TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
SAGARMATHA ENGINEERING COLLEGE

A PROJECT REPORT ON
NEPALI HANDWRITTEN TEXT RECOGNITION

BY
JEEVAN KAPHLE 26365
KESHAB SAUD 26366
NAKUL POUDEL 26373
NIRAJAN GYAWALI 26375

A PROJECT REPORT SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR IN COMPUTER ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
SANEPA, LALITPUR, NEPAL

April 12, 2022

NEPALI HANDWRITTEN TEXT RECOGNITION

BY

JEEVAN KAPHLE 26365

KESHAB SAUD 26366

NAKUL POUDEL 26373

NIRAJAN GYAWALI 26375

Project Supervisor

Er. Saurav Raj Pant

A project report submitted in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering

Department of Electronics and Computer Engineering

Sagarmatha Engineering College

Tribhuvan University

Lalitpur, Nepal

April 12, 2022

COPYRIGHT©

DECLARATION

The dissertation is submitted for fulfillment of degree in **Bachelor's in Computer Engineering** to Department of Electronics and Computer Engineering, Sagarmatha Engineering College, under the title **"NEPALI HANDWRITTEN TEXT RECOGNITION"**. We promise this work has not been done previously and is completed by our own work. This work has not been previously submitted to the university to fulfill any academic award.

Department of Electronics and Computer Engineering, Sagarmatha Engineering College has authorized to freely distribute this project to other intuitions or to any individuals for further scholarly research work.

Jeevan Kaphle 26365
Keshab Saud 26366
Nakul Poudel 26373
Nirajan Gyawali 26375
Date: April 12, 2022

ACKNOWLEDGEMENT

We are very grateful to the Department of Electronics and Computer Engineering, Sagarmatha Engineering College for providing us the opportunity to undertake this project as the successful output of the course called 'Major Project' considered as the partial fulfillment of B.E. Computer Engineering, IOE including it as a part of the curriculum in B.E. Computer Engineering.

We are extremely thankful to Er. Saurav Raj Pant, our Project Supervisor for providing us with his invaluable guidance and support throughout every project development phase. His useful suggestions and continuous motivation are sincerely acknowledged.

Also, we express sincere thanks to our colleagues and respected seniors and juniors for their positive advice and support.

Last but not least, we would like to thank everyone who helped and motivated us to work on this project.

Jeevan Kaphle 26365

Keshab Saud 26366

Nakul Poudel 26373

Nirajan Gyawali 26375

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
BLSTLM	Bidirectional Long Short Term Memory
CNN	Convolutional Neural Network
CRNN	Convolutional Recurrent Neural Network
CSS	Cascading Style
CTC	Connectionist Temporal Classification
DNN	Deep Neural Network
GRU	Gated Recurrent Unit
HTML	Hypertext Markup Language
HTR	Handwritten Text Recognition
LSTM	Long Short Term Memory
ML	Machine Learning
MLSTM	Multidirectional Long Short Term Memory
NN	Neural Network
OCR	Optical Character Recognition
RNN	Recurrent Neural Network
SDLC	Software Development Life Cycle

CHAPTER 1

INTRODUCTION

Handwritten character recognition is a technique to convert handwritten images into text files that can be edited [1]. Handwritten character recognition is a very complex task in contrast to printed character recognition as the characters are written in various styles and sizes. Every individual has a different way of writing. So, 100 % accuracy is difficult to achieve as everyone has a different way of writing style. Nepali Handwritten Text Recognition is one of the methods of conversion of pictures of composed, written by hand, or printed content into machine-encoded text. The word is recognized following different algorithms.

1.1 Background

Nepal is a multi-linguistic country. The languages are spoken in Nepal such as Nepali, Maithili, Bhojpuri, Newari, Tharu, Hindi, Sanskrit, etc. use the Devanagari script. Most of the people speak Nepali as it is the national language. All the learning and government work uses Nepali as an official language. So for the digitization of the Nepali language, Devanagari OCR can be a landmark. Handwritten OCR is one of the most challenging and demanding areas of interest for researchers in the domain of image processing and pattern recognition [1]. Many researchers have worked with recognition of characters of different languages but there is less work carried out for Devanagari script. Handwriting recognition has gained a lot of attention in the field of pattern recognition and machine learning due to its application in various fields. Character classification is an important part of many computer vision problems like Optical character recognition, license plate recognition, and also to digitize texts, papers, notebooks, etc. It can have contributions in automation and especially to develop applications for helping visually impaired people. It mainly deals with training thousands of images of a certain pattern and predicting the output of test data given afterward. The accuracy rate of learning depends mainly upon the numbers of data trained and the selection of the learning algorithms [2]. Handwritten Devanagari character

recognition is more complex due to many possible variations in order, number, modifiers used, direction, and shape of constituent strokes [3]. There are big challenges to processing its documents due to linguistic-based criticalities, large character set, complex conjuncts, the typical geometrical structure of character, and due to the presence of header line "shirorekha" that connects Devanagari character to form a word [4]. The presence of the header line makes the segmentation process of characters even more difficult. Handwriting is written in various styles and sizes and every individual has a unique way of writing which adds to the complexity [5]. In the Devanagari script, there are 12 vowels and 36 consonants along with 10 numerals. There is no concept of upper and lowercase in Devanagari as in English. All the characters have a horizontal line in the upper part known as "Shirorekha". Other constituent symbols in Devanagari are a set of modifiers known as "Matras" [1].

1.2 Problem Definition

Nepal has not progressed much in the field of digitalization. Despite the advancements in science and technology, we are still entangled in tedious paper works. Most of the paper works in Nepal is done in Nepali and uses the Devanagari script. It's time to shift from paperwork to digital so that all the traditional documents written in Nepali paper by hand could be preserved and the coming generations can get access to that valuable information about the past. To do so, it will be more complicated to write all the documents to text manually. So, we have come up with a project which focuses on the recognition of handwritten and printed text images which will convert the handwritten or printed text images to editable text format. The proposed work in this report addresses the Recognition of Devanagari text from images by using CRNN Model.

1.2.1 Challenges in Handwriting Recognition

- Huge variability and ambiguity of strokes from person to person.
- The handwriting style of an individual person also varies from time to time and is inconsistent.

- Poor quality of the source document/image due to degradation over time.
- Text in printed documents sit in a straight line whereas humans need not write a line of text in a straight line on white paper.
- Text in handwriting can have variable rotation to the right which is in contrast to the printed text where all the text sits up straight.
- Collecting a good labeled data set to learn is not cheap.

1.3 Objective

This research work is conducted in order to:

- To recognize the handwritten Nepali words.

1.4 Features

- It converts the text from the image into machine-editable format.
- It recognizes the words containing modifiers.
- It can extract texts from image formats such as jpg, png, jpeg.
- Our web app has an attractive, simple, and easy-to-use interface. Overall, it provides a better user experience.

1.5 Feasibility

The proposed project "Nepali Handwritten Text Recognition" is feasible. The various aspects of feasibility are described below.

1.5.1 Technical Feasibility

Our project is considered to be technically feasible because it has the necessary expertise, infrastructure, and capital to develop, install, operate and maintain the system. We have gone through many of the research articles and we learned the necessary technical skills needed for doing this project. We have used softwares like google Colab, python, NumPy, Keras, and many more applications. Most of them

are open-sourced for years making it feasible in the software department which makes the project technically feasible.

1.5.2 Operational Feasibility

Our project is beneficial because it can be turned into the systems that will recognize the handwritten words present into various documents. It can save the time of manual typing. The digital documents provides more security and are reliable than traditional handwritten documents. So, this project is operationally feasible.

1.5.3 Economic Feasibility

Our project needs a minimum economic cost to implement, so it can be easily implemented and is economically feasible.

1.6 System Requirements

1.6.1 Software Requirements

- Google Colab
- Google Chrome
- Windows 7 or Ubuntu 18 and above OS

1.6.2 Hardware Requirements

- NVidia Tesla K80 GPU for training model.
- Google Colab 12 GB RAM, 78 GB Disk
- Intel Core i5 64-bit Processor

CHAPTER 2

LITERATURE REVIEW

Nepali Optical Character Recognition is a field where significant research hasn't been carried out. Text recognition has been one of the major topics for machine vision. Handwritten character recognition's main purpose is to digitize the document written in papers and notebooks. In today's world, the digitalized format of a document is much more convenient for sharing, storing, and security [1]. Optical character recognition is a sequence of multiple processes – segmentation, feature extraction, and classification. Different models or techniques are proposed for character segmentation. Various segmentation, feature extraction, and classification techniques have been proposed by different researchers. Classification is one of the major steps in OCR and the design of a good classifier is also a challenging task. Mostly supervised learning is used for the classification of characters [6]. In Nepal, some of the works have been carried out to recognize Devanagari characters. The paper "Nepali OCR Using Hybrid Approach of Recognition" has proposed a hybrid OCR system for printed Nepali text using the Random Forest (RF) algorithm. It incorporates two different techniques of OCR – firstly, the Holistic approach and secondly, the Character Level Recognition approach. The system first tries to recognize a word as a whole and if it is not confident about the word, the character level recognition is performed. The performance of 78.87% and 94.80% recognition rates are achieved for the character-level recognition approach and the hybrid approach respectively [6]. Similarly, the paper" Optical Character Recognition System for Nepali Language Using Convnet" describes the implementation of a CNN (Convolutional Neural Network) based Optical Character Recognition System for the Nepali Language. The system has been developed in python using Keras library on top of Theano and NumPy for character recognition. The system has given encouraging results for a limited domain only [7]. The paper" Deep Learning-Based Large Scale Handwritten Devanagari Character Recognition" by Acharya et al. [8] uses a relatively small dataset of handwritten character sets to classify 46 characters. This paper introduces the use of Convolutional Neural

Networks for classification. However, this project only addresses a small field of recognition. Another related paper is "Optical Character Recognition for Nepali, English Character and Simple Sketch Using Neural Network" by Shakya et al. [9]. In this paper they proposed an OCR for Nepali text in Devanagari Script, using multi-layer feed-forward back propagation Artificial Neural Network(ANN).

In a Bangla character segmentation article , an attempt is made for the development of a tri-level (line, word, and character) segmentation scheme without any normalization[10]. The proposed approach of line segmentation is a hybrid approach. This word segmentation is based on connected component analysis. Character segmentation is performed on a relatively large number of word sets using an existing zone and character segmentation method. The main drawback of this technique lies in word and character segmentation where it fails to reduce over-segmentation. Average accuracy of 90.46% was obtained for line segmentation.
The research paper [11] is based on header line detection, upper modifier detection, lower modifier detection, character detection, and contour following technique. Paper introduced the concept of resizing the image so that the system is able to reduce space as well as time complexity. Firstly, the contour following after header line detection correctly separates the upper modifier then character and lower modifier. Secondly, this paper provides a brief review of text line segmentation techniques for handwritten text. This paper used a structural approach in which the system looked at the similarities in the structure of different characters, like the location of the vertical bar and the joint characters that are made up of half consonant and full character, etc. The uncertainty associated with the structures of characters arises from different writing styles. To take account of this uncertainty, the paper used windows to enclose certain portions of the characters for making the decisions such as whether the character is simple or joint. This paper gives more stress to the general conditions that are applicable to most of the characters. In our project, we will use the Deep Learning approach using Recurrent Neural Network as this approach has proved to be promising in the field of Optical Character Recognition. We will use previously collected real-world data sets and those data sets we have collected to get a promising result in Devanagari OCR.

CHAPTER 3

RELATED THEORY

Handwritten Character Recognition is a technique that scans the file and converts it to equivalent text. Comparatively, English Text Recognition is easier than that of Nepali Text Recognition. In this project, we deal with the Nepali Handwritten Text Recognition. Optical Character Recognition (OCR) is a technique that takes an image as an input and converts it to a recognizable form. The OCR that is present in today's world mostly uses the machine learning technique that includes deep learning.

3.1 Devanagari Script

Like other languages, Nepali is one of the most widely used languages all over Nepali for setting the records of the individuals of the country. As the English language uses the English alphabet, the Nepali language uses a script called Devanagari. This script is unique from all other languages as it consists of vowels and consonants along with modifiers and half words as well so, it is not easy learning English [12].

3.2 Characteristics of Devanagari Script

Devanagari script has 36 consonants and 12 vowels which are the basic characters. Besides these basic characters, there are also special symbols in the script known as Modifiers. These are known as the modifiers of the script and are placed above, below, left, or right of a character. Also, there are compound characters in the script which are formed by the combination of basic characters [13].

Vowels	अ	आ	इ	ई	उ	ऊ	ऋ
	ॠ	ए	ऐ	ओ	औ	अं	अः

Figure 3.1: Devanagari Vowels

क	ख	ग	घ	ङ
च	छ	ज	झ	ञ
ट	ठ	ड	ढ	ण
त	थ	द	ध	न
प	फ	ब	भ	म
य	र	ल	व	
श	ष	स	ह	
क्ष	त्र	ज्ञ		

Figure 3.2: Devanagari Consonant

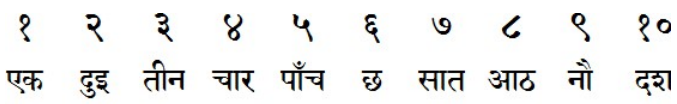

Figure 3.3: Devanagari Numerals

3.3 Deep Learning

Deep learning is one of the topics of machine learning in artificial intelligence. It has networks that are capable of learning unsupervised data. Most modern deep learning models are based on artificial neural networks, specifically recurrent neural networks (RNNs). The deep learning concept is widely used in most projects in today's world.

3.4 Convolutional Neural Network (CNN)

CNNs are introduced to acoustic modeling in which convolution was introduced over windows of acoustic frames that overlap in time to learn more about more robust acoustic features for the mobile, speaker, and gender groups[44]. Compared with conventional speech features (MFCC, PLP, LPC, etc.), CNN can use local filtering and maximum pooling techniques to get more robust features. CNN was originally developed to tackle issues like computer vision. The spectral characteristics of speech signals may therefore be considered an image [14]. CNN is a neural network that has a common structure. CNN, the first layer consisting of several characteristic graphs, is called a convolution layer. The growing neuron receives input from a local receptive field representing features of a small frequency range in the convolution layer. Neurons belonging to the same function map share the same weights (also called filters or kernels) but obtain different frequency-shifting inputs. As a consequence, the convolution layer performs a kernel convolution with the activation of the lower layer. The different operations that are held under the CNN can be described by the figure given below:

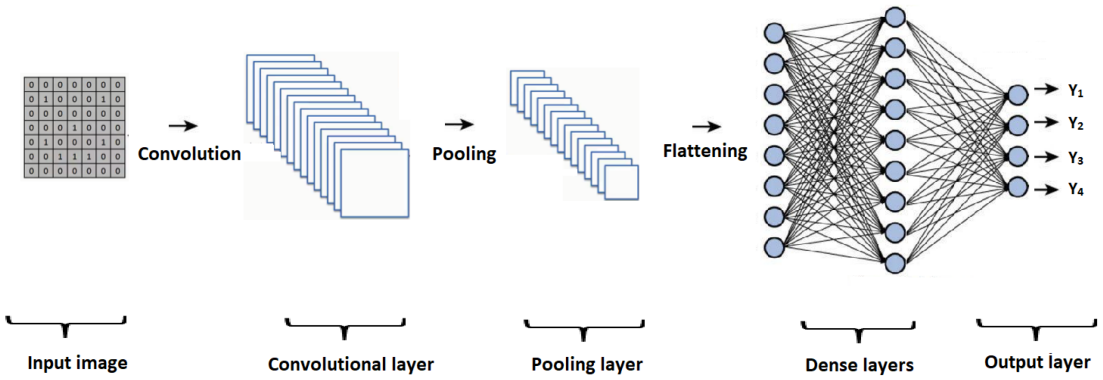

Figure 3.4: Different Operations in CNN

3.5 Multi-dimensional Recurrent Neural Networks

Recurrent Neural Networks (RNN) or Long Short-Term Memory (LSTM), which is a specific RNN architecture, are known to be able to deal with sequential data to identify temporal patterns and generate results. At least the most common configurations of these kinds of architectures. Nevertheless, there's a significant limitation: they can only handle 1D data and as a consequence, these architectures can't be applied to image data directly [15].

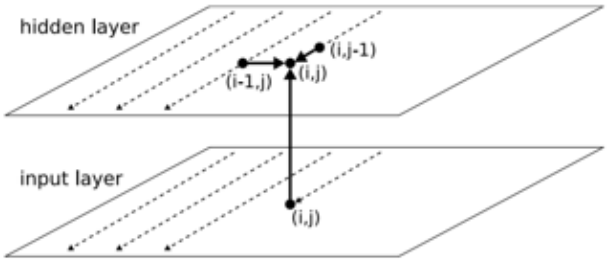

Figure 3.5: Two-dimensional MDRNN

Here in the image, the thick lines show connections to the current point (i,j). The connections within the hidden layer plane are recurrent. The dashed lines show the scanning strips along which previous points were visited, starting at the top left corner.

The difference between a traditional RNN and the proposed multi-dimensional RNN is described in the example. Usually, in general RNNs, hidden layer i receives

state from a previous hidden layer in time i-1. But in cases of multi-dimensional RNNs, let's take the example of a 2-dimensional RNN structure, the hidden layer (i, j) receives states from multiple previous hidden layers, being (i-1, j) and (i, j-1). Therefore, such architectures capture context from both height and width in an image which is pivotal for getting a clear understanding of the local region [15]. You can further extend that to also get information from future layers. So, this is pretty similar to how Bidirectional Long Short-Term Memory (BI-LSTM) networks receive information from both t-1 and t+1. To get back to our hypothetical 2-dimensional RNN structure, here we would now be able to receive information in both directions as well, e.g. (i-1, j), (i, j-1), (i+1, j), (i, j+1) and thus, capturing context in all directions just as with the BI-LSTM.

So, what we can see is basically the entire multi-dimensional RNN structure. We can simply replace the RNN block with an LTSM block. The network's input is divided into blocks of size 3×4 which is fed into the multi-dimensional RNN layers respectively the network's hierarchical structure of multi-dimensional RNN layers followed by feed-forward layers in tandem. Feed-forward layers are classical artificial neural networks. The final output is converted into a 1D vector and is passed to a Connectionist Temporal Classification (CTC) function to generate an output [16].

3.5.1 Gated Recurrent Unit Network

A Gated Recurrent Unit (GRU) is part of a particular Recurrent Neural Model Network which intends to use the connections to perform a sequence of nodes machine learning tasks related, for example, to memory and clustering in speech recognition. Gated recurrent units help change the input weights of the neural network to solve the vanishing gradient problem common to recurrent neural networks.

Update gate and reset gate is used by the GRU units [17]. The Sigma notation above describes those gates that allow a GRU to transfer information over several periods of time in order to affect a future time span. In certain words, for a certain amount of time, the value is stored in memory and at a certain stage, taking the

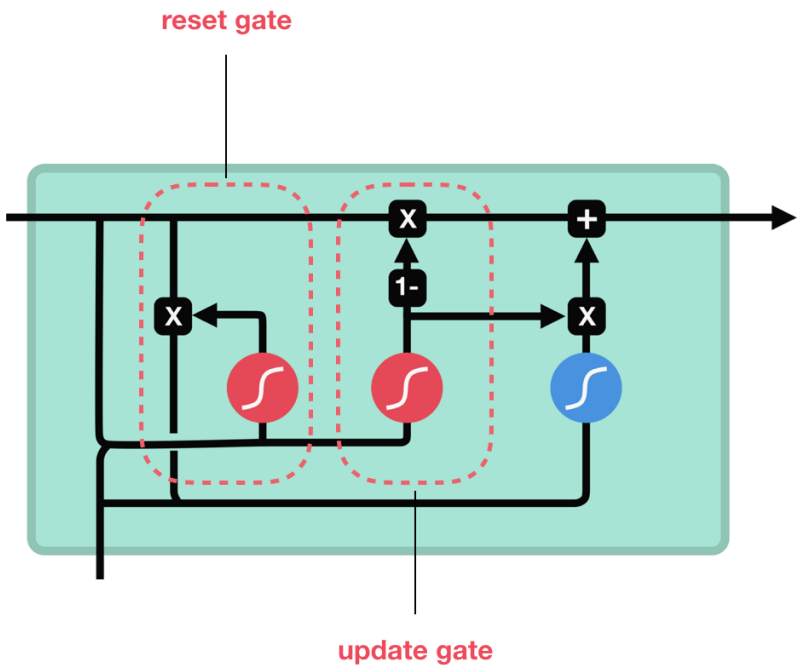

Figure 3.6: Structure of GRU

value out and using it with the current state to change at a future state.

Update gate

This gate calculates update gate Z_t by executing the following steps at the time step

$$Z_t = \sigma(W_z[h_t - 1, x_t]) \tag{3.1}$$

1. The X_t input is multiplied by a W_{zx} weight

2. H_{t-1} is the previous output that contains data from previous units multiplied by a W_{zh} weight

3. These are applied together and a sigmoid function is used to compress output from 0 to 1.

Reset Gate

Reset gate calculates the update gate Rt, performing the following steps at the time stage t.

$$r_t = \sigma(W_z[h_t - 1, x_t]) \tag{3.2}$$

1. The X_t input is multiplied by a W_{rx} weight

2. The previous output H_{t-1} which contains previous unit knowledge multiplied by a weight W_{rh}

3. Both are combined and a sigmoid function is applied to squeeze the Output scale 0 to 1.

Current Memory Unit

$$\hat{h} = \tanh(r_t.W_r[h_t - 1, x_t]) \tag{3.3}$$

1. X_t input is multiplied by W_x in weight

2. Apply multiplication element-wise to reset gate R_t and previous output H_t-1, only valid past details can be transferred

3. Both are added together and a tanh function is applied.

Final Memory at Current Time Step
Last but not least, the unit will measure the Ht vector that holds information for the current unit and will transfer it down to the network further. The Upgrade gate Zt plays a key role in this process.

$$h = (1 - z_t) * h_t - 1 + z_t * h_t \tag{3.4}$$

1. Update gate Z_t and Ht using element-wise multiplication

2. Multiplication element-wise 1 Z_t and H'$_t$

3. These both get added.

3.6 Common Activation Functions

An activation function determines whether a neuron should be activated. The nonlinear functions typically convert the output of a given neuron to a value between 0 and 1 or -1 and 1. 't' is used to determine the output of a neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. [18]. Some of the most commonly used functions are defined as follows:

3.6.1 Softmax Function

Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector.

The most common use of the Softmax function in applied machine learning is in its use as an activation function in a neural network model. Specifically, the network is configured to output N values, one for each class in the classification task, and the Softmax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one. Each value in the output of the Softmax function is interpreted as the probability of membership for each class [19].

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{3.5}$$

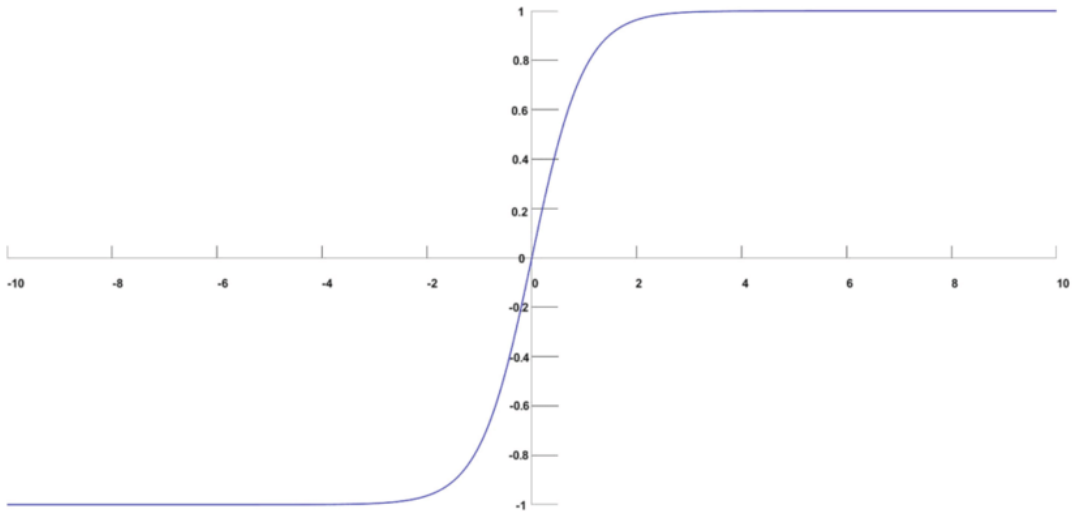

Figure 3.7: Graph Of Softmax Function

3.6.2 ReLU (Rectified Linear Unit) Activation Function

The ReLU is the most used activation function in the world right now. Since it is used in almost all Convolutional Neural Networks or deep learning.

Range: [0 to infinity) the function and its derivative both are monotonic. But the issue is that all the negative values become zero immediately which decreases

the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turn affects the resulting graph by not mapping the negative values appropriately [20].

$$\mathrm{R}(z) = max(0, z) \tag{3.6}$$

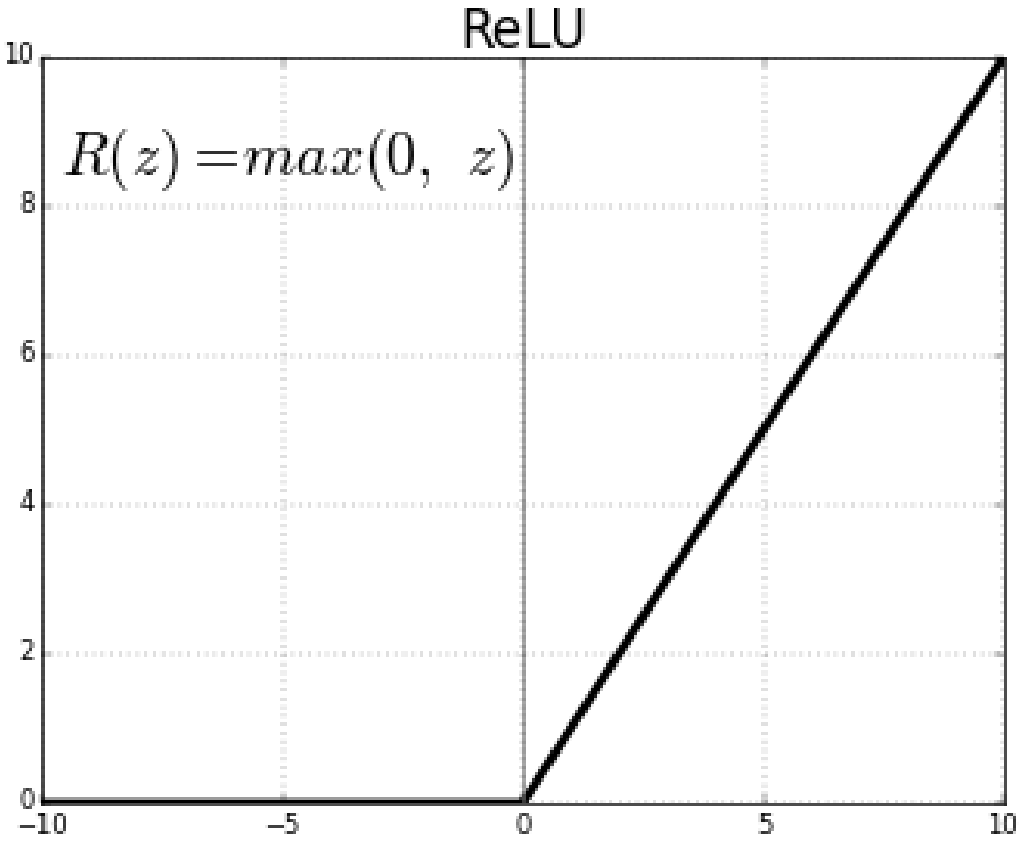

Figure 3.8: Graph Of ReLU Function

3.7 Optimizers

Optimizers are algorithms or methods used to change the attributes of the neural network such as weights and learning rate to reduce the losses. Optimizers are used to solve optimization problems by minimizing the function [21]. Optimization algorithms are responsible for reducing the losses and providing the most accurate

results possible. The weight is initialized using some initialization strategies and is updated with each epoch according to the update equation.

$$Wnew = Wold - lr * (\nabla wL) * Wold \tag{3.7}$$

3.7.1 RMS-Prop (Root Mean Square Propagation)

RMS-Prop is a special version of AdaGrad in which the learning rate is an exponential average of the gradients instead of the cumulative sum of squared gradients. RMS-Prop basically combines momentum with Adaptive Gradient Decent (AdaGrad).
In RMS-Prop learning rate gets adjusted automatically and it chooses a different learning rate for each parameter. Similarly, it has a slow learning rate.

3.7.2 Adam (Adaptive Moment Estimation)

Adam optimizer is one of the most popular and famous gradient descent optimization algorithms. It is a method that computes adaptive learning rates for each parameter. It stores both the decaying average of the past gradients, similar to momentum, and also the decaying average of the past squared gradients, similar to RMS-Prop and Adadelta. Thus, it combines the advantages of both methods [21].

3.8 Learning Rate

The learning rate is a configurable hyper parameter used in the training of neural networks that has a small positive value, often in the range between 0.0 and 1.0. The learning rate controls how quickly the model is adapted to the problem. A large learning rate allows the model to learn faster, at the cost of arriving on a sub-optimal final set of weights. A smaller learning rate may allow the model to learn a more optimal or even globally optimal set of weights but may take significantly longer to train. Normally we use 0.01 as learning rate.

3.9 Segmentation

Segmentation is the process of segmenting the whole document image into recognizable units. In the segmentation stage, an image consisting of a sequence of characters is decomposed into sub-images of individual characters. The main goal is to divide an image into parts that have a strong correlation with objects or areas of the real world contained in the image. Segmentation is very important for the recognition system. Segmentation is an important stage because the extent one can reach in the separation of words, lines, or characters directly affects the recognition rate of the script. Image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics. In character recognition techniques, segmentation is the most important process. Segmentation is done to make the separation between the individual characters of an image. The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries in images. Script segmentation is done by executing the following operations: Line segmentation, Word segmentation, and Character segmentation [22].
Individual images respective to an individual digits or words are separated from the bi-level image containing multiple digits or alphabets. For this first of the regions of contours are detected in the image then a bounding rectangle is drawn respective to each contour which is cropped out of the image. After performing all the above-given steps, the image is then resized to 32x32 pixel which is converted to matrix rows and columns equal to 28 and is fed into the network [23].

3.10 Canny Edge Detection

Canny Edge Detection algorithm is used for edge detection. It involves the following steps:

1. Noise Reduction: The noise in the image is removed with a 5x5 Gaussian filter. Edge detection is susceptible to noise in the image so noise reduction is important.

2. Finding Intensity Gradient of the Image: After noise reduction, a smooth

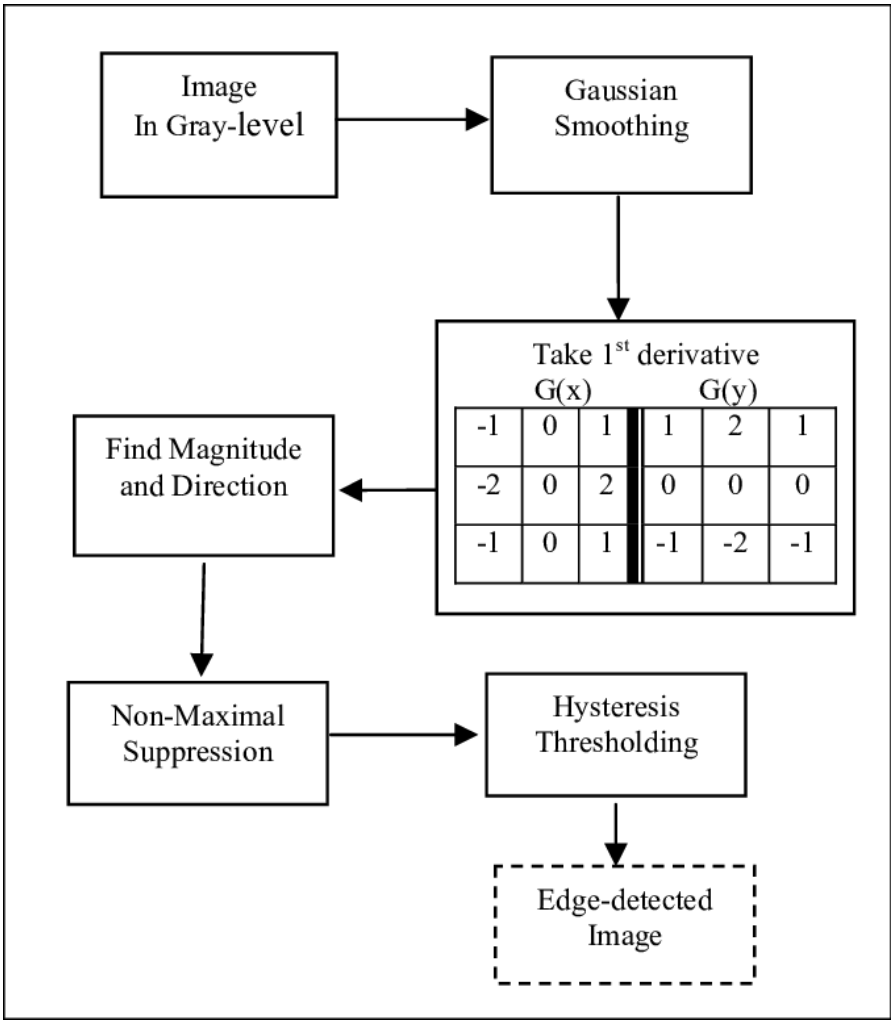

Figure 3.9: Canny Edge Detection Process

image is obtained which is filtered using a Sobel operator in a box in X and Y direction to give derivative of G_x and G_y. From this, we obtain the gradient of intensity matrix and gradient angles as:

$$Edge\_Gradient(G) = \sqrt{{G_x}^2 + {G_y}^2} \tag{3.8}$$

$$Angle(\theta) = \tan^{-1}\frac{G_x}{G_y} \tag{3.9}$$

Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal, and two diagonal directions. The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically, it is used to find the approximate absolute gradient magnitude at

each point in an input grayscale image.

3. Non-max Suppression:

   The output obtained after calculating gradient intensity of the image, the final image must have thin images. After this, we perform Non-max suppression to thin out the images. Here, the algorithm goes through all the points on the image and finds the pixel with the maximum value of gradient in edge detection.

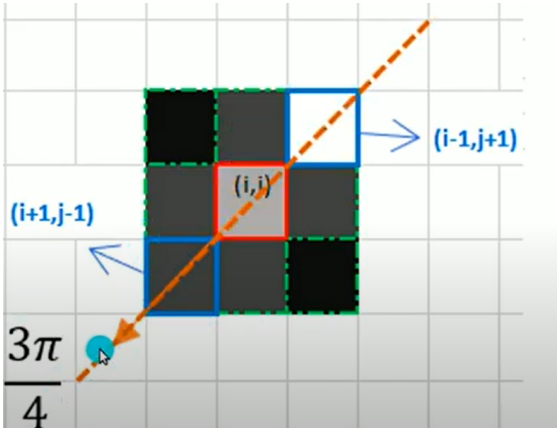

Figure 3.10: Non-max Suppression

4. Double Threshold and Hysteresis Threshold

   For this we need two thresh hold values i.e. minVal and maxVal. Any edges with an intensity gradient more than maxVal are sure to be edge and those minVal are considered to be non-edge and so discarded. They are classified as edges or non-edges based on their connectivity. If they are connected to" sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded.

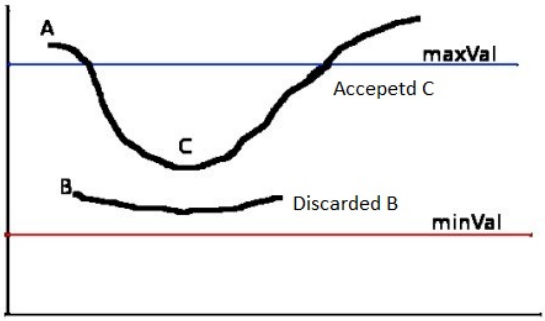

Figure 3.11: Hysteresis Thresh holding

The edge A is above the maxVal, so considered as" sure-edge". Although edge C is below maxVal, it is connected to edge A, so that is also considered a valid edge and we get that full curve. But edge B, although it is above minVal and is in the same region as that of edge C, it is not connected to any "sure-edge", so that is discarded. So it is very important that we have to select minVal and maxVal accordingly to get the correct result. This stage also removes small pixels noises on the assumption that edges are long lines. So what we finally get is strong edges in the image.
Finally, an image with an only edge is obtained. The closing morphological transformation is applied to close gaps between edges or small black points on the object.

The contours are identified using OpenCV findContours function. Contours are the curve joining all the continuous points (along the boundary), having the same color or intensity.

The coordinates of the four sides of the page are determined. The coordinates are sorted in the order top-left, bottom-left, bottom-right, and top-right. The border of 5px is drawn around the image which is then subtracted from the image coordinates. The image is perspective transformed from start points to target points. The four coordinates are used to determine the height and width of the image using Euclidean distance. The target point has coordinates as:

[[0, 0], [0, height], [width, height], [width, 0]]

The 0pencv function getPerspectiveTransform is used to determine the transformation matrix which is then passed to the warpPerspective function to obtain a transformed image.

CHAPTER 4

METHODOLOGY

4.1 System Block Diagram

The block diagram of the proposed system is given below. Handwritten text images from different sources are taken as input and passed to the CRNN model from which the handwritten text is converted into machine editable format. Users can easily input handwritten text images into a simple web interface, and gets the editable text as an output.

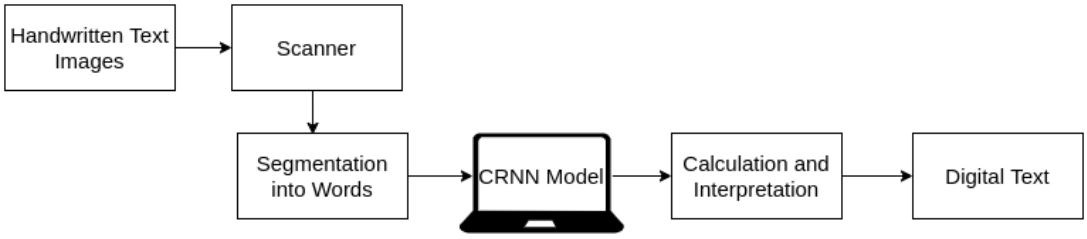

Figure 4.1: Block Diagram of System

4.2 Model Training and Recognition

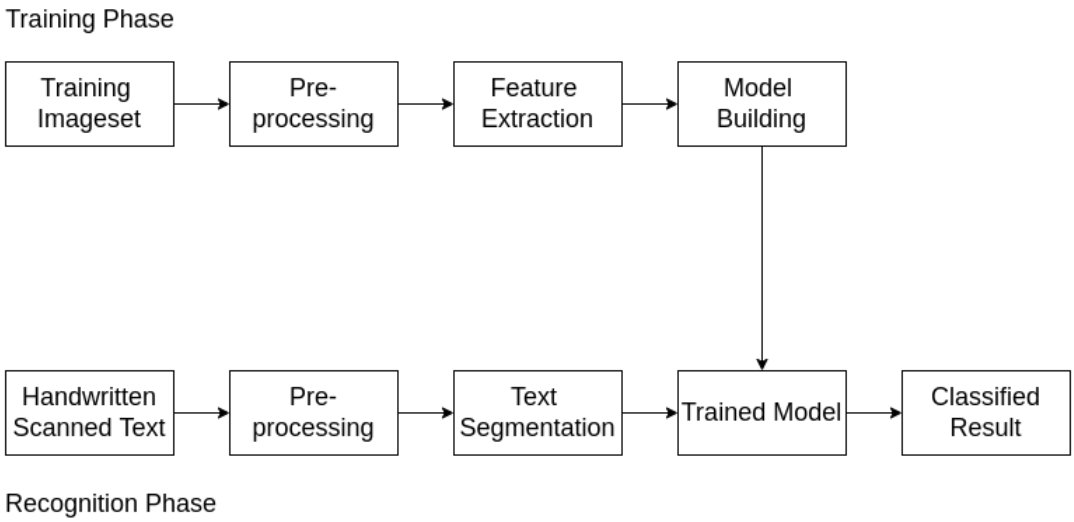

Figure 4.2: Training and Recognition Phase of System

Building an HTR system passes through a series of steps and processes. Data

preparation, Preprocessing, Segmentation, feature extraction, model building are major steps which are divided into two phases i.e Training phase and Recognition phase. The overall processes involved in the methodology is represented by the block diagram.

The training phase consists of data set preparation, image preprocessing,feature extraction and model building which are explained below:

4.2.1 Data Set Preparation

Data collection refers to the process of obtaining pictures of the handwritten text. Since, this is the deep learning-based project so, huge amount of training data incorporating the different varieties of handwriting needed to be collected. The word level devanagari dataset is collected from CVIT [24]. It comprises 96000 word level devanagari dataset written by different people. It is scanned and segmented properly. Since, the length of words are varying, the images of words are also of variable size. Resizing operation is performed in image preprocessing to make all images of the same size. In a text file every image is properly labeled giving image path and ground truth text in image. It is then used as a dataset to train and validate the model.

4.2.2 Image Preprocessing

A series of operations are performed on the scanned input image. The various steps involved in image processing are image resize, transpose, grayscale conversion, and normalization. After all these steps the image is ready for training the model. The image also passes through the same steps for the recognition phase [25].

1. Grayscale conversion

   The scanned image is converted into a grayscale image using the OpenCV library. The image is read using the imread function with image and OpenCV function to read the image passed as a parameter. The grayscale image has only a single channel and pixel intensity varies between 0 (black) to 255 (white).

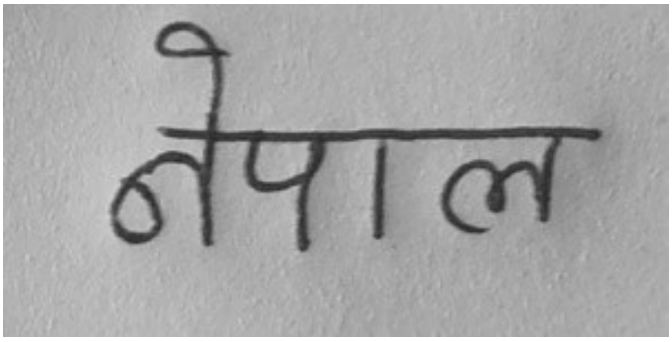

Figure 4.3: Gray-Scale Conversion

2. Image resize

   The scanned words are of different shapes. In this step, the words are resized into 128 * 32 pixels image such that the width of the image becomes 128 pixels and the height becomes 32 pixels. The image is scaled preserving the aspect ratio. The scaling value f is calculated as:

$$f = max(fx, fy) \left\{ Where, fx = \frac{w}{128}, fy = \frac{h}{32} \right. \tag{4.1}$$

   The image is reshaped into the size obtained using the scaling value f. The target image of size 128*32 pixels is created with each pixel value equal to 255. The resized image is inserted into the target image. Finally, we obtain an image of the required shape.

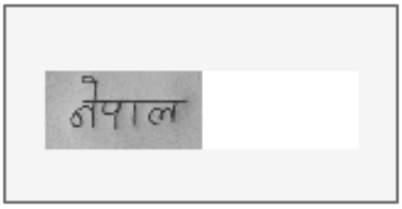

Figure 4.4: An Image of Required Size

3. Transpose

   The target image is transposed using the OpenCV transpose function with the target image obtained after scaling passed as a parameter.

4. Normalization

This step involves normalizing the data to a uniform mean of 0 and a standard deviation of 1, such that faster convergence is achieved. The mean and standard deviation of the image pixel values is calculated using the OpenCV library meanstdDev. The image is normalized as:

$$img = \frac{img - m}{s} \begin{cases} Where, m = Mean\ Deviation\ of\ image, \\ s = Standard\ Deviation\ of\ image \end{cases} \tag{4.2}$$

4.2.3 Feature Extraction

The input image is fed into the CNN layers. These layers extract relevant features from the image. Each layer consists of three operations. The layers in the convolution layer are defined. Activation function used is Relu. CNNs consists of the following layers:

- Convolution layer:

  The kernel of size defined above is used for five layers of CNN, which is passed over the image and a dot product of the original pixel values with weights defined in the kernel is calculated. This matrix is then passed through an activation function "ReLu" that converts every negative value in the matrix to zero.

- Max Pooling layer: A "pooling matrix" of size defined above i.e 2X2 for first and second layer and 1*2 for all other layers , is passed over the matrix to reduce the size of the matrix so as to highlight only the important features of the image. In max pooling, the maximum value present inside the pooling matrix is put inside the final matrix. The 256 feature maps of size 32x1 are extracted from the CNN layer i.e (32, 1, 256). Or The output feature map (or sequence) has a size of 32×256.

4.2.4 Model Building

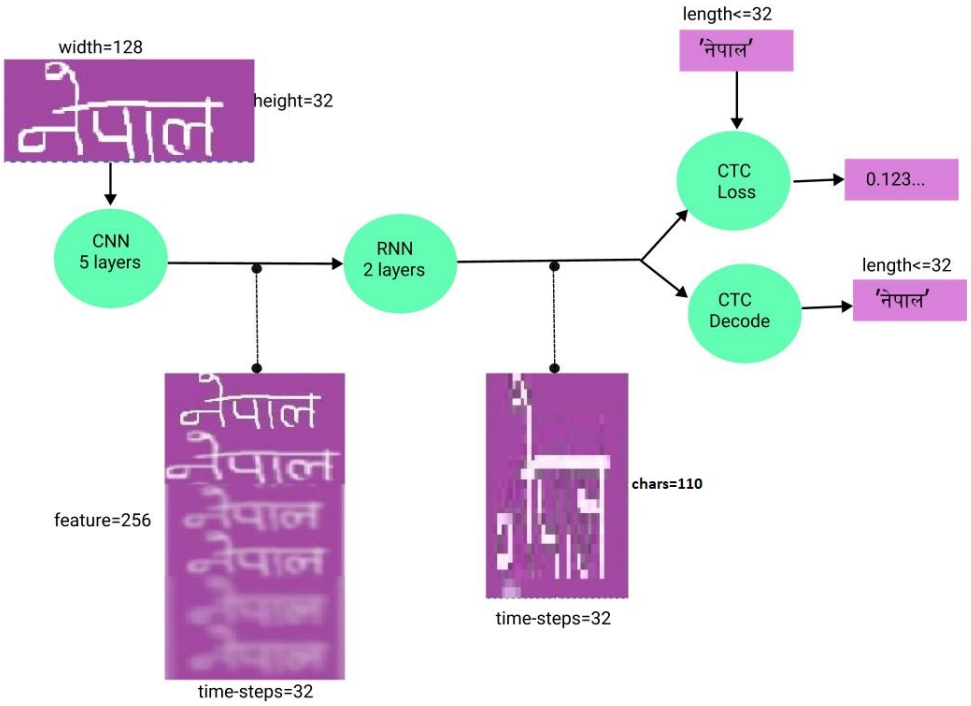

Figure 4.5: Overview of the NN operations

The feature extracted image is passed to the RNN layer. We use CTC to compute the final classification score of the Handwritten Text Recognition System.
Taking a look at how the NN works, we can view it in two ways. On a more formal level, the NN can be seen as a function that maps an image M of size W×H to a character sequence with a length between 0 and L. In other words, the text is recognized on character-level.

$$NN : M \rightarrow (C1, C2.....Cn) \left\{ (W \times H) 0 \leq n \leq L \right. \tag{4.3}$$

- Operation

1. Data: Input is a gray-scale value image of size 128×32. Usually, the images from the dataset do not have exactly this size, therefore we resize it (without distortion) to a width of 128 and a height of 32. Then, we copy the image into a (white) target image of size 128×32. This process is shown in Fig. Finally, we normalize the gray-values of the image which simplifies the task for the NN. Data augmentation can easily be integrated by copying the image

to random positions instead of aligning it to the left or by randomly resizing the image.

Figure 4.6: Resizing Input Image

An image from the dataset with an arbitrary size is taken as input. After that it is scaled to fit the target image of size 128×32, the empty part of the target image is filled with white color.

2. CNN: The input image is fed into the CNN layers. These layers are trained to extract relevant features from the image. Each layer consists of three operations. First, the convolution operation, which applies a filter kernel of size 5×5 in the first two layers and 3×3 in the last three layers to the input. Then, the non-linear RELU function is applied. Finally, a pooling layer summarizes image regions and outputs a downsized version of the input. While the image height is downsized by 2 in each layer, feature maps (channels) are added, so that the output feature map (or sequence) has a size of 32×256.

3. RNN: The feature sequence contains 256 features per time step, the RNN propagates relevant information through this sequence. The popular Long Short-Term Memory (LSTM) implementation of RNNs is used, as it is able to propagate information through longer distances and provides more robust training characteristics than vanilla RNN. The RNN output sequence is mapped to a matrix of size 32×111. The dataset consists of 110 different characters, further one additional character is needed for the CTC operation (CTC blank label), and therefore there are 111 entries for each of the 32 time-steps.

4. CTC (Connectionist Temporal Classification): In training, the NN is given the output matrix and the ground truth text and it computes a loss value. In inference, only the output matrix is given and it decodes it into a final

text. Both the ground truth text and the recognized text can be at most 32 characters long. The ground truth text is encoded as a sparse tensor. The length of the input sequences must be passed to both CTC operations. Mathematically, the CTC objective for each pair (X,Y) is:

$$p(Y \mid X) = \sum_{A \in \mathcal{A}_{X,Y}} \prod_{t=1}^{T} p_t(a_t \mid X)$$

The CTC conditional **probability** | **marginalizes** over the set of valid alignments | computing the **probability** for a single alignment step-by-step.

To estimate the per time-step probabilities P(Y X), CTC models typically use a Recurrent Neural Network(RNN). A RNN works fine for this application because it accounts for the context in the input. However, any other learning algorithm which produces a distribution over output classes given a fixed-slice of input can be used.

- Architecture of the model

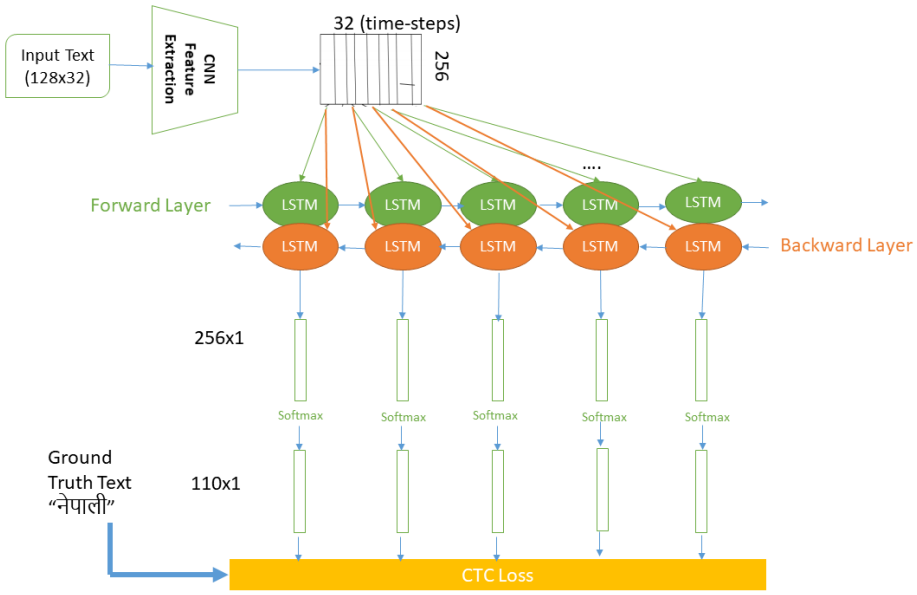

Figure 4.7: Architecture of the model

We use a Neural Network for our task. It consists of convolutional NN (CNN) layers, recurrent NN (RNN) layers and a final Connectionist Temporal Classification (CTC) layer. Fig above shows an overview of our system. The image size of (128*32) is given as input text. The input image is fed into the CNN layers. These layers are trained to extract relevant features from the image. Each layer consists of three operation. First, the convolution operation, which applies a filter kernel of size 5×5 in the first two layers and 3×3 in the last three layers to the input. Then pooling layer image regions and downsize the image. While the image height is downsized by 2 in each layer, feature maps (channels) are added, so that the output feature map (or sequence) has a size of 32×256.

In RNN, the feature sequence contains 256 features per time-step, the RNN propagates relevant information through this sequence. The popular Long Short-Term Memory (LSTM) implementation of RNNs is used, as it is able to propagate information through longer distances and provides more robust training-characteristics than vanilla RNN.

While training the NN, the Connectionist temporal classification (CTC) is given

to the RNN output matrix and the ground truth text and it computes the loss value. While inferring, the CTC is only given the matrix and it decodes it into the final text. Both the ground truth text and the recognized text can be at most 32 characters long.

- Implementation using TF

  Tensorflow is an open-source platform for machine learning. It is a deep learning framework, we can use TensorFlow to build our systems for handwritten text recognition, also it can be used for for object detection, and number plate recognition. This also solves the accuracy issue.

  The network is made up of 5 CNN and 2 RNN layers and workflow can be divided into 3 steps:

  Step 1: Create 5 Convolutional Neural Network (CNN) layers

  The input image is fed into the CNN layers. These layers are extract relevant features from the image. Each layer consists of three operation. First, the convolution operation, which applies a filter kernel of size 5×5 in the first two layers and 3×3 in the last three layers to the input. Then, the non-linear RELU function is applied. Finally, a pooling layer summarizes image regions and outputs a downsized version of the input. While the image height is downsized by 2 in each layer, feature maps (channels) are added, so that the output feature map (or sequence) has a size of 32×256.

  Step 2: Create a Recurrent Neural Network (RNN) layer and return its output

  The input sequence is traversed from front to back and the other way round. As a result, we get two output sequences fw and bw of size 32×256, which we later concatenate along the feature-axis to form a sequence of size 32×512. Finally, it is mapped to the output sequence (or matrix) of size 32×110 which is fed into the CTC layer.

  Step 3:

  For training, we used the RMSProp optimizer with the standard learning rate of each model to incrementally update the parameters using CTC loss gradient. While inferring, the CTC is only given the matrix and

it decodes it into the final text. Both the ground truth text and the recognized text can be at most 32 characters long.

- Implementation modules

  The implementation consists of 4 modules:

  1. Sample preprocessor

     Prepares the images from the dataset for the Neural Network. A series of operations are performed on the scanned input image. The various steps involved in image processing are image resize, transpose, grayscale conversion, and normalization. After all these steps the image is ready for training the model. The image also passes through the same steps for the recognition phase

  2. Data loader

     Reads samples, puts them into batches, and provides an iterator-interface to go through the data. It is for loader of a dataset at a given location, preprocess images and text according to parameters. We use our own path and trick to read the images with the text file. We remove the new lines in the text file and also ignore the commented line. Ground Truth text is columns starting at the first position of the text file. Similarly, we check if it contains an image or not and then finally we put sample in the list. After this, we split into training and validation sets: 90% - 10%. Then we put the words in a list. We choose the sample list for training. CTC loss can't compute loss if it cannot find a mapping between text label and input so we truncate it. If a too-long label is provided, CTC loss returns an infinite gradient. Then we define the training set and validation sets within a function. Iterative it checks if it has got the next sample and after this, we define how it gets the next image.

  3. Model

     Creates the model as described above, loads and saves models, manages the TF sessions and provides an interface for training and inference. In this section, we define the model used in our project. We create CNN layers and return the output of these layers with ReLu as an activation

function. After this, we create a pooling layer to CNN. The output of CNN is fed to RNN. We create RNN layers and return output of these layers. We create LSTM and BLSTM layers. After all these the output is fed to the CTC_loss function and returned it. The loss is calculated for each batch and compute the label of probability. Then we choose among the best path decoding or beam search decoding. Finally, the model is saved with a certain name and the model is stored in certain directory.

4. Main

   All the modules are created and imported in the main file and accessed from it. In this section, we define the batch size, conditions for epochs, and stopping criteria. If best validation accuracy so far, we save model parameters. We stop training if no more improvement in the last 5 epochs. We finally print the validation result. At the end, we infer text on the test image.

- Summary of model

Table 4.1: Summary of Model

Layer(Type)	**Configuration**
Input	128*32;gray-Scale image
Convolution	maps:32;k=5*5;p:same
Maxpooling	Window:2*2;s:2
Convolution	maps:64;k=5*5;p:same
Maxpooling	Window:2*2;s:2
Convolution	maps:128;k=3*3;p:same
Maxpooling	Window:1*2;s:2
Convolution	maps:128;k=3*3;p:same
Maxpooling	Window:1*2;s:2
Convolution	maps:256;k=3*3;p:same
Maxpooling	Window:1*2;s:2
LSTM1	hidden units:256
LSTM2	hidden units:256
Concantenate	LSTM1,LSTM2
k,s,p	kernel size,stride size and padding size

After the model building, a trained model is obtained. The workflow now reaches to the recognition phase in which the trained model is used to recognize the handwritten text image. The recognition phases consist of scanning of handwritten text, preprocessing, text segmentation and using trained model.

4.2.5 Word Segmentation

Word segmentation is the process of splitting a string of written language into its component words. At this level of segmentation, we are provided with an image containing a single line that consists of a sequence of words. The objective of Word Level Segmentation is to segment the image into words. The process includes the following steps.

1. Read image

   The handwritten text image is scanned and is read using OpenCV imread function. We have also used a pillow library for processing so the image color order is changed into RGB from default BGR bypassing cv2.COLOR_BGR2RGB as a parameter while reading the image.

2. Resizing image

   The input image is resized. The maximum height of the image is taken as 800 pixels. If the height of the image is greater than the maximum height, the image is resized by preserving the aspect ratio using OpenCV resize function.

$$ratio = \frac{800}{heightofimage} \tag{4.4}$$

$$newsize = (ratio * width\ of\ image\ , 800) \tag{4.5}$$

   The resized RGB image is converted into a grayscale image. The grayscale image has only a single channel and pixel intensity varies between 0 (black) to 255 (white). The filtering is performed in an image using OpenCV bilateralFilter() function. The diameter of each pixel neighborhood is taken as 9 and the sigma color and sigma space value as 75. A bilateral filter is used for smoothening images and reducing noise while preserving edges.

   Adaptive thresholding with Gaussian is applied to the filtered image. Adaptive

thresholding is the method where the threshold value is calculated for smaller regions. In this, a weighted sum of the blockSize×blockSize neighborhood of a point is subtracted from constant to determine the threshold value.

The median blur is performed in which the central element of the image is replaced by the median of all the pixels in the kernel area. It is used to remove the thin details. cv2.copyMakeBorder() method is used to create a border around the image like a photo frame. The border width in the number of pixels in all directions is kept at 5. Canny Edge Detection algorithm is used for edge detection. It involves: Noise Reduction, finding intensity of gradient image and non-man supression. Finally a image with only edge is obtained.

3. Words detection

   The cropped image is used to detect words from the images. The OpenCV GaussianBlur function is used with a kernel size of (5, 5). The blurred image is used for edge detection using the Sobel operator. These are the kernels used for Sobel Edge Detection.

X – Direction Kernel

-1	0	1
-2	0	2
-1	0	1

Y – Direction Kernel

-1	-2	-1
0	0	0
1	2	1

Figure 4.8: Sobel Operation

These kernels are convolved with the original image, to obtain the 'Sobel edge image'. Let G_x and G_y represent the intensity gradient in the x and y directions respectively. If A and B denote the X and Y kernels defined above:

$$Gradient\_along\_X\_direction(G_x) = A * I \tag{4.6}$$

$$Gradient\_along\_Y\_direction(G_y) = B * I \tag{4.7}$$

$$\Big\{ Where, * \textit{ denotes the convolution operator, and } I \textit{ represents the input image}$$

The final approximation of the gradient magnitude, can be computed as:

$$Edge\_Gradient(G) = \sqrt{{G_x}^2 + {G_y}^2} \tag{4.8}$$

A simple thresholding operation is performed on the edge detected image. The threshold value which is used to classify the pixel values is set as 50. The pixel values exceeding the threshold are assigned the maximum value of 255 (white). The closing morphological transformation is applied to close gaps between edges or small black points on the object. The contour is determined using OpenCV findContours methods. For each contour, the coordinates, height, and width of the bounding rectangle are determined using findContours. These coordinates will be used to determine the words from the cropped image. Finally, words are sorted in order from left to right, top to bottom, and stored in a directory.

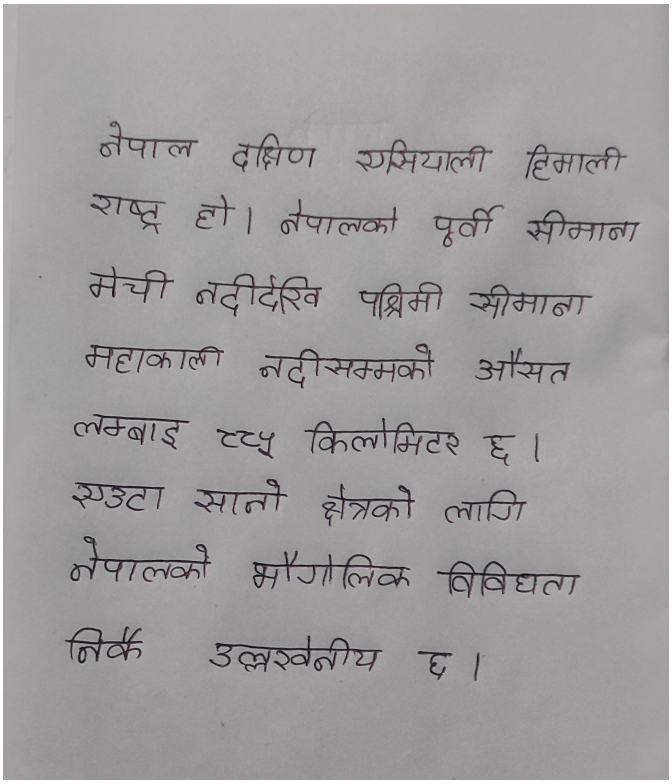

Figure 4.9: Original Image

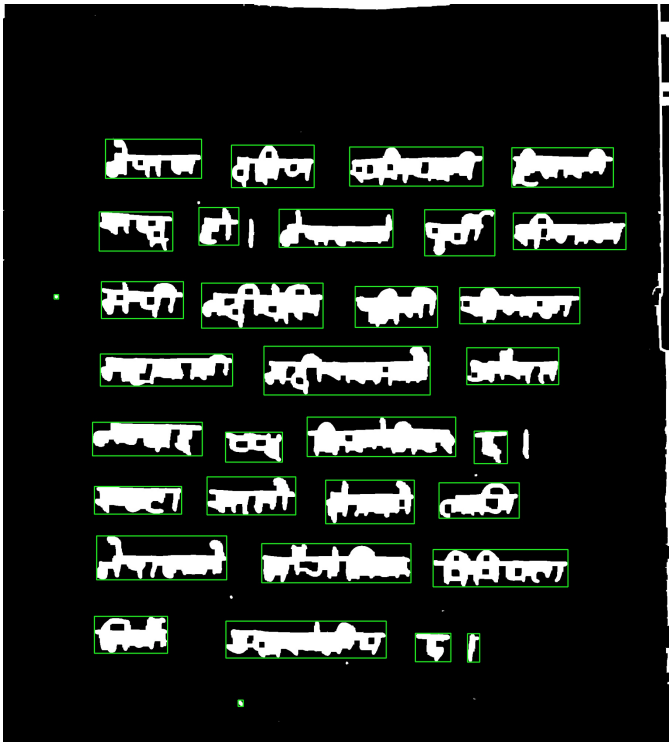

Figure 4.10: Segmented Image

4.2.6 Word recognition

Recognition is final stage of the system. In this stage, we load the saved model. Now this loaded model can be used to recognize the input image after passing image through the image processing phase. The model provide the text present in the word as an output in web interface implemented using python library flask. User can copy the text displayed using copy text button.

4.3 Algorithm and Flowchart

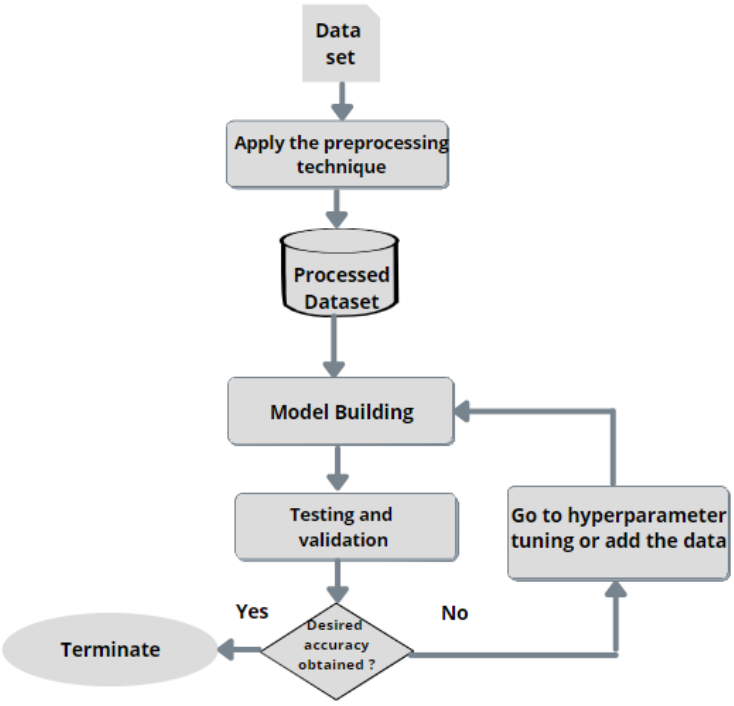

Figure 4.11: Flowchart of Handwritten Text Recognition

The above figure describes the overall flow of our project. Initially, the data from different sources are collected. After applying different preprocessing techniques to the data sets, all the noise content and undesirable outlier are removed to get a well-managed or processed data set which are applied to train the RNN model. Testing and validation are done to check if the output is of the desired accuracy. If not, the we continue the training the model and testing and validating it until the desired accuracy is obtained. We train the RNN model varying the hyperparameters like learning rate, batch size, epochs. The data can also be added to improve the accuracy. When the desired accuracy is obtained we stop the process. In this way, our system works to convert images containing text into an editable text format.

4.4 UML Diagrams

4.4.1 Use Case Diagram

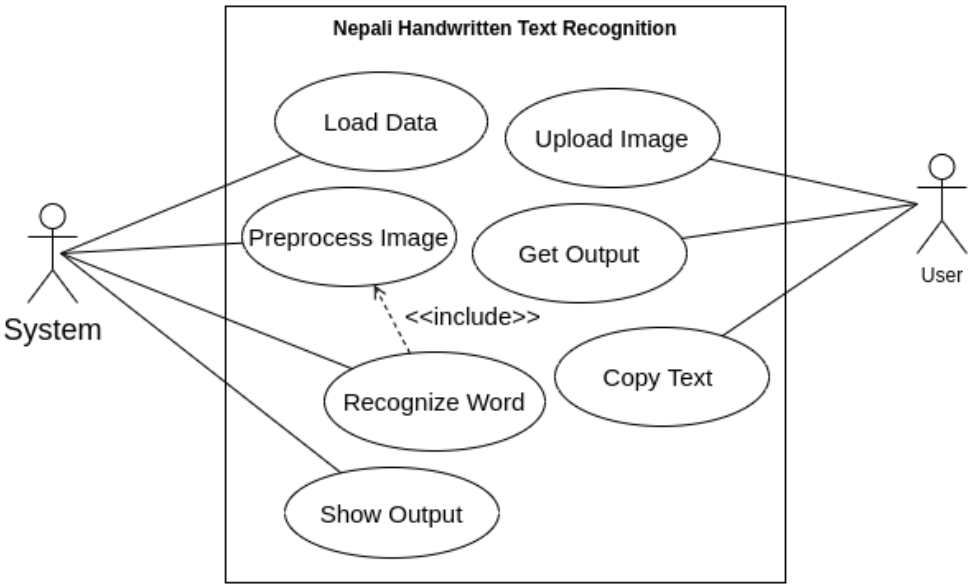

Figure 4.12: Use Case Diagram of System

4.4.2 Sequence Diagram

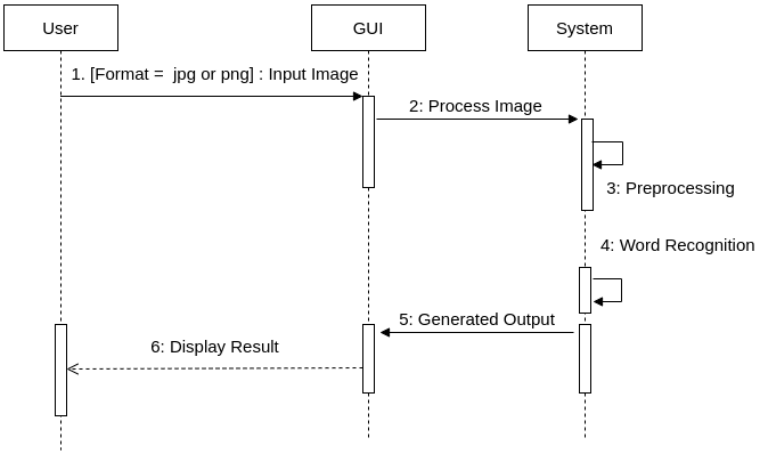

Figure 4.13: Sequence Diagram of System

4.5 Software Development Life Cycle

We have followed the iterative model of SDLC for our project. The implementation of iterative software development life cycle, includes the initial development based on the initial requirements and more features are added to the base software product with the ongoing iterations until the final system is created.The iterative model breaks down the software development process of a very big application into smaller pieces, so that complexity reduces. The benefit of this model is that it is employed during the earlier stages of SDLC.

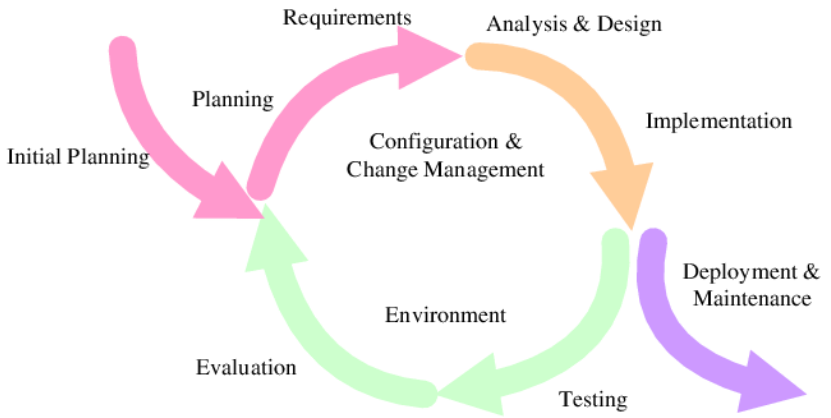

Figure 4.14: Iterative Software Development Model

We begin by creating the initial planning. The requirements of the project are gathered. Algorithms are developed after requirements have been gathered in machine learning, but there will be the fixed plan defined at the beginning of the model. The analysis and design is performed using various tools. The design involves drawing use case diagram which describe the high-level functions and scope of a system and the interaction between the system and its actors. Sequence diagram shows object interactions arranged in time sequence. The initial implementation of the project involve coding in python programming language. For the development of user interface languages like javascript and various other frontend tools like HTML and CSS are used. The developed system is tested and evaluated provided different set of input data. The cycle repeats iteratively i.e system passes through planning, design, implementation, and testing stages again and again until the requirement is fulfilled or the desired accuracy is obtained.

4.6 System Specification and Configuration

The training of the developed system was performed on a computer system without running any other external applications during the entire period of training with the specifications as mentioned below in the specification table:

Table 4.2: System Specifications Table

S.N.	Items	System Specifications
1	Computer Model	Dell Inspirion 13 5000 series
2	Computer Processor	Intel ® Core™ I5-7200U CPU @2.50GHz 2.70
3	Installed RAM	8.00 GB(7.87GB available
4	System Type	64bit Windows 10 Home ,x64-based processor

For the training of the system being developed, the parameters required were configured as mentioned below in the training configuration table:

Table 4.3: Configuration Table

S.N.	Particulars	Applied Configuration
1	Number of Epoch	50
2	Batch Size	25
3	Optimizer	RMSProp
4	Learning Rate	0.0001
5	Loss Function	CTC Loss
6	Training of Image Sample	25000
7	Validation of Image Sample	2500

4.7 Model Development Tools

Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data. During the development of our system, we have used the following tools and techniques.

4.7.1 Tensorflow

Tensorflow makes it easy for beginners and experts to create machine learning models for desktop, mobile, web, and cloud. It is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that

simplify the process built on top of Tensorflow. We have used tensorflow version 1.8 in our project.

4.7.2 Python(Flask)

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. Flask is used for developing web applications using python. Advantages of using Flask framework are: There is a built-in development server and a fast debugger provided.

4.7.3 NumPy

NumPy is a python library used for working with arrays. It also has functions for working in the domain of linear algebra, Fourier transform, and matrices. We have used NumPy in our project implementation.

4.7.4 Codecs

The codecs module defines a set of base classes that define the interface and can also be used to easily write our own codecs for use in Python. Each codec has to define four interfaces to make it usable as a codec in Python: stateless encoder, stateless decoder, stream reader, and stream writer.

4.7.5 Editdistance

The edit distance between two strings refers to the minimum number of character insertions, deletions, and substitutions required to change one string to the other. In our project, we have used to calculate the minimum distance between the other words.

4.7.6 Argparse

Argparse is the "recommended command-line parsing module in the Python standard library. The argparse module in Python helps create a program in a command-line environment in a way that appears not only easy to code but also improves interaction.

4.7.7 OpenCV

It is a library of programming functions mainly aimed at real-time computer vision. All the OpenCV array structures are converted to and from NumPy arrays. This also makes it easier to integrate with other libraries that use NumPy such as SciPy and Matplotlib. In our project, we have used OpenCV2 for image pre- processing.

CHAPTER 5

RESULT AND ANALYSIS

The main target of this project was to create a system that could recognize text from an image. Among all the processes the most important task was to obtain a stable model with the highest level of accuracy by training the Convolutional Neural Network along with Recurrent Neural Network and LSTM with our own training data set. So, to achieve the objective of this project, we implemented RMSProp as an optimizer, Connectionist Temporal Classification (CTC) function as the loss function in a Convolutional Neural Network along with RNN and LSTM.

The final analysis of the obtained results are shown below in the plot between training loss and validation loss and also the plot between train accuracy and validation accuracy and character error rate, RMSProp for which the network was trained. We can see that the train and validation accurecy gradually increases as the number of epoch increases. At 50 epoch the train accuracy is obtained as 97% and test accuracy above 87% is obtained. The the train and validation loss decreases as the number of epoch increases. Similarly, the character error rate decreases with the number of epoch.

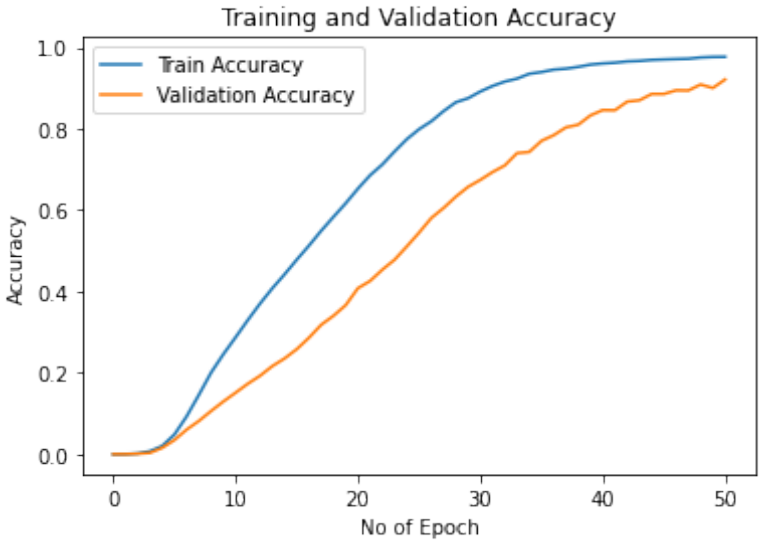

Figure 5.1: Plot for Training and Validation Accuracy

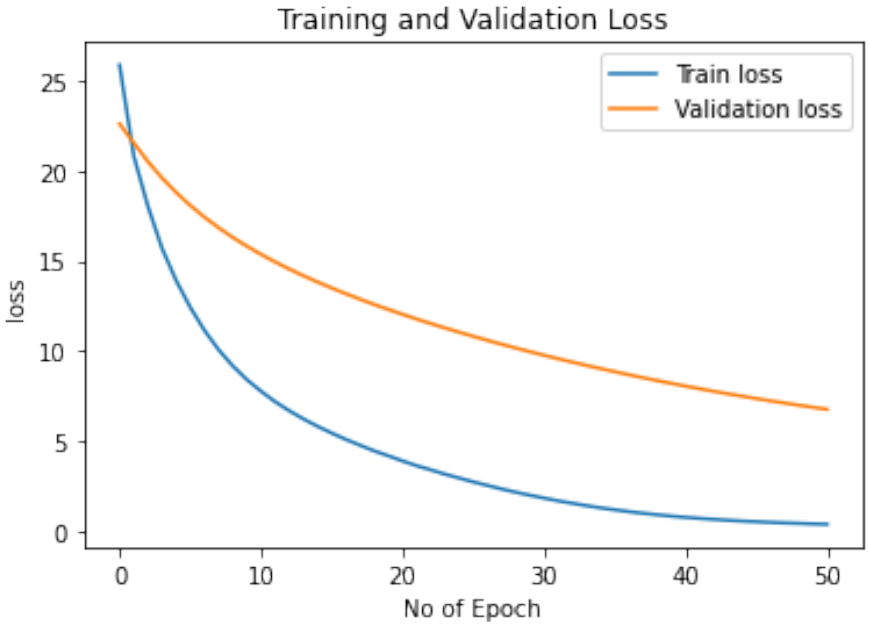

Figure 5.2: Plot between Training and Validation Loss

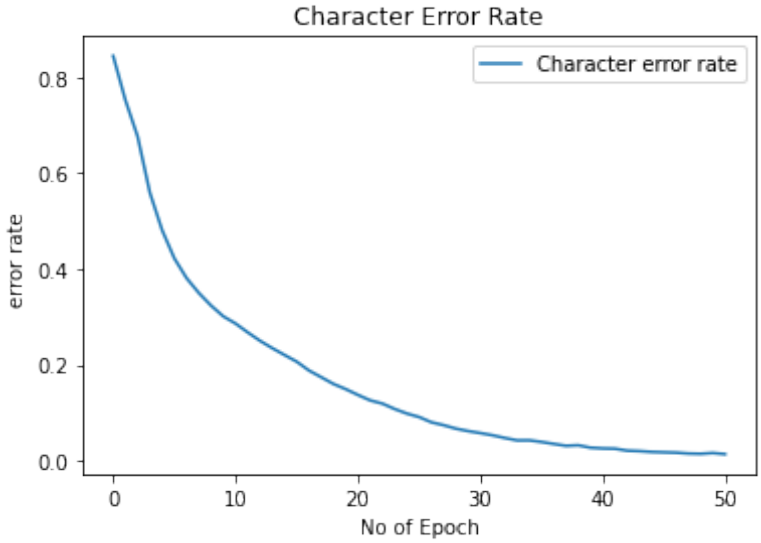

Figure 5.3: Plot for Character Error Rate

5.1 Output

After obtaining the stable hypothesis, we used the model to recognize the handwritten sample from our friends and colleagues. Some images were recognized with 98% accuracy whereas some were mispredicted. Some of the snapshots of our web interfaces are shown below:

Figure 5.4: Home Page

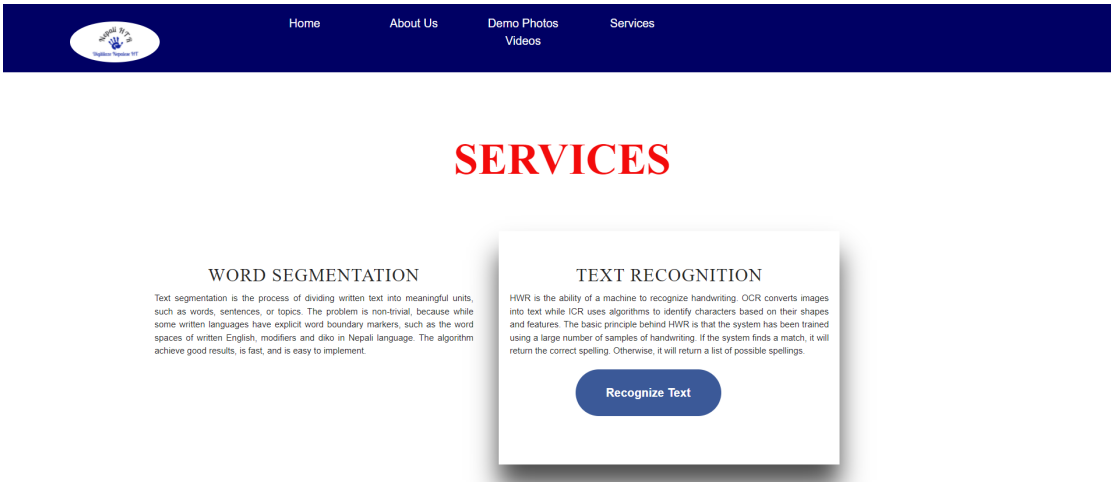

Figure 5.5: Service Page

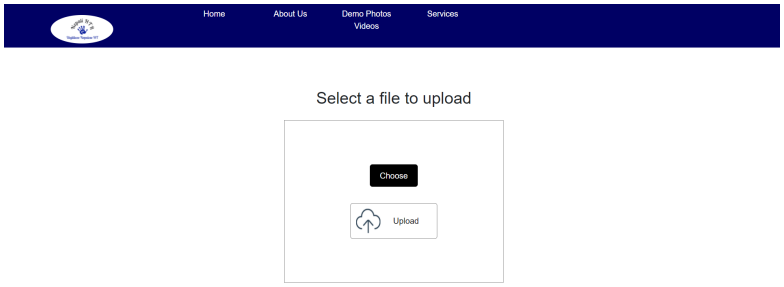

Figure 5.6: Upload Image Section

The overall performance of the system for total images used to be identified was always above 87% accurate. Some of the samples of completely recognized images are shown below.

Figure 5.7: Recognized Text After Uploading Image

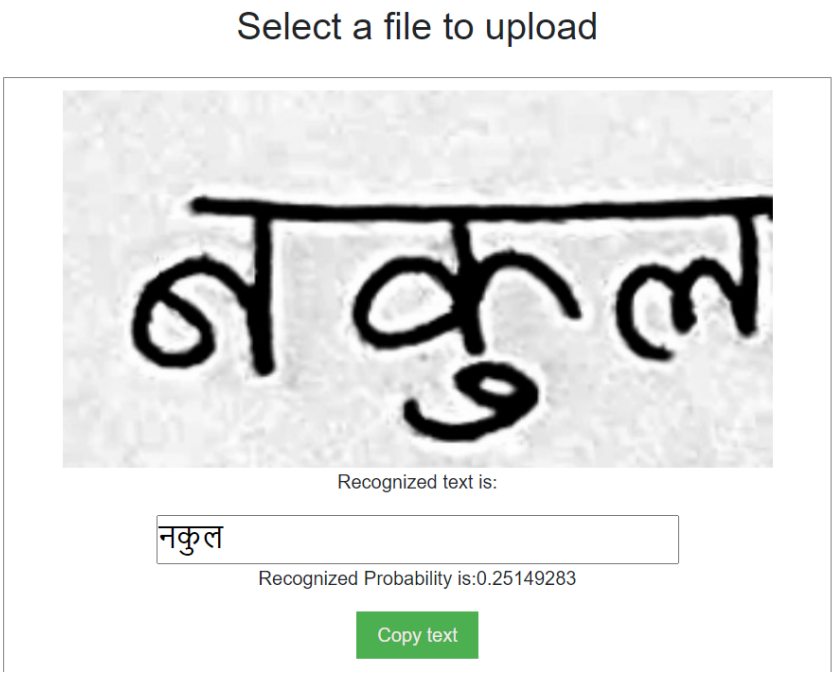

Figure 5.8: Recognized Text After Uploading Image

The snapshots of the images being recognized show the predicting capacity of our model. All the images included are handwritten in plain white paper and passed to the the model. The model is able to recognize these handwritten words. The probability of recognition decreases for the second word as it consists of modifiers and more number of characters than the first word.

5.2 Conclusion

This project work is a successful output of the course called 'Major Project' considered as the partial fulfillment of B.E. Computer Engineering at IOE. The main objective of the project was to recognize the Handwritten Nepali text in an image and convert it to an editable format. The system was implemented in Python programming language with Flask using static webpage with HTML/CSS/JavaScript and its performance was tested on real images.

This system is one kind of AI program. The knowledge of image processing and CRNN that implements the OCR technology has been used to extract the text from an image and finally, high level of accuracy i.e. 86% of validation accuracy with RMSProp optimizer. The system performs very well with the other real world's images as well, in the recognition phase.

This project work has been a great achievement for us even though some limitations still exist. Finally, the overall result was satisfactory, and we successfully carried out this project as a part of our course work and also developed the hands-on experience of working on a project.

CHAPTER 6

LIMITATIONS AND FURTHER WORK

Handwriting Recognition has been quite an interesting field today. Many technical teams around the world are still working together to get satisfactory result. Several types of research have been done and some are still ongoing in this field due to advancements in technology and efficient new models, it has made it possible to make further enhancements in this field to get more accurate results. Like other projects, our project has also got limitations and the enhancements that can be made in the future. Some of the limitations of our project are:

1. It cannot recognize the text images consisting of many modifiers.
2. A paragraph must be segmented into words and an image consisting of the word must be given as input.
3. When a random image is given it also takes as input and prediction is incorrect.

The main objective of this project was to obtain a model with the highest accuracy that can predict real-world images. But yet the model we have obtained is not perfectly trained. So, the basic tasks yet to be done are:

1. Better Image preprocessing such as: reducing background noise to handle real-time images more accurately.
2. Use Data Augmentation before training to improve accuracy. Deep Layers of BLSTM or MDLSTM can be used but again we have to compromise with training complexity and overfitting.
3. Use Better CTC Decoders such as Word Level Decoder and better Language model for decoder output correction.
4. Increase input size (if the input of NN is large enough, complete text-lines can be used)
5. Replace LSTM by 2D-LSTM

REFERENCES

[1] Divakar Yadav, Sonia Sánchez-Cuadrado, and Jorge Morato. Optical character recognition for hindi language using a neural-network approach. *JIPS*, 9(1):117–140, 2013.

[2] Anupama Thakur and Amrit Kaur. Devanagari handwritten character recognition using neural network. *Int. J. Sci. Technol. Res*, 8(10), 2019.

[3] Oka Sudana, I Wayan Gunaya, and I Ketut Gede Darma Putra. Handwriting identification using deep convolutional neural network method. *Telkomnika*, 18(4):1934–1941, 2020.

[4] Shalini Puri and Satya Prakash Singh. An efficient devanagari character classification in printed and handwritten documents using svm. *Procedia Computer Science*, 152:111–121, 2019.

[5] Shalaka Prasad Deore and Albert Pravin. Devanagari handwritten character recognition using fine-tuned deep convolutional neural network on trivial dataset. *Sādhanā*, 45(1):1–13, 2020.

[6] NIRAJAN PANT. *NEPALI OCR USING HYBRID APPROACH OF RECOGNITION*. PhD thesis, Kathmandu University, 2016.

[7] Manish K Sharma and Bidhan Bhattarai. Optical character recognition system for nepali language using convnet. In *Proceedings of the 9th International Conference on Machine Learning and Computing*, pages 184–189, 2017.

[8] Shailesh Acharya, Ashok Kumar Pant, and Prashnna Kumar Gyawali. Deep learning based large scale handwritten devanagari character recognition. In *2015 9th International conference on software, knowledge, information management and applications (SKIMA)*, pages 1–6. IEEE, 2015.

[9] Subarna Shakya, Abinash Basnet, Suman Sharma, and Amar Bdr Gurung. Optical character recognition for nepali, english character and simple sketch

using neural network. In *Recent Advances in Information and Communication Technology 2016*, pages 45–53. Springer, 2016.

[10] Payel Rakshit, Chayan Halder, Subhankar Ghosh, and Kaushik Roy. Line, word, and character segmentation from bangla handwritten text—a precursor toward bangla hocr. In *Advanced Computing and Systems for Security*, pages 109–120. Springer, 2018.

[11] Ms Vaishali G Bhujade and Ms Chhaya M Meshram. A technique for segmentation of handwritten hindi text. *Int. J. Eng. Res. Technol*, 3:1491–1495, 2014.

[12] Brijeshwar Dessai and Amit Patil. A deep learning approach for optical character recognition of handwritten devanagari script. In *2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, volume 1, pages 1160–1165. IEEE, 2019.

[13] Ashok Kumar Pant, Sanjeeb Prasad Panday, and Shashidhar Ram Joshi. Off-line nepali handwritten character recognition using multilayer perceptron and radial basis function neural networks. In *2012 Third Asian Himalayas International Conference on Internet*, pages 1–5. IEEE, 2012.

[14] Chongchong Yu, Yunbing Chen, Yueqiao Li, Meng Kang, Shixuan Xu, and Xueer Liu. Cross-language end-to-end speech recognition research based on transfer learning for the low-resource tujia language. *Symmetry*, 11(2):179, 2019.

[15] Gundram Leifert, Roger Labahn, and Tobias Strauß. Citlab argus for arabic handwriting. *arXiv preprint arXiv:1412.6061*, 2014.

[16] handwritten text recognition documenttation - google search. https://www.google.com/search?q=handwritten+text+recognition+documenttationoq=han 8. (Accessed on 03/03/2022).

[17] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[18] Jianli Feng and Shengnan Lu. Performance analysis of various activation functions in artificial neural networks. In *Journal of physics: conference series*, volume 1237, page 022030. IOP Publishing, 2019.

[19] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[20] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *towards data science*, 6(12):310–316, 2017.

[21] optimizer in neural network - google search. `https://www.google.com/search?q=optimizer+in+neural+network&oq=optimizer+&aqs=chrome.1.69i59j0i512l3j69i57j69i60l3.8290j0j7&sourceid=chrome&ie=UTF-8`. (Accessed on 03/03/2022).

[22] J Pradeep, E Srinivasan, and S Himavathi. Performance analysis of hybrid feature extraction technique for recognizing english handwritten characters. In *2012 World Congress on Information and Communication Technologies*, pages 373–377. IEEE, 2012.

[23] Rohit Verma and Jahid Ali. A-survey of feature extraction and classification techniques in ocr systems. *International Journal of Computer Applications & Information Technology*, 1(3):1–3, 2012.

[24] Kartik Dutta, Praveen Krishnan, Minesh Mathew, and C. V. Jawahar. Offline handwriting recognition on devanagari using a new benchmark dataset. In *DAS*, 2018.

[25] Esam MA Hussein. *Computed Radiation Imaging: Physics and Mathematics of Forward and Inverse Problems*. Elsevier, 2011.