TRIBHUVAN UNIVERSITY

Prime College

Nayabazar, Kathmandu, Nepal

A Project Report

On

"PrachaRead:

Newari Text Recognition"

SUBMITTED BY:

ARYAN KHADGI (5-2-410-255-2020)

POEM MAHARJAN (5-2-410-270-2020)

PRADYUMNA RAJBHANDARI (5-2-410-272-2020)

A Project Report Submitted in partial fulfillment of the requirement of **Bachelor of Science in Computer Science & Information Technology (BSc.CSIT) 7th Semester** of Tribhuvan University, Nepal

January, 2025

“PrachaRead:
Newari Text Recognition”
[CSC 412]

A project report submitted for the partial fulfillment of the requirement for the degree of Bachelor of Science in Computer science & Information Technology awarded by Tribhuvan University.

Submitted By

ARYAN KHADGI (5-2-410-255-2020)

POEM MAHARJAN (5-2-410-270-2020)

PRADYUMNA RAJBHANDARI (5-2-410-272-2020)

Submitted To

Prime College

Department of Computer science

Affiliated to Tribhuvan University

Khusibun, Nayabazar, Kathmandu

January, 2025

Date: 2nd Magh, 2081

SUPERVISOR'S RECOMMENDATION

It is my pleasure to recommend that a report on **"PRACHAREAD: NEWARI TEXT RECOGNITION"** has been prepared under my supervision by Aryan Khadgi, Poem Maharjan, Pradyumna Rajbhandari in partial fulfillment of the requirement of the degree of Bachelor of Science in Computer Science and Information Technology (BSc.CSIT). Their report is satisfactory to process for the future evaluation.

……

Mr. Sudan Prajapati

Supervisor

Department of Computer Science & IT

Prime College

Date: 2nd Magh, 2081

CERTIFICATE OF APPROVAL

This is to certify that this report has been read and recommended to the Department of Computer Science and Information Technology for acceptance of report entitled **"PRACHAREAD: NEWARI TEXT RECOGNITION"** submitted by Aryan Khadgi, Poem Maharjan, Pradyumna Rajbhandari in partial fulfillment for the degree of Bachelor of Science in Computer Science and Information Technology (BSc.CSIT), Institute of Science and Technology, Tribhuvan University.

……	**……**
Ms. Rolisha Sthapit	**Mr. Sailesh Karmacharya**
Vice Principal	**Research Coordinator**
……	**……**
Mr. Sudan Prajapati	**Mr.**
Supervisor	**External Examiner**

ACKNOWLEDGEMENT

We would like to extend our sincere gratitude to all those who contributed to the successful completion of this report. In particular, we express our deep appreciation to Mr. Sudan Prajapati, our project supervisor, for his invaluable guidance, thoughtful recommendations, and unwavering support, which greatly facilitated the coordination of our project, especially in the preparation of this report.

We are also grateful to the expert panels and our secondary supervisor for their constructive advice during our project presentation. Their feedback and insightful suggestions significantly enhanced the quality of our work.

Our heartfelt thanks go to our colleagues for their continuous encouragement and assistance throughout the project. Their support was instrumental in its completion.

Finally, we would like to thank everyone who contributed to the successful completion of this project. Your help, support, and motivation have been deeply appreciated.

With respect,

Aryan Khadgi (5-2-410-255-2020)

Poem Maharjan (5-2-410-270-2020)

Pradyumna Rajbhandari (5-2-410-272-2020)

ABSTRACT

Newari text is a low resource language which hasn't had much attention from the IT community. There are multiple different tools and methods for optical character recognition for English and other high resource languages. But for newari language the tools are limited. Newari text recognition is the process of recognizing handwritten characters and digital words from images and converting them into editable digital text. This project uses neural networks and computer vision techniques to convert the optical text to digital editable text. The project frontend was created using HTML, CSS and JavaScript while the backend was done using Django. The project uses CNN model to classify and convert the handwritten characters into digital editable characters. The CNN model gives an accuracy of 99%. The character dataset was self-created and numeral characters were combined with dataset of Kaggle. This model accurately recognizes the handwritten characters and gives their corresponding digital characters. To convert words into editable text, CRNN model was used due to its performance in sequence-to-sequence task. The CRNN model gives and accuracy of 83.85% accuracy, CER of 4.16% and WER of 16.15%. The model predicts the words in the images and outputs the corresponding digital editable text.

Keywords: *Convolutional Neural Network, Convolutional Recurrent Neural Network, Computer Vision, Handwritten Characters, Sequence-to-Sequence*

Table of Contents

LIST OF ABBREVIATIONS

Adam:	Adaptive Moment Estimation
AI:	Artificial Intelligence
CER:	Character Error Rate
CNN:	Convolutional Neural Network
CPU:	Central Processing Unit
CRNN:	Convolutional Recurrent Neural Network
CSS:	Cascading Stylesheets
CTC:	Connectionist Temporal Classification
GAN:	Generative Adversarial Network
GPU:	Graphical Processing Unit
HTML:	Hypertext Markup Language
JS:	JavaScript
LSTM:	Long Short Term Memory
MLP:	Multi-Layer Perceptron
NAdam:	Nesterov-accelerated Adaptive Moment Estimation
OCR:	Optical Character Recognition
RAM:	Random Access Memory
ReLU:	Rectified Linear Unit
SGD:	Stochastic Gradient Descent
WER:	Word Error Rate

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
Introduction

1.1 Introduction

PrachaRead: Newari Text Recognition system is a system that converts handwritten newari characters in images into editable digital characters and digital words in images to editable digital text. This project uses neural networks and computer vision to convert the text in images to digital editable text.

For converting handwritten characters into digital characters, the system takes input images of handwritten characters and converts it into digital characters. The image is preprocessed and classified by a CNN model which then maps the output class to corresponding characters.

For converting digital words into editable text, the system takes input as images of words and gives digital editable text as output. The system extracts the features from the images and makes predictions using the CRNN model and decodes those predictions to give the output which is the extracted digital editable text.

The models were constructed using python for programming language for the implementation of neural networks in Jupyter Notebook. The frontend of the system was done using simple HTML, CSS and JavaScript and the backend was done using Django framework of python.

1.2 Problem Statement

There are many models today that can digitize handwritten text. However these models are only present for high resource languages such as English and there are very less models that can digitize handwritten text for low resource languages such as Newa Bhasa. One such script for Newa Bhasa is Prachalit lipi.

There are tons of historic literatures that are written in Prachalit lipi. There's a growing trend of digitization and a pressing need for preserving and promoting lesser-known languages and scripts such as Prachalit script. Due to the low amount of resources and recognition systems for Prachalit lipi, it is very difficult to digitize the manuscripts, store and use them.

1.3 Objectives

- To convert handwritten characters written in Prachalit lipi into digital characters using Convolutional Neural Network (CNN) model.
- To convert words in images in Prachalit lipi into digital editable text using (Convolutional Recurrent Neural Network) CRNN model.

1.4 Scope and Limitation

1.4.1 Scope

The project's scope is to develop a system that is capable of recognizing handwritten characters and word images in Prachalit lipi into digital text format. This can be further used to digitize the historic manuscripts, documents or any other handwritten materials which have been written in this script which can then be used to preserve and promote the culture of Newari community. The primary focus of this project is to accurately recognize handwritten characters and words using neural networks and provide an efficient platform for digitization.

1.4.2 Limitation

- Low resources of handwritten content in Prachalit lipi.
- The system may not perform well on highly stylized handwriting or unclear handwritings.
- Can only handle images not live video.

1.5 Development Methodology

For this project we have used the incremental model as shown in Figure 1.1 as in this model functions are developed iteratively and added to the system in increments. This model is best suited as there need to be made many adjustments in the model to improve the accuracy and robustness of the model.

1. **Iteration 1:**

Extensive research was conducted and it was found CNN was best for extracting

features from an image and classification tasks. The dataset was collected from kaggle from Prachalit Newa Lipi Dataset and more data was manually created by writing the characters. The CNN model was developed and trained on the collected dataset. The model was trained to recognize individual characters of Prachalit script. The model performed satisfactory only giving 75% accuracy in validation data.

2. Iteration 2:

To improve the performance of the model, the CNN model was modified. The dataset was by further creating the data. The model was trained on the new data and this time the accuracy was 98% on the validation data. After further research CRNN model was found to be the best was sequence-to-sequence task such as word recognition. After this the creation of words dataset was started which proved extremely difficult due to the complexity of the language and characters. So words dataset was created manually from drawing words in the images. The CRNN model was trained on the newly created dataset which gave an accuracy of 40%.

3. Iteration 3:

The CRNN model was modified and upgraded using two bidirectional LSTM networks to further capture the features of the images and their relations. This increased the accuracy of the model to 83% which improved by using different training parameter values. After this the development of frontend and backend was done to deploy the model to a website so that users could use the website to convert newari text. The project was documented thoroughly from the start to the end.

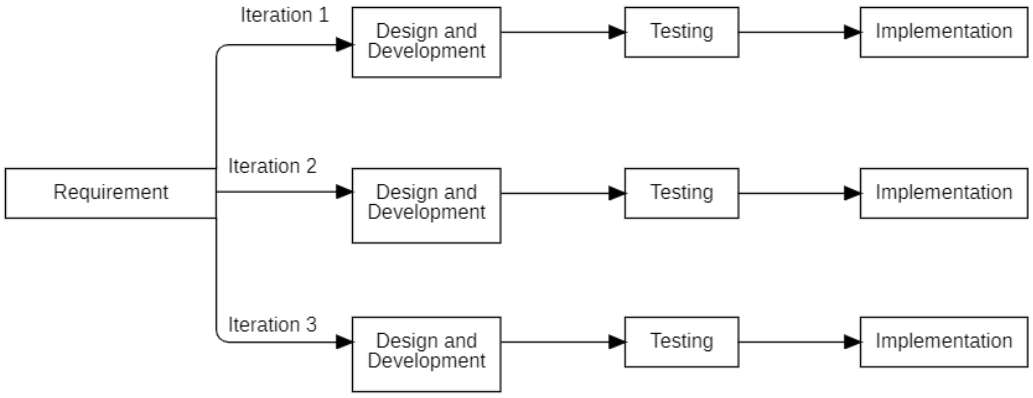

Figure 1.1: Incremental Model

1.6 Report Organization

This report of “PrachaRead: Newari Text Recognition” is separated into 6 different chapters.

Chapter 1: This chapter introduces the project along with its objectives, scope, limitations and the development methodology used to develop the project.

Chapter 2: This chapter discusses the background relating to the project and the summary of past research papers and reports related to the current project.

Chapter 3: This chapter includes the analysis of the requirements of the project which contains use case diagrams and scenarios, class diagrams, activity diagrams and sequence diagrams along with Gantt chart to visualize the project schedule.

Chapter 4: This chapter includes system design and UML diagrams of the system such as refined class diagrams, refined sequence diagrams and refined activity diagram.

Chapter 5: This chapter describes about the tools used in the development of this project. It also contains details of testing done and the result analysis of the system.

Chapter 6: This chapter contains the conclusion and the possible future recommendations for this project.

CHAPTER 2
BACKGROUND STUDY AND LITERATURE REVIEW

2.1 Background Study

Handwritten text recognition, or HTR, is a type of technology that helps convert handwritten letters into digital text. It is a part of Optical Character Recognition (OCR). HTR is commonly used for major languages like English, Chinese, and Arabic, but for lesser-known languages and scripts, it’s still not very common. One of these scripts is Prachalit Lipi, used for writing Newa Bhasa (Newari). Unfortunately, Prachalit Lipi has not received much attention in the field of handwritten text recognition.

Prachalit Lipi is an ancient script used by the Newar community, especially for writing documents, manuscripts and religious texts. Lately, there’s been growing interest in preserving these texts by converting them into digital format. However, there aren't many tools available for digitizing Prachalit Lipi, which makes it hard to store and use these historical documents. This lack of resources makes it difficult to archive the rich cultural heritage of Newari people.

With recent advances in artificial intelligence (AI) and deep learning techniques, especially Convolutional Neural Networks (CNNs), it's now possible to create systems that can recognize handwritten characters more accurately. CNNs are good at identifying patterns in images, making them useful for tasks like handwriting recognition. However, for scripts like Prachalit Lipi, challenges still exist, such as different handwriting styles, the limited amount of training data, and the complexity of some characters. The PrachaRead system aims to solve these problems by using CNNs to recognize Prachalit Lipi characters and convert them into digital text. This can help preserve Newar manuscripts and make them easier to store and access in the future.

2.2 Literature Review

The Nepal Script text recognition project is presented in this publication. With the help of line and word segmentations, the implementation incorporates the Nepal Script text recognizer, which is based on the CRNN CTC architecture. With the use of a meticulously selected dataset that includes printed and handwritten texts in Nepal Script, this study has produced a WER of 13.11% and a CER of 6.65%. The Nepal

Script Text Dataset on Kaggle is the dataset utilized in this project. The study also examines the difficulties that arise because of the script's complexity, including conjuncts, modifiers, and variants, as well as the script's present condition. [1]

The public database for Ranjana script handwritten characters (RHCD), the first of its type for scholars, is presented in this work. Derived from the Brahmi system, the Ranjana script contains 10 numerals, 16 vowels, and 36 consonants. Developing a new database for Ranjana character identification, testing it with CNN methods such as LeNET-5, AlexNET, and ZFNET, and then recommending a new model with hyperparameter adjustment to increase accuracy are the three objectives of the project. Data collection, digitization, pre-processing, splitting, and data augmentation are all part of the research. They discovered that their model had flawless precision, recall, and F1-score, and it achieved the highest test accuracy of 99.73% at 64x64 pixels. [2]

The main goal of this article is to detect Devanagari letters using deep convolutional neural networks (CNN). Twelve Indian languages utilize the Devanagari script. By selecting the optimal hyperparameters, this project enhances the network. The outcomes demonstrate this method's efficacy on a benchmark dataset.. [3]

A comparison of many machine learning and deep learning models for Devanagari character recognition is presented in this research. Because so much study has previously been done on Devanagari text recognition using Convolutional Neural Networks (CNN), the MLP Mixer technology is used. Comparing the more recent MLP Mixer model with more conventional models is the aim. CNN is used to the issue as a stand-alone deep learning model, mostly for feature extraction. Additionally, these retrieved complicated characteristics are used to create a number of conventional machine learning classifiers. [4]

In order to digitize visual information for simpler retrieval and more effective data processing, this work concentrated on identifying letters and numbers in images. Three main components of the study were image processing, dataset preparation, and the application of a Convolutional Neural Network (CNN) as a classifier. Creating a CNN with the right settings and training it on a unique dataset was the main goal. The next step in the study was to feed an input image into the trained network after processing it into the necessary format in order to predict the classes in the image. The study also

contrasted Adam and NAdam, two popular optimizers in image classification. [5]

The goal of this work was to employ optical character recognition algorithms to identify text from scanned images. It suggested studies to use a Convolutional Neural Network (CNN) to study and detect handwritten Nepali characters. 92,000 images with 32x32 Nepali characters were used in the study; these images were preprocessed using clipping, cropping, and grayscale conversion. The Back Propagation and Gradient Descent algorithms were used to update weights during training and testing in order to test recognition using a template matching technique. The findings showed that the CNN model outperformed the Feed Forward Neural Network in terms of character recognition accuracy. [6]

CHAPTER 3
SYSTEM ANALYSIS

3.1 Project Analysis

3.1.1 Requirement Analysis

i. Functional Requirements

The system accurately recognizes handwritten characters and digital words in images. It must be capable of detecting and distinguishing between various handwritten characters and digital words of Prachalit script, using a comprehensive database to match the input text with predefined characters. Upon matching, the system convert the recognized handwritten characters and digital word images into a digital format.

Users upload images of handwritten characters which the system preprocesses by converting to grayscale, normalizing and resizing. The system then use a Convolutional Neural Network (CNN) for character recognition. The system outputs the recognized characters.

Users upload digital word images which is preprocessed by the system by converting to grayscale, resizing, normalizing and expanding the dimensions. The system then uses a Convolutional Recurrent Neural Network (CRNN) for prediction which is decoded and outputted as digital words.

- **Use Case Diagram**

In Figure 3.1, the user uploads the image containing handwritten Newari characters which then the system processes to give the output digital text.

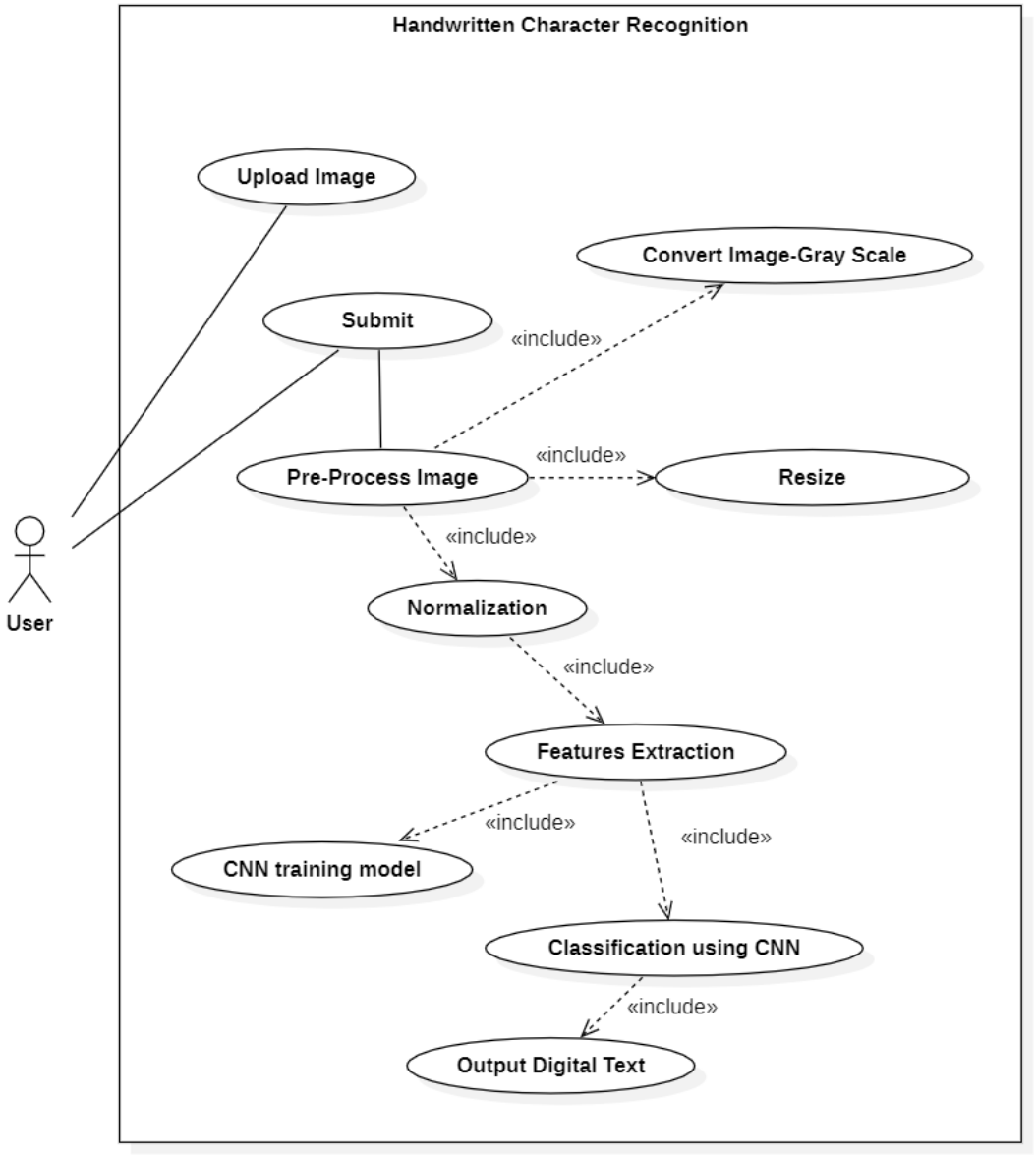

Figure 3.1: Use case diagram of Handwritten Character Recognition

Table 3.1: Use-case scenario of Handwritten Character to Digital Text

Use case: Handwritten Text to Digital Text	
Primary Actor:	User
Secondary Actor:	System
Flow of event:	i. User has handwritten character in Prachalit script. User wants to digitize the text so user uses PrachaRead. ii. User uploads images() of the content and submits() iii. System Pre-Processes() by convert-image to gray scale(), resize() and normalization() iv. System does features extraction () using CNN training model () and classifies using CNN (). v. System generates output digital text ().
Pre-condition:	The system must be in running state.
Post-condition:	The output digital text should match the inputted image text content.
Quality:	The conversion should be completed as soon as possible.

- **Use Case Diagram**

In Figure 3.2, the user uploads the image containing Newari words which then the system processes to give the output digital text.

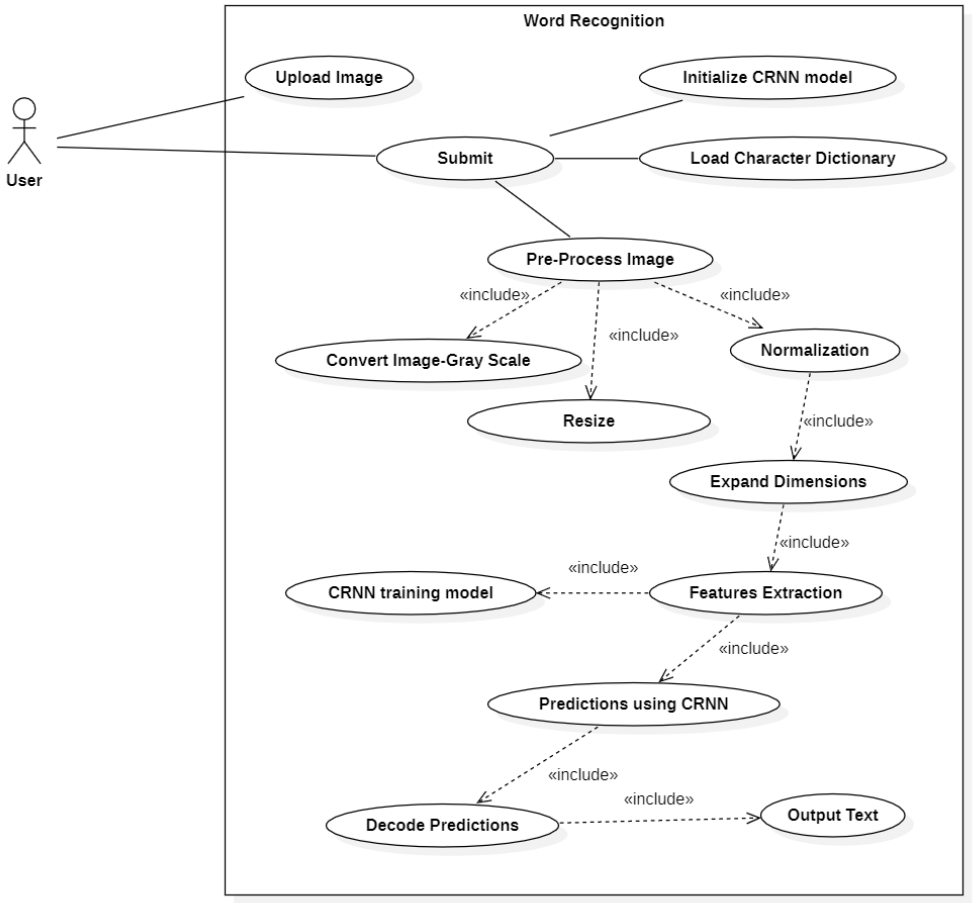

Figure 3.2: Use case diagram of Word Recognition

Table 3.2: Use-case scenario of words to digital text

Use case: Words to Digital Text	
Primary Actor:	User
Secondary Actor:	System
Flow of event:	vi. User has images of words in Prachalit script. User wants to digitize the text so user uses PrachaRead. vii. User uploads images() of the content and submits() iii. System Pre-Processes() by convert-image to gray scale(), resize(), normalization() and Expand Dimensions() ix. System does features extraction () using CRNN training model () and predicts using CRNN (). x. System decodes the predictions () and generates output text ().
Pre-condition:	The system must be in running state.
Post-condition:	The output digital text should match the inputted image text content.
Quality:	The conversion should be completed as soon as possible.

Non-Functional Requirements

- **Usability:** The system will be user-friendly and easy to navigate. It will feature clearly labeled buttons, straightforward options and simple instructions to ensure a smooth user experience for all users.
- **Reliability:** The system will be dependable and consistent, operating without unexpected interruptions or failures. It will reliably recognize handwritten as well as digital input, delivering accurate results.

- **Accessible:** The system will translate recognized handwritten characters and word images into digital text enhancing convenience for users who wish to digitize handwritten characters and digital words.

3.1.2 Feasibility Study

i. Technical Feasibility

- **Hardware Requirements**

The project was run on a local machine with 6 GB of GPU with a processor of Intel Core I5 13th gen with 16 GB of RAM. The training required 97% of memory of the GPU, 95% of CPU and 91% of total RAM. So to train the model it is recommended to use at least a hardware system that can match the GPU capability as mentioned above.

- **Software Requirements**

The software requirements for this project involves using the language Python to develop handwriting recognition and image classification algorithms, supported by machine learning libraries like Pytorch for training and running models. The project's frontend is built using HTML, CSS and JavaScript while Django serves as the backend.

ii. Operational Feasibility

The Handwritten Recognition System operates by allowing users to upload images of characters or words for real-time text conversion. The system processes the uploaded images and then translates them into digital text. The project also incorporates machine learning techniques to accurately detect and recognize the handwritten characters and digital words provided by users. By converting input into digital formats, the system enhances convenience for users who wish to work with documents written in Prachalit scripts.

iii. Economic Feasibility

This analysis accounts for all the key expenses related to the final development of the handwriting recognition system like potential additional costs for enhancing RAM for the training of the Neural Network. The system is designed to efficiently process image inputs and provide their digitized format all while staying within the existing budget.

As no further purchases are required for deployment, the project is cost-effective and economically feasible.

iv. Schedule Feasibility

We have used a Gantt chart to highlight the time period for the project.

Table 3.3: Schedule Feasibility

Name	Start Date	End Date	Duration
Iteration 1	**Tue 7/30/24**	**Sun 9/8/24**	**35 days**
Design and Development	Tue 7/30/24	Wed 8/21/24	20 days
Testing	Thu 8/22/24	Thu 8/29/24	7 days
Implementation	Fri 8/30/24	Sun 9/8/24	8 days
Iteration 2	**Mon 9/9/24**	**Wed 10/30/24**	**45 days**
Design and Development	Mon 9/9/24	Fri 10/11/24	29 days
Testing	Sun 10/13/24	Sun 10/20/24	7 days
Implementation	Mon 10/21/24	Wed 10/30/24	9 days
Iteration 3	**Thu 10/31/24**	**Sun 12/8/24**	**33 days**
Design and Development	Thu 10/31/24	Thu 11/14/24	13 days
Testing	Fri 11/15/24	Mon 11/25/24	9 days
Implementation	Tue 11/26/24	Sun 12/8/24	11 days
Documentation	**Tue 7/30/24**	**Sun 12/8/24**	**113 days**

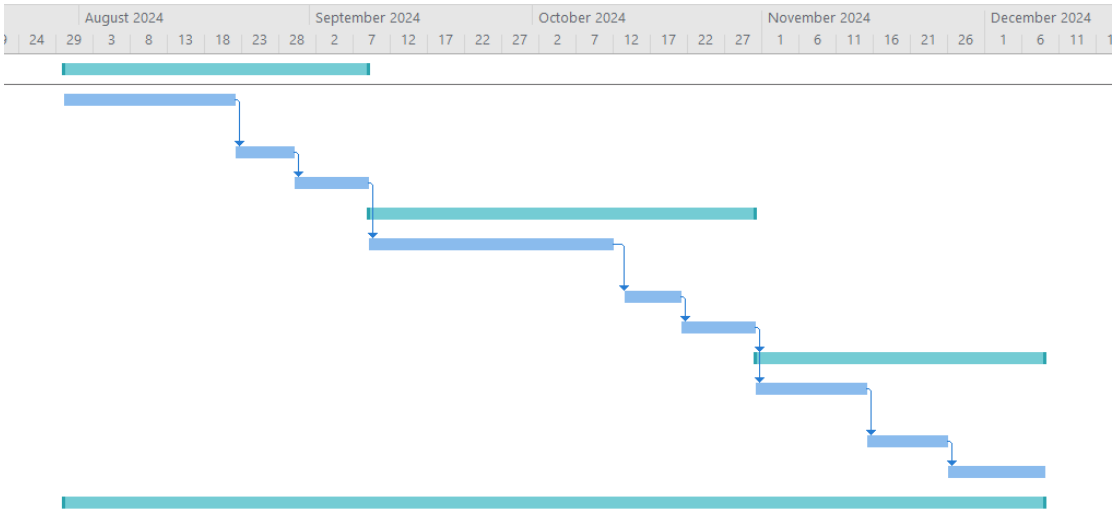

Figure 3.3: Gantt chart

3.1.3 Analysis

- **Object modelling using Class and Object Diagrams**

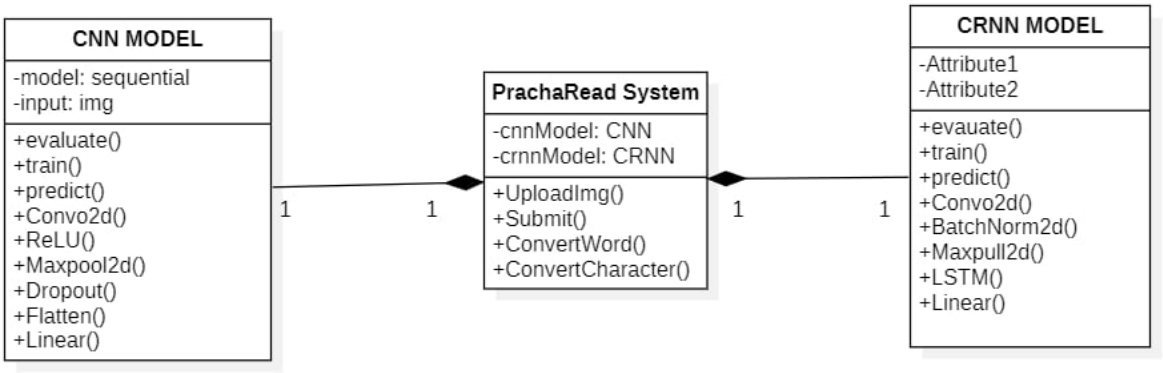

Figure 3.4: Class Diagram for PrachaRead System

The class diagram in Figure 3.4 depicts a static view of PrachaRead System, designed for character and word recognition. It consists of three main classes CNN model, CRNN model, and PrachaRead system. The CNN model represents a Convolutional Neural Network that looks after image processing and feature extraction. It does character recognition in the system and has methods to evaluate, train and prediction for managing the training and prediction workflow. It also has core CNN methods like convolutional and pooling layers, dropout layers and flattening with ReLU as activation function. The CRNN model upgrades the capabilities of CNN Model integrating convolutional layers and recurrent neural network. This is responsible for word recognition. It has methods like BatchNorm2d for normalization, Maxpool2d for down-sampling and LSTM for sequential data dependencies.

- **Dynamic modelling using State and Sequence Diagrams**

A sequence diagram shows the sequence of messages passed between the objects.

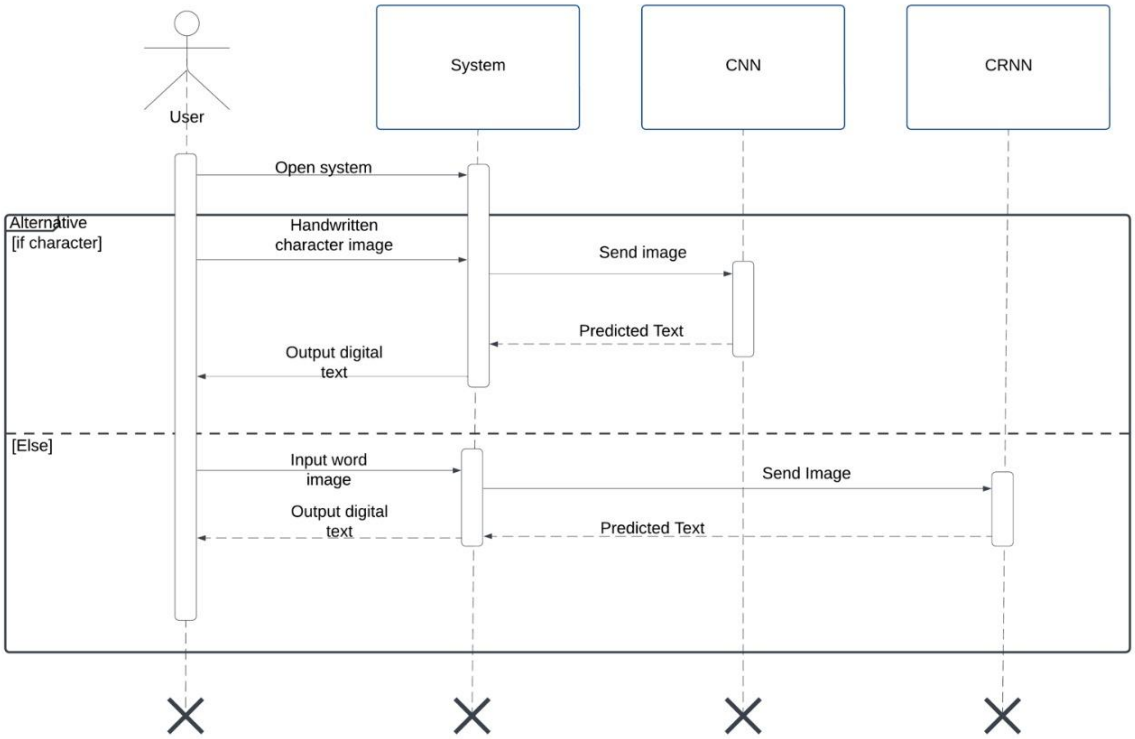

Figure 3.5: Sequence Diagram for PrachaRead System

The Sequence diagram in Figure 3.5 shows how handwritten text is converted into digital form. The process starts when the user opens the system and provides an input image, a single handwritten character or a whole word. If the input is a character, the system uses a Convolutional Neural Network (CNN) to identify it and then give the recognized character as digital text. Otherwise, if the input is a word, the system uses a Convolutional Recurrent Neural Network (CRNN) to process and recognize the word. Once identified, the system presents the word as digital text.

- **Process modelling using Activity Diagrams**

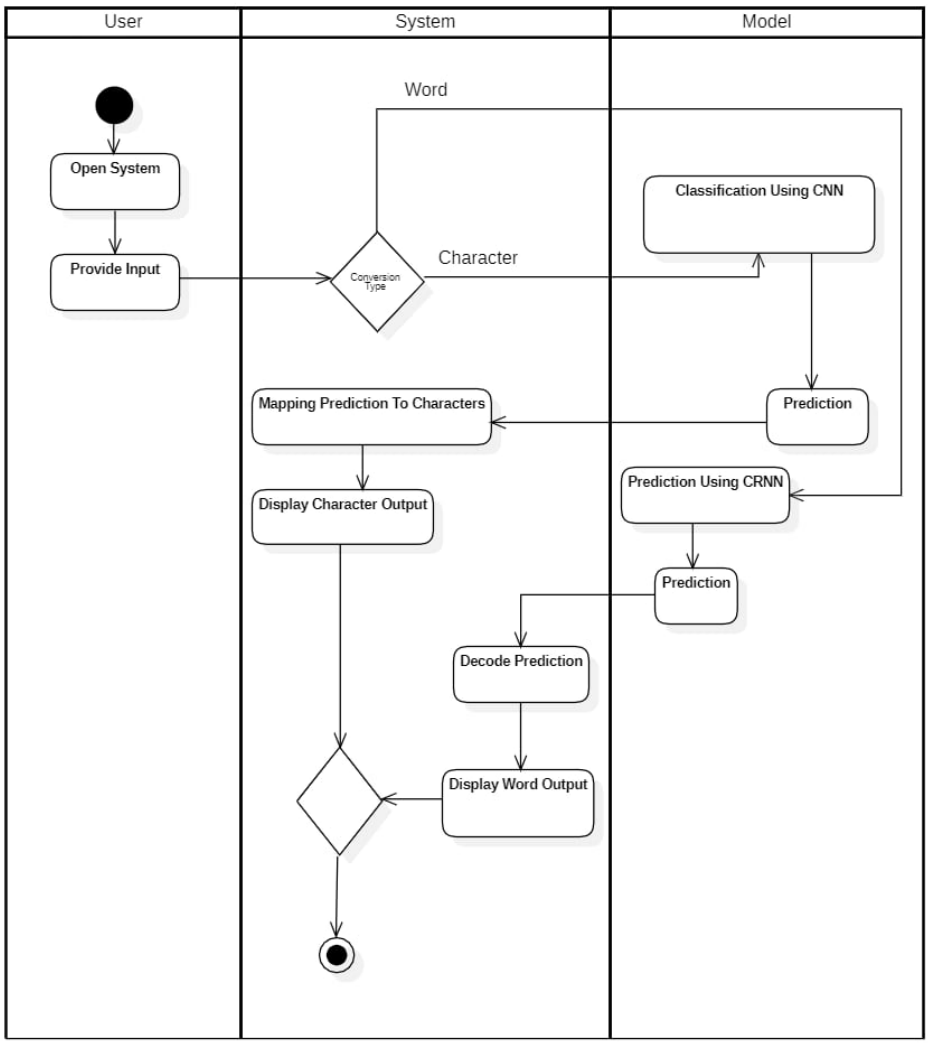

Figure 3.6: Activity Diagram for PrachaRead System

In the Figure 3.6, the user initiates the process by opening the system and uploading the image input. Then, the conversion type of the image is chosen. If its character, the classification is done through CNN to make predictions. Then, the prediction's label is mapped to characters. If the conversion is word, then prediction is done through CRNN. The prediction is then decoded. Finally, output is displayed as character output or word output based on the initial conversion type provided by the user. The process ends when the results are displayed.

CHAPTER 4
SYSTEM DESIGN

4.1 Design

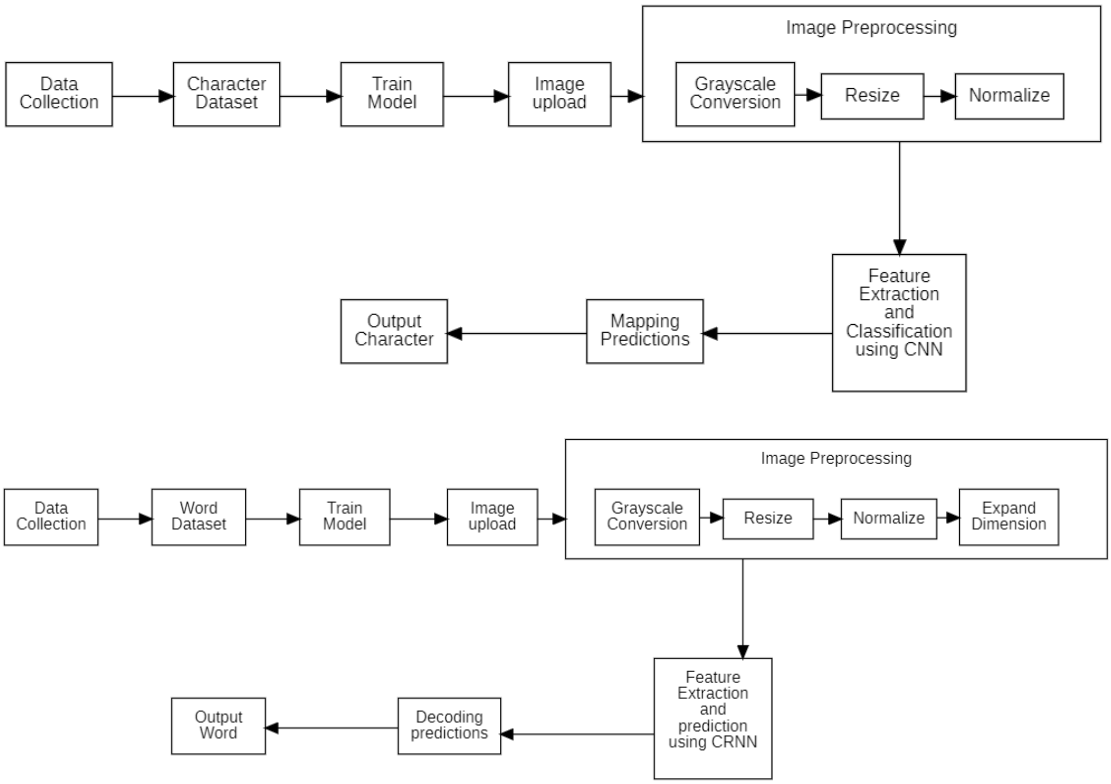

Figure 4.1: System Architecture for PrachaRead

System design involves development of an architecture defining components, interfaces, and modules, while also providing essential information to understand their integration and implementation.

- **Character Dataset**

Figure 4.2: Character Dataset

- **Word Dataset**

Figure 4.3: Word Dataset

- **Image Upload for character**

The character image provided by the user is the input data for the PrachaRead system. All the Newari characters were labeled and sorted out. Now the labeled characters have to match with the inputted data. After the CNN model is trained, the uploaded image is passed through the system for prediction.

- **Image Preprocessing for character**

Uploaded image is preprocessed to ensure that the picture is in a suitable format for the system. Firstly, the image is gray scaled to remove unwanted color information while retaining essential features. Then, the image is resized. Resizing helps to standardize the input size for the neural network so that the model can handle input consistently. Normalization is the process of adjusting pixel vales to a range (for e.g., 0 to 1) to improve model's performance.

- **CNN model**

CNN has been trained on Prachalipi script and custom dataset handwritten by our members. The model is used for classification of input images, which are preprocessed. The class with the highest probability is then selected as the predicted class for that image.

- **Character Output**

The prediction from the model is mapped to corresponding character. This final output is displayed to users ending the recognition process. In summary, the uploaded image is preprocessed into a gray-scaled standardized form, applies a trained CNN model to recognize the character and provides a character output.

- **Image Upload for word**

If the user gives input as a character, then the CNN model gives an character output, but users can also give a word as inputs too.

- **Image Preprocessing for word**

Same as characters, the word input also requires standardization. The image is once again grayscaled, resized, and normalized to prepare them for the model. Expand dimensions is also done on top of it. When images are initially uploaded, they are typically 2D. However, deep learning models like CRNN require input tensors to be in a 3D or 4D format. Expand dimension reshapes the 2D image into a 3D tensor.

- **CRNN model**

The CRNN model extracts sequential features from the image, analyzing spatial and temporal dependencies. It helps to predict the sequences of characters that make up a word.

- **Word Output**

The predicted sequence of characters is decoded into a complete word, which ends the recognition process for word. In summary, the system takes the word input given by users, preprocessed into a standardized form with dimension expansion, uses CRNN model to recognize the sequence of characters, decodes the prediction, and finally provides a word output.

4.1.1 Refinement of Class Diagram

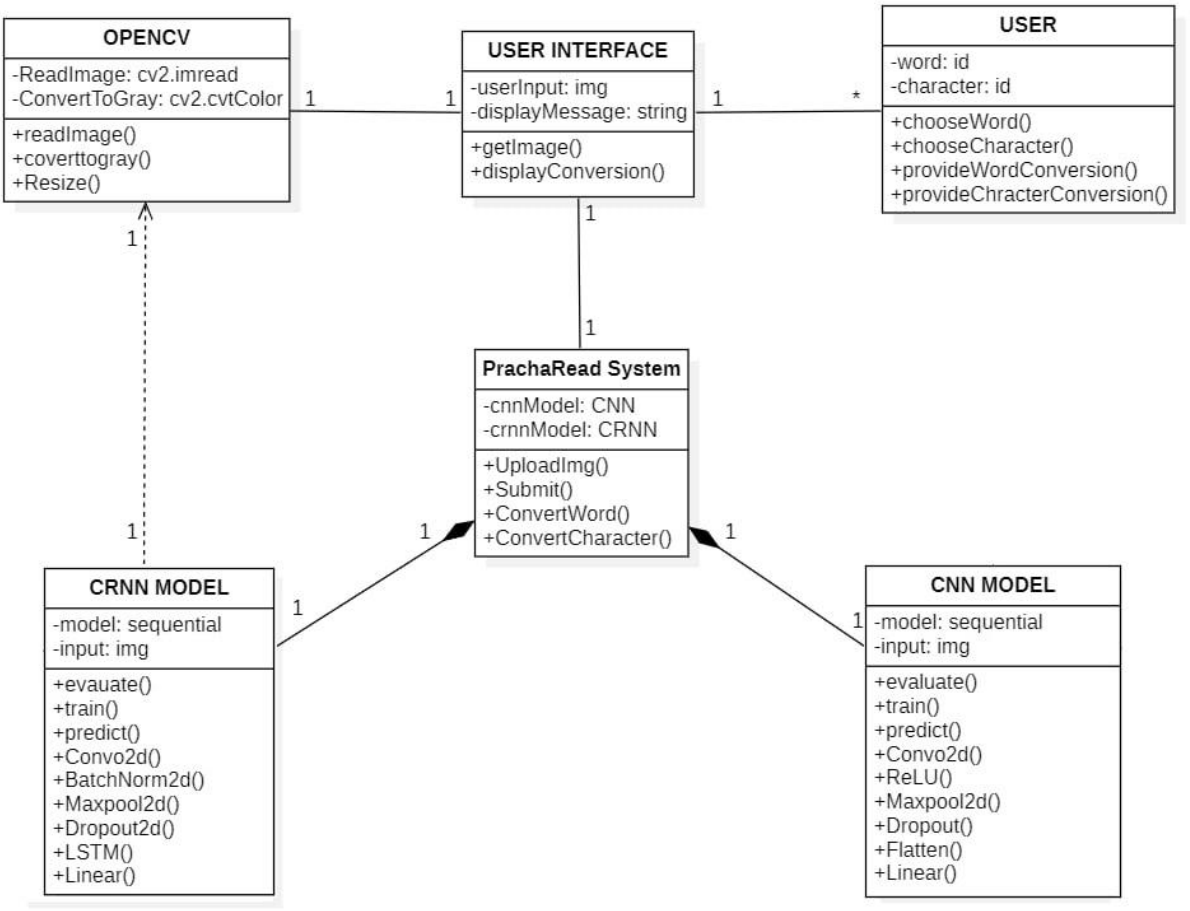

Figure 4.4: Refined Class Diagram for PrachaRead System

Figure 4.4 shows the refined class diagram consisting of 6 main classes. PrachaRead System, CNN model, CRNN model, User Interface, User and OpenCV. The user class allows user to choose word or character recognition. The OpenCV class is responsible for reading, converting into gray scale and resizing the image.

The PrachaRead System centralizes the operation, incorporating CNN for character recognition and CRNN for word recognition. The CNN model processes input images by extracting features using convolutional and pooling layers for robust pattern detection. The CRNN model furthers CNN processes by handling sequential data, utilizing layers like LSTM to recognize sequences effectively. Together, these components ensures character and word conversion from images to texts

4.1.2 Refinemend of Sequence Diagram

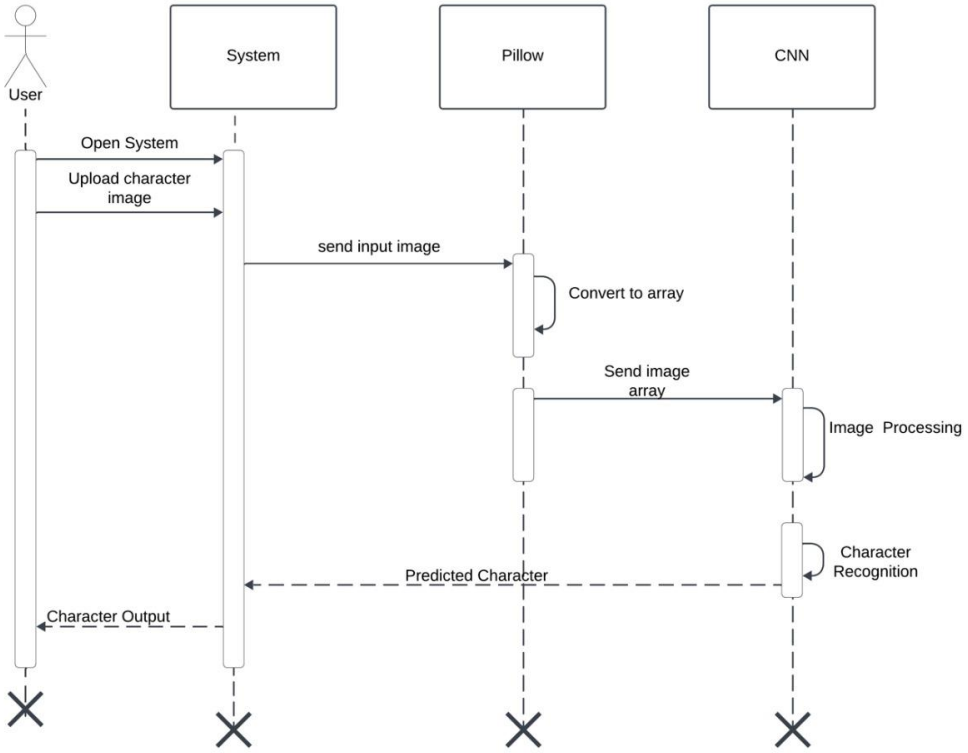

Figure 4.5: Refined Sequence Diagram for Character Recognition

In Figure 4.5 first the user opens the system and uploads an image that contains handwritten characters. The system then processes this image by sending it to the Pillow Library. Pillow converts the image into a numerical array which makes it easier for the system to process.

Next this array is sent to a Convolutional Neural Network (CNN) model to recognize handwritten characters. CNN works by analyzing the image and identifying key features which helps it correctly recognize the characters.

Once CNN identifies the characters, it sends the recognized characters back to the system. The system then displays the recognized characters to the user in a digital format.

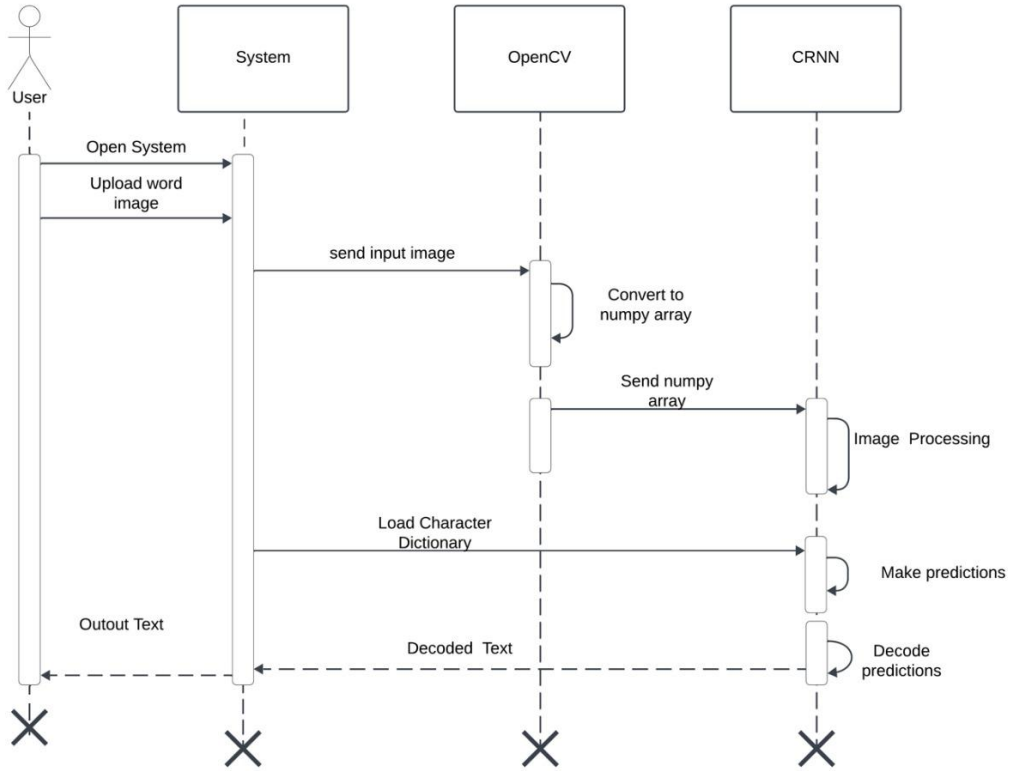

Figure 4.6: Refined Sequence Diagram for Word Recognition

The refined sequence diagram shown in Figure 4.6 explains the process for extracting text from an image using OpenCV and a CRNN model. When a user opens the system and uploads a picture with handwritten text, the procedure starts. After uploading the picture, the system converts it into a NumPy array using OpenCV. Python commonly uses this format to efficiently handle and analyze picture data. After that, the CRNN model receives the NumPy array. Based on its training experience, the model analyzes the image, identifies key features, and predicts the text. A character dictionary is then used to translate the anticipated text so that the user can read and understand it. Finally, the decoded text is returned to the system and displayed to the user as the output text.

4.1.3 Refinement of Activity Diagram

Refined activity diagram provides the detailed representation of the flow of events in a system.

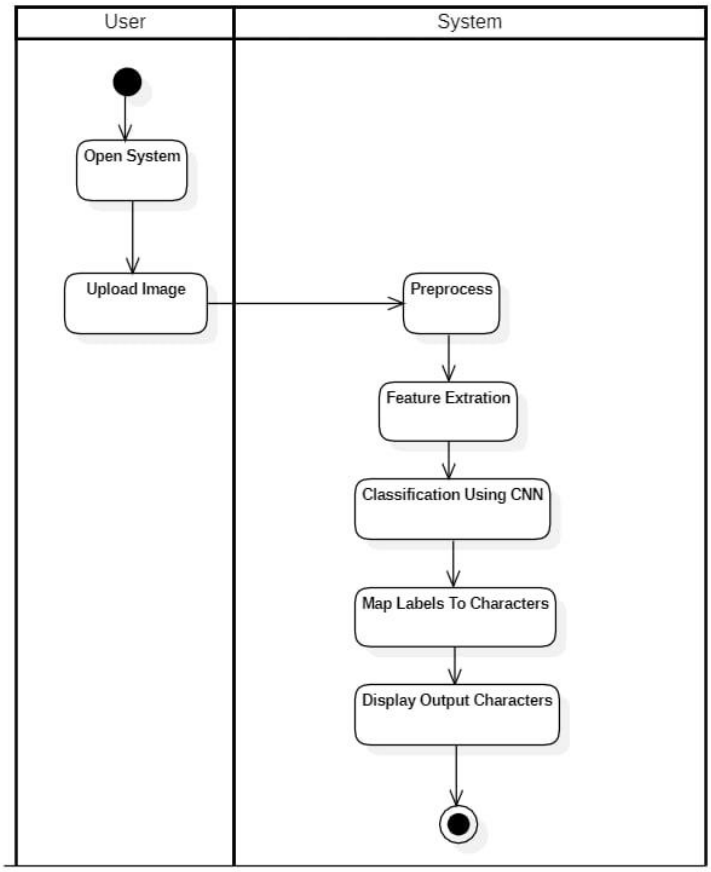

Figure 4.7: Refined Activity Diagram of Character recognition

The refined activity diagram in Figure 4.7 shows the character recognition for PrachaRead system. Here, the user initiates the system by opening the system and uploading the input image. Then, preprocessing of the image is done to extract the features of the image provided. If its character, the classification is done through CNN model. The predicted labels is mapped to characters and the character output is displayed.

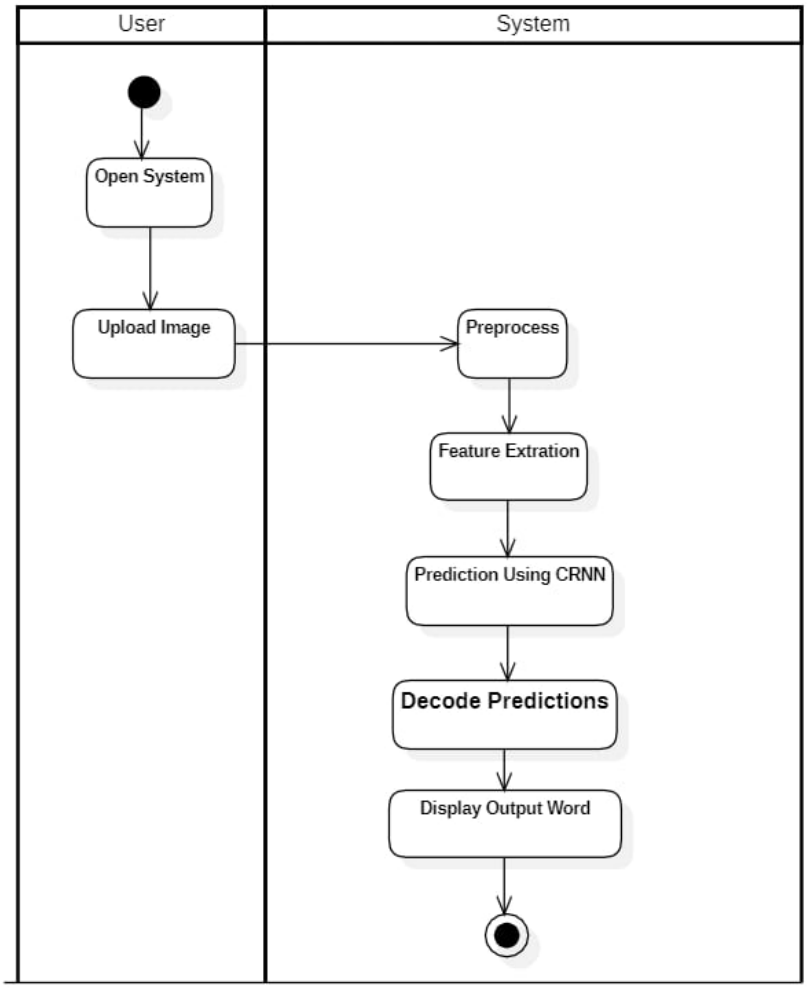

Figure 4.8: Refined Activity Diagram for Word Recognition

In the refined activity diagram for word recognition as shown in Figure 4.8 shows the word recognition for PrachaRead system. Similarly like the figure 4.8, the user initiates the system, opening the system and uploading the image. Preprocessing is done to extract the features of the image. Since, it's a word, the prediction is done through CRNN model. The prediction is then decoded, and the word output is displayed. The process ends when the result is displayed to the user

- **Component Diagram**

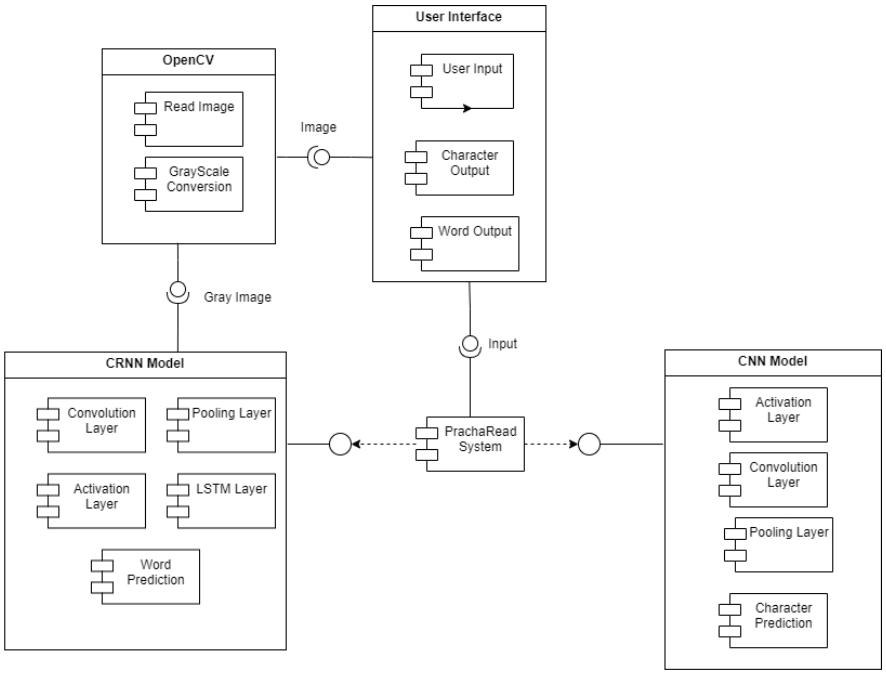

Figure 4.9: Component diagram of PrachaRead System

A component diagram divides the system into distinct functional units. Each component has a specific task or function and communicates only with other crucial components.

The component diagram in the Figure 4.9 shows the process of this system through its separate components. The process starts with OpenCV, which handles the image uploaded by the user. OpenCV's job is to load the image and convert it to grayscale (black and white) format. This conversion makes it easy to analyze and process the image data. Then with the help of the user interface users can upload their images and see the output in the form of recognized characters and words. Once the image is converted to grayscale, it is sent to two different models: the CRNN model and the CNN model. The CRNN model uses convolution and pooling layers to identify features in the image. It analyzes the sequence of those features using LSTM layers, which helps in identifying the entire word. On the other hand, the CNN model focuses on identifying individual characters in the image, using activation, convolution, and pooling layers. The central part of the PrachaRead system is the system that combines the results from both models to produce the result. This system processes the input image, recognizes characters and words, and presents the result through a user-friendly interface.

- **Deployment Diagram**

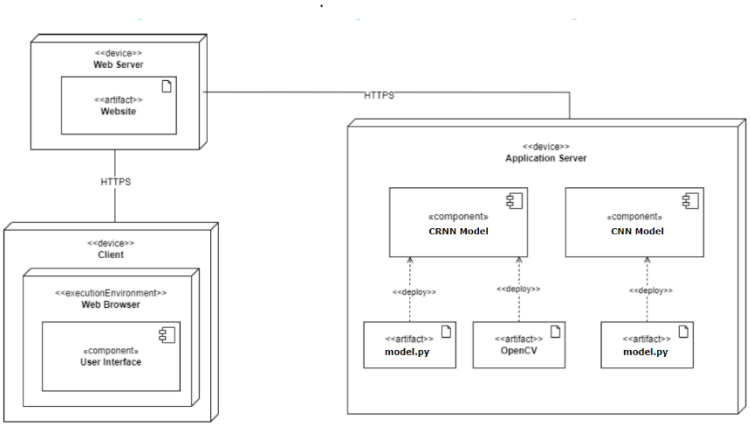

Figure 4.10: Deployment Diagram of PrachaRead System

The deployment diagram in Figure 4.10 shows how the various components of the system function and are interconnected. The client, web server, and application server are the three primary components. The client is the user's device, such as a computer or smartphone, on which they utilize a web browser to communicate with the system. Using a User Interface to enter data and see the results is helpful. Between the client and the application server, the web server serves as a bridge. Requests from the client are received, forwarded to the application server, and then returned to the user. All of these procedures are carried out safely over HTTPS. The system's primary functions, such as analyzing pictures with a CNN Model and identifying sequential data with a CRNN Model, are carried out by the Application Server. Tools like OpenCV and Python scripts (`model.py`) are used to make these models function. The complete system receives requests from the client, handles them on the application server, and securely and efficiently returns the results.

4.2 Algorithm Details

The project implements the following algorithms:

4.2.1 Convolutional Neural Network

Convolutional Neural Network (CNN) is a type of feed-forward neural network that learns features using kernels. This algorithm extracts features from images and updates the weights of the network associated with the corresponding labels. The trained network is capable of classifying unlabeled images.

CNN consists of mainly consists of three layers which are: convolution layer, pooling layer and fully connected layer.

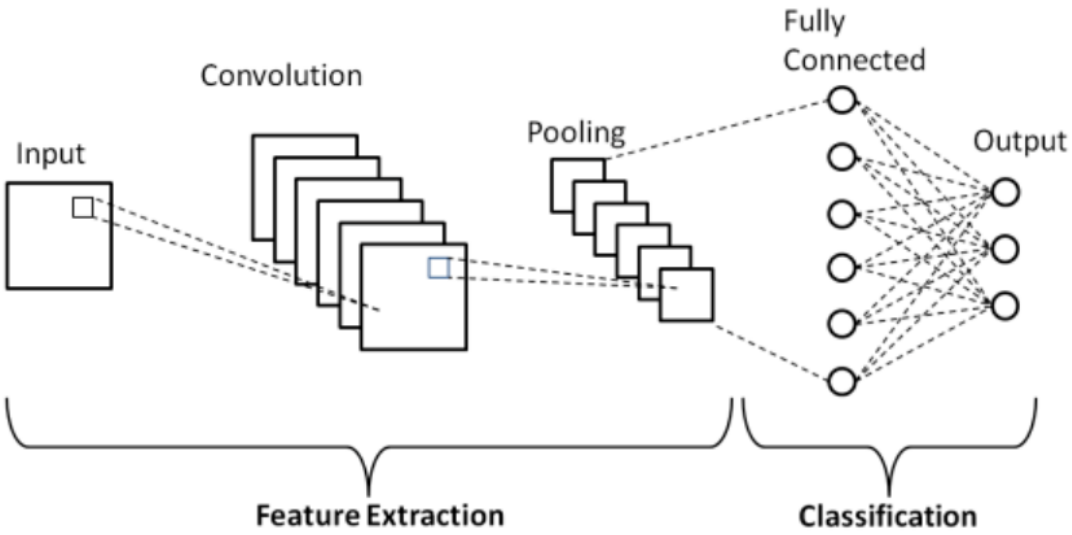

Figure 4.11: Convolutional Neural Network *[10]*

- **Convolution Layer**

This layer is the main layer in the CNN model. This layer detects features such as edges, patterns or textures from the images. It performs dot product between the kernel or filter and the input to create an activation map in which the kernel slides over the entire image.

For a kernel 'K' of size M * M and depth 'D' applied to an input patch 'I' of the same size, the dot product is computed as below:

$$\boldsymbol{Dot\ Product} = \sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{k=1}^{D} \boldsymbol{K}[i,j,k].\boldsymbol{I}[i,j,k] \quad ---------\text{eqn i}$$

Where K[i, j, k] is the value of the kernel at depth k position (i, j), I[i, j, k] is the value of the input patch at depth k position (i, j) and M is the size of the kernel.

This results in a single value which is placed in an output feature map which is repeated as the kernel slides across the entire input image. After the activation map is created, the output size 'W_{out}' for input size 'W', kernel size 'M', padding 'P' and stride 'S' is computed as follows:

$$\boldsymbol{W_{out} = \frac{W-M+2P}{S} + 1} \quad \text{---------- eqn ii}$$

This gives us an output volume size of $W_{out} \times W_{out} \times D_{out}$ where D_{out} is the number of kernels.

- **Pooling Layer**

After the output feature map is computed, the pooling layer downsamples the feature map. It reduces the spatial size of the feature map which means the height and the width of the feature map. This helps in reducing the computation power required and makes the model less prone to change due to small distortions. Max pooling selects the maximum value from a small region from the feature map. The output size after the pooling is given by:

$$\boldsymbol{W_{out} = \frac{W-M}{S} + 1} \quad \text{---------- eqn iii}$$

Where W is the input size, M is the kernel size and S is the Stride.

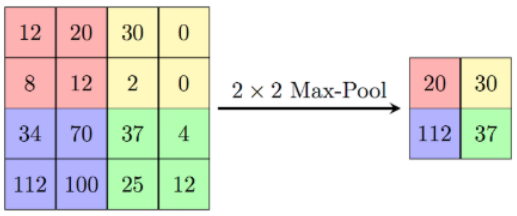

Figure 4.12: Max pooling *[9]*

- **Fully Connected Layer**

In fully connected layer, the every neuron is connected to every neuron in the previous layers which allows the combination of features that were detected earlier. The output of this layer is used for making decisions or classifications.

- **ReLU activation Layer**

The Rectified Linear Unit is applied usually after the convolution to introduce non-linearity which helps the model to learn complex patterns. It converts the negative to zero and other values as it is. It is computationally efficient which makes the model faster. It can be implemented as:

$$\boldsymbol{ReLU}(\boldsymbol{x}) = \boldsymbol{max}(\mathbf{0}, \boldsymbol{x}) \quad \text{eqn iv}$$

4.2.2 Convolutional Recurrent Neural Network

Convolutional Recurrent Neural Network (CRNN) is a hybrid model which combines CNN and RNN. It is designed to handle sequence based data such as handwritten recognition, speech recognition etc. It has the capabilities of both models such as the feature extraction of CNN and sequence-to-sequence learning capabilities of RNN. It extracts features from an image and model the temporal dependencies across those details. It consists of CNN model which extracts the features and RNN model which model the sequential relationships in the extracted features.

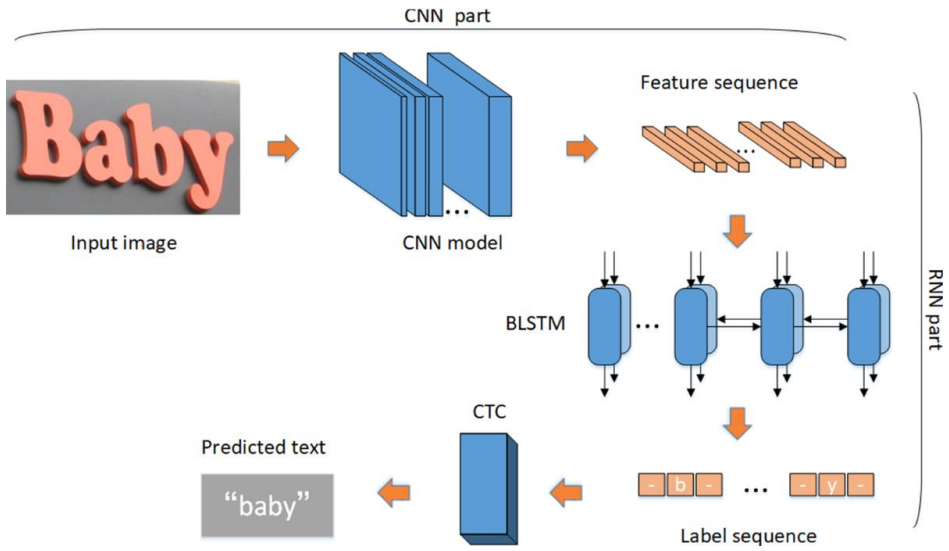

Figure 4.13: Convolutional Recurrent Neural Network *[7]*

- **CNN Layer**

This layer consists of layers which have been described above in the CNN algorithm

such as convolution layer, pooling layer, fully connected layers and ReLU.

- **Recurrent Layer**

This layer contains the LSTM layer which captures long-range dependencies in data using gates to control the information flow. A LSTM consists of

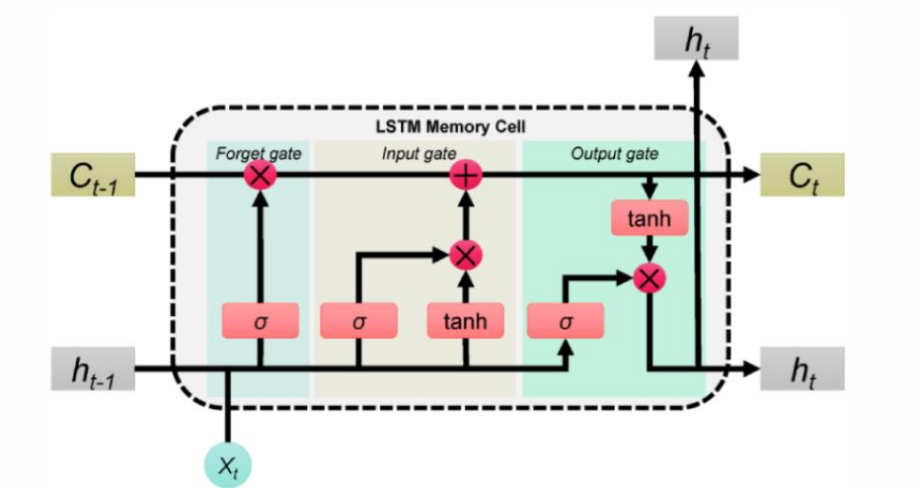

Figure 4.14: Long Short Term Memory Unit *[8]*

Cell State (C_t) which is the memory of the network which allows information to flow across timestamps.

$$\boldsymbol{C_t = f_t . C_{t-1} + i_t . \widetilde{C}_t} \quad \text{---------- eqn v}$$

Hidden State (h_t) which encodes the short term information from the current timestamp.

$$\boldsymbol{h_t = o_t . tanh(C_t)} \quad \text{---------- eqn vi}$$

Forget Gate (f_t) which decides the information to forget or discard from the previous cell.

$$\boldsymbol{f_t = \sigma(W_f . [h_{t-1}, x_t] + b_f} \quad \text{---------- eqn vii}$$

Input Gate (i_t) which decides the new information to add to the cell state.

$$\boldsymbol{i_t = \sigma(W_i . [h_{t-1}, x_t] + b_i} \quad \text{---------- eqn viii}$$

Output Gate (o_t) which determines what part of the cell state to output.

$$\boldsymbol{o_t = \sigma(W_o . [h_{t-1}, x_t] + b_o} \quad \text{---------- eqn ix}$$

Where $\boldsymbol{\sigma}$ is sigmoid function, W is weight for each gate, b is bias for each gate and $\tilde{C}_t$ is candidate cell state.

- **Connectionist Temporal Classification (CTC) loss**

CTC loss calculates the negative log probability of the target sequence Y given the input sequence X which is summed over all possible alignments. It is uses in sequence-to-sequence tasks where the alignment between the input and output sequence is not known or unknown. If $X=(x_1, x_2, \ldots x_T)$ is the input sequence and $Y=(y_1, y_2, \ldots., y_{L)}$ is the target sequence and $P(a_t \mid x_t)$ represents the probability of label a_t at time step t then CTC loss is calculated as:

$$\boldsymbol{CTC\ loss = -log\left(\sum_{a \in A(Y)} \prod_{t=1}^{T} P(a_t|x_t)\right)} \quad \text{---------- eqn x}$$

Where ***A(Y)*** is the set of all the possible alignments of the target sequence ***Y*** with the input sequence ***X*** and $\boldsymbol{a_t}$ represents one of the many possible alignments at time ***t*** of the target.

CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

5.1.1 Tools Used

- **Software Development Tools**

Python was the main programming language used in this project due to its multiple and varieties of libraries that make the implementation of the system easier. Numpy and Pandas are the most used and well known libraries when it comes to manipulating arrays and for data analysis. Torch was used to construct the neural networks used in this project. Matplotlib was used to visualize the graphs for accuracies and losses. OpenCV was used to manipulate and process images. The frontend of the project was done using HTML, CSS and JavaScript while the backend was done using a Django.

- **Hardware Tools**

A powerful CPU and GPU was used to process the images, train the model, speed the training and perform inference operations. The project needs minimum of 8GB of RAM and a GPU to perform in satisfactory levels.

- **Diagram Tools**

In this project we used StarUML as the primary diagramming tool which allowed us to create diagrams to visualize the process of construction of the system. We also used luciddraw.io as well for creating diagrams. Using StarUML we createds use case diagrams which shows the interactions and use case between different actors and the system, class diagrams which help us to model the structure of the system and activity diagrams to visualize the workflow. Using luciddraw.io, we designed sequence diagrams to visualize the order of interactions between different objects.

- **Dataset**

The character dataset contains 15000 images of handwritten characters which includes roughly 235 images per character. The word dataset contains 100000 images of digital words. In both cases the train and validation data were spilt in 8:2 ratios of total data.

5.1.2 Implementaion Details of Modules

- **Training Module for handwritten character model**

```
class CNNModel(nn.Module):
    def __init__(self, num_classes=63):
        super(CNNModel, self).__init__()
        self.num_classes = num_classes
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.pool1 = nn.MaxPool2d(2, 2)  # Pooling layer

        self.conv2 = nn.Conv2d(32, 32, kernel_size=3, padding=1)
        self.pool2 = nn.MaxPool2d(2, 2)

        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.pool3 = nn.MaxPool2d(2, 2)

        self.conv4 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.pool4 = nn.MaxPool2d(2, 2)

        self.batch_norm = nn.BatchNorm2d(128)
        self.dropout = nn.Dropout(0.5)

        # Fully connected layer
        self.fc1 = nn.Linear(128 * 1 * 1, self.num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)

        x = F.relu(self.conv2(x))
        x = self.pool2(x)

        x = F.relu(self.conv3(x))
        x = self.pool3(x)

        x = F.relu(self.conv4(x))
        x = self.pool4(x)

        x = self.batch_norm(x)
        x = self.dropout(x)

        # Flatten the tensor
        x = x.view(x.size(0), -1)

        x = self.fc1(x)
        return F.log_softmax(x, dim=1)
```

Figure 5.1: Training module for CNN model

The training module for character model as shown in Figure 5.1 uses a CNN model to classify the images into 63 classes. It consists of several blocks which includes convolutional layers, pooling layers, batch normalization and dropout for regularization. After that, There is a convolutional layer with 32 filters, 3 * 3 kernel size and ReLU activation with padding 1. It also contains a maxpooling layer with 2*2 pooling size which reduces the spatial dimensions by half. There is a convolutional layer with 32 filters, 3*3 kernel size and ReLU activation with padding set to 1. It also contains a maxpooling layer with 2*2 pooling size which reduces the spatial dimensions further. After these, there is a convolutional layer with 64 filters, 3*3 kernel size and ReLU activation with padding set to 1. It also contains a maxpooling layer with 2*2 pooling size which reduces the spatial dimensions again. Then there is a convolutional layer with 128 filters, 3*3 kernel size and ReLU activation with padding set to 1. It also contains a maxpooling layer with 2*2 pooling size which reduces the dimensions even further. It also contains a batch normalization layer which stabilizes training and improves convergence. A dropout layer with a rate of 0.5 which randomly disables 50% of the neurons during training to prevent overfitting is also present in this block. Finally, it contains flattening and fully connected layers. The flattening layers converts the multidimensional output from the previous block to a 1D array using the view method. The fully connected layer contains 128*1*1 input features and the num_classes amount of output neurons. The final activation is log_softmax which outputs log probabilities for classification.

- **Training Module for word model**

```
class CRNN(nn.Module):
    def __init__(self, img_size, num_chars, dropout_rate=0.5):
        super(CRNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.dropout1 = nn.Dropout2d(dropout_rate)

        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.dropout2 = nn.Dropout2d(dropout_rate)

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.pool3 = nn.MaxPool2d(2, 2)
        self.dropout3 = nn.Dropout2d(dropout_rate)

        # Input dimension for LSTM
        self.rnn_input_dim = img_size[0] // 8 * 128

        # First LSTM layer (bidirectional)
        self.lstm1 = nn.LSTM(self.rnn_input_dim, 256, bidirectional=True, batch_first=True)

        # Second LSTM layer (bidirectional)
        self.lstm2 = nn.LSTM(512, 256, bidirectional=True, batch_first=True)

        self.dropout_rnn = nn.Dropout(dropout_rate)
        self.fc = nn.Linear(512, num_chars)

    def forward(self, x):
        # Convolutional layers
        x = self.pool1(torch.relu(self.bn1(self.conv1(x))))
        x = self.dropout1(x)

        x = self.pool2(torch.relu(self.bn2(self.conv2(x))))
        x = self.dropout2(x)

        x = self.pool3(torch.relu(self.bn3(self.conv3(x))))
        x = self.dropout3(x)

        # Prepare the input for LSTM
        b, c, h, w = x.size()
        x = x.permute(0, 3, 1, 2)  # Change shape to (batch_size, width, channels, height)
        x = x.view(b, w, -1)  # Reshape to (batch_size, width, rnn_input_dim)

        # Pass through the first LSTM layer
        x, _ = self.lstm1(x)

        # Pass through the second LSTM layer
        x, _ = self.lstm2(x)

        x = self.dropout_rnn(x)

        # Fully connected layer
        x = self.fc(x)
        return x
```

Figure 5.2: Training module for CRNN model

The training module for word model as shown in Figure 5.2 uses a CRNN model to predict the words in the images. It consists of multiple blocks which contain convolutional layers, batch normalization layers, pooling layers and dropout layers to improve regularization.

It contains a convolutional layer with 32 filters, 3*3 kernel size and padding set to 1. It also contains a batch normalization layer that normalizes the values to improve the training speed. It also contains a max pooling layer with pool size 2*2 that reduces the spatial dimensions by half. It also contains a dropout layer that randomly deactivates certain percentage of neurons during training to prevent overfitting. Then, it contains a convolutional layer with 64 filters, 3*3 kernel size and padding set to 1. It also contains a batch normalization layer that normalizes the values to improve the training speed. It also contains a max pooling layer with pool size 2*2 that further reduces the spatial dimensions by half. It also contains a dropout layer that randomly deactivates certain percentage of neurons during training to prevent overfitting. After that, it contains a convolutional layer with 128 filters, 3*3 kernel size and padding set to 1. It also contains a batch normalization layer that normalizes the values to improve the training speed. It also contains a max pooling layer with pool size 2*2 that again reduces the spatial dimensions by half. It also contains a dropout layer that randomly deactivates certain percentage of neurons during training to prevent overfitting.

The output from the convolutional layers is a 4D tensor (batch, channels, height, width) is transformed to send it as input to the LSTM layers. It is permuted to align the width with the sequence length which results in the shape of (batch, width, channels, height). It is then reshaped into (batch, width, rnn_input_dim) where rnn_input_dim combines the channels and height from the output of convolutional layers. This reshaped data is fed to the two bidirectional LSTM layers for learning sequential features. The first LSTM layer has 256 hidden units in each direction which results in 512 outputs for each timestep. The second LSTM layer also has 256 hidden units in each direction which further refines the learned pattern sequence. It also contains a dropout layer that prevents overfitting and improves regularization. Finally a fully connected layer which contains 512 input features maps the sequence-level LSTM outputs to the number of target classes (num_chars). This layers produces class scores for each timestep in the sequence which enables the model to classify the sequences.

- **Inference Module for word**

```python
def infer(image_path, model_path, char_file, img_size=(64, 256), is_gray=True):
    device = "cuda" if torch.cuda.is_available() else "cpu"

    # Load character dictionary
    char_dict = load_char_dict(char_file)

    # Load model
    num_chars = len(char_dict)
    model = CRNN(img_size, num_chars)
    model.load_state_dict(torch.load(model_path, map_location=device))
    model.to(device)
    model.eval()

    # Preprocess image
    img = preprocess_image(image_path, img_size, is_gray).to(device)

    # Predict
    with torch.no_grad():
        preds = model(img)
        preds = preds.softmax(2).argmax(2)  # Convert to probabilities and get max indices
        preds = preds.squeeze(0).cpu().numpy().tolist()

    # Decode predictions
    decoded_text = decode_predictions([preds], char_dict)[0]
    return decoded_text
```

Figure 5.3: Inference Module for Word recognition

The module shown in Figure 5.3 is responsible for making the predictions on the images uploaded by the user. First the character dictionary is loaded. Then the trained model is loaded. Then the image is preprocessed and fed to the model to get the predictions. The predictions are then decoded and the output is returned.

5.2 Testing

Testing is the process of checking the results of the system with the expected results to make sure that the system is performing as it should.

5.2.1 Test Case for Unit Testing

Unit testing tests the individual components or units of the system. It ensures each unit of the system is performing as it should be.

Table 5.1: Test Case for Character Recognition

S.N.	Test Cases	Test Description	Input Test Data	Expected Results	Actual Result	Remarks
1	TC-CHR-1	Recognize handwritten character	Image of handwritten characters	Output corresponding digital character	Output corresponding digital character	Pass
2	TC-CHR-2	Invalid Character recognition	Image without any characters	Output no character detected	Output no character detected	Pass

Table 5.2: Test Case for Word Recognition

S.N.	Test Cases	Test Description	Input Test Data	Expected Results	Actual Result	Remarks
1	TC-WRD-1	Recognize digital words	Image of digital words	Output corresponding digital word	Output corresponding digital word	Pass
2	TC-WRD-2	No word recognition	Image without any words	Blank Output	Blank Output	Pass

5.2.2 Test Case for System Testing

System testing tests the entire system as a whole to ensure that the system performs as it should. It makes sure that the system is working properly and whether it can be delivered to the end users or not.

Table 5.3: Test Case for PrachaRead System

S.N.	Test Cases	Test Description	Input Test Data	Expected Results	Actual Result	Remarks
1	TC-SYS-1	Handle unsupported file format	Non image file	Invalid file format	Invalid file format	Pass
2	TC-SYS-2	Evaluate recognition speed for character recognition	Image with characters	Corresponding output character within 1 second	Corresponding output character within 1 second	Pass
3	TC-SYS-3	Evaluate recognition speed for word recognition	Image with words	Corresponding output word within 1 second	Corresponding output word within 1 second	Pass

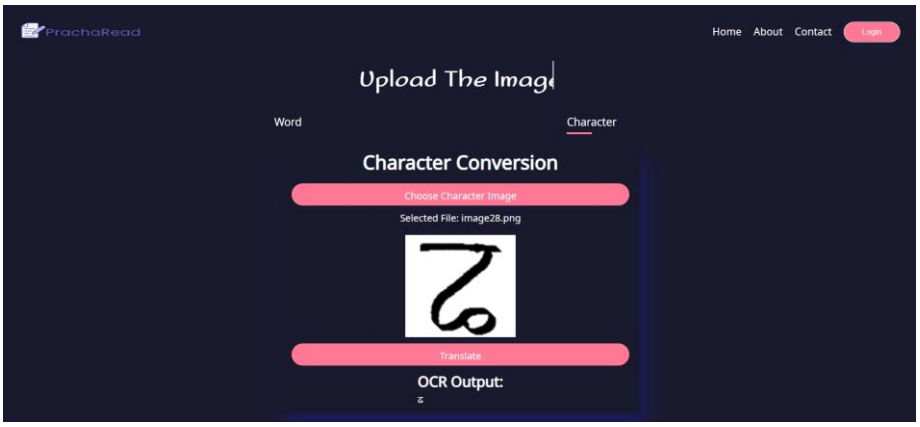

Figure 5.4: System Testing for Character Recognition Speed

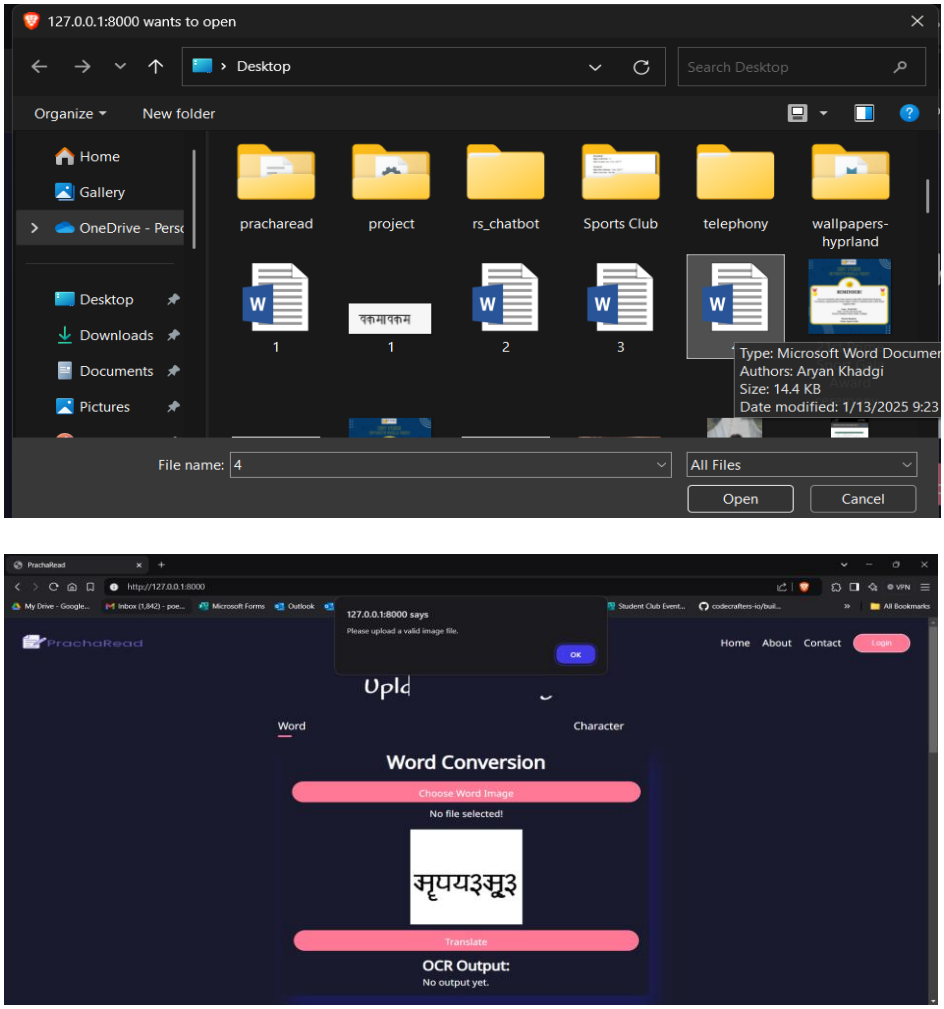

Figure 5.6: System Testing for Unsupported File Format

Figure 5.5: System testing for Word Recognition Speed

5.3 Result Analysis

The project was tested thoroughly using multiple unit and system testing to ensure that the system performed as per it expectations and requirements. A detailed performance report was generated to evaluate the various models employed which highlights their effectiveness in accurately classifying handwritten characters and digital words. This report serves as a valuable resource for future improvements which guides further development to increase the model's efficiency and accuracy.

The dataset was tested using different hyper parameters such as different batch sizes, optimizers and learning rates.
The following table is for character recognition.

Table 5.4: Analysis Table for Character Recognition

Batch Size	Lr	Optim.	Epoch	Network Architecture	Train Acc.	Train Loss	Val. Acc.	Val Loss
128	0.001	Adam	200	4 convolutional layers, 4 pooling layers, 1 batch normalization layer, 1 dropout layer, 1 fully connected layer	99.83 %	0.005	99.60 %	0.037
64	0.001	Adam	200	7 convolutional layers, 1 Batch normalization layer, 1 dropout layer, 1 fully connected layer	98.94 %	0.034	99.44 %	0.0185
64	0.0002	Adam	200	7 convolutional layers, 1 Batch normalization layer, 1 dropout layer, 1 fully connected layer	99.82 %	0.0121	99.44 %	0.0195

128	0.0001	Adam	200	7 convolutional layers, 1 Batch normalization layer, 1 dropout layer, 1 fully connected layer	99.77%	0.0299	99.68%	0.0172
64	0.001	Adam	200	4 convolutional layers, 4 pooling layers, 1 batch normalization layer, 1 dropout layer, 1 fully connected layer	99.83%	0.0049	99.29%	0.0303
128	0.0003	Adam	200	4 convolutional layers, 4 pooling layers, 1 batch normalization layer, 1 dropout layer, 1 fully connected layer	99.85%	0.0049	99.21%	0.1143
128	0.001	SGD	200	4 convolutional layers, 4 pooling layers, 1 batch normalization layer, 1 dropout layer, 1 fully connected layer	92.41%	0.4535	92.77%	0.4056
32	0.0001	Adam	200	4 convolutional layers, 4 pooling layers, 1 batch normalization layer, 1 dropout layer, 1 fully connected layer	99.97%	0.0025	99.52%	0.0125
128	0.0001	Adam	200	4 convolutional layers, 4 pooling layers, 1 batch normalization layer, 1 dropout layer, 1 fully connected layer	99.90%	0.0072	98.89%	0.052

128	0.0001	Adam	50	4 convolutional layers, 4 pooling layers, 1 batch normalization layer, 1 dropout layer, 1 fully connected layer	97.21 %	0.1681	96.43 %	0.162
128	0.0001	Adam	100	4 convolutional layers, 4 pooling layers, 1 batch normalization layer, 1 dropout layer, 1 fully connected layer	99.80 %	0.0211	99.60 %	0.0322

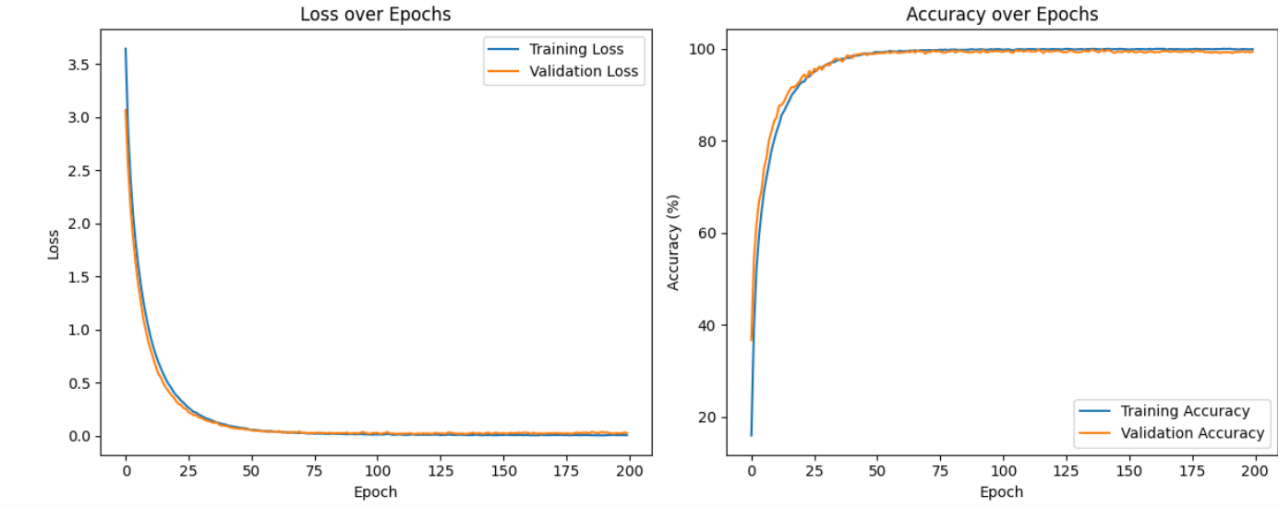

Figure 5.7: Accuracy and Loss for Character Recognition

The following table is for word recognition:

Table 5.5: Analysis Table for Word Recognition

Batch Size	Lr	Optim.	Epoch	Network Architecture	Train Acc.	Train Loss	Val. Acc.	Val Loss	CER	WER
32	0.003	SGD	40	3 blocks: 1 convolutional layer, 1 batch normalization ,1 pooling, 1 dropout, 1 block: 2 LSTM layers, 1 fully connected layer	79.54%	0.1744	82.05%	0.1561	4.61%	17.95%
32	0.001	SGD	40	3 blocks: 1 convolutional layer, 1 batch normalization ,1 pooling, 1 dropout, 1 block: 2 LSTM layers, 1 fully connected layer	73.51%	0.2728	78.12%	0.2151	6.17%	21.88%
64	0.0001	Adam	40	3 blocks: 1 convolutional layer, 1 batch normalization ,1 pooling, 1 dropout, 1 block: 2 LSTM layers, 1 fully connected layer	86.04%	0.0828	83.70%	0.1574	4.05%	16.3%
128	0.0001	Adam	40	3 blocks: 1 convolutional layer, 1 batch normalization ,1 pooling, 1 dropout, 1 block: 2 LSTM layers, 1 fully connected layer	85.60%	0.0895	83.85%	0.1488	4.01%	16.15%
128	0.001	Adam	20	3 blocks: 1 convolutional layer, 1 batch normalization ,1 pooling, 1 dropout, 1 block: 2 LSTM layers, 1 fully connected	82.23%	0.1334	83.28%	0.1412	4.14%	16.72%

				layer						
128	0.001	Adam	15	3 blocks: 1 convolutional layer, 1 batch normalization ,1 pooling, 1 dropout, 1 block: 2 LSTM layers, 1 fully connected layer	80.76%	0.1534	82.85%	0.1487	4.39%	17.15%
128	0.001	Adam	10	3 blocks: 1 convolutional layer, 1 batch normalization ,1 pooling, 1 dropout, 1 block: 2 LSTM layers, 1 fully connected layer	76.79%	0.2099	79.04%	0.1935	5.65%	20.96%
32	0.003	Adagrad	40	3 blocks: 1 convolutional layer, 1 batch normalization ,1 pooling, 1 dropout, 1 block: 2 LSTM layers, 1 fully connected layer	73.05%	0.2818	76.67%	0.2375	6.81%	23.3%
32	0.001	Adadelta	40	3 blocks: 1 convolutional layer, 1 batch normalization ,1 pooling, 1 dropout, 1 block: 2 LSTM layers, 1 fully connected layer	2.77%	4.127	3%	4.078	97%	97%

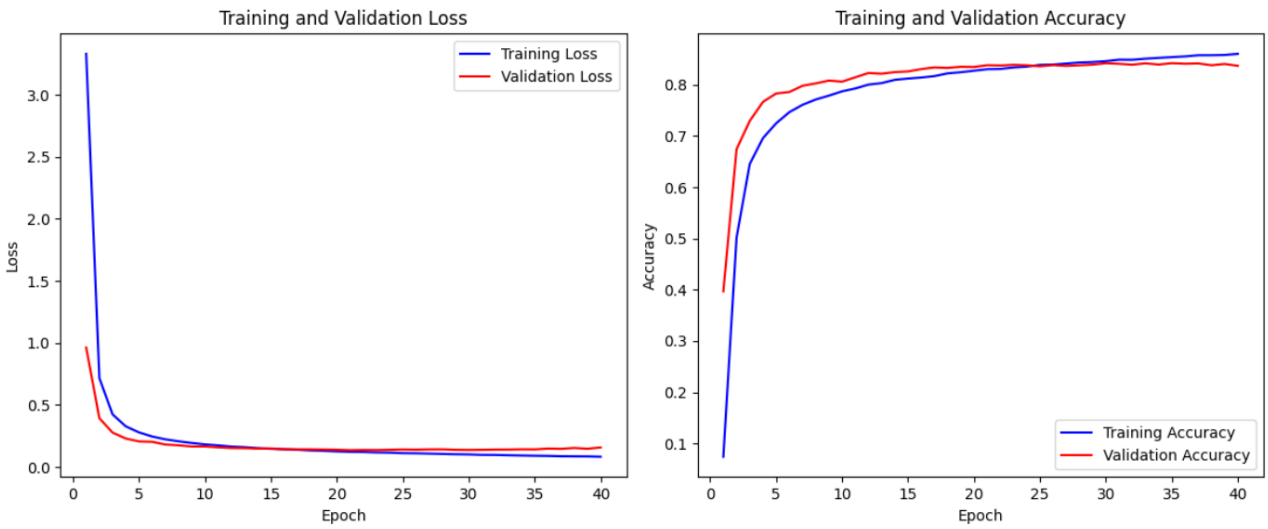

Figure 5.8: Accuracy and Loss for Word Recognition

5.3.1 Performance Metrics for CNN classification of Newari Characters

The following is the summary of performance metrics for the CNN model.

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	20
1	1.00	1.00	1.00	28
10	0.96	1.00	0.98	22
11	0.94	1.00	0.97	17
12	1.00	1.00	1.00	16
13	0.89	1.00	0.94	17
14	1.00	1.00	1.00	25
15	1.00	1.00	1.00	16
16	1.00	0.83	0.91	29
17	1.00	1.00	1.00	23
18	1.00	1.00	1.00	16
19	1.00	1.00	1.00	25
2	1.00	1.00	1.00	22
20	1.00	1.00	1.00	15
21	1.00	1.00	1.00	17
22	1.00	0.90	0.95	20
23	0.95	1.00	0.97	18
24	1.00	1.00	1.00	16
25	1.00	1.00	1.00	21
26	1.00	1.00	1.00	22
27	1.00	1.00	1.00	22
28	1.00	1.00	1.00	17
29	1.00	1.00	1.00	15
3	1.00	1.00	1.00	12
30	0.96	1.00	0.98	26
31	1.00	1.00	1.00	18
32	1.00	1.00	1.00	12
33	1.00	1.00	1.00	29
34	1.00	1.00	1.00	17
35	1.00	1.00	1.00	18
36	1.00	1.00	1.00	19
37	1.00	1.00	1.00	18
38	1.00	1.00	1.00	18
39	1.00	1.00	1.00	27
4	1.00	1.00	1.00	20
40	1.00	1.00	1.00	26
41	1.00	1.00	1.00	25
42	1.00	1.00	1.00	13
43	1.00	1.00	1.00	15
44	1.00	1.00	1.00	17
45	1.00	1.00	1.00	10
46	1.00	1.00	1.00	16
47	0.94	1.00	0.97	16
48	1.00	1.00	1.00	22
49	1.00	1.00	1.00	17
5	1.00	1.00	1.00	17
50	1.00	1.00	1.00	23
51	1.00	1.00	1.00	23
52	1.00	1.00	1.00	17
53	1.00	1.00	1.00	21
54	1.00	1.00	1.00	20
55	1.00	1.00	1.00	22
56	1.00	1.00	1.00	27
57	1.00	1.00	1.00	20
58	1.00	1.00	1.00	23
59	0.97	1.00	0.98	31
6	1.00	1.00	1.00	18
60	1.00	0.97	0.98	32
61	1.00	1.00	1.00	16
62	1.00	1.00	1.00	20
7	1.00	1.00	1.00	20
8	1.00	1.00	1.00	17
9	1.00	1.00	1.00	22
accuracy			0.99	1259
macro avg	0.99	1.00	0.99	1259
weighted avg	0.99	0.99	0.99	1259

Figure 5.9: Performance Metrics for CNN model

In figure 5.9 which shows the classification report for the character recognition, we see that the model's overall accuracy is 99% which means the model correctly classified the instances 99% of the time. The precision, recall and f1 scores of the classes are 1 or almost equal to 1 which shows exceptional performance of the model in all classes. The macro average and weighted average is 0.99 which shows the strong performance of the model across all classes.

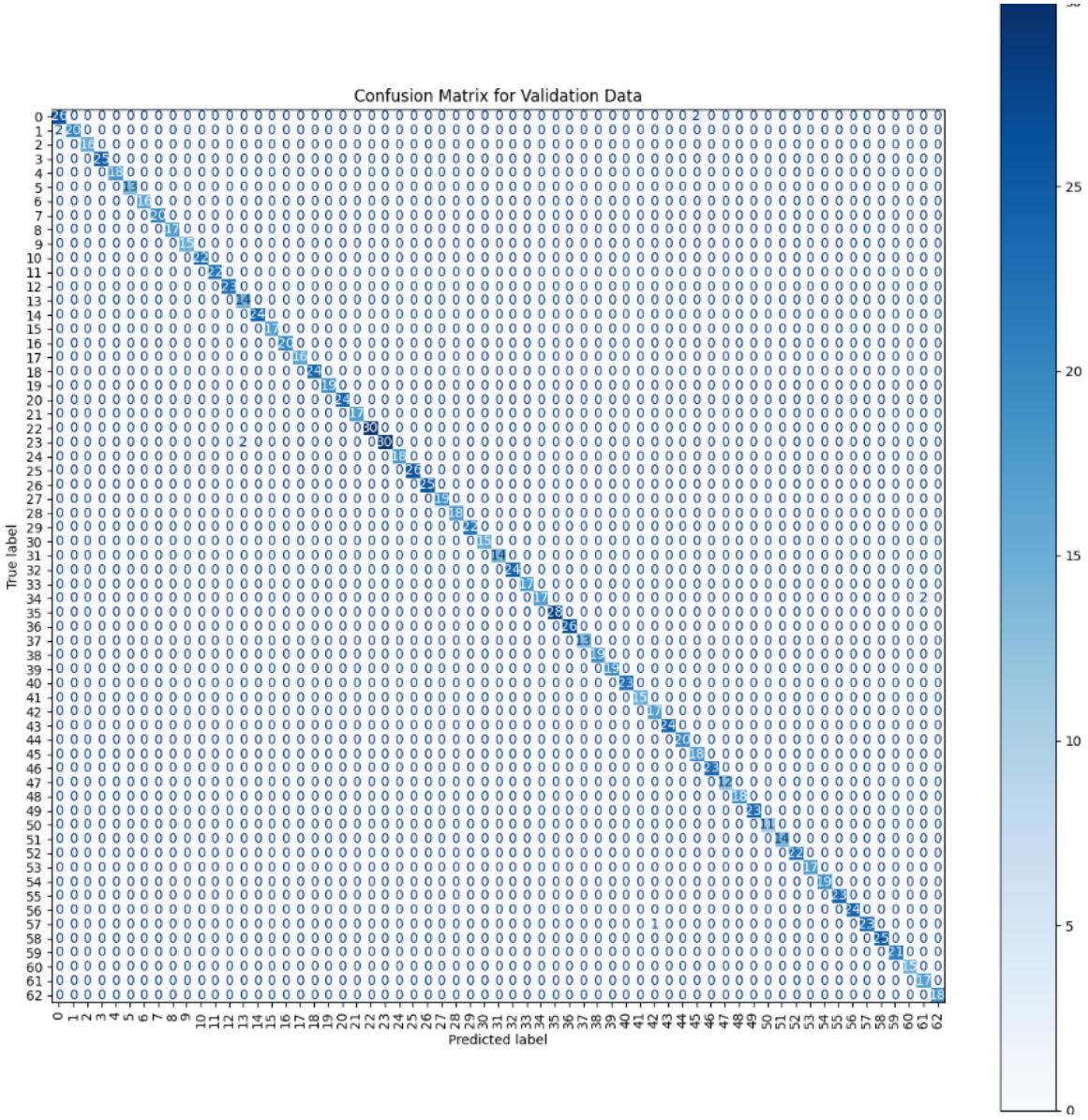

Figure 5.10: Confusion Matrix for CNN model

In figure 5.10, the confusion matrix for the character model can be seen where the rows represent the true labels and the columns represent the predicted labels.

- **Diagonal Elements:** The diagonal elements represents the instances where the predicted labels is equal to the true labels.

- **Non-diagonal elements:** The non-diagonal elements show the instances where the predicted labels are not equal to the true labels.

From the confusion matrix we can see that almost all the classes are correctly classified with only some which are misclassified such as 41 and 57 with only one instance where they were misclassified.

5.3.2 Performance Metrics for CRNN recognition of Newari Words

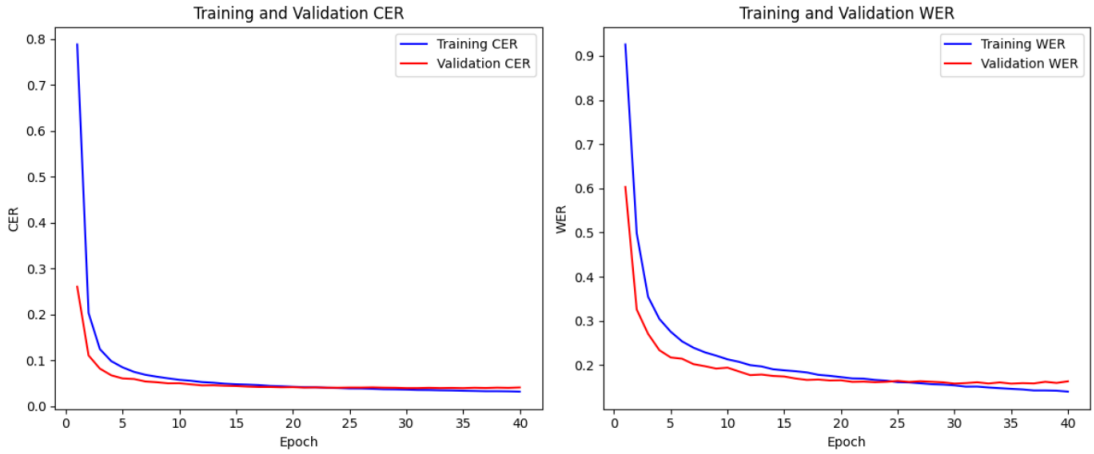

Figure 5.11: CER and WER of CRNN model

The figure 5.11 shows the training and validation CER (Character Error Rate) and WER (Word Error Rate) for the CRNN model. It is clear that the CER and WER are both decreasing across the epoch reaching up to 4.05% CER and 16.30% WER. It shows that the systems makes 4.05% error at the character level and 16.30% error at the word level.

CHAPTER 6
CONCLUSION AND FUTURE RECOMMENDATIONS

6.1 Conclusion

In conclusion, the development of this system that recognizes handwritten characters and digital words is a significant step towards preserving and digitizing the newari historical scripts. This system achieved this through the help machine learning algorithms such as CNN and CRNN and computer vision libraries such as OpenCV. CNN is used to extract features from images and classify them. CRNN is used to extract features form images and learn the sequential dependencies and make predictions based on them.

This project accurately recognizes the handwritten vowels, consonants and numbers of Prachalit script with an accuracy of 99%. This model performs great with images with clear distinction between background and the characters. This project also accurately predicts the digital words present in images with an accuracy of 83%.

Hence, this project allows further progress in the field of OCR development of newari languages which can facilitate the digitization of scripts with cultural and historical importance of Newari community.

6.2 Future Recommendation

There are tons of future recommendations that could be added to this project. They are:

- **Recognition of handwritten words, lines and documents:** This project only recognizes the handwritten characters of newari script. So addition of recognition of words, lines and documents would help the project to improve a lot.

- **Context understanding:** Currently this project only recognizes words and doesn't involve understanding the context and meaning of the words. Addition the context understanding further helps to improve the accuracy of the model to correctly output the correct and meaningful words.

- **Handwriting generation:** This project recognizes the handwritten characters. The addition of generation of handwritten characters can help users to type digital text and convert them handwritten form.

References

[1] S. Nakarmi, S. Sthapit, A. Shakya, R. Chulyadyo and B. K. Bal, "Nepal Script Text Recognition Using CRNN CTC Architecture," ELRA and ICCL, Torino, Italia, 2024.

[2] J. Bati and R. P. Dawadi, "Ranjana Script Handwritten Character Recognition using CNN," Dhulikhel, 2023.

[3] S. Ram, S. Gupta and B. Agarwal, "Devanagri character recognition model using deep convolution neural network," *Journal of Statistics and Management Systems,* vol. 21, no. 4, pp. 593-599, 2018.

[4] A. S. A. J. A. K. A. Gupta, "Devanagari Character Recognition Using MLP-Mixer and CNN Extracted Features," in *2023 IEEE World Conference on Applied Intelligence and Computing (AIC),* Sonbhadra, India, 2023.

[5] A. Baral, D. Khanal, P. Baskota and P. Dahal, "HANDWRITTEN NEPALI CHARACTER RECOGNITION AND NARRATION SYSTEM USING DEEP CNN," Purwanchal Campus, Institute of Engineering, Dharan, 2019.

[6] A. Ghimire, A. Chapagain, U. Bhattarai and A. Jaiswal, "Nepali Handwriting Recognition using Convolution," *International Research Journal of Innovations in Engineering and Technology (IRJIET),* vol. 4, no. 5, pp. 5-9, 2020.

[7] "Springer Nature Link," [Online]. Available: https://link.springer.com/article/10.1007/s00138-018-0942-y. [Accessed 26 12 2024].

[8] "Dida," [Online]. Available: https://dida.do/what-is-an-lstm-neural-network. [Accessed 20 12 2024].

[9] "Paperswithcode," [Online]. Available: https://paperswithcode.com/method/max-pooling. [Accessed 20 12 2024].

[10] "Research Gate," [Online]. Available: https://www.researchgate.net/figure/Schematic-diagram-of-a-basic-convolutional-neural-network-CNN-architecture-26_fig1_336805909. [Accessed 20 12 2024].

Appendices

Figure: Home Page

Figure: Word Conversion Page

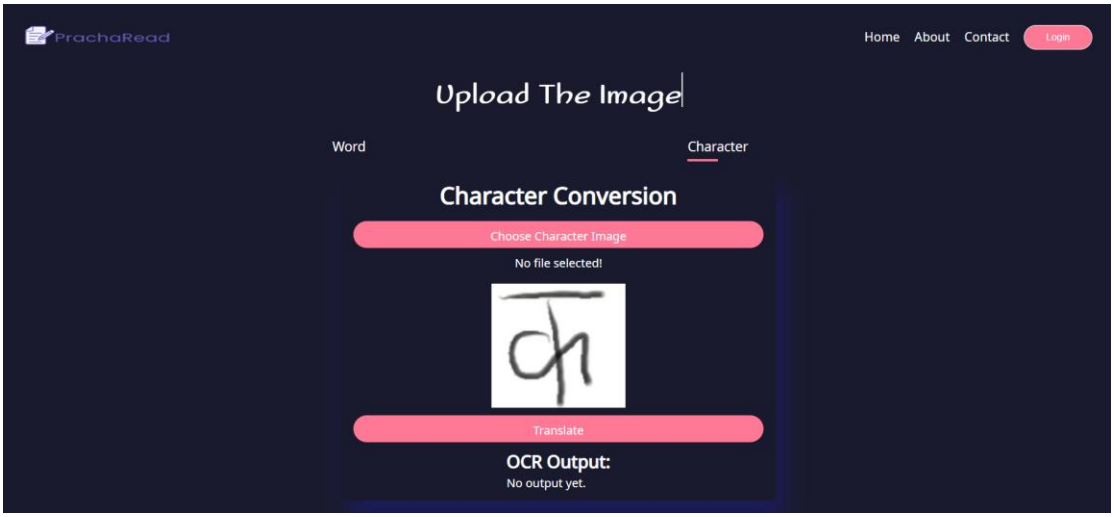

Figure: Character Conversion Page

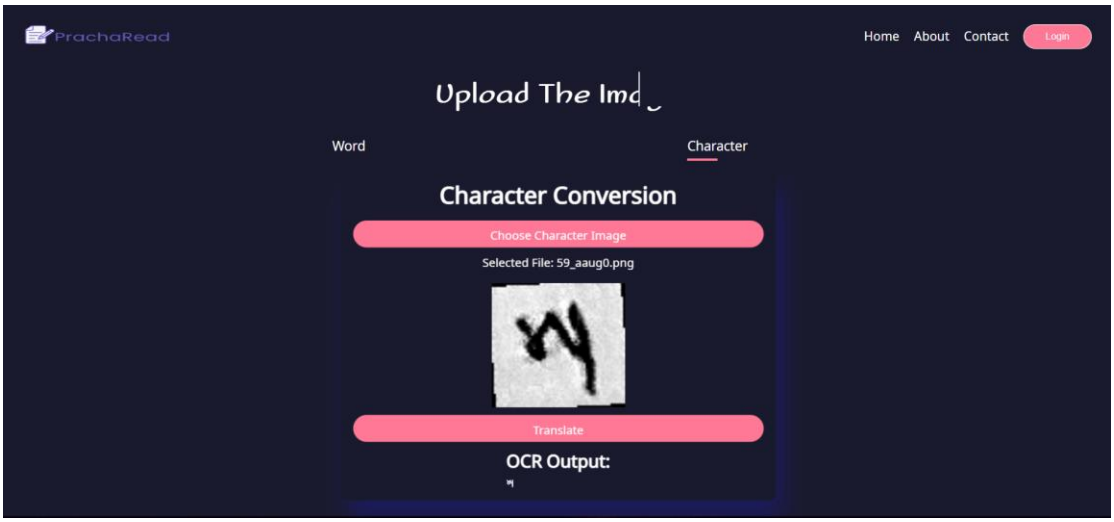

Figure: Character Converted Demo

Figure: Word Conversion Demo

Figure: No character in Character conversion

Figure: Black image in word conversion

PrachaRead: Newari Text recognition

ORIGINALITY REPORT

5%
SIMILARITY INDEX

MATCH ALL SOURCES (ONLY SELECTED SOURCE PRINTED)

★www.serri.org 3%
Internet

EXCLUDE QUOTES	ON	EXCLUDE SOURCES	< 1%
EXCLUDE BIBLIOGRAPHY	ON	EXCLUDE MATCHES	< 10 WORDS