HANDWRITTEN TEXT RECOGNITION

A Report submitted

***in partial fulfillment* for the Degree of**

B. Tech in

Computer Science and Engineering

by

AYUSH NEGI (200970101013)

ABHISHEK RANA (200970101004)

SAURABH BHATT (200970101044)

pursued in

Department of Computer Science and Engineering

THDC Institute of Hydropower Engineering and Technology

To

VEER MADHO SINGH BHANDARI UTTARAKHAND TECHNICAL UNIVERSITY

JUNE, 2024

CERTIFICATE

This is to certify that the project report entitled **Handwritten Text Recognition** submitted by **Ayush Negi, Abhishek Rana** and **Saurabh Bhatt** to the THDC Institute of Hydropower Engineering and Technology, New Tehri, in partial fulfilment for the award of the degree of **B. Tech in Computer Science and Engineering** is a *bona fide* record of project work carried out by them under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institution or University for the award of any degree or diploma.

Mr. Pallaw Aswal
Assistant Professor
Department of Computer Science
and Engineering, THDC IHET

External Examiner

New Tehri
June, 2024

Counter signature of HOD

DECLARATION

We declare that this project report titled **Handwritten Text Recognition** submitted in partial fulfillment of the degree of **B. Tech in Computer Science and Engineering** is a record of original work carried out by us under the supervision of **Mr. Pallaw Aswal**, and has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Ayush Negi
(200970101013)

Abhishek Rana
(200970101004)

Saurabh Bhatt
(200970101044)

THDC-IHET
JUNE 2024

AKNOWLEDGEMENTS

We would like to express our profound gratitude and thanks to our project guide Mr. *Pallaw Aswal*, Assistant Professor, Department of Computer Science and Engineering, THDC IHET for introducing the present topic and intellectual support, inspiring guidance and his invaluable encouragement, suggestions and cooperation helped a lot to completion of project work successfully.

We are also thankful to Mr. *Vivek Kumar*, Head of Department, Computer Science and Engineering, THDC IHET, for providing necessary information and guidance regarding the project.

We would like to thank THDC IHET for providing us this wonderful opportunity and our parents for their kind co-operation and support which helped us to complete our project.

Ayush Negi

Abhishek Rana

Saurabh Bhatt

ABSTRACT

While there's no shortage of technological writing aids available, many still opt for the traditional method of pen and paper when taking notes. However, handwriting has its drawbacks. Storing and accessing physical documents can be challenging, and searching through them efficiently or sharing them with others is not always easy.

As a result, valuable knowledge often goes unviewed or gets lost because handwritten documents aren't transferred to digital formats. Recognizing this issue, we've embarked on a project aimed at addressing it. We believe that the superior manageability of digital text over handwritten text will enable people to access, search, share, and analyze their records more effectively, all while allowing them to continue using their preferred writing method.

Our project focuses on the task of classifying handwritten text and converting it into a digital format. Handwritten text encompasses various styles, so we narrowed our focus to classifying images of individual handwritten words, whether in cursive or block writing. We aim to create a fully functional model that assists users in converting handwritten words into digital format. This model will prompt users to photograph a word for conversion.

TABLE OF CONTENTS

DESCRIPTION..PAGE NUMBER

CERTIFICATE..III

DECLARATION..IV

AKNOWLEDGEMENTS..V

ABSTRACT..VI

TABLE OF CONTENTS..VII

LIST OF FIGURES..IX

ABBREVIASTIONS/ NOTATIONS/ NOMENCLATURE..X

INTRODCUTION..1

1.1 MOTIVATION..1
1.2 PROBLEM STATEMENT..2
1.3 OBJECTIVE..2

LITERATURE SURVEY..3

2.1 CONVOLUTIONAL NEURAL NETWORK..3
2.2 RECURRENT NEURAL NETWORK..5
2.3 CONNECTIONIST TEMPORAL CLASSIFICATION (CTC)..8

REQUIREMENT ANALYSIS..11

3.1 INTRODUCTION..11
3.2 SOFTWARE AND HARDWARE REQUIREMENTS..12
3.2 DATASET..14
3.3.1 IAM Handwriting Dataset..14
3.2 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS..15

SYSTEM DESIGN..16

IMPLEMENTATION..18

5.1 DATA PREPROCESSING..18
5.1.1 Data Augmentation..18
5.1.2 Grayscale..19
5.1.3 Image Resizing..19
5.1.4 Transpose image..20
5.1.5 Normalization..20
5.2 TRAINING THE MODEL..21
5.2.1 Training..21
5.3 SOURCE CODE..27

TABLE OF CONTENTS

SYSTEM TESTING 30
6.1 RESULT 30
6.2 EVALUATION 31
6.2.1 Character Error Rate 31
CONCLUSION 32
FUTURE SCOPE 33
REFERENCES 34
APPENDIX 35

LIST OF FIGURES

FIGURE: DESCRIPTION…………………………………......PAGE NUMBERS

FIGURE 2.1: A SIMPLE CNN ARCHITECTURE, COMPRISED OF JUST FIVE LAYER.........4
FIGURE 2.2: IMPORTANCE OF CONTEXT IN HANDWRITING RECOGNITION. 5
FIGURE 2.3: ILLUSTRATION OF THE VANISHING GRADIENT PROBLEM. 6
FIGURE 2.4: LSTM MEMORY BLOCK WITH ONE CELL. 6
FIGURE 2.5: PRESERVATION OF GRADIENT INFORMATION BY LSTM. 7
FIGURE 2.6: TYPICAL USE CASE FOR DIFFERENT TYPES OF DECODERS.......................9
FIGURE 3.1 SAMPLES OF IAM DATASET...14
FIGURE 4: SYSTEM BLOCK DIAGRAM 17
FIGURE 5.1: AN IMAGE FROM THE DATASET WITH AN ARBITRARY SIZE IS SCALED TO FIT THE TARGET IMAGE OF SIZE 128×32. 19
FIGURE 5.2: OVERVIEW OF THE NN OPERATIONS AND THE DATA FLOW THROUGH THE NN...21
FIGURE 5.3 256 FEATURE PER TIME-STEP ARE COMPUTED BY THE CNN LAYERS. ...22
FIGURE 5.4: OUTPUT MATRIX OF THE RNN LAYERS. 23
FIGURE 5.5: A MORE DETAILED VIEW ON MODEL ARCHITECTURE 24
FIGURE 5.6: CRNN MODEL...24
FIGURE 6.1: PREDICTION DONE BY HTR SYSTEM. 30
FIGURE 10.1: HOME PAGE OF THE WEBUI...35
FIGURE 10.2: UPLOAD PAGE...35
FIGURE 10.3: PREDICTION 1..36
FIGURE 10.4: CONSOLE OUTPUT...36

ABBREVIASTIONS/ NOTATIONS/ NOMENCLATURE

ANN	Artificial Neural Network
BLSTM	Bidirectional Long Short Term Memory
BRNN	Bidirectional Recurrent Neural Network
CER	Character Error Rate
CNN	Convolutional Neural Network
CRNN	Convolutional Recurrent Neural Network
CTC	Connectionist Temporal Classification
HMM	Hidden Markov Model
HTR	Handwritten Text Recognition
LM	Language Model
LSTM	Long Short Term Memory
MDLSTM	Multidimensional LSTM
ReLU	Rectifying Linear Unit
RNN	Recurrent Neural Network
WAR	Word Accuracy Rate

CHAPTER 1

INTRODCUTION

Handwritten text recognition (HTR) has emerged as a crucial technology in various domains, including document digitization, automated form processing, and historical manuscript transcription. Unlike printed text, handwritten text presents unique challenges due to the variability in individual writing styles, diverse character shapes, and varying degrees of legibility. Overcoming these challenges requires sophisticated techniques capable of accurately interpreting and converting handwritten content into digital text.

In this project, we delve into the development and implementation of a handwritten text recognition system leveraging Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). CNNs, renowned for image processing tasks, are adept at identifying spatial hierarchies in data, making them particularly suitable for recognizing complex patterns in handwritten text. CNNs, with their ability to model complex relationships and learn from vast datasets, complement CNNs by refining the recognition process and enhancing the overall accuracy of the system.

Our approach integrates the strengths of CNNs and RNNs to create a robust and efficient handwritten text recognition system. The project encompasses various stages, including data preprocessing, model architecture design, training, and evaluation. We aim to achieve high accuracy in recognizing handwritten text, paving the way for more efficient document digitization and information retrieval process.

1.1 Motivation

The ability to accurately recognize handwritten text has profound implications across various fields, from digitizing historical documents to enabling efficient data entry and enhancing accessibility technologies. With the rapid advancements in machine learning and neural networks, we now have the tools to tackle the complexities associated with handwritten text recognition. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have shown remarkable performance in image and pattern recognition tasks, making them ideal candidates for developing robust handwritten text recognition systems.

1.2 Problem Statement

- Handwritten text recognition poses significant challenges due to the inherent variability in handwriting styles, inconsistencies in character formation, and differences in writing tools and surfaces.

- Traditional Optical Character Recognition (OCR) systems struggle to maintain high accuracy levels when dealing with handwritten text.

- This project aims to address these challenges by leveraging the strengths of CNNs in feature extraction and the classification capabilities of ANNs to create a comprehensive solution that improves recognition accuracy and robustness.

1.3 Objective

The primary objective of this project is to develop an efficient and accurate handwritten text recognition system using a combination of Convolutional Neural Networks (CNN), Bidirectional LSTMs and CTC loss function. The project will:

1. Design and implement a CNN to effectively extract relevant features from handwritten text images.
2. Integrate an RNN (bidirectional LSTM) for classifying these features into corresponding text characters.
3. Evaluate the performance of the combined CRNN model on diverse datasets of handwritten text.
4. Compare the proposed model's accuracy and efficiency with existing handwritten text recognition systems.

By achieving these objectives, the project aims to contribute to the development of more reliable and practical handwritten text recognition technologies, facilitating advancements in various applications that rely on accurate text digitization.

CHAPTER 2

LITERATURE SURVEY

The most traditional approaches to HTR are based on N -gram language models (LM) and optical modeling of characters by means of Hidden Markov Models with Gaussian mixture emission distributions (HMM-GMM). However, significant improvements in optical modeling were demonstrated by approaching emission probabilities with multilayer perceptrons (HMM-MLP) and also by training the HMM-GMMs with discriminative training techniques. Optical models are trained with pairs of line images and their corresponding transcripts. If transcripts exactly follow the text written in the images, they are usually called “diplomatic”. Language Models, on the other hand, are usually trained using only training text, typically just the transcripts of the training images. (Toselli et al., 2004; Bluche, 2015).

Current state-of-the-art optical modeling HTR technologies are based on deeply layered neural network models which consist of a stack of several *convolutional laye*rs followed by one or more layers of RNNs composed of special “neurons” called *Bidirectional Long Short Term Memory* (BLSTM) units. Finally, a softmax output layer computes an estimate of the probabilities of each character in the training alphabet plus a special “non-character” symbol. The overall architecture is often referred to as *Convolutional-Recurrent Neural Networks* (CRNN). (Graves et al., 2009; Bluche, 2015; J.Puigcerver, 2017).

Graves and Schmidhubler (2008) introduced a more complex “multidimensional” version of BLSTM architecture called *Multidimensional Recurrent Neural Networks* (MDLSTM), which has become fairly popular because of superior performance.

2.1 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are analogous to traditional ANNs in that they are comprised of neurons that self-optimize through learning. The only notable difference between CNNs and traditional ANNs is that CNNs are primarily used in the field of pattern recognition within images. This allows us to encode image-specific features into the architecture, making the network more suited for image-focused tasks - whilst further reducing the parameters required to set up the model.

CNN are comprised of neurons organized into three dimensions, the spatial dimensionality of the input (height and the width) and the depth. The depth does not refer to the total number of layers within the ANN, but the third dimension of an activation volume.

CNNs are comprised of three types of layers. These are convolutional layers, pooling layers and fully-connected layers. When these layers are stacked, a CNN architecture has been formed. A simplified CNN architecture for MNIST classification is illustrated in Figure 2.1.

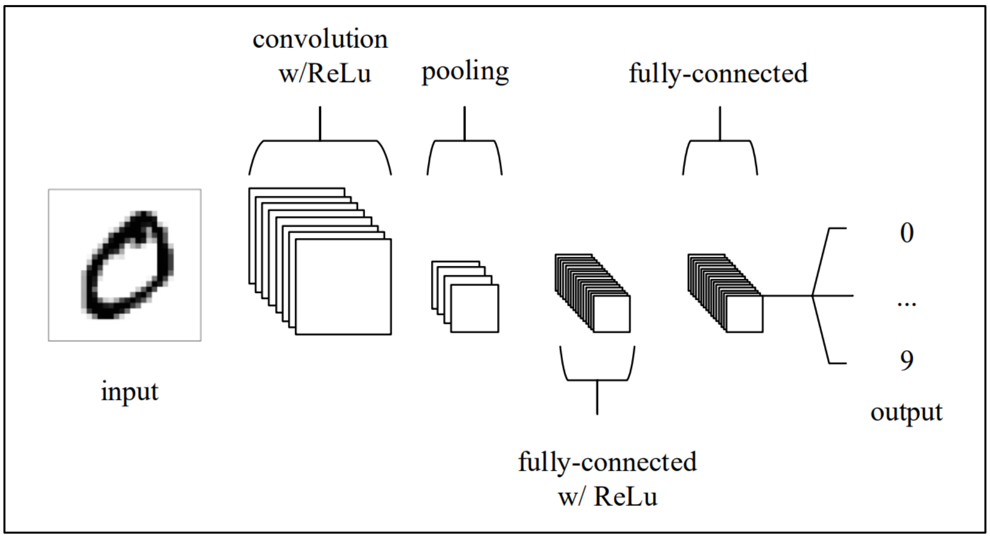


Figure 2.1: A simple CNN architecture, comprised of just five layer

1. As found in other forms of ANN, the input layer will hold the pixel values of the image.

2. The convolutional layer will determine the output of neurons of which are connected to local regions of the input through the calculation of the scalar product between their weights and the region connected to the input volume. The rectified linear unit (commonly shortened to ReLu) aims to apply an 'elementwise' activation function such as sigmoid to the output of the activation produced by the previous layer.

3. The pooling layer will then simply perform downsampling along the spatial dimensionality of the given input, further reducing the number of parameters within that activation.

4. The fully-connected layers will then perform the same duties found in standard ANNs and attempt to produce class scores from the activations, to be used for classification. It is also suggested that ReLu may be used between these layers, as to improve performance.

2.2 Recurrent Neural Network

Recurrent neural networks (RNNs) are a connectionist model containing a self-connected hidden layer. One benefit of the recurrent connection is that a ‘memory’ of previous inputs remains in the network’s internal state, allowing it to make use of past context. Context plays an important role in handwriting recognition, as illustrated in Figure 2.2. Another important advantage of recurrence is that the rate of change of the internal state can be finely modulated by the recurrent weights, which builds in robustness to localized distortions of the input data.

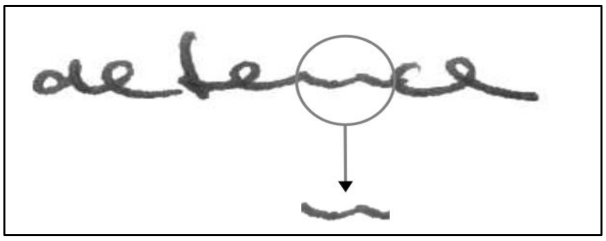


Figure 2.2: Importance of context in handwriting recognition. The word ‘defence’ is clearly legible, but the letter ‘n’ in isolation is ambiguous.

2.2.1 Long Short Term Memory (LSTM)

Unfortunately, the range of contextual information that standard RNNs can access is in practice quite limited. The problem is that the influence of a given input on the hidden layer, and therefore on the network output, either decays or blows up exponentially as it cycles around the network’s recurrent connections. This shortcoming (as the vanishing gradient problem makes it hard for an RNN to bridge gaps of more than about 10 time steps between relevant input and target events. The vanishing gradient problem is illustrated schematically in Figure 2.3.

Long Short-Term Memory (LSTM) is an RNN architecture specifically designed to address the vanishing gradient problem. An LSTM hidden layer consists of recurrently connected subnets, called memory blocks. Each block contains a set of internal units, or cells, whose activation is controlled by three multiplicative gates: the input gate, forget gate and output gate. Figure 2.4 provides a detailed illustration of an LSTM memory block with a single cell.

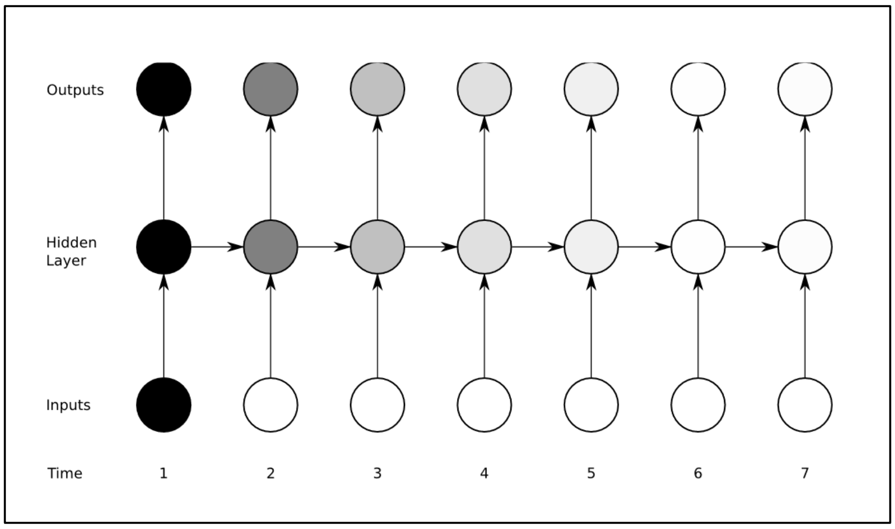


Figure 2.3: Illustration of the vanishing gradient problem. The diagram represents a recurrent network unrolled in time. The units are shaded according to how sensitive they are to the input at time 1 (where black is high and white is low).

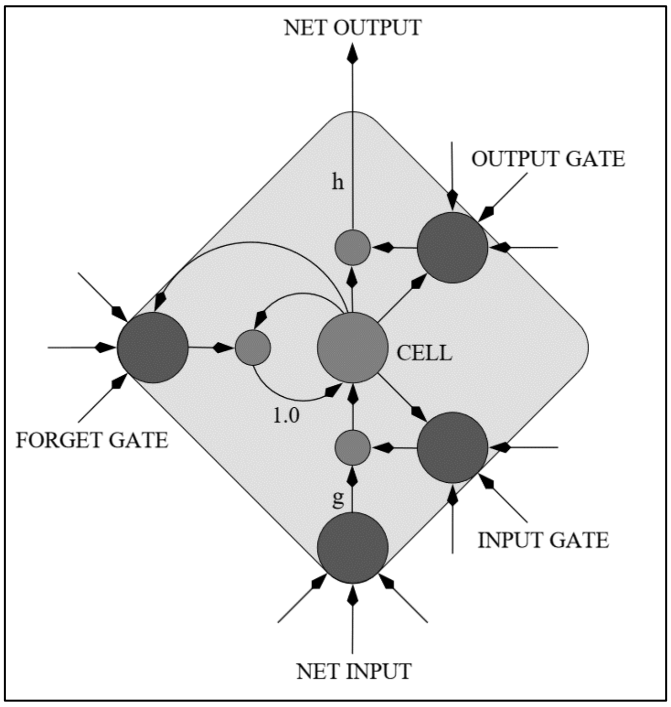


Figure 2.4: LSTM memory block with one cell. The cell has a recurrent connection with fixed weight 1.0. The three gates collect input from the rest of the network, and control the cell via multiplicative units (small circles). The input and output gates scale the input and output of the cell, while the forget gate scales the recurrent connection of the cell. The cell squashing functions (g and h) are applied at the indicated places. The internal connections from the cell to the gates are known as peephole weights.

The effect of the gates is to allow the cells to store and access information over long periods of time. For example, as long as the input gate remains closed (i.e. has an activation close to 0), the activation of the cell will not be overwritten by the new inputs arriving in the network. Similarly, the cell activation is only available to the rest of the network when the output gate is open, and the cell's recurrent connection is switched on and off by the forget gate.

Figure 2.5 illustrates how an LSTM block maintains gradient information over time. Note that the dependency is 'carried' by the memory cell as long as the forget gate is open and the input gate is closed, and that the output dependency can be switched on and off by the output gate, without affecting the hidden cell.

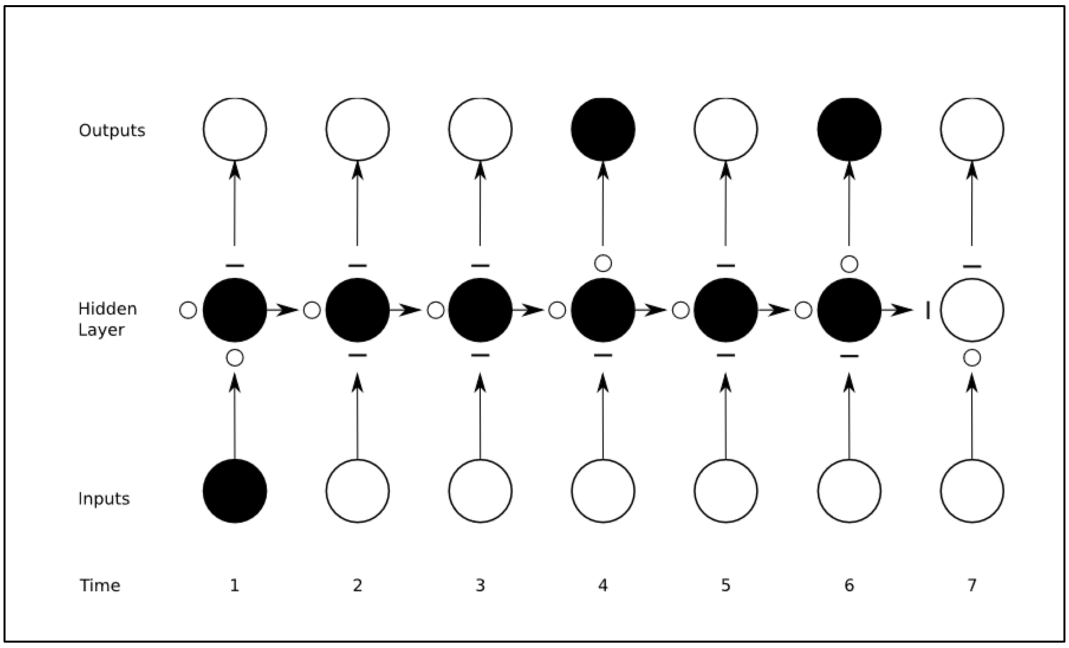


Figure 2.5: Preservation of gradient information by LSTM. The diagram represents a network unrolled in time with a single hidden LSTM memory block. The input, forget, and output gate activations are respectively displayed below, to the left and above the memory block. As in Figure 10, the shading of the units corresponds to their sensitivity to the input at time 1. For simplicity, the gates are either entirely open ('O') or entirely closed ('—').

2.2.2 Bidirectional Recurrent Neural Network

Bidirectional recurrent neural networks (BRNNs) are able to access context in both directions along the input sequence. BRNNs contain two separate hidden layers, one of which processes the input sequence forwards, while the other processes it backwards. Both hidden layers are connected to the same output layer, providing it with access to the past and future context of every point in the sequence.

BRNNs have outperformed standard RNNs in several sequence learning tasks, notably protein structure prediction and speech processing.

Combining BRNNs and LSTM gives bidirectional LSTM (BLSTM). BLSTM has previously outperformed other network architectures, including standard LSTM, BRNNs and HMM-RNN hybrids, on phoneme recognition.

2.3 Connectionist Temporal Classification (CTC)

Connectionist temporal classification (CTC) is an RNN output layer specifically designed for sequence labelling tasks. It does not require the data to be presegmented, and it directly outputs a probability distribution over label sequences. CTC has been shown to outperform both HMMs and RNN-HMM hybrids on a phoneme recognition task. CTC can be used for any RNN architecture.

A CTC output layer contains as many units as there are labels in the task, plus an additional 'blank' or 'no label' unit. The output activations are normalised using the softmax function so that they sum to 1 and are each in the range (0, 1):

$$y_k^t = \frac{e^{a_k^t}}{\sum_{k'} e^{a_{k'}^t}},$$

where at a_k^t is the unsquashed activation of output unit k at time t, and y_k^t k is the final output of the same unit.

The normalised outputs are used to estimate the conditional probabilities of observing label k at time t in the input sequence x, i.e. $y_k^t = \mathrm{p}(\mathrm{k}, \mathrm{t}|\mathrm{x})$. Note that each output is conditioned on the entire input sequence. For this reason, CTC is best used in conjunction with an architecture capable of incorporating long range context in both input directions. One such architecture is bidirectional LSTM, as described in the previous section.

The conditional probability p(π|x) of observing a particular path π through the lattice of label observations is found by multiplying together the label and blank probabilities at every time step:

$$p(\pi|x) = \prod_{t=1}^{T} p(\pi t, t|x) = \prod_{t=1}^{T} y_{\pi_t}^t$$

where πt is the label observed at time t along path π.

Paths are mapped onto label sequences by an operator B that removes first the repeated labels, then the blanks. For example, both $B(a, -, a, b, -)$ and $B(-, a, a, -, -, a, b, b)$ yield the labelling (a, a, b). Since the paths are mutually exclusive, the conditional probability of some labelling l is the sum of the probabilities of all the paths mapped onto it by B:

$$p(l|x) \;=\; \sum_{\pi \in B^{-1}(1)} p(\pi|x)$$

The above step is what allows the network to be trained with unsegmented data. The intuition is that, because we don't know where the labels within a particular transcription will occur, we sum over all the places where they could occur.

CTC Decoder

Given the above formulation, the output of the classifier should be the most probable labelling for the input sequence:

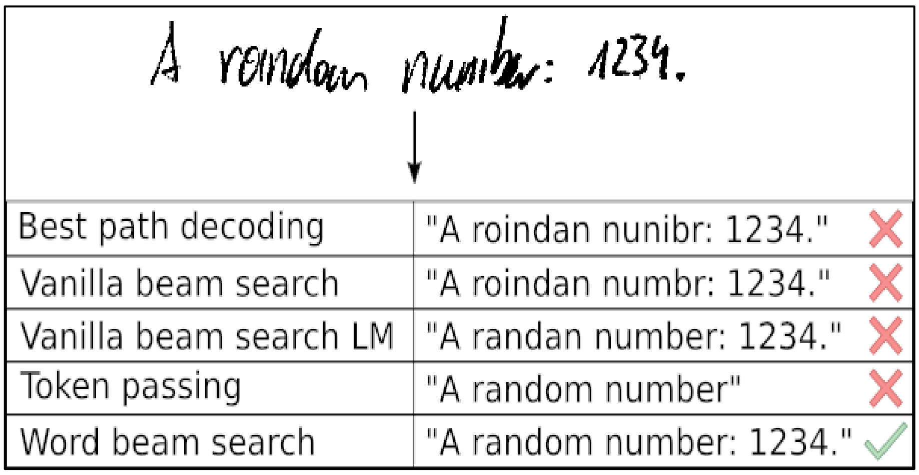
$$h(x) = \; arg \; \max_{\mathrm{l} \in \mathrm{L} \leq \mathrm{T}} p(l|x).$$

We refer to the task of finding this labelling as decoding. Unfortunately, we do not know of a general, tractable decoding algorithm for our system. However the following two approximate methods give good results in practice. The first method (best path decoding) is based on the assumption that the most probable path will correspond to the most probable labelling:

$$h(x) \;\approx\; B(\pi^*)$$

$$\text{where } \pi^* \;=\; arg \; \max_{\pi \in N^t} p(\pi|x)\,.$$

Best path decoding is trivial to compute, since π^* is just the concatenation of the most active outputs at every time-step. However it is not guaranteed to find the most probable labelling.



Best path decoding	"A roindan nunibr: 1234." ✗
Vanilla beam search	"A roindan numbr: 1234." ✗
Vanilla beam search LM	"A randan number: 1234." ✗
Token passing	"A random number" ✗
Word beam search	"A random number: 1234." ✓

Figure 2.6: Typical use case for different types of decoders.

Scheidle et al. describes another better CTC decoding algorithm known as Word Beam Search [1]which is modification of Vanilla Beam Search Decoder. It is

[1] GitHub - githubharald/CTCWordBeamSearch: Connectionist Temporal Classification (CTC) decoder with dictionary and language model.

used for sequence recognition tasks like handwritten text recognition or automatic speech recognition. The four main properties of word beam search are:

- Words constrained by dictionary
- Allows arbitrary number of non-word characters between words (numbers, punctuation marks)
- Optional word-level Language Model (LM)
- Faster than token passing

CTC loss

The output received from the RNN layer is a tensor that contains the probability of each label for each receptive field. But, how does this translate to the output? That's when Connectionist Temporal Classification (CTC) loss comes in. CTC loss is responsible for training the network as well as the inference that is decoding the output tensor. CTC works on the following major principles:

- Text encoding: CTC solves the issue when a character takes more than one time step. CTC solves this by merging all the repeating characters into one. And, when that word ends it inserts a blank character "-". This goes on for further characters. For example in fig -04, 'S' in 'STATE' has three time steps. The network might predict those time steps as 'SSS'. Now, the CTC will merge those outputs and predict the output as 'S'. For the word, a possible encoding could be SSS-TT-A-TT-EEE, Hence the output 'STATE'.

- Loss Calculation: For a model to learn, loss needs to be calculated and back propagated into the network. Here the loss is calculated by adding up all the score of possible alignments at each time step, that sum is the probability of the output sequence. Finally, the loss is calculated by taking a negative logarithm of the probability, which is then used for back-propagation into the network.

- Decoding: At the time of inference, we need a clear and accurate output. For this, CTC calculates the best possible sequence from the output tensor or matrix by taking the characters with the highest probability per time step. Then it involves decoding which is the removal of blanks "-" and repeated characters.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 Introduction

Requirement analysis is an essential part of product development. It plays an important role in determining the feasibility of an application. Requirement analysis defines the software and hardware necessities that are required to develop the product or application. Requirement analysis mainly consists of software requirements, hardware requirements and functional requirements.

Software requirements: This mainly refers to the needs that solve the end user problems using the software. The activities involved in determining the software requirements are:

1. Elicitation: This refers to gathering of information from the end users and customers.
2. Analysis: Analysis refers to logically understanding the customer needs and reaching a more precise understanding of the customer requirements.
3. Specification: Specification involves storing the requirements in the form of use cases, user stories, functional requirements and visual analysis.
4. Validation: Validation involves verifying the requirements that has been specifies.
5. Management: During the course of development, requirements constantly change and these requirements have to be tested and updated accordingly.

Hardware requirements: Hardware requirements are the physical aspects that are needed for an application. All the software must be integrated with hardware in order to complete the development process. The hardware requirements that are taken into account are:

1. Processor Cores and Threads
2. GPU Processing Power
3. Memory
4. Secondary Storage
5. Network Connectivity

3.2 Software and Hardware Requirements

SOFTWARE REQUIREMENTS: Listed below are the software requirements for performing time series analysis on the fraud data:

1. **Python** : Python is a high-level, interpreted programming language known for its simplicity, readability, and extensive ecosystem of libraries and frameworks specifically designed for machine learning and neural networks

2. **Anaconda:** Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. Package versions are managed by the package management system conda.

3. **Spyder:** Spyder is an open-source cross-platform integrated development environment for scientific programming in the Python language. Uses include: data cleaning and transformation, numerical simulation, statistical modelling, data visualization, machine learning, and much more.

4. **CUDA:** CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model created by NVIDIA. It allows developers to utilize the computational power of NVIDIA GPUs (Graphics Processing Units) for training of deep learning models.

5. **Dependencies**:

    a. **OpenCV**: OpenCV, short for Open Source Computer Vision Library, is an open-source computer vision and machine learning software library. It provides a wide range of functions and algorithms for real-time computer vision tasks like image processing, object detection, video analysis, and more.

    b. **Tensorflow**: TensorFlow is an open-source machine learning framework developed by Google Brain for building and training machine learning models. It's designed to provide a flexible, scalable, and easy-to-use platform for implementing and deploying machine learning algorithms and deep learning models.

    c. **Numpy**: NumPy is a fundamental package for scientific computing in Python. It provides support for arrays, matrices, and a collection of mathematical functions to operate on these arrays.

d. **Flask**: Flask is a lightweight and flexible web framework for Python. It's designed to make building web applications and APIs in Python

e. **CTCWordBeamSearch**: A CTC decoder used for sequence recognition tasks like handwritten text recognition or automatic speech recognition.

HARDWARE REQUIREMENTS:

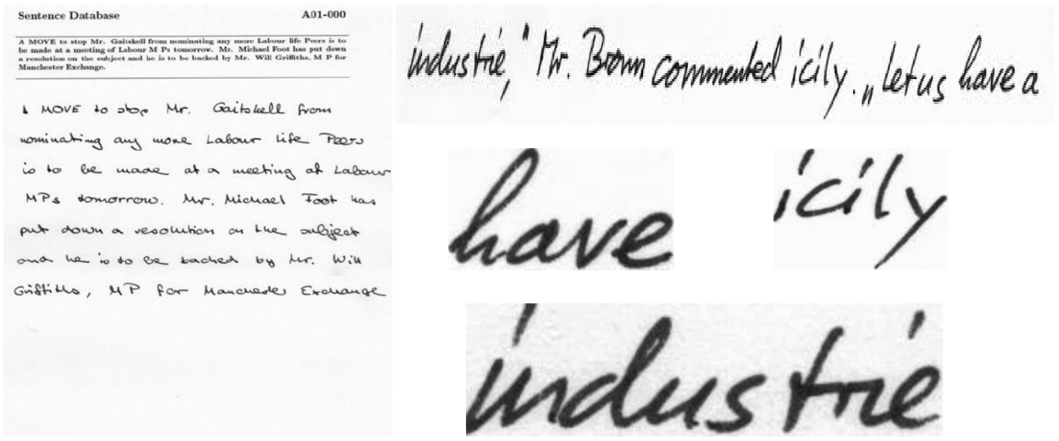
1. Processor: Intel i5 2.5 GHz up to 3.5 GHz (or AMD equivalent)
2. GPU (preferred): dedicated GPU from NVIDIA or AMD with 4GB VRAM
   : We used NVIDIA RTX 3050 (4GB VRAM)
3. Memory: minimum 8GB RAM
4. Secondary Storage: minimum 128GB SSD or HDD
5. Network Connectivity: bandwidth ~ 10 Mbps 3 75 Mbps

3.2 Dataset

3.3.1 IAM Handwriting Dataset

The IAM Handwriting Database contains forms of handwritten English text which can be used to train and test handwritten text recognizers and to perform writer identification and verification experiments.

The database was first published in [1][2] at the ICDAR 1999. The database contains forms of unconstrained handwritten text, which were scanned at a resolution of 300dpi and saved as PNG images with 256 gray levels. The figure below provides samples of a complete form, a text line and some extracted words.



Sentence Database A01-000

A MOVE to stop Mr. Gaitskell from nominating any more Labour life Peers is to be made at a meeting of Labour M Ps tomorrow. Mr. Michael Foot has put down a resolution on the subject and he is to be backed by Mr. Will Griffiths, M P for Manchester Exchange.

A MOVE to stop Mr. Gaitskell from
nominating any more Labour life Peers
is to be made at a meeting of Labour
MPs tomorrow. Mr. Michael Foot has
put down a resolution on the subject
and he is to be backed by Mr. Will
Griffiths, MP for Manchester Exchange

industrie," Mr. Brown commented icily. " Let us have a

have icily

industrie

Figure 3.1 Samples of IAM dataset

Characteristics

The IAM Handwriting Database 3.0 is structured as follows:

- 657 writers contributed samples of their handwriting
- 1'539 pages of scanned text
- 5'685 isolated and labeled sentences
- 13'353 isolated and labeled text lines
- 115'320 isolated and labeled words

[2] [1] U. Marti and H. Bunke. A full English sentence database for off-line handwriting recognition. In Proc. of the 5th Int. Conf. on Document Analysis and Recognition, pages 705 - 708, 1999.

3.2 Functional and Non-Functional Requirements

FUNCTIONAL REQUIREMENTS: Functional requirements are those which represent the functions of the system. Functional requirements may involve calculations, data manipulation, technical details and data processing. The functional requirements are:

1. Data Pre-processing: This process involves cleaning, transforming and reducing data to convert raw data in a useful manner.

2. Training: Initially, the system has to train based on the data set given. The training period is when the system learns how to perform the required task based on the inputs given through the dataset.

3. Forecasting: Forecasting is the process of making predictions of the future based on past and present data and most commonly by analysis of trends.

4. Validation: This is necessary in order to determine the accuracy of the system.

NON-FUNCTIONAL REQUIREMENTS: Non-functional requirements are the specifics that can be used to measure the operation of the system. These include:

1. Accuracy: This refers to the number of correct outputs to the total number of outputs.

2. Openness: The system must be guaranteed to work efficiently for a certain period of time.

3. Portability: The system must be designed in a way which is independent of the platform on which it is run. This makes it possible to run on many systems without making a lot of changes.

4. Reliability: The system has to produce fast and accurate result

CHAPTER 4

SYSTEM DESIGN

The aim of this project is to develop an HTR system that can accurately recognize and convert handwritten text into digital format. To achieve this goal, we propose a system design that utilizes a CRNN (Convolutional Recurrent Neural Network) model for training our HTR system.

- Image pre-processing: The raw input images are pre-processed to remove noise, enhance image data and features, and produce a clean image that can be easily recognized by the OCR system. The pre-processing steps include image resizing, normalization, grayscale conversion, and binarization. All of these steps will be discussed in *Chapter 5*.

- Data preparation: The pre-processed images are then divided into training and testing sets. Ground truths are developed for each training image, which is a text string that represents the actual text in the image. The ground truths are used for training the CRNN model.

- Model training: The CRNN model is trained using the training set images and their corresponding ground truths. The model architecture consists of a combination of convolutional and recurrent neural networks that work together to recognize and classify the input images. The training process involves iterative updates of the model weights to minimize the loss function.

- Model evaluation: The trained model is evaluated using the testing set images and their corresponding ground truths. The evaluation metrics used to assess the performance of the OCR system include character error rate (CER) and word accuracy (WAR).

- Inference: Once the model is trained and evaluated, it can be used for inference on new input images. The pre-processing steps are applied to the input image to produce a clean image, which is then fed into the CRNN model for recognition. The recognized text is then output as digital text.

- User Interface: The user interface (UI) of our HTR project is designed using Flask, popular web development framework and simple HTML-CSS scripts. The UI design was intended to be user friendly and easy to navigate, with a clean and simple layout. The website included a portal for data collection, where users could either provide their own handwritten texts or annotate texts from provided images. The data collection portal was designed to be intuitive and easy to use, with clear instructions and prompts for users. The UI also included a section for the OCR output, where users could see the converted text from their provided images or handwritten texts. The output section was designed to be visually appealing and easy to read.

A DFD of our HTR system is given in Figure 4. Data Flow Diagram is a graphical representation of the flow of data through information through information system, modeling its prospects. DFDs are useful for the visualization of data processing

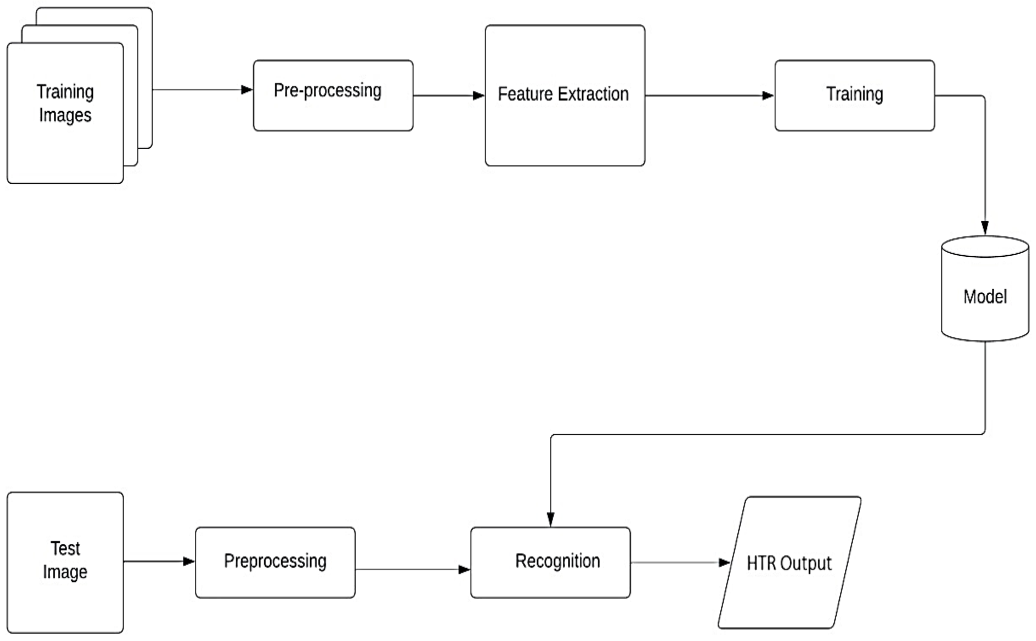


Figure 4: System Block Diagram

CHAPTER 5

IMPLEMENTATION

5.1 Data Preprocessing

The input to the NN model is a gray-value image of size 128×32. Usually, the images from the dataset do not have exactly this size, therefore we resize it (without distortion) until it either has a width of 128 or a height of 32. Then, we copy the image into a (white) target image of size 128×32. This process is shown in Fig. 5.1. 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.

5.1.1 Data Augmentation

In order to improve the performance of our HTR system, we used data augmentation techniques to increase the size and diversity of our training dataset. Data augmentation involves generating new training examples by applying transformations to existing images, such as rotation, scaling, elastic deformation, random zoom, and cropping. All these techniques are discussed briefly below:

- Random rotation: Random rotation involves rotating the images by a random angle, which helps to improve the HTR system’s ability to recognize characters at different orientations. We implemented random rotation using the openCV library in Python, which provides a variety of image augmentation functions.

- Random scaling: We also used random scaling as a data augmentation technique to increase the diversity of our training data. Random scaling involves resizing the images to a random size, which helps to improve the HTR system’s ability to recognize characters at different sizes.

- Elastic deformation: Elastic deformation involves randomly distorting the images by applying small displacements to their pixels, simulating the effect of stretching and compressing the images. This technique helps to increase the variability of the dataset and improve the robustness of the HTR system to variations in character shape and size.

Overall, these data augmentation were an effective technique in increasing the variability of our training data, and helped to improve the HTR system’s ability to recognize characters at different angle.

5.1.2 Grayscale

This process converts an image with RGB channels into an image with a single grayscale channel. An RGB image has Red, Green, and Blue values of a pixel ranging from 0 to 255. These values are converted into a single gray value. The value of each grayscale pixel is calculated as the weighted sum of the corresponding red, green, and blue pixels. Humans perceive green more strongly than red and red more strongly than blue. Since Humans don't perceive all colors equally the weighted sum method is used to better simulate how the human eye perceives the image.

The weighted sum method of grayscale conversion results in a more dynamic grayscale image.

5.1.3 Image Resizing

Image resizing is a crucial preprocessing step in our project. Here, we describe our approach to resizing images without distortion and then copying them into a target image of size 128×32.

- **Resizing without Distortion:** To resize images without distortion, we aim to maintain the original aspect ratio of the images. This prevents squishing or stretching of the image content, preserving its visual integrity. We use the following steps to achieve this:
  - Calculate the aspect ratio of the original image:
    Aspect ratio = Width / Height.
  - Determine the target aspect ratio: We aim for a width of 128 or a height of 32. If the original aspect ratio is closer to 128:32 (or 4:1), we resize based on width. Otherwise, we resize based on height.
  - Compute the scaling factor:
    Scaling factor = Target dimension / Original dimension
  - Resize the image using the scaling factor: We scale both the width and height by the same factor to maintain the aspect ratio. This ensures that the resized image fits within the target dimensions while preserving its original proportions.

- **Copying into a Target Image:** After resizing each image, we copy it into a target image of size 128×32. We initialize the target image as a blank white image to ensure that the copied image stands out clearly against the background.

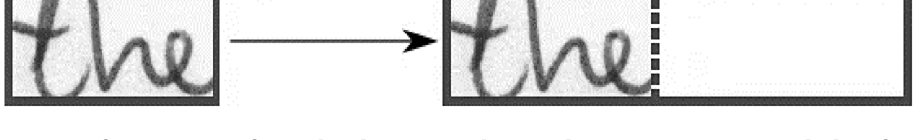


Figure 5.1: Left: an image from the dataset with an arbitrary size. It is scaled to fit the target image of size 128×32, the empty part of the target image is filled with white color.

5.1.4 Transpose image

Transposing a grayscale image involves swapping its rows and columns. However, for a grayscale image, where each pixel corresponds to a single intensity value, the transpose operation doesn't change the image's content significantly. It merely changes its orientation. It increases efficiency for training the NN and inferring from the model.

5.1.5 Normalization

Normalization is a technique used to adjust the gray values in an image to a specific range in order to enhance its visual appearance or facilitate further image processing tasks. The process involves scaling the pixel values so that they fall within a desired range, typically between 0 and 255 for 8-bit grayscale images.

In this project, we calculate the mean and standard deviation of pixel values in the input image, then subtracts the mean from each pixel and divides by the standard deviation to normalize the pixel values. This process helps standardize the intensity distribution of the image, which can be useful for various image processing tasks.

5.2 Training the Model

5.2.1 Training

Training Data:

For training of model, we put samples from dataset into batches. A batch is split into training set (95%) and validation set (5%). Each batch represents an epoch of training. Every input image is of size 128x32.

Model Architecture:

The HTR model used the CRNN algorithm with three stages:

1. Multi-scale feature Extraction --> Convolutional Neural Network (5 Layers)
2. Sequence Labeling (BLSTM-CTC) --> Recurrent Neural Network (2 layers of LSTM) with CTC
3. Transcription --> Decoding the output of the RNN (CTC decode)

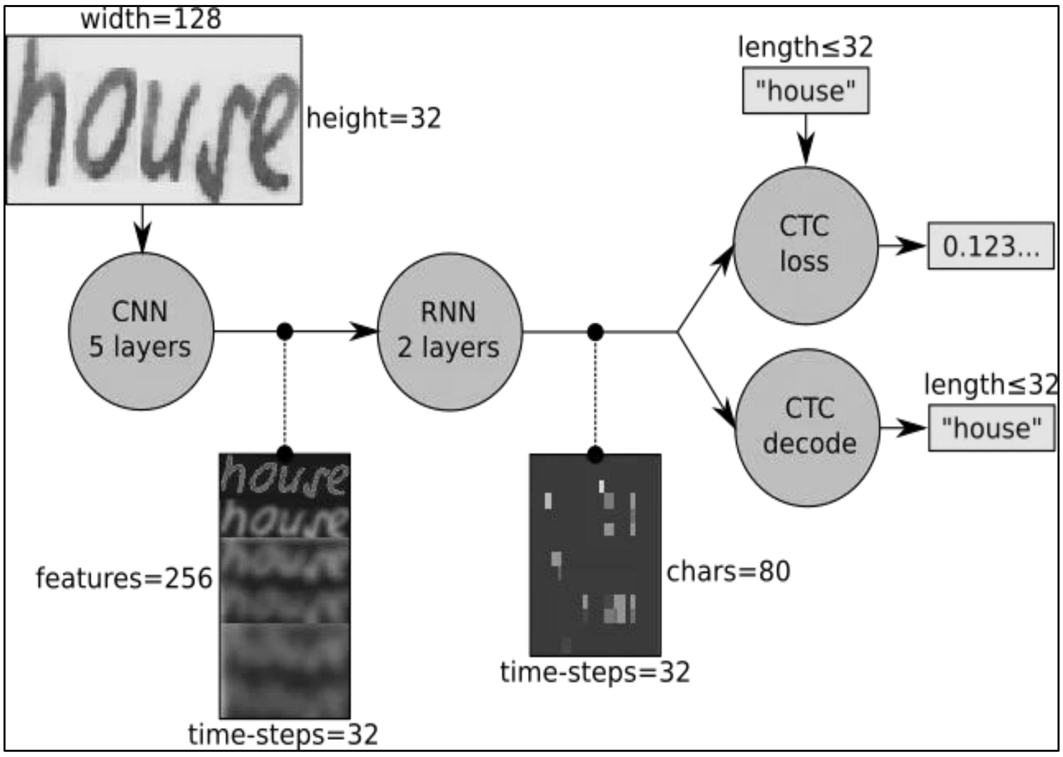


Figure 5.2: Overview of the NN operations and the data flow through the NN

Model Overview

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 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.

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 32×80. The IAM dataset consists of 79 different characters, further one additional character is needed for the CTC operation (CTC blank label), and therefore there are 80 entries for each of the 32 time-steps

CTC: while training the NN, the CTC is given 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.

CNN output: Figure 5.3 shows the output of the CNN layers which is a sequence of length 32. Each entry contains 256 features. Of course, these features are further processed by the RNN layers, however, some features already show a high correlation with certain high-level properties of the input image: there are features which have a high correlation with characters (e.g. "e"), or with duplicate characters (e.g. "tt"), or with character-properties such as loops (as contained in handwritten "l"s or "e"s).

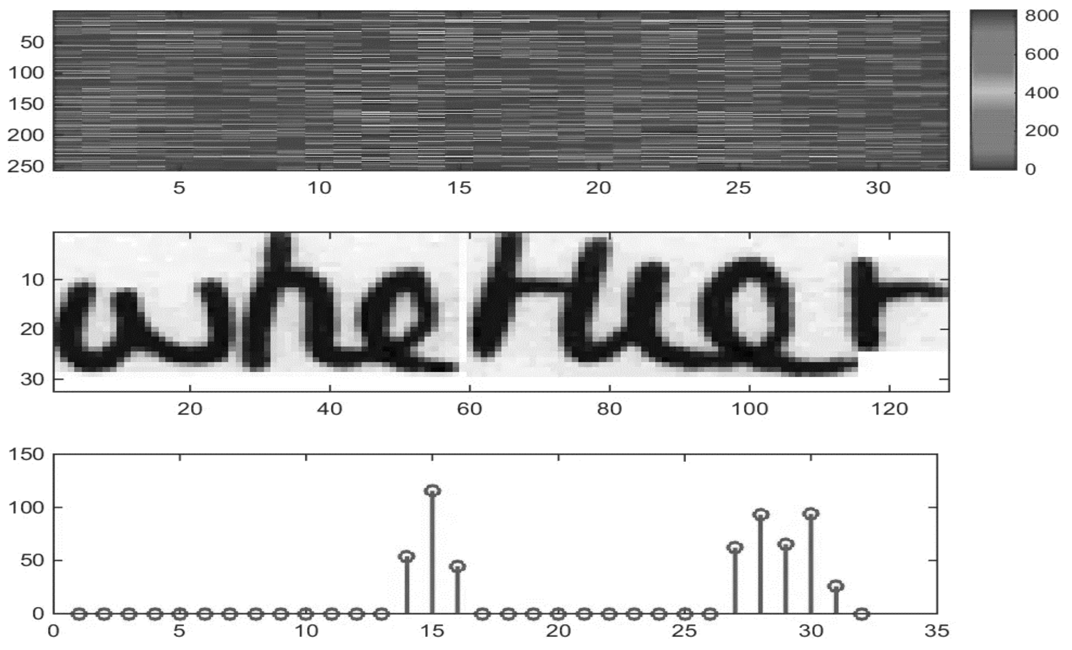


Figure 5.3 Top: 256 feature per time-step are computed by the CNN layers. Middle: input image. Bottom: plot of the 32nd feature, which has a high correlation with the occurrence of the character "e" in the image.

RNN output: Figure 5.4 shows a visualization of the RNN output matrix for an image containing the text “little”. The matrix shown in the top-most graph contains the scores for the characters including the CTC blank label as its last (80th) entry. The other matrix-entries, from top to bottom, correspond to the following characters:

“ !”#&’() ∗ +, −./0123456789: ; ? ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz”

It can be seen that most of the time, the characters are predicted exactly at the position they appear in the image (e.g. compare the position of the “i” in the image and in the graph). Only the last character “e” is not aligned. But this is OK, as the CTC operation is segmentation-free and does not care about absolute positions. From the bottom-most graph showing the scores for the characters “l”, “i”, “t”, “e” and the CTC blank label, the text can easily be decoded: we just take the most probable character from each time-step, this forms the so called best path, then we throw away repeated characters and finally all blanks:

“l---ii--t-t--l-…-e” → “l---i--t-t--l-…-e” → “little”.

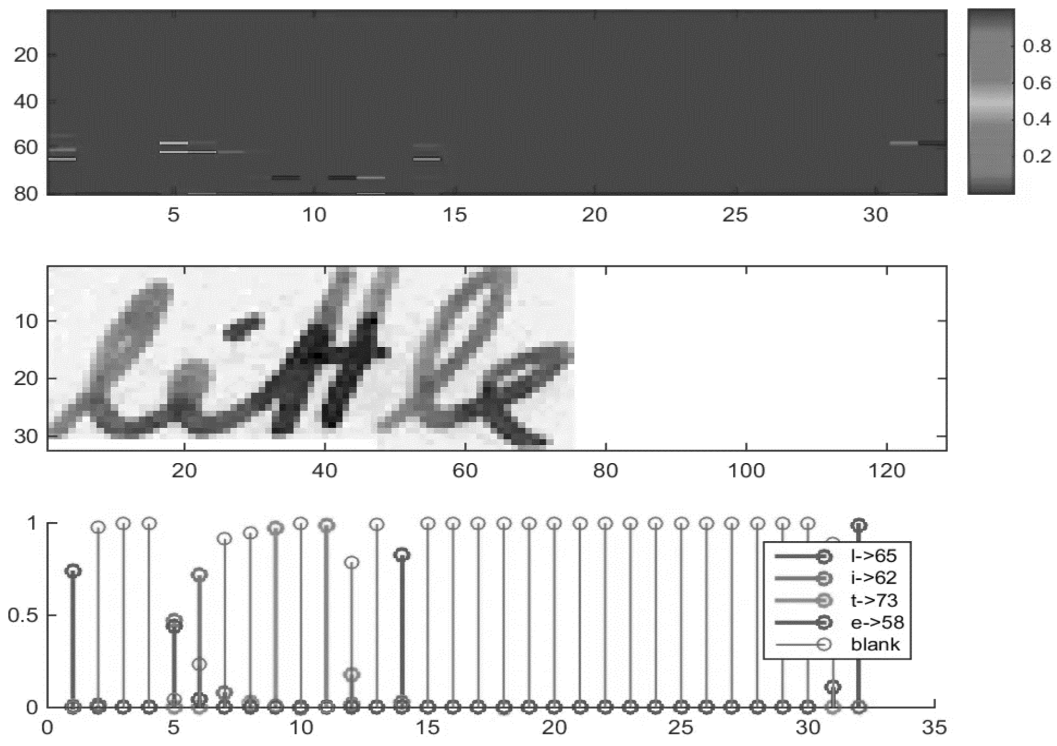


Figure 5.4: Top: output matrix of the RNN layers. Middle: input image. Bottom: Probabilities for the characters “l”, “i”, “t”, “e” and the CTC blank label.

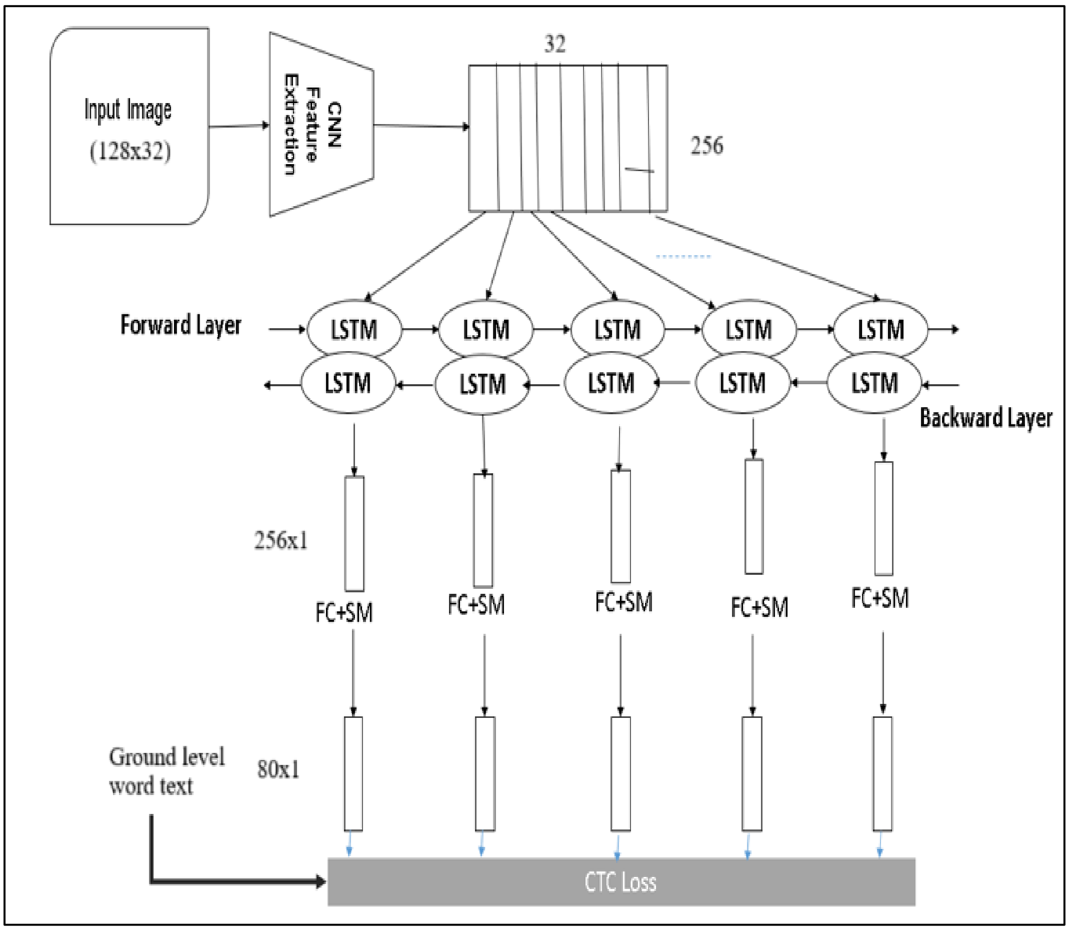
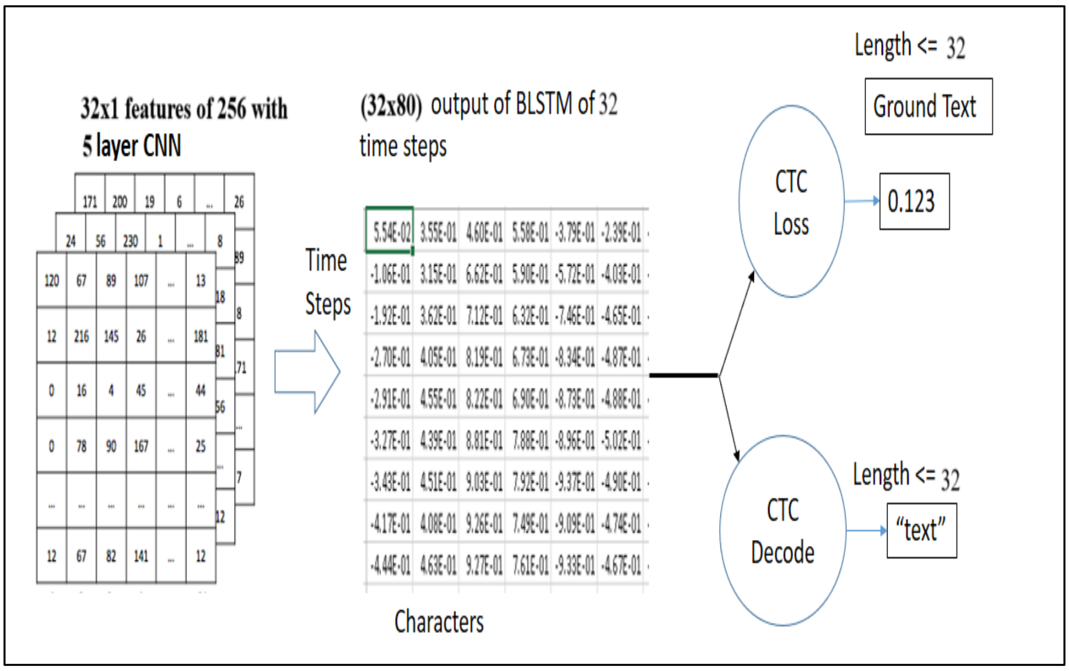


Figure 5.5: A more detailed view on model architecture



5.54E-02	3.55E-01	4.60E-01	5.58E-01	-3.79E-01	-2.39E-01
-1.06E-01	3.15E-01	6.62E-01	5.90E-01	-5.72E-01	-4.03E-01
-1.92E-01	3.62E-01	7.12E-01	6.32E-01	-7.46E-01	-4.65E-01
-2.70E-01	4.05E-01	8.19E-01	6.73E-01	-8.34E-01	-4.87E-01
-2.91E-01	4.55E-01	8.22E-01	6.90E-01	-8.73E-01	-4.88E-01
-3.27E-01	4.39E-01	8.81E-01	7.88E-01	-8.96E-01	-5.02E-01
-3.43E-01	4.51E-01	9.03E-01	7.92E-01	-9.37E-01	-4.90E-01
-4.17E-01	4.08E-01	9.26E-01	7.49E-01	-9.09E-01	-4.74E-01
-4.44E-01	4.63E-01	9.27E-01	7.61E-01	-9.33E-01	-4.67E-01

Figure 5.6: CRNN Model

Training procedure:

Training was held for multiple days in RTX 3050 with CUDA and cuDNN which were installed according to the specifications outlined in the NVIDIA documentation. We used this GPU to train our deep learning model, which required a significant amount of computational resources to complete the process efficiently.

HyperParameter Tuning:

The training process did not happen overnight. After completing each stage of training evaluation, we collected and analyzed the model's bad cases in real-world scenarios. Based on this analysis, we made targeted adjustments to the proportion of training data and added synthetic data as needed. We then repeated this process of training and evaluation through multiple iterations, which allowed us to continually optimize the effectiveness of our model. As a result of this approach, we were able to improve the model's performance over time and achieve better results in real-world scenarios. For our training, we used the following hyperparameters:

– Optimizer: Adam

– Learning Rate: 0.01, 0.001, 0.0001 (decay learning rate)

– Batch size: 50

Evaluation metrics:

The performance of the HTR model was evaluated using the Character Error Rate (CER), which measures the percentage of incorrect characters in the predicted text.CER stands for Character Error Rate, which is a metric used to evaluate the performance of a text recognition system. CER measures the rate at which the recognition system makes errors in converting an input text sequence to an output text sequence. To calculate CER, you need to first obtain the edit distance between the input text and the recognized text. The edit distance is the minimum number of operations (insertion, deletion, or substitution) required to transform the input text into the recognized text? Once you have the edit distance, you can calculate CER as follows:

$$CER = \frac{total\ number\ of\ errors}{total\ number\ of\ characters\ in\ the\ input\ text}$$

For example, suppose the input text is "hello world" and the recognized text is "helo word". The edit distance between the two texts is 3 (one deletion and one substitution for the character "l", and one deletion for the character "d"). The total number of characters in the input text is 11. Therefore, the CER is 3/11, which is approximately 0.27 or 27.

Equation

CER calculation is based on the concept of Levenshtein distance, where we count the minimum number of character-level operations required to transform the ground truth text (aka reference text) into the HTR output. It is represented with this formula:

$$CER = \frac{S + D + I}{N}$$

where

- S = Number of Substitutions
- D = Number of Deletions
- I = Number of Insertions
- N = Number of characters in reference text (also known as ground truth)

Note: The denominator N can alternatively be computed with: N = S + D + C (where C = number of correct characters)

The output of this equation represents the percentage of characters in the reference text that was incorrectly predicted in the HTR output. The lower the CER value (with 0 being a perfect score), the better the performance of the HTR model.

5.3 Source Code

The implementation consists of 4 modules:

1. SamplePreprocessor.py: prepares the images from the IAM dataset for the NN.

2. DataLoader.py: reads samples, puts them into batches and provides an iterator-interface to go through the data.

3. Model.py: creates the model as described above, loads and saves models, manages the TF sessions and provides an interface for training and inference.

4. main.py: puts all previously mentioned modules together.

CNN:

For each CNN layer, kernel of size k×k is used in the convolution operation.

```
kernel = tf.Variable(tf.random.truncated_normal
         ([kernelVals[i], kernelVals[i],
          featureVals[i], featureVals[i + 1]],
          stddev=0.1))

conv = tf.nn.conv2d(pool, filters=kernel,
        padding='SAME',  strides=(1,1,1,1))
```

Then, the result of the convolution is fed into the RELU operation and then again to the pooling layer with size px×py and step-size sx×sy.

```
learelu = tf.nn.leaky_relu(conv_norm, alpha=0.01)

pool = tf.nn.max_pool2d(input=learelu, ksize=(1,
       poolVals[i][0], poolVals[i][1], 1),
       strides=(1, strideVals[i][0],
       strideVals[i][1], 1),
        padding='VALID')
```

These steps are repeated for all layers in a for-loop.

RNN:

Two RNN layers with 256 units each are created and stacked.

```
cells = [tf.compat.v1.nn.rnn_cell.LSTMCell
         (num_units=numHidden, state_is_tuple=True)
         for _ in range(2)] # 2 layers
```

```
stacked = tf.compat.v1.nn.rnn_cell.MultiRNNCell(cells,
          state_is_tuple=True)
```

Then, bidirectional RNN is created from it, such that 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 32×80 which is fed into the CTC layer.

```
((fw,bw),_) = tf.compat.v1.nn.bidirectional_dynamic_rnn
             (cell_fw=stacked, cell_bw=stacked,
              inputs=rnnIn3d, dtype=rnnIn3d.dtype)
```

CTC:

For loss calculation, we feed both the ground truth text and the matrix to the operation. The ground truth text is encoded as a sparse tensor. The length of the input sequences must be passed to both CTC operations.

```
self.gtTexts = tf.SparseTensor(tf.compat.v1.placeholder
              (tf.int64, shape=[None, 2]) ,
               tf.compat.v1.placeholder(tf.int32,
                 [None]),
               tf.compat.v1.placeholder(tf.int64, [2]))

self.seqLen = tf.compat.v1.placeholder(tf.int32,
             [None])
```

We now have all the input data to create the loss operation and the decoding operation.

```
self.lossPerElement = tf.compat.v1.nn.ctc_loss
                      (labels=self.gtTexts,
                      inputs=self.savedCtcInput,
                      sequence_length=self.seqLen,
                      ctc_merge_repeated=True)

if self.decoderType == DecoderType.BestPath:
      self.decoder = tf.nn.ctc_greedy_decoder
                     (inputs=self.ctcIn3dTBC,
                      sequence_length=self.seqLen)

elif self.decoderType == DecoderType.BeamSearch:
      self.decoder = tf.nn.ctc_beam_search_decoder
                     (inputs=self.ctcIn3dTBC,
                     sequence_length=self.seqLen,
                     beam_width=50)

elif self.decoderType == DecoderType.WordBeamSearch:
      chars = str().join(self.charList)
      wordChars = open('../model/wordCharList.txt')
                 .read().splitlines()[0]
```

```
     corpus = open('../data/corpus.txt').read()

     from word_beam_search import WordBeamSearch
     self.decoder = WordBeamSearch(50, 'Words', 0.0,
                    corpus.encode('utf8'),
                     chars.encode('utf8'),
                    wordChars.encode('utf8'))
     self.wbs_input = tf.nn.softmax(self.ctcIn3dTBC,
                                    axis=2)
```

Training:

The mean of the loss values of the batch elements is used to train the NN: it is fed into an optimizer such as Adam.

```
self.optimizer = tf.compat.v1.train.AdamOptimizer
                 (learning_rate=self.learningRate)
                 .minimize(self.loss)
```

CHAPTER 6

SYSTEM TESTING

6.1 Result

English handwritten text, written on blank paper is captured using a smartphone camera and is uploaded to the application for conversion. After a few seconds of processing, the result is displayed on the interface. The result obtained is the conversion of handwritten text into digital form. The result obtained can also be edited if any error is present in the output.

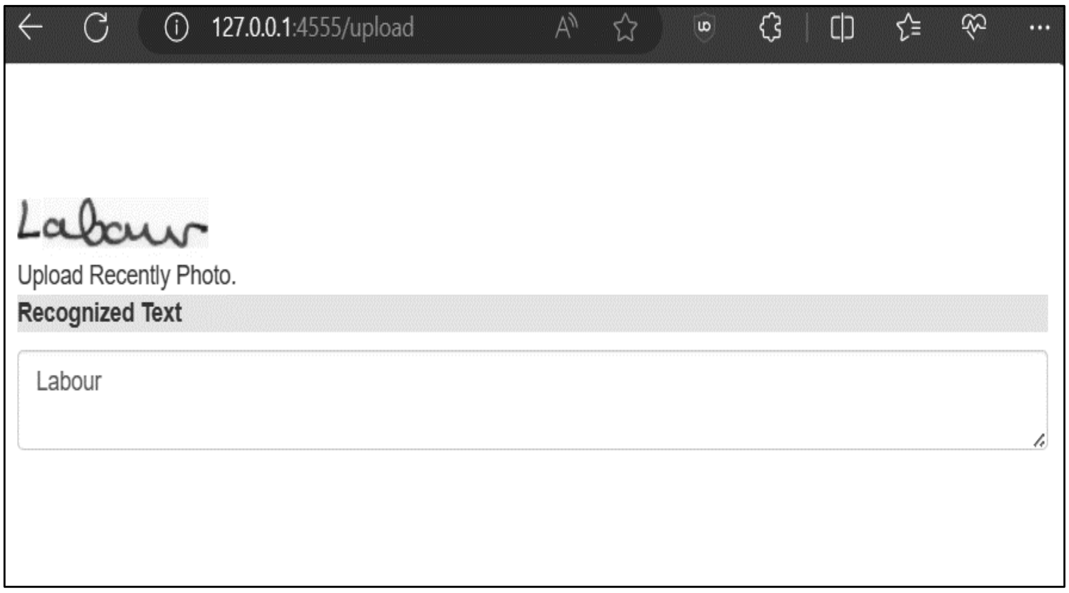


Figure 6.1: Prediction done by HTR system.

6.2 Evaluation

6.2.1 Character Error Rate

Character Error Rate The character error rate (CER) is a commonly used metric for evaluating the performance of character recognition systems. It measures the percentage of characters that are incorrectly recognized by the system. In this section, we will report the CER achieved by our system and compare it with the performance of existing systems.

Our system achieved a CER of 9.11 percent on the evaluation dataset. This means that for every 100 characters in the dataset, our system incorrectly recognized 9.11 of them. The Word Accuracy of the system is 82.88 percent.

$$CER = \frac{total\ number\ of\ errors}{total\ number\ of\ characters\ in\ the\ input\ text}$$

Equation

CER calculation is based on the concept of Levenshtein distance, where we count the minimum number of character-level operations required to transform the ground truth text (aka reference text) into the HTR output. It is represented with this formula:

$$CER = \frac{S + D + I}{N}$$

where

- S = Number of Substitutions
- D = Number of Deletions
- I = Number of Insertions
- N = Number of characters in reference text (also known as ground truth)

Note: The denominator N can alternatively be computed with: N = S + D + C (where C = number of correct characters)

The output of this equation represents the percentage of characters in the reference text that was incorrectly predicted in the HTR output. The lower the CER value (with 0 being a perfect score), the better the performance of the HTR model.

CONCLUSION

In conclusion, we developed an HTR system using a combination of deep learning models including CNN for text detection and BLSTM-CTC for text recognition. The system was trained on a dataset of handwritten text images that was downloaded from IAM Online Dataset. Our experiments show that the developed HTR system achieved high accuracy on recognizing handwritten text in English language.

However, there are some limitations to our approach that should be addressed in future work. One limitation of our approach is that the HTR system heavily relies on the quality of the input images. Even with the data filtering process we used to select high-quality images for training, there are still many factors that can affect the accuracy of the system such as lighting, orientation, and quality of handwriting. In future work, we could investigate more advanced preprocessing techniques to improve image quality and reduce the impact of these factors on HTR accuracy. Additionally, we could explore other deep learning models and architectures that may be better suited for recognizing specific types of handwriting or languages.

Finally, we should note that there are existing HTR systems such as Google Cloud Vision that have already achieved high accuracy on recognizing both printed and handwritten text. However, these systems may come with high costs or limitations on usage depending on the specific requirements of the application. Therefore, our developed HTR system could serve as a more cost-effective and customizable solution for applications that require recognizing handwritten text in specific languages or handwriting styles.

Overall, our developed HTR system has demonstrated promising results in recognizing handwritten text in English language. However, further research and development are needed to address the limitations and improve the robustness and generalizability of the system.

FUTURE SCOPE

1. HTR efficiency can be improved by use of Deslanting Algortihm to remove slanting from the handwritten text.

2. By expanding the input size of CNN, complete text lines can be fed to NN. IAM lines dataset can be used for training such NN.

3. More CNN layers can be added to improve Character error rate.

4. A Language Model (LM) can be used to only include the word contained in a dictionary.

5. Using MDLSTM to recognize whole paragraph at once Scan, Attend and Read: End-to-End Handwritten Paragraph Recognition with MDLSTM Attention[3]

6. Line segmentations can be added for full paragraph text recognition. For line segmentation you can use A* path planning algorithm or CNN model to separate paragraph into lines.

7. AutoEval: An automated evaluation system that automatically grade exams, quizzes, or assignments based on predefined criteria.

[3] https://arxiv.org/abs/1604.03286 Scan, Attend and Read: End-to-End Handwritten Paragraph Recognition with MDLSTM Attention.

REFERENCES

1. H. Toselli, A. Juan, D. Keysers, J. Gonz´alez, I. Salvador, H. Ney, E. Vidal, F. Casacuberta, Integrated handwriting recognition and interpretation using finite-state models, International Journal of Pattern Recognition and Artificial Intelligence 18 (4) (2004) 519–539.

2. T. Bluche, Deep neural networks for large vocabulary handwritten text recognition, Ph.D. thesis, Ecole Doctorale Informatique de Paris-Sud - Laboratoire d'Informatique pour la M´ecanique et les Sciences de l'Ing´enieur, discipline : Informatique (May 2015).

3. A. Graves, M. Liwicki, S. Fern´andez, R. Bertolami, H. Bunke, J. Schmid-huber, A novel connectionist system for unconstrained handwriting recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 31 (5) (2009) 855–868.

4. J. Puigcerver, Are multidimensional recurrent layers really necessary for handwritten text recognition?, in: International Conference on Document Analysis and Recognition, Vol. 01, 2017, pp. 67–72.

5. A. Graves, J. Schmidhuber, Offline handwriting recognition with multi-dimensional recurrent neural networks., in: NIPS, 2008, pp. 545–552.

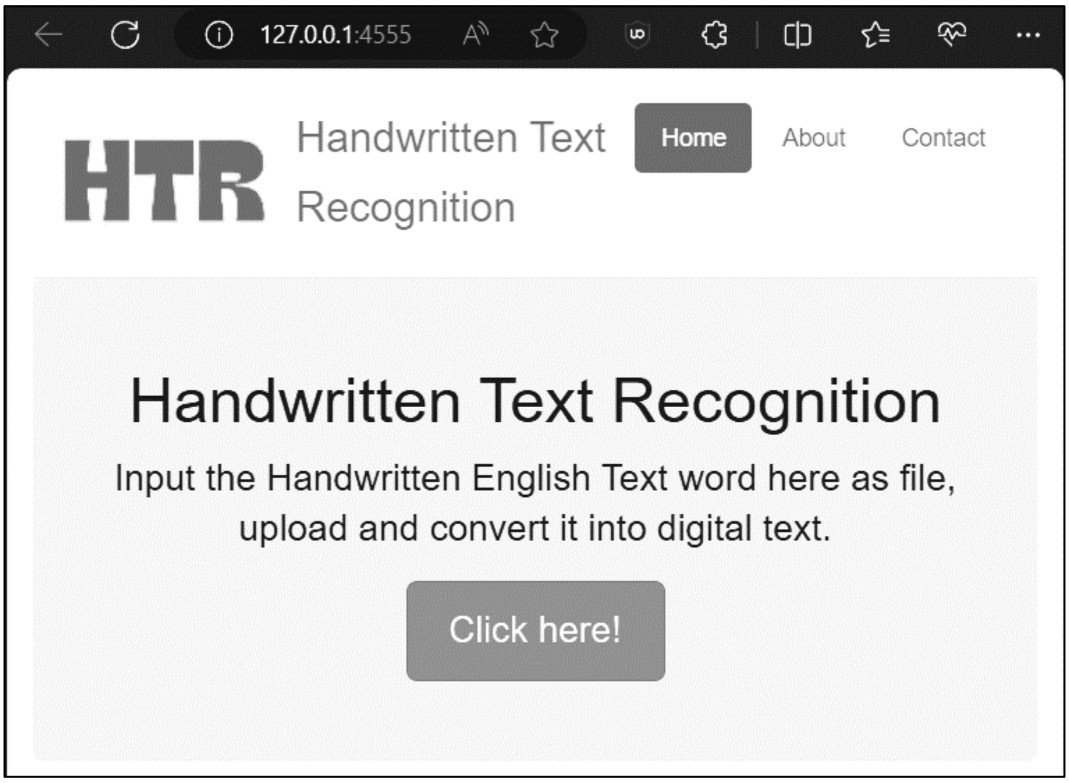
6. Sánchez Peiró, JA.; Romero, V.; Toselli, AH.; Villegas, M.; Vidal, E. (2019). A Set of Benchmarks for Handwritten Text Recognition on Historical Documents. Pattern Recognition. 94:122-134.

7. Graves, Alex, et al. "A novel connectionist system for unconstrained handwriting recognition." *IEEE transactions on pattern analysis and machine intelligence* 31.5 (2008): 855-868.

8. Scheidl, Harald, Stefan Fiel, and Robert Sablatnig. "Word beam search: A connectionist temporal classification decoding algorithm." *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2018.

9. O'shea, Keiron, and Ryan Nash. "An introduction to convolutional neural networks." *arXiv preprint arXiv:1511.08458* (2015).

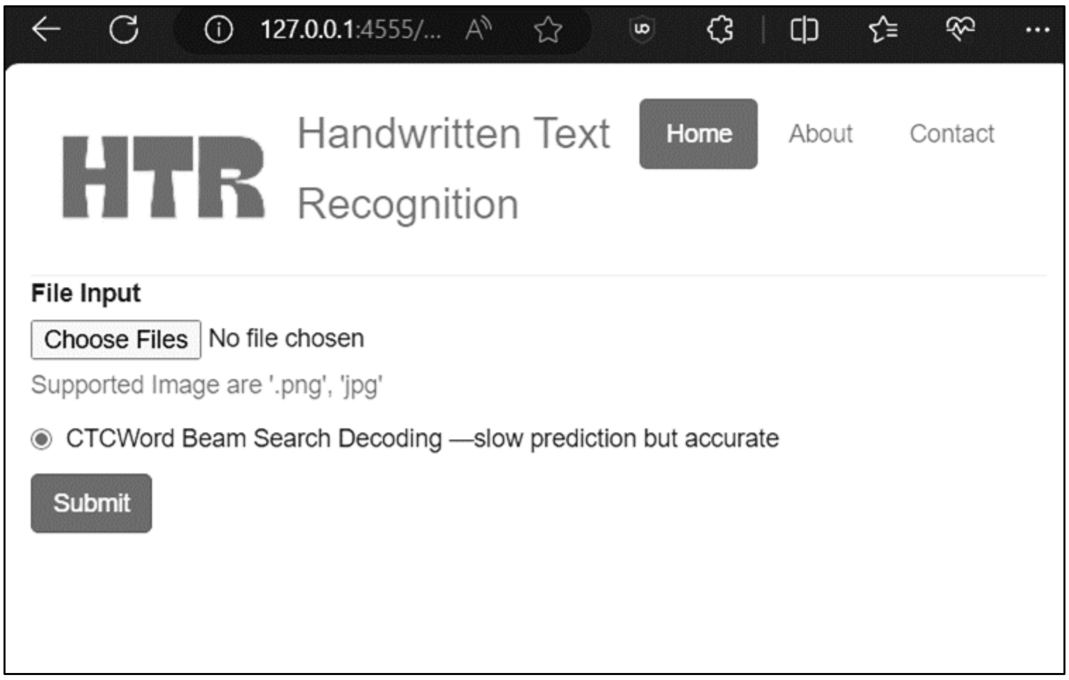
APPENDIX



Handwritten Text Recognition

Input the Handwritten English Text word here as file, upload and convert it into digital text.

Figure 10.1: Home Page of the webUI



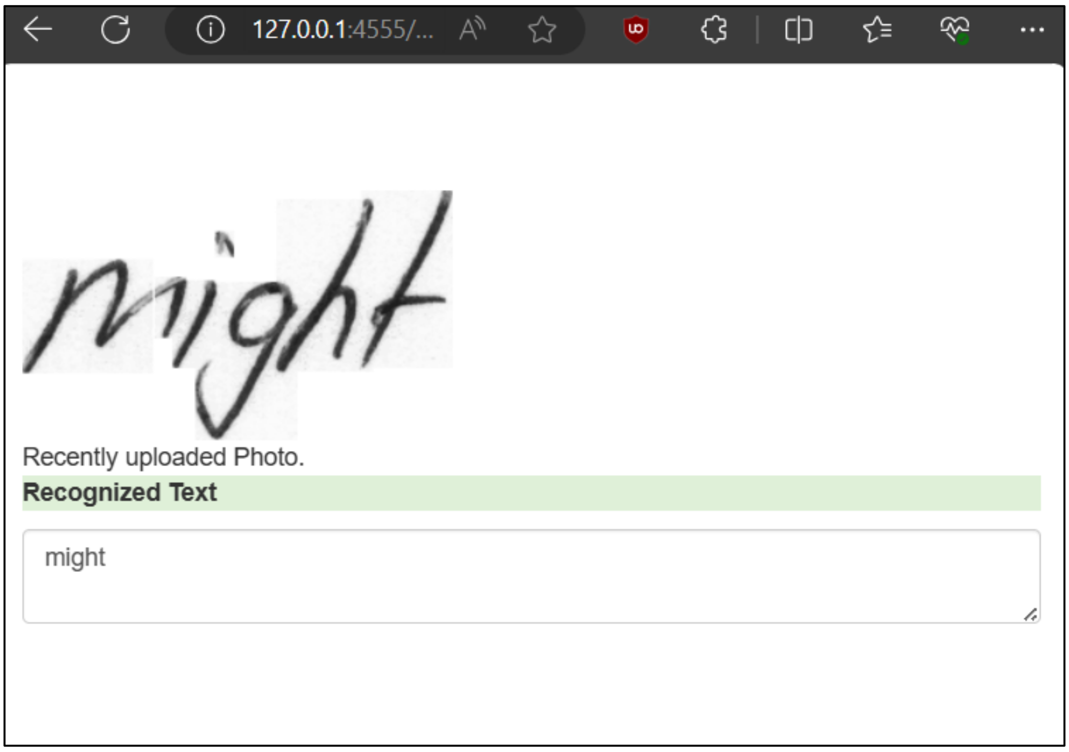
File Input

No file chosen

Supported Image are '.png', 'jpg'

(●) CTCWord Beam Search Decoding —slow prediction but accurate

Figure 10.2: Upload Page

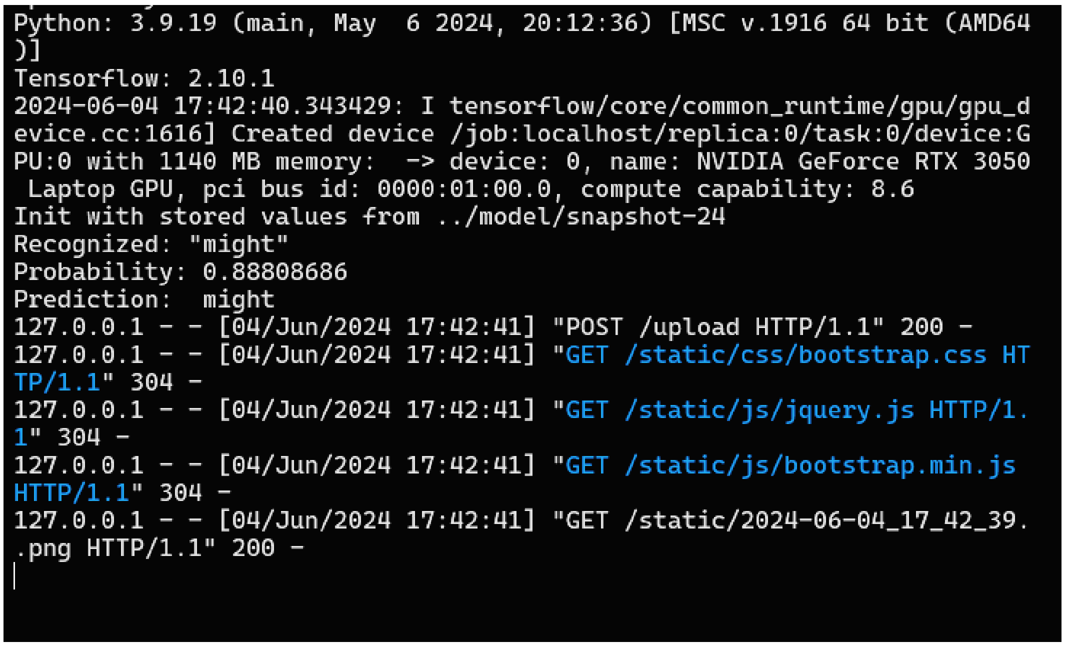


Recently uploaded Photo.

Recognized Text

might

Figure 10.3: Prediction 1



```
Python: 3.9.19 (main, May  6 2024, 20:12:36) [MSC v.1916 64 bit (AMD64
)]
Tensorflow: 2.10.1
2024-06-04 17:42:40.343429: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1616] Created device /job:localhost/replica:0/task:0/device:G
PU:0 with 1140 MB memory:  -> device: 0, name: NVIDIA GeForce RTX 3050
 Laptop GPU, pci bus id: 0000:01:00.0, compute capability: 8.6
Init with stored values from ../model/snapshot-24
Recognized: "might"
Probability: 0.88808686
Prediction:  might
127.0.0.1 - - [04/Jun/2024 17:42:41] "POST /upload HTTP/1.1" 200 -
127.0.0.1 - - [04/Jun/2024 17:42:41] "GET /static/css/bootstrap.css HT
TP/1.1" 304 -
127.0.0.1 - - [04/Jun/2024 17:42:41] "GET /static/js/jquery.js HTTP/1.
1" 304 -
127.0.0.1 - - [04/Jun/2024 17:42:41] "GET /static/js/bootstrap.min.js
HTTP/1.1" 304 -
127.0.0.1 - - [04/Jun/2024 17:42:41] "GET /static/2024-06-04_17_42_39.
.png HTTP/1.1" 200 -
```

Figure 10.4: Console Output

AYUSH NEGI

+91 9456142158, devayushnegi@gmail.com
Tehri Garhwal, Uttarakhand

EDUCATION

2020- 24	B.Tech (CSE)	: THDC Institute of Hydropower Engineering and Technology
2018 – 20	Intermediate (PCM)	: All Saints Convent School
2018	Higher Secondary	: Carmel School

PERSONAL INFORMATION

Date of Birth	:	3 August 2003
Work Experience	:	Fresher
Gender	:	Male
Major	:	Computer Science and Engineering (CSE)
Area of Interest	:	Optical Character Recognition, Digital Image Processing, Deep Learning

ABHISHEK RANA

+91 9068334516, abhishek2731175@gmail.com
Tehri Garhwal, Uttarakhand

EDUCATION

2020- 24	B.Tech (CSE)	:	THDC Institute of Hydropower Engineering and Technology
2018 – 20	Intermediate (PCM)	:	All Saints Convent School
2018	Higher Secondary	:	Carmel School

PERSONAL INFORMATION

Date of Birth	:	6 February 2003
Work Experience	:	Fresher
Gender	:	Male
Major	:	Computer Science and Engineering (CSE)
Area of Interest	:	Java, C, C++, Python, Data Science, Mathematics

SAURABH BHATT

+91 8057850721, saurabhbhattlearn@gmail.com
Tehri Garhwal, Uttarakhand

EDUCATION

2020- 24	B.Tech (CSE)	: THDC Institute of Hydropower Engineering and Technology
2018 – 20	Intermediate (PCM)	: All Saints Convent School
2018	Higher Secondary	: Carmel School

PERSONAL INFORMATION

Date of Birth	:	11 September 2003
Work Experience	:	Fresher
Gender	:	Male
Major	:	Computer Science and Engineering (CSE)
Area of Interest	:	Java, C, C++, Python, Data Science, Mathematics