),activation='relu',padding='same'))
        model.add(Convolution2D(256,(3,3),activation='relu',padding='same'))
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(Dropout(0.25))
        model.add(Flatten())
        model.add(Dense(512))
        model.add(Dropout(0.25))
        model.add(Dense(256))
        model.add(Dropout(0.25))
        model.add(Dense(459,activation='softmax'))
```

Figure 4.4: Snippet 3

Loading Model:

```
In [8]:  model.compile(loss='categorical_crossentropy',optimizer='rmsprop',metrics=['accuracy'])

In [10]: filepath = 'new_data2_acc.hdf5'
         filepath2 = 'new_data2_loss.hdf5'

         checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy',verbose=1, save_best_only=True,mode='max')
         checkpoint2 = ModelCheckpoint(filepath=filepath2, monitor='val_loss',verbose=1, save_best_only=True,mode='min')
         stop=EarlyStopping(monitor="val_loss",patience=5,verbose=1,mode="min",restore_best_weights=True)
         callbacks = [checkpoint,checkpoint2,stop]
         output=model.fit_generator(x_train,steps_per_epoch=len(x_train),callbacks=callbacks,epochs=70,validation_data=x_test,val

         C:\Users\USER\AppData\Local\Temp\ipykernel_13836\3276262421.py:8: UserWarning: `Model.fit_generator` is deprecated and
         will be removed in a future version. Please use `Model.fit`, which supports generators.
           output=model.fit_generator(x_train,steps_per_epoch=len(x_train),callbacks=callbacks,epochs=70,validation_data=x_test,
         validation_steps=len(x_test))
```

Figure 4.5: Snippet 4

```
In [13]: model1 = load_model(filepath)
         model2 = load_model(filepath2)
         x=image.img_to_array(image.load_img("C:/Users/USER/Desktop/Major/Data HW/MegaData/test/fullstop/fullstop_original_2023-03-1
         x = np.squeeze(x)
         x = np.stack((x,) * 3, axis=-1)
         print(x.shape)
         x=np.expand_dims(x,axis=0)/255
         print(index[np.argmax(model1.predict(x),axis=1)[0]])
         print(index[np.argmax(model2.predict(x),axis=1)[0]])

         (64, 64, 3)
         1/1 [==============================] - 0s 38ms/step
         fullstop
         1/1 [==============================] - 0s 35ms/step
         fullstop

In [ ]:

In [14]: x=image.img_to_array(image.load_img("C:/Users/USER/Desktop/Major/Data HW/MegaData/test/घे/36378_rlt.png",color_mode="graysca
         x = np.squeeze(x)
         x = np.stack((x,) * 3, axis=-1)
         print(x.shape)
         x=np.expand_dims(x,axis=0)/255
         print(index[np.argmax(model1.predict(x),axis=1)[0]])
         print(index[np.argmax(model2.predict(x),axis=1)[0]])

         (64, 64, 3)
         1/1 [==============================] - 0s 43ms/step
         घे
         1/1 [==============================] - 0s 33ms/step
         घे
```

Figure 4.6: Snippet 5

Recognizing the characters: Using the CNN, Characters are recognized here.

In [83]:

```
from tensorflow.keras.preprocessing import image
import os
import numpy as np
from sklearn.metrics import classification_report

# Set test data directory and target size
test_data_dir = 'C:/Users/USER/Desktop/Major/Data HW/MegaData/test'
target_size = (64, 64)

# Initialize lists to store test data and labels
x_test = []
y_test = []

# Loop over subdirectories in test data directory
for subdir in os.listdir(test_data_dir):
    # Get class label from subdirectory name
    class_label = subdir

    # Loop over images in subdirectory
    for filename in os.listdir(os.path.join(test_data_dir, subdir)):
        # Load image and resize it to target size
        img = image.load_img(os.path.join(test_data_dir, subdir, filename), target_size=target_size)

        # Convert image to numpy array and rescale pixel values
        img_array = image.img_to_array(img) / 255.0

        # Append image array and class label to test data and labels lists
        x_test.append(img_array)
        y_test.append(class_label)

# Convert test data and labels lists to numpy arrays
x_test = np.array(x_test)
y_test = np.array(y_test)

# Get predicted class probabilities
y_pred = model.predict(x_test)

# Convert predicted probabilities to class labels
y_pred_classes = np.argmax(y_pred, axis=1)
```

Figure 4.7: Snippet 6

In [85]:

```
from keras.preprocessing.image import ImageDataGenerator

# Create ImageDataGenerator for test data
test_datagen = ImageDataGenerator(rescale=1./255)

# Create DirectoryIterator for test data
test_iterator = test_datagen.flow_from_directory(
    'C:/Users/USER/Desktop/Major/Data HW/MegaData/test',
    target_size=(64, 64),
    batch_size=256,
    class_mode='categorical'
)

# Get class label to index mapping
class_indices = test_iterator.class_indices

# Convert true class labels to integer labels
y_test = np.array([class_indices[label] for label in y_test])
# Get classification report
report = classification_report(y_test, y_pred_classes, output_dict=True)

# Get class labels
class_labels = report.keys()

# Sort classes based on f1-score
sorted_classes = sorted(class_labels, key=lambda x: report[x]['f1-score'])

# Print sorted classes with their f1-score
for class_name in sorted_classes:
    print(f"{class_name}: {report[class_name]['f1-score']}")
```

```
Found 67447 images belonging to 459 classes.
```

Figure 4.8: Snippet 7

The API code to process images on the docker container:

.

```python
from PIL import Image, ImageDraw
from tesserocr import PyTessBaseAPI, RIL
import numpy as np
from sklearn.base import check_array
import statistics
import docx
import os
import base64
from io import BytesIO
from tensorflow import keras
from tensorflow.keras.preprocessing import image
import cv2
import requests
from flask import Flask, request

app = Flask(__name__)
```

Figure 4.9: Backend 1

```python
    model = keras.models.load_model("model.hdf5")
    document = docx.Document()
    print("Total Images",len(encoded_imgs))
    for encoded_img in encoded_imgs:
        decoded_data = base64.b64decode(encoded_img)
        nparr = np.frombuffer(decoded_data, np.uint8)
        img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        _, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
        mask = np.zeros(img.shape, np.uint8)
        mask.fill(255)
        mask[thresh > 0] = [0, 0, 0]
        result = cv2.bitwise_and(mask, mask)
        cv2.imwrite("img.jpg", result)
        Text=""
        with PyTessBaseAPI(lang='hin') as api:
            api.SetImageFile("img.jpg")
            lines=api.GetComponentImages(RIL.TEXTLINE, 1)
            os.remove("img.jpg")
```

Figure 4.10: Backend 2

Continuation of the docker backend code:
Here the uploaded image is segmented and then recognized. The recognized characters are then reconstructed into words, sentences and paragraphs. This is converted into a word file and sent.

```
        for i in range(len(lines)):
            Line=""
            para = document.add_paragraph()
            lines[i][0].save( 'line'+str(i)+'.png', 'png')
            try:
                with PyTessBaseAPI(lang='hin') as api:
                    api.SetImageFile('line'+str(i)+'.png')
                    os.remove('line' + str(i) + '.png')
                    words=api.GetComponentImages(RIL.WORD, 1)
            except:
                continue
            if len(words)<1:
                continue
            for j in range(len(words)):
                Word=""
                words[j][0].save('line'+str(i)+'-word'+str(j)+'.png', 'png')
                word = Image.open('line'+str(i)+'-word'+str(j)+'.png')
                os.remove('line'+str(i)+'-word'+str(j)+'.png')
                vertical_hist = (np.sum(word,axis=0,keepdims=True)/255)[0]
                avg=np.full(len(vertical_hist), vertical_hist.mean())
                z=[]
                w=[]
                y=0
                thr=5
                per=20
                for I in range(0,len(vertical_hist)):
                    if I==0:
                        x=vertical_hist[0]
                        y=0
                    try:
                        if vertical_hist[I]==x:
                            x=vertical_hist[I]
                            if y>thr:
                                for J in reversed(range(thr)):
                                    z.append(vertical_hist[I-J]*255)
                                    w.append(I-J)
                                y=0
                            else:
                                y=y+1
```

```
                for I in range(len(totalw)+1):
                    if I==len(totalw):
                        w=word.size[0]-1
                    else:
                        w=totalw[I]
                    char=word.crop((x, 0,w, y))
                    try:
                        char.save('line'+str(i)+'-word'+str(j)+'-char'+str(I)+'.png')
                    except:
                        continue
                    x = image.img_to_array(image.load_img('line' + str(i) + '-word' + str(j) + '-char' + str(I) + '.png', color_mode="rgb", target_size=(64, 64, 3)))
                    os.remove('line' + str(i) + '-word' + str(j) + '-char' + str(I) + '.png')
                    x = np.expand_dims(x, axis=0) / 255
                    predictions = model.predict(x)
                    top_indices = np.argsort(np.squeeze(predictions))[-5:][::-1]
                    top_indices = np.argsort(np.squeeze(predictions))[-5:][::-1]
                    probabilities = predictions[0, top_indices] / np.sum(predictions[0, top_indices])
                    class_labels = [index[i] for i in top_indices]
                    sorted_class_labels = [x for _, x in sorted(zip(probabilities, class_labels), reverse=True)]
                    sorted_probabilities = sorted(probabilities, reverse=True)
                    Word=Word+sorted_class_labels[0]
                    char_run = para.add_run(sorted_class_labels[0])
                    if sum(sorted_probabilities)/len(sorted_probabilities)>0.3*sorted_probabilities[0]:
                        char_run.font.color.rgb = docx.shared.RGBColor(255, 0, 0)
                    if I!=len(totalw):
                        x=totalw[I]
                word_run = para.add_run(" ")
                Line=Line+Word+" "
            Text=Text+Line[:-1]+"\n"
        document.add_paragraph().add_run().add_break(docx.enum.text.WD_BREAK.PAGE)
    document.save('result.docx')
    with open('result.docx', 'rb') as doc_file:
        encoded_file = base64.b64encode(doc_file.read())
    print("\n \n \n \n")
    print("Encoded file", encoded_file)
    print("\n \n \n \n")
    return {
            'word_file': encoded_file.decode('utf-8')
        }
```

Figure 4.11: Backend 3

Chapter 5

Testing

Testing is an organized summary of testing objectives, activities, and results. It is created and used to help stakeholders (product managers, analysts, testing teams, and developers) understand product quality and decide whether a product, feature or defect resolution is on track for release. Test documentation includes all files containing information on the testing team's strategy, progress, metrics, and results. The combination of all available data measures the testing effort, control test coverage, and track future project requirements.

5.1 Functional Testing

5.1.1 Segmentation

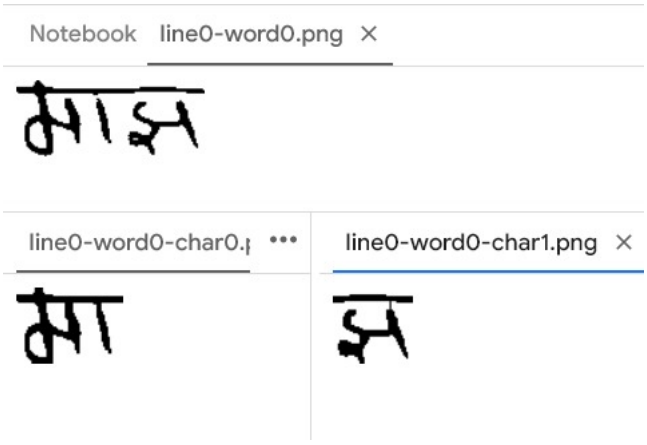


Figure 5.1: Character segmentation

5.1.2 Character Recognition

Recognition of Characters: This shows the recognition of single characters in the program.

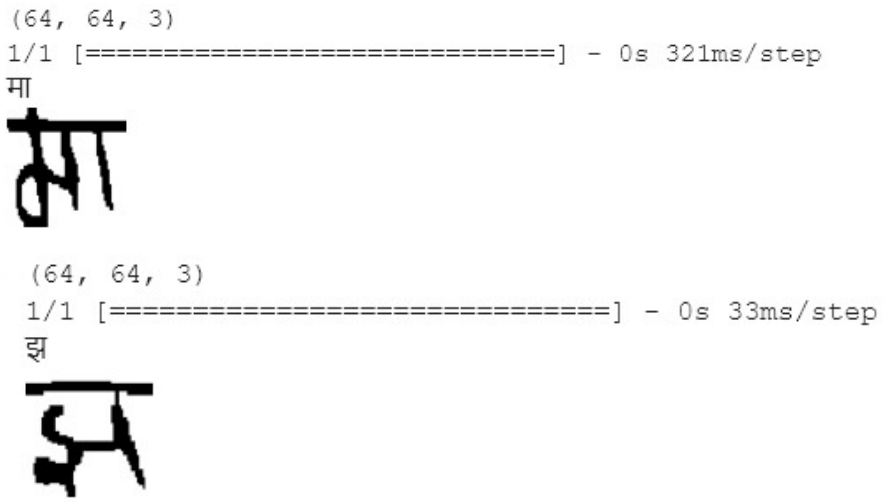


Figure 5.2: Snippet

5.1.3 Various Test Cases:

Test Case	Expected Result	Actual Result
1	Image uploaded successfully	Image uploaded successfully
2	Image is pre-processed correctly	Image pre-processing failed
3	Borders of image identified correctly	Borders not identified; image analysis incorrect
4	Image converted to greyscale successfully	Image converted
5	Image normalized to correct size	Image normalized
6	Noise removed from image	Noise still present, recognition accuracy reduced
7	Skew correction performed correctly	Skew correction failed; characters misaligned
8	Characters segmented accurately	Characters not segmented properly; recognition accuracy reduced
9	Character recognition accurate	Character recognition inaccurate, errors in output
10	Output text in correct digital format	Output in incorrect format, further processing required
11	Robust pre-processing handles various input images	Pre-processing fails on certain image types
12	Comprehensive database ensures accurate recognition	Database missing some characters, recognition accuracy reduced

Figure 5.3: Test Cases

Chapter 6

Result

The proposed system has yielded favourable results on our machines. The accuracy obtained is on average 84% and validation accuracy of about 97%. Despite having to train a large number of classes, the results were favourable. The model is trained on 384 classes as opposed to 46 classes in [1] and is trained in an average of 4 hours. The model was trained on 3 different machines 2 of which had integrated graphics and one having a dedicated graphics card.

PC	Specifications	Accuracy	Training Time(minutes)
PC 1	i7 9th gen 16 GB RAM	0.9742	130
PC 2	i3 11th gen 8 GB RAM	0.9818	230
PC 3	i5 10th gen 8 GB RAM	0.9761	226

Table 6.1: Computer Specifications of the Members.

This table shows the training time of our model done on different machines with different specifications. All of the machines had Windows 11 as their OS.

During the training process, the model's accuracy improves as the number of epochs increases. In our proposed system, a significant increase in accuracy during the first few epochs was observed, after which the accuracy growth plateaued. The validation accuracy also increased steadily during the initial epochs before stabilizing at around 97%.

These observations suggest that the model was able to effectively learn the features of the Devanagari characters and achieve high accuracy with relatively fewer epochs.

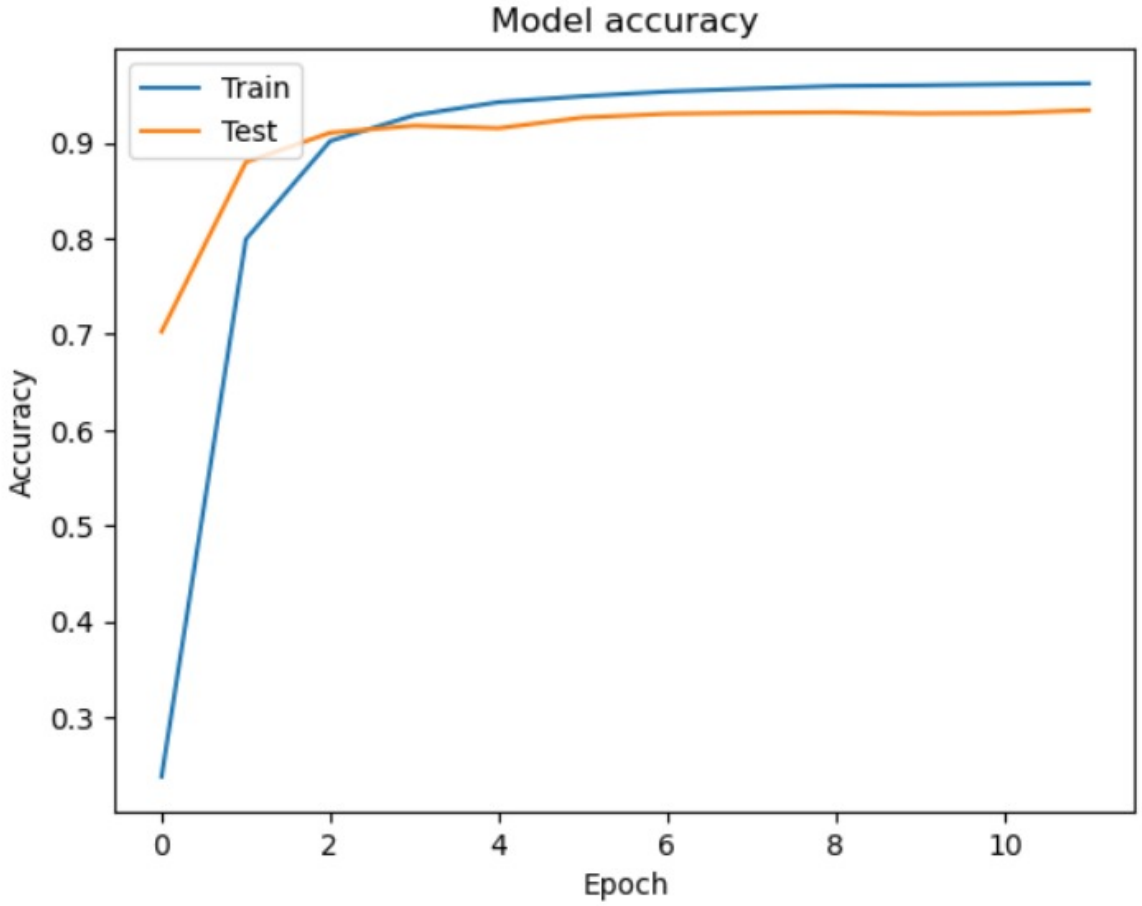


Figure 6.1: Epoch accuracy

The currently trained model has high accuracy. The summary of the CNN can be seen in the below table:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 32)	896
conv2d_1 (Conv2D)	(None, 64, 64, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	0
dropout (Dropout)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 64)	18496
conv2d_3 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_1 (Dropout)	(None, 16, 16, 64)	0
conv2d_4 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_5 (Conv2D)	(None, 16, 16, 128)	147584
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_2 (Dropout)	(None, 8, 8, 128)	0
conv2d_6 (Conv2D)	(None, 8, 8, 256)	131328
conv2d_7 (Conv2D)	(None, 8, 8, 256)	262400
conv2d_8 (Conv2D)	(None, 8, 8, 256)	590080
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_3 (Dropout)	(None, 4, 4, 256)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 512)	2097664
dropout_4 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
dropout_5 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 459)	117963

Total params: 3,617,771
Trainable params: 3,617,771
Non-trainable params: 0

Figure 6.2: CNN Summary

Chapter 7

Conclusions and Future Scope

There are a lot of handwritten Devanagari texts written in India now and thousands are currently being written. Most of them are in government offices or in academic settings. The process of digitization is mostly done manually with human effort which is prone to errors and is slow. There are a lot of causes that can wipe out a big chunk of information in the form of handwritten data like natural disasters such as floods, tsunamis, fires, pests and earthquakes or even human errors. Over time, information can be forgotten and lost forever if there are no digital copies available. Therefore, it is crucial to digitize handwritten texts. Our approach, along with previous methods, has demonstrated that digitizing documents can be done at a low cost with high accuracy and minimal errors. Our proposed method performs better than other handwritten character recognition systems in terms of accuracy and performance. Our approach is one step closer towards a perfect system with characters having 'matra' being recognized as well and the next foreseeable step is to have a model that recognizes 'jodshabda' or consonant clusters as well.

The present research demonstrates the feasibility of achieving high accuracy in digitizing Devanagari's handwritten text, covering a larger set of characters than previously explored in related works. This work explores 384 classes, surpassing previous studies in this area. Further improvements could involve expanding the dataset to include characters with consonant clusters or 'jodshabda,' which are connected characters. Another potential application for this work is to integrate it into text-to-speech software, providing blind individuals with the ability to read handwritten Devanagari text.
In addition, the proposed approach can be extended to recognize handwritten text in other languages, as the basic architecture and techniques used are not specific to the Devanagari script. The use of data augmentation and transfer learning can further improve the performance of the model with limited training data. Moreover, the model can be integrated into mobile applications for on-the-go digitization, enabling users to capture and digitize handwritten text in real time. These potential applications demonstrate the practicality and usefulness of the proposed approach in various domains.

Bibliography

[1] Pande, S. D., Jadhav, P. P., Joshi, R., Sawant, A. D., Muddebihalkar, V., Rathod, S., Gurav, M. N., & Das, S. (2022). "Digitization of handwritten Devanagari text using CNN transfer learning – A better customer service support". Neuroscience Informatics, 2(3), 100016. https://doi.org/10.1016/j.neuri.2021.100016

[2] Deore, S.P., Pravin, A. "Devanagari Handwritten Character Recognition using fine-tuned Deep Convolutional Neural Network on trivial dataset". Sādhanā 45, 243 (2020). https://doi.org/10.1007/s12046-020-01484-1

[3] K. Dutta, P. Krishnan, M. Mathew and C. V. Jawahar, "Offline Handwriting Recognition on Devanagari Using a New Benchmark Dataset," 2018 13th IAPR International Workshop on Document Analysis Systems (DAS), 2018, pp. 25-30, doi: 10.1109/DAS.2018.69.

[4] Patil, Jyoti A., and Dr S. R. Patil. "Optical Handwritten Devnagari Character Recognition Using Artificial Neural Network Approach." International Journal of Innovations in Engineering Research and Technology, vol. 5, no. 3, 2018, pp. 1-5.

[5] I. Kissos and N. Dershowitz, "OCR Error Correction Using Character Correction and Feature-Based Word Classification," 2016 12th IAPR Workshop on Document Analysis Systems (DAS), 2016, pp. 198-203, doi: 10.1109/DAS.2016.44.

[6] P. K. Sonawane and S. Shelke, "Handwritten Devanagari Character Classification using Deep Learning," 2018 International Conference on Information, Communication, Engineering and Technology (ICICET), 2018, pp. 1-4, doi: 10.1109/ICICET.2018.8533703.

[7] https://archive.ics.uci.edu/ml/datasets /Devanagari+Handwritten+Character+Dataset

[8] https://nanonets.com/blog/ocr-with-tesseract/

[9] https://likegeeks.com/python-image-processing/

[10] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. and Kudlur, M., 2016. TensorFlow: a system for Large-Scale machine learning. In 12th USENIX symposium on operating systems design and implementation (OSDI 16) (pp. 265-283).

[11] F. Chollet & others, Keras, Available at: https://github .com /fchollet /keras, 2015.

[12] Dataset used: https://www.kaggle.com/datasets/sushantshetty/shabd-complete-hindi-characters-dataset

[13] https://towardsdatascience.com/segmentation-in-ocr-10de176cf373

[14] https://en.wikipedia.org/wiki/Devanagari

[15] Susan, S., Malhotra, J. (2020). Recognising Devanagari Script by Deep Structure Learning of Image Quadrants. DESIDOC Journal of Library Information Technology, 40(05), 268-271. https://doi.org/10.14429/djlit.40.05.16336

Appendices

This section shows all the libraries and technologies that have been used in our project.

Appendix-I: Installation of all Libraries

7.0.1 Libraries Used

1. camera: A Flutter plugin for accessing the camera. It provides APIs to capture photos and videos.

2. image_picker: A Flutter plugin for picking images and videos from the device's gallery or camera.

3. http: A Dart package for making HTTP requests. It provides APIs for performing GET, POST, PUT, DELETE, and other HTTP methods.

4. open_file: A Flutter plugin for opening files with the default OS application. It supports various file types such as images, videos, PDFs, and more.

5. path_provider: A Flutter plugin for accessing the device's file system. It provides APIs for getting the app's documents directory, cache directory, and more.

7.0.2 Default Libraries in Dart:

1. dart:convert: A library for converting between Dart objects and JSON.
2. dart:typed_data: A library for working with byte buffers.
3. dart:ui: A library for creating and manipulating graphics and UI elements.

7.0.3 Default Libraries in Flutter:

1. package:flutter/rendering.dart: A library for rendering and painting UI elements.
2. dart:io: A library for working with files and sockets.

3. dart:async: A library for working with asynchronous operations.

4. package:flutter/material.dart: A library for creating UI elements and building user interfaces.

To use these libraries in your Flutter app, add them to the dependencies section of your project's **pubspec.yaml** file and run **flutter pub get** command in the terminal or command prompt in the project directory to install them.

7.0.4 Python Libraries Used:

1. tesserocr==2.5.2: A Python wrapper for Tesseract OCR library, which is used for optical character recognition.

2. gunicorn: A Python WSGI HTTP Server for UNIX. It is used to deploy Python applications to production.

3. numpy: A library for scientific computing with Python. It provides arrays, matrices, and mathematical functions for working with large datasets.

4. tensorflow: An open-source machine learning platform for building and training models. It is widely used for image and speech recognition, natural language processing, and more.

5. Pillow: A Python Imaging Library. It provides tools for working with images, such as image processing and manipulation.

6. scikit-learn: A Python library for machine learning. It provides tools for data mining and data analysis, including classification, regression, and clustering.

7. statistics: A Python built-in library for statistical calculations.

8. python-docx: A Python library for creating and updating Microsoft Word (.docx) files.

9. opencv-python-headless: A Python package for computer vision tasks. It provides tools for image and video processing, object detection, and more.

10. requests: A Python library for making HTTP requests. It provides a simple interface for sending HTTP/1.1 requests.

11. flask: A Python web framework for building web applications. It provides tools for handling HTTP requests and responses, routing URLs, and more.

12. pandas: A Python library for data manipulation and analysis. It provides tools for working with tabular data, such as filtering, sorting, and summarizing data.

13. torch: A PyTorch is an open-source machine learning framework for building and training neural networks. It provides tools for building and training models for a variety of tasks, including image classification, object detection, and natural language processing.

14. ESRGAN: A GitHub repository for the Enhanced Super-Resolution Generative Adversarial Networks (ESRGAN) project. It provides a pre-trained model for upscaling low-resolution images to high-resolution images.

To use these libraries in your Python project, you can install them using **pip** package manager by running **pip install (library-name)** command in your terminal or command prompt. You can also include them in a **requirements.txt** file and run **pip install -r requirements.txt** command to install all the libraries at once.

Make sure to import the libraries in your Python files using the import statement to use their functionalities in your project.

Publication

Paper entitled **"Developing Real-time Digitization System Using Deep Learning for Handwritten Documents"** is presented at **"2023 International Conference on Contemporary Challenges in Science and its Engineering Applications (IC3SEA 2023)"** by **"Atharv Joshi", " Siddhesh Puranik", "Niranjan Ram", "Sonal Balpande", "Jayashree Jha".**