

**HANDWRITTEN TEXT RECOGNITION MODEL**

**A PROJECT REPORT**

*Submitted by*

**ABHISHEIK KTR (160701005)**

**AJAY KRISHNA RV (160701014)**

***in partial fulfilment for the award of the degree***

***of***

**BACHELOR OF ENGINEERING**

**IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

**SRI VENKATESWARA COLLEGE OF ENGINEERING**

**ANNA UNIVERSITY, CHENNAI**

**APRIL 2020**

**ANNA UNIVERSITY, CHENNAI**

**BONAFIDE CERTIFICATE**

Certified that this project report **“HANDWRITTEN TEXT RECOGNITION MODEL”** is the bonafide work of **“ABHISHEIK KTR (160701005), AJAY KRISHNA (160701014)”** who carried out the project work under my supervision.

**SIGNATURE SIGNATURE**

Dr. S. Muthukumar Mrs. S. Kalyani

**HEAD OF THE DEPARTMENT SUPERVISOR**

Professor Assistant Professor

Electronics & Communication Electronics & Communication Engineering Engineering

Sri Venkateswara College of Sri Venkateswara College of

Engineering Engineering

Pennalur, Pennalur,

Sriperumbudur – 602 117 Sriperumbudur – 602 117

Submitted to the Project viva-voce Examination held on …………………………

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**ACKNOWLEDGEMENT**

We would like to thank our principal, **Prof. Dr. S. GANESH VAIDYANATHAN**, for providing us the necessary infrastructure and constant support to complete our project.

We thank **Dr. S. MUTHUKUMAR**, Head of the Department, Department of Electronics and Communication Engineering for his continuous support in the completion of the project.

We are extremely grateful to, **Mrs. S. KALYANI** Assistant Professor, Department of Electronics and Communication Engineering, for guiding us throughout the project work. We sincerely express our gratitude to her

We wish to thank theproject coordinator, **Ms. K. S. SUBHASHINI**  for encouraging us throughout our project.

We also extend our sincere thanks to all teaching and non-teaching staff of Department of Electronics and Communication Engineering for their valuable support.

**ABSTRACT**

Many studies on (Offline) Handwritten Text Recognition (HTR) systems have focused on building state-of-the-art models for line recognition on small corpora. However, adding HTR capability to a large scale multilingual OCR system poses new challenges. This paper addresses three problems in building such systems: data, efficiency, and integration. Firstly, one of the biggest challenges is obtaining sufficient amounts of high-quality training data. We address the problem by using online handwriting data collected for a large scale production online handwriting recognition system. We describe our image data generation pipeline and study how online data can be used to build HTR models. We show that the data improve the models significantly under the condition where only a small number of real images is available, which is usually the case for HTR models. It enables us to support a new script at substantially lower cost. Secondly, we propose a line recognition model based on neural networks without recurrent connections. The model achieves a comparable accuracy with LSTM-based models while allowing for better parallelism in training and inference. Finally, we present a simple way to integrate HTR models into an OCR System. These constitute a solution to bring HTR capability intoa large scale OCR system.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER** | **TITLE** | **PAGE NO.** |
|  | **ACKNOWLEDGEMENT** | **iii** |
|  | **ABSTRACT** | **iv** |
|  | **LIST OF FIGURES** | **vi** |
|  | **LIST OF ABBREVIATIONS** | **viii** |
|  | **LIST OF TABLES** | **xi** |
|  |  |  |
| **1** | **INTRODUCTION** | **12** |
|  | **1.1 Optical Character Recognition(OCR)** | **12** |
|  | **1.1.1 Handwritten Text Recognition(HTR)** | **14** |
|  | **1.2 Why is it Needed** | **17** |
|  | **1.2.1 Digitization of Old Manuscripts** | **17** |
|  | **1.2.2 Aid for Visually Impaired** | **19** |
|  | **1.3 History of Practices** | **21** |
|  | **1.3.1 Old Methods**  **1.3.2 Modern Technology**  **1.3.3** | **21**  **22**  **25** |
|  | **1.4 Research And Advancements**  **1.4.1 Conferences**  **1.4.1.1 ICFHR**  **1.4.1.2 ICDAR** | **27**  **28**  **29**  **30** |
| **2** | **LITRATURE SURVEY** | **31** |
| **3** | **PROPOSED SYSTEM** | **32** |
|  | **3.1 Model Overview** | **32** |
|  | **3.2 Library Modules Used (Python)**  **3.2.1 Tensorflow**  **3.2.2 Keras**  **3.3 Operations Done**  **3.3.1 CNN**  **3.3.2 RNN**  **3.3.3 CTC**  **3.3.4 LSTM**  **3.4 Database**  **3.4.1 Kaggle - MNIST**  **3.4.2 IAM**  **3.5 Advancements Over Other Methods**  **3.6 Modifications**  **3.7 Training**  **3.8 Testing**  **3.9 Improving**  **3.10 Output** | **39**  **39**  **42**  **44**  **44**  **46**  **46**  **47**  **49**  **49**  **51**  **53**  **54**  **54**  **54**  **54**  **54** |
| **4.** | **IMPLEMENTATION AND RESULT** | **55** |
|  | **4.1 Software Requirements**  **4.1.1 Python**  **4.1.1.1 Features**  **4.1.1.2 Typing**  **4.1.1.3 Libraries**  **4.1.2 Jupyter Notebook**  **4.1.3 Spyder** | **55**  **55**  **56**  **56**  **57**  **57** |
|  | **4.2 Installation and Setup**  **4.2.1 Jupyter Notebook Installation**  **4.2.2 Spyder Installation**  **4.3 Integrated Word Beam Search Decoding**  **4.4 Training the Model**  **4.5 Result** | **58**  **58**  **58**  **61**  **61**  **61** |
| **5** | **CONCLUSION AND FUTUREWORK** | **66** |
|  | **5.1 Conclusion** | **66** |
|  | **5.2 Futurework** | **67** |
|  |  |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO** | **DESCRIPTION OF FIGURE** | **PAGE NO.** |
| **1.1** | **Optical Character Recognition** | **12** |
| **1.2** | **Handwritten Text Recognition** | **14** |
| **1.3** | **Digitizing An Old Book** | **17** |
| **1.4** | **Basic TTS Architecture** | **19** |
| **1.5** | **Optophone For OCR** | **22** |
| **1.6** | **Augmentation through Machine Learning** | **25** |
| **1.7** | **ICFHR 2019** | **29** |
| **3.1** | **Conversion of Handwritten Text To Digital Text** | **32** |
| **3.2** | **Model Overview** | **32** |
| **3.3** | **Image from the dataset** | **35** |
| **3.4** | **256 feature per time-step are computed** | **35** |
| **3.5** | **RNN output matrix** | **37** |
| **3.6** | **Basic CNN layout** | **45** |
| **3.7** | **CTC Architecture** | **47** |
| **3.8** | **MNIST Dataset** | **51** |
| **3.9** | **IAM Dataset** | **52** |
| **4.1** | **Command To be Typed in python shell** | **59** |
| **4.2** | **Command to be Typed to create a new notebook** | **59** |
| **4.3** | **New notebook Webpage** | **60** |
| **4.4** | **New notebook** | **60** |
| **4.5** | **Spyder Interface after installation** | **61** |
| **4.6** | **Word Search** | **62** |
| **4.7** | **Training of the model** | **64** |
| **4.8** | **Scanned word** | **65** |
| **4.9** | **Output of the scanned word** | **65** |
| **4.10** | **Code execution** |  |

**LIST OF ABBREVIATIONS**

|  |  |
| --- | --- |
| **API** | Application programming interface |
| **ASCII** | American Standard Code for Information Interchange |
| **CNN** | Convolutional Neural Network |
| **CPU** | Central Processing Unit |
| **CTC** | Connectionist Temporal Classification |
| **HTR** | Handwritten Text Recognition |
| **HWR** | Handwritten Word Recognition |
| **IAM** | Department of Computer Science and Applied Mathematics |
| **IAPR** | International Association for Pattern Recognition |
| **ICDAR** | International Conference On Document Analysis and Recognition |
| **ICFHR** | International Conference on Frontiers in Handwriting Recognition |
| **ICR** | Intelligent Character Recognition |
| **IWFHR** | International Workshop On Frontiers Of Handwriting Recognition |
| **IWR** | Intelligent Word Recognition |
| **OCR** | Optical Character Recognition |
| **PDF** | Portable Document Format |
| **RELU** | Rectified Linear Unit |
| **RNN** | Recurrent Neural Networks |
| **TF** | TensorFlow |
| **TPU** | Tensor Processing Unit |
| **TTS** | Text-To-Speech |

**CHAPTER 1**

# INTRODUCTION

# 1.1 OPTICAL CHARACTER RECOGNITION (OCR)

# Optical character recognition or optical character reader (OCR) is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example from a television broadcast).

# 

Fig 1.1 Optical Character Recognition

# Widely used as a form of data entry from printed paper data records – whether passport documents, invoices, bank statements, computerized receipts, business cards, mail, printouts of static-data, or any suitable documentation – it is a common method of digitizing printed texts so that they can be electronically edited, searched, stored more compactly, displayed on-line, and used in machine processes such as cognitive computing, machine translation, (extracted) text-to-speech, key data and text mining. OCR is a field of research in pattern recognition, artificial intelligence and computer vision.Early versions needed to be trained with images of each character, and worked on one font at a time. Advanced systems capable of producing a high degree of recognition accuracy for most fonts are now common, and with support for a variety of digital image file format inputs. Some systems are capable of reproducing formatted output that closely approximates the original page including images, columns, and other non-textual components.

# Optical character recognition (OCR) – targets typewritten text, one glyph or character at a time.

# Types of OCR

# Optical word recognition – targets typewritten text, one word at a time (for languages that use a space as a word divider). (Usually just called "OCR".)

# Intelligent character recognition (ICR) – also targets handwritten printscript or cursive text one glyph or character at a time, usually involving machine learning.

# Intelligent word recognition (IWR) – also targets handwritten printscript or cursive text, one word at a time. This is especially useful for languages where glyphs are not separated in cursive script.

# 

# Our Project is Based on ICR (Intelligent Character Recognition) or IWR Intelligent Write Recognition) Collectively called Handwritten Text Recognition (HTR).

# 1.1.1 Handwritten Text Recognition (HTR)

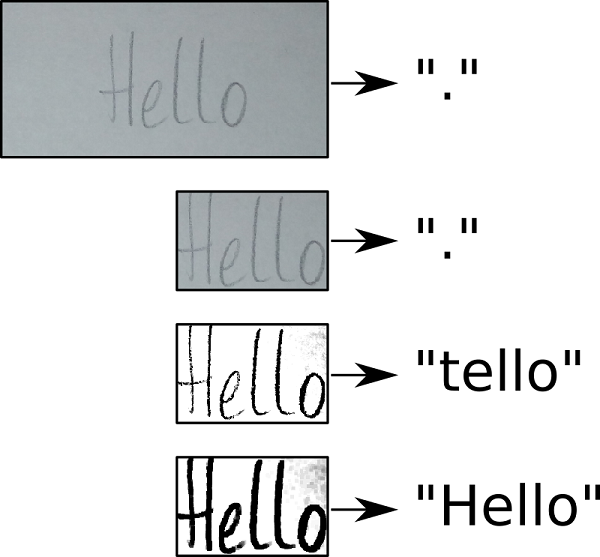


Fig 1.2 Recognizing “Hello” text from Writing Image

Handwriting recognition (HWR), also known as Handwritten Text Recognition (HTR), is the ability of a computer to receive and interpret intelligible handwritten input from sources such as paper documents, photographs, touch-screens and other devices. The image of the written text may be sensed "off line" from a piece of paper by optical scanning (optical character recognition) or intelligent word recognition. Alternatively, the movements of the pen tip may be sensed "on line", for example by a pen-based computer screen surface, a generally easier task as there are more clues available. A handwriting recognition system handles formatting, performs correct segmentation into characters, and finds the most plausible words.

**Off-line recognition**

Off-line handwriting recognition involves the automatic conversion of text in an image into letter codes which are usable within computer and text-processing applications. The data obtained by this form is regarded as a static representation of handwriting. Off-line handwriting recognition is comparatively difficult, as different people have different handwriting styles. And, as of today, OCR engines are primarily focused on machine printed text and ICR for hand "printed" (written in capital letters) text.

**Traditional techniques**

**Character extraction**

Off-line character recognition often involves scanning a form or document written sometime in the past. This means the individual characters contained in the scanned image will need to be extracted. Tools exist that are capable of performing this step. However, there are several common imperfections in this step. The most common is when characters that are connected are returned as a single sub-image containing both characters. This causes a major problem in the recognition stage. Yet many algorithms are available that reduce the risk of connected characters.

**Character recognition**

After the extraction of individual characters occurs, a recognition engine is used to identify the corresponding computer character. Several different recognition techniques are currently available.

**Feature extraction**

Feature extraction works in a similar fashion to neural network recognizers. However, programmers must manually determine the properties they feel are important. This approach gives the recognizer more control over the properties used in identification. Yet any system using this approach requires substantially more development time than a neural network because the properties are not learned automatically.

**Modern techniques**

Where traditional techniques focus on segmenting individual characters for recognition, modern techniques focus on recognizing all the characters in a segmented line of text. Particularly they focus on machine learning techniques which are able to learn visual features, avoiding the limiting feature engineering previously used. State-of-the-art methods use convolutional networks to extract visual features over several overlapping windows of a text line image which an RNN uses to produce character probabilities.

**1.2 WHY IS IT NEEDED**

**1.2.1 Digitization Of Books.**



Fig 1.3 Digitizing an Old Book

Book scanning or book digitization (also: magazine scanning or magazine digitization) is the process of converting physical books and magazines into digital media such as images, electronic text, or electronic books (e-books) by using an image scanner.

Digital books can be easily distributed, reproduced, and read on-screen. Common file formats are DjVu, Portable Document Format (PDF), and Tagged Image File Format (TIFF). To convert the raw images optical character recognition (OCR) is used to turn book pages into a digital text format like ASCII or other similar format, which reduces the file size and allows the text to be reformatted, searched, or processed by other applications.

Image scanners may be manual or automated. In an ordinary commercial image scanner, the book is placed on a flat glass plate (or platen), and a light and optical array moves across the book underneath the glass. In manual book scanners, the glass plate extends to the edge of the scanner, making it easier to line up the book's spine. Other book scanners place the book face up in a v-shaped frame, and photograph the pages from above. Pages may be turned by hand or by automated paper transport devices. Glass or plastic sheets are usually pressed against the page to flatten it.

After scanning, software adjusts the document images by lining it up, cropping it, picture-editing it, and converting it to text and final e-book form. Human proofreaders usually check the output for errors.

**Non-destructive scanning**

An example of a DIY non-destructive book scanner/digitizer, with the book downwards design, allowing gravity to flatten pages

Software driven machines and robots have been developed to scan books without the need of unbinding them in order to preserve both the contents of the document and create a digital image archive of its current state. This recent trend has been due in part to ever improving imaging technologies that allow a high quality digital archive image to be captured with little or no damage to a rare or fragile book in a reasonably short period of time.

**1.2.2 Text-to-Speech**

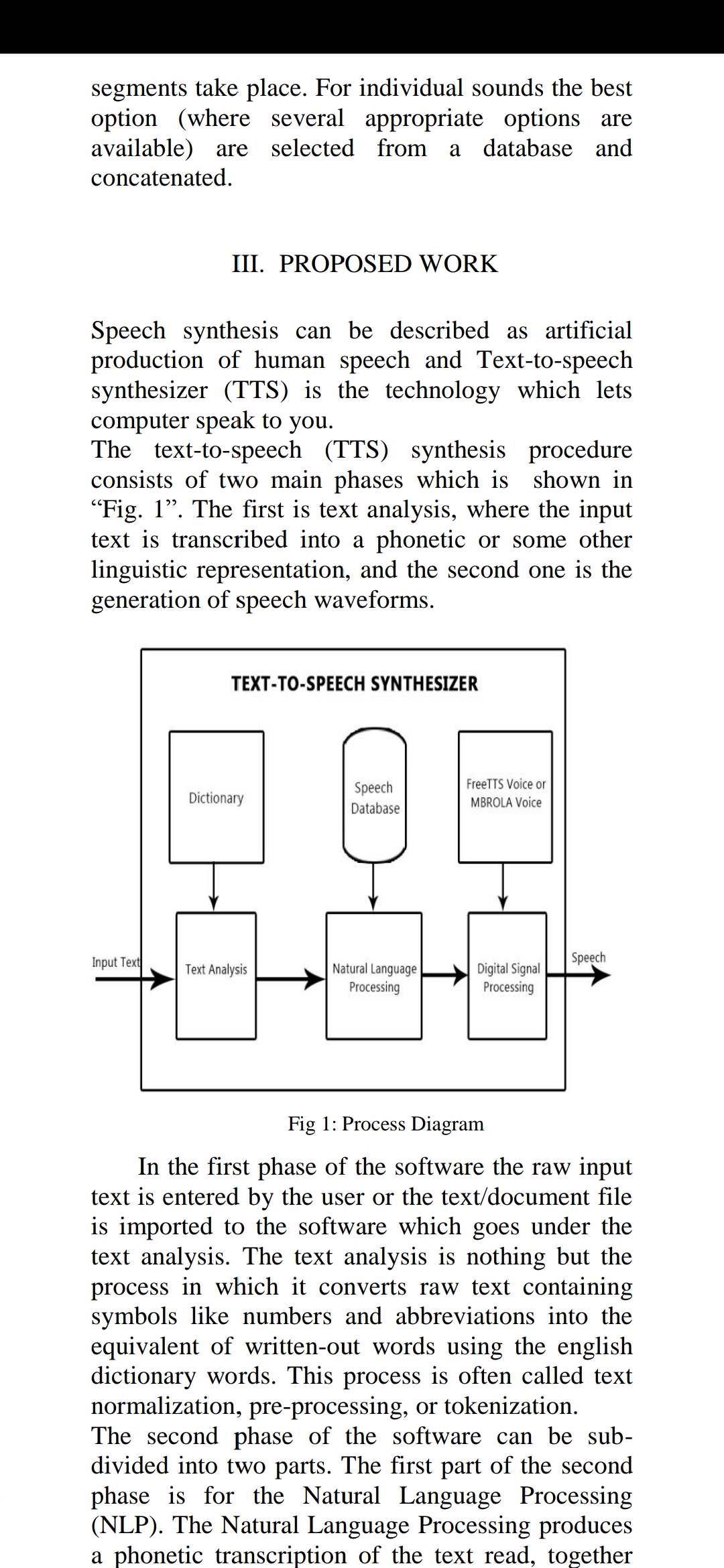


Fig 1.4 Basic TTS Architecture

Speech synthesis is the artificial production of human speech. A computer system used for this purpose is called a speech computer or speech synthesizer, and can be implemented in software or hardware products.

A text-to-speech (TTS) system converts normal language text into speech; other systems render symbolic linguistic representations like phonetic transcriptions into speech.

Synthesized speech can be created by concatenating pieces of recorded speech that are stored in a database. Systems differ in the size of the stored speech units; a system that stores phones or diphones provides the largest output range, but may lack clarity. For specific usage domains, the storage of entire words or sentences allows for high-quality output. Alternatively, a synthesizer can incorporate a model of the vocal tract and other human voice characteristics to create a completely "synthetic" voice output.

The quality of a speech synthesizer is judged by its similarity to the human voice and by its ability to be understood clearly. An intelligible text-to-speech program allows people with visual impairments or reading disabilities to listen to written words on a home computer. Many computer operating systems have included speech synthesizers since the early 1990s.

A text-to-speech system (or "engine") is composed of two parts: a front-end and a back-end. The front-end has two major tasks. First, it converts raw text containing symbols like numbers and abbreviations into the equivalent of written-out words.

This process is often called text normalization, pre-processing, or tokenization. The front-end then assigns phonetic transcriptions to each word, and divides and marks the text into prosodic units, like phrases, clauses, and sentences.

The process of assigning phonetic transcriptions to words is called text-to-phoneme or grapheme-to-phoneme conversion. Phonetic transcriptions and prosody information together make up the symbolic linguistic representation that is output by the front-end. The back-end—often referred to as the synthesizer—then converts the symbolic linguistic representation into sound. In certain systems, this part includes the computation of the target prosody (pitch contour, phoneme durations), which is then imposed on the output speech.

**1.3 HISTORY OF OCR**

**1.3.1 Old Methods(Optophone)**

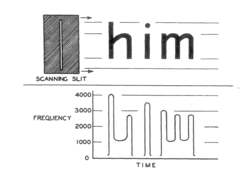
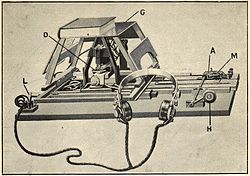
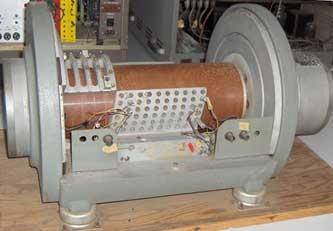


Fig 1.5 Optophone For OCR

The optophone is a device, used by the blind, that scans text and generates time-varying chords of tones to identify letters. It is one of the earliest known applications of sonification. Dr. Edmund Fournier d'Albe of Birmingham University invented the optophone in 1913, which used selenium photosensors to detect black print and convert it into an audible output which could be interpreted by a blind person. The Glasgow company, Barr and Stroud, participated in improving the resolution and usability of the instrument.

**1.3.2 After Computers**

1870- American inventor Charles R. Carey invents the retina scanner, an image transmission system using a mosaic of photocells, considered the first OCR invention in the world.



1885- Image scanner Paul Nipkow invents the Nipkow disk, an image scanning device that later will be a major breakthrough both for modern television and reading machines.

1900- Russian scientist Tyurin envisions the first OCR machine to serve as an aid to the visually handicapped, but never manages to develop it.

1912- Text-to-speech Edmund Fournier d'Albe develops the Optophone, a handheld scanner that when moved across a printed page, produces tones that corresponded to specific letters or characters, so as to be interpreted by a blind person.

1916- American engineer John B. Flowers patents the "One-Eyed Machine Stenographer", a machine capable of reading and typing a script. It worked by superimposing all the letters to find a point that marked each of them.

1921- Text-to-tactile sensations Italian professor Ciro Codelupi envisions the "Reading machine for the blind", capable of transforming luminous sensations into tactile sensations.

1929- Austrian engineer Gustav Tauschek creates the first OCR device called the "Reading Machine", with a photo-sensor pointing light on words when they corresponded to a content template in its memory.

1931- Text-to-telegraph Israeli physicist and inventor Emanuel Goldberg is granted a patent for his "Statistical machine" (US Patent 1838389), which was later acquired by IBM. It was described as capable of reading characters and converting them into standard telegraph code.

1938- MIT professor Vannebar Bush develops the Microfilm Rapid Selector, a similar but simpler Goldberg' statistical machine, and 40 times faster.

1949- Engineers working on the Radio Corporation of America start a project to help the blind and the U.S. Department of Veterans Affairs, using the first text-to-speech techniques.

1951- Text & Morse-to-speech American cryptoanalyst David H. Shepard and Harvey Cook Jr. build "Gismo", a machine able to read aloud letter by letter and interpret Morse code (U.S. Patent 2,663,758).

1952- The Intelligent Machines Research Corporation isfounded by D. Shepard and William Lawless Jr, to commercialise Gismo (later renamed to "Analysing Reader")

1954- American magazine Reader's Digest becomes the first business to install an OCR reader, used to convert typewritten sales reports into punched cards.

1962- Portability Stanford professor John Linvill develops the Optacon, the first portable reading device for the blind.

1965- Reader's Digest expands its OCR use to digitise serial numbers of coupons. with a RCA 501 computer.

1966- Handwriting scanner The IBM Rochester lab develops the IBM 1287, the first scanner capable of reading any handwritten numbers.

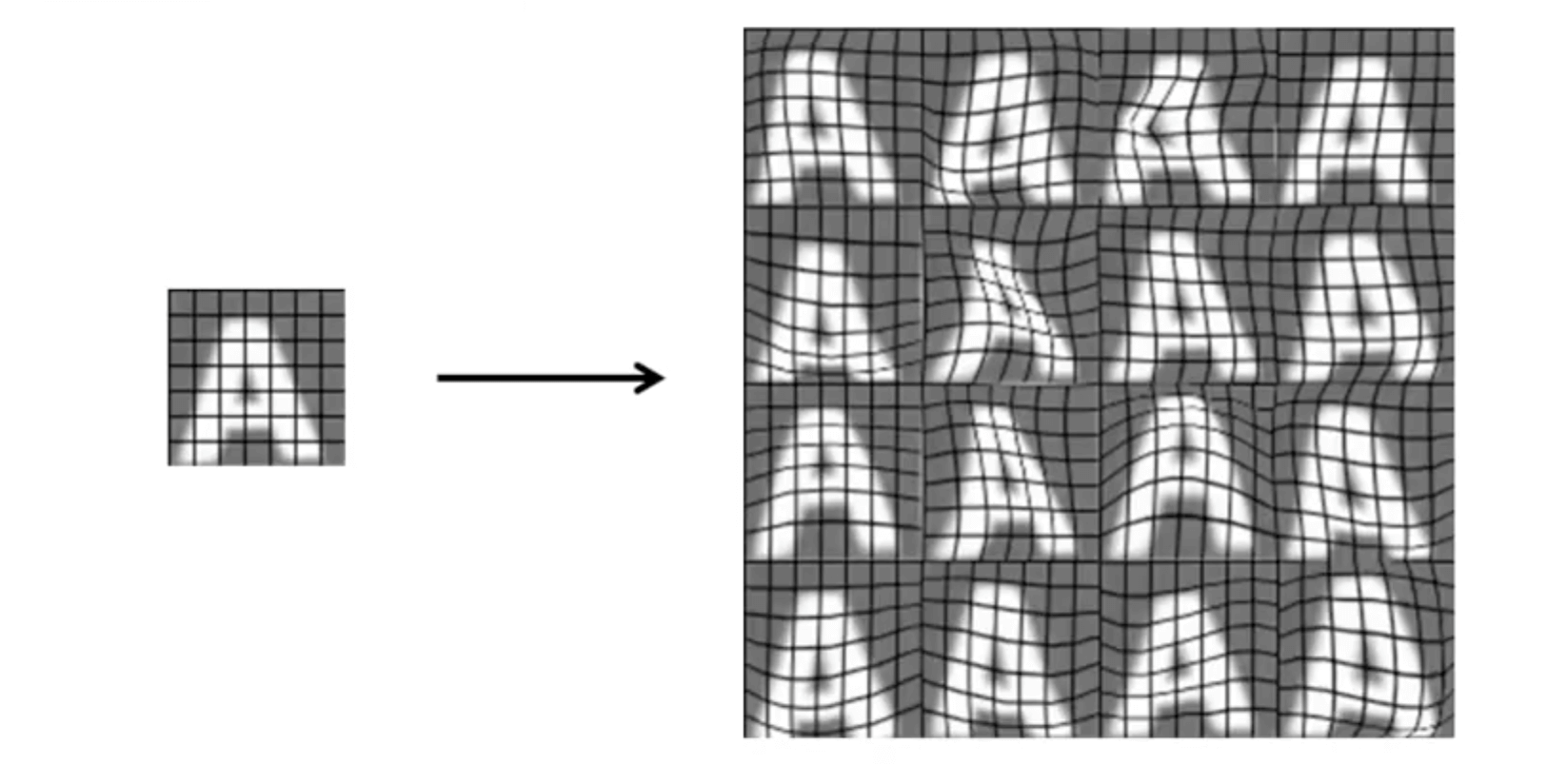


Fig 1.6 Augmentation through Machine Learning

**1.3.3 After Machine Learning**

1992- The first program that recognizes Cyrillic is invented by Russian company OKRUS.

2000- OCR technology is made available online as a service (WebOCR), in a cloud computing environment, as well as in mobile applications like real-time translation of foreign-language signs on a smartphone.

2005- The free cross-platform OCR engine Tesseract is published by Hewlett Packard and the University of Nevada, Las Vegas.

2008- Adobe Acrobat starts including support for OCR on any PDF file

2011- Word-frequency lookup Google Ngram Viewer is developed to chart frequencies of words on any source printed from 1950 to 2008.

2013- The MNIST database is created to train machine learning models in pattern recognition.

2015- Open access Google offers OCR tools to scan any Google Drive files in over 200 languages for free.

**1.4 Research and Advancements**

Optical Character Recognition, or OCR, is a technology that enables you to convert different types of documents, such as scanned paper documents, PDF files or images captured by a digital camera into editable and searchable data.

Imagine you’ve got a paper document - for example, magazine article, brochure, or PDF contract your partner sent to you by email. Obviously, a scanner is not enough to make this information available for editing, say in Microsoft Word. All a scanner can do is create an image or a snapshot of the document that is nothing more than a collection of black and white or colour dots, known as a raster image. In order to extract and repurpose data from scanned documents, camera images or image-only PDFs, you need an OCR software that would single out letters on the image, put them into words and then - words into sentences, thus enabling you to access and edit the content of the original document.

an image can be accomplished by digitally scanning an existing photograph (2D) or by using a video image to acquire a live picture of a subject (3D).

**1.4.1 Conferences**

Handwriting Recognition has an active community of academics studying it. The biggest conferences for handwriting recognition are the International Conference on Frontiers in Handwriting Recognition (ICFHR), held in even-numbered years, and the International Conference on Document Analysis and Recognition (ICDAR), held in odd-numbered years. Both of these conferences are endorsed by the IEEE and IAPR. Active areas of research include:

**Online Recognition**

**Offline Recognition**

**Signature Verification**

**Postal-Address Interpretation**

**Bank-Check Processing**

**Writer Recognition**

Once it detects a face, the system determines the head's position, size and pose. As stated earlier, the subject has the potential to be recognized up to 90 degrees, while with 2D, the head must be turned at least 35 degrees toward the camera.

**1.4.1.1 ICFHR**



Fig 1.7 ICFHR 2019

The International Conference on Frontiers of Handwriting Recognition (ICFHR), formerly called International Workshop on Frontiers of Handwriting Recognition (IWFHR), is the most important scientific venue in the field of handwriting recognition. The aim of this conference is to bring together international experts from academia and industry to share their experiences and to promote research and development in all aspects of handwriting recognition and applications.

system then measures the curves of the face on a sub-millimetre (or microwave) scale and creates a template.

**1.4.1.2 ICDAR**

is a very successful and flagship conference series, which is the biggest and premier international gathering for researchers, scientist and practitioners in the document analysis community. The conference is endorsed by IAPR-TC 10/11 and it was established nearly three decades ago.

**CHAPTER 2**

**LITERARY SURVEY**

**1. Harald Scheidl - A minimalistic neural network implementation which can be trained on the CPU**

The NN consists of 5 CNN and 2 RNN layers and outputs a character-probability matrix. This matrix is either used for CTC loss calculation or for CTC decoding. An implementation using TF is provided and some important parts of the code were presented.

**2. Harald Scheidl, Stefan Fiel, and Robert Sablatnig: Word Beam Search:**

Beam search allows arbitrary character strings, which is needed to decode numbers and punctuation mark. Token passing limits its output to dictionary words, which avoids spelling mistakes. Of course, the dictionary must contain all words which have to be recognized. A combination of both properties: when we see a word, we only allow words from a dictionary, but in any other case, we allow arbitrary character strings.

**3.Build a Handwritten Text Recognition System using Tensor Flow.**

**4.Marti - The IAM-database: an English sentence database for offline handwriting recognition.**

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.

**CHAPTER 3.**

**PROPOSED SYSTEM**

**3.1 Model Overview**

Offline Handwritten Text Recognition (HTR) systems transcribe text contained in scanned images into digital text, an example is shown in Fig. 1. We will build a Neural Network (NN) which is trained on word-images from the IAM dataset. As the input layer (and therefore also all the other layers) can be kept small for word-images, NN-training is feasible on the CPU (of course, a GPU would be better). This implementation is the bare minimum that is needed for HTR using TF.

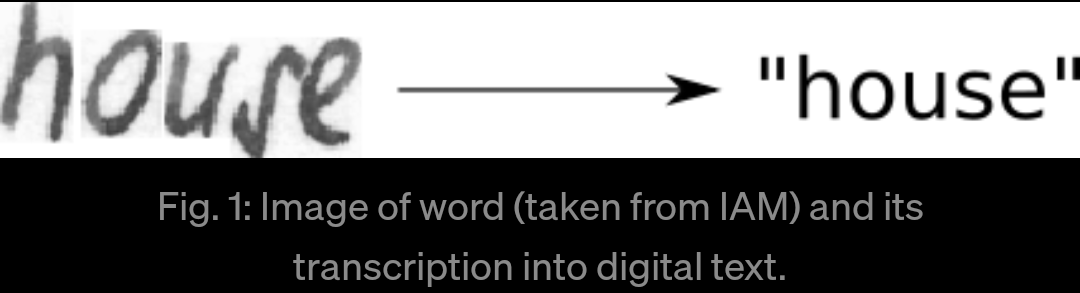


Fig. 3.1: Image of word (taken from IAM) and its transcription into digital text.

**Model**

We use a NN for our task. It consists of convolutional NN (CNN) layers, recurrent NN (RNN) layers and a final Connectionist Temporal Classification (CTC) layer. Fig. 3.2 shows an overview of our HTR system.

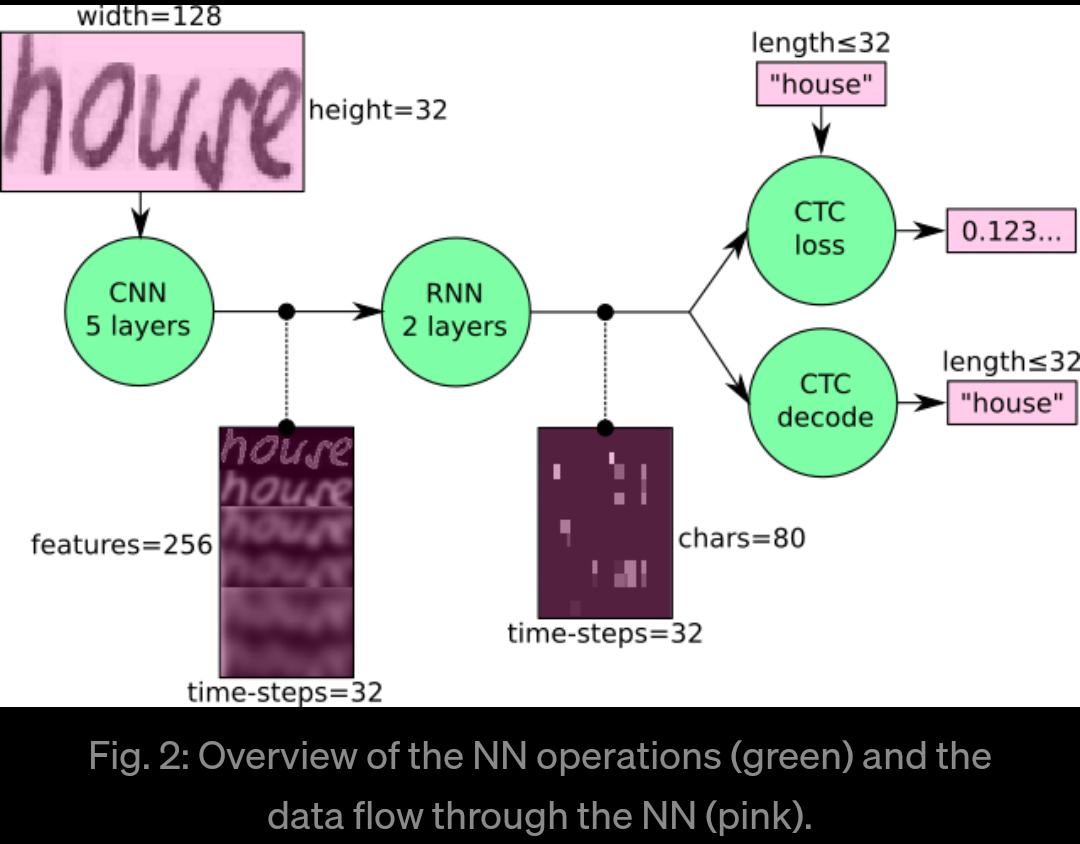
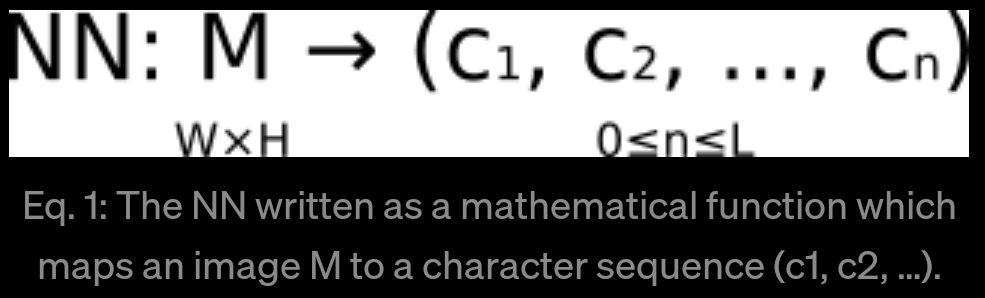


Fig. 3.2: Overview of the NN operations (green) and the data flow through the NN (pink).

We can also view the NN in a more formal way as a function (see Eq. 1) which maps an image (or matrix) M of size W×H to a character sequence (c1, c2, …) with a length between 0 and L. As you can see, the text is recognized on character-level, therefore words or texts not contained in the training data can be recognized too (as long as the individual characters get correctly classified).



Eq. 3.1: The NN written as a mathematical function which maps an image M to a character sequence (c1, c2, …).

Operations

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), 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.

Data

Input: it 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. 3. 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.

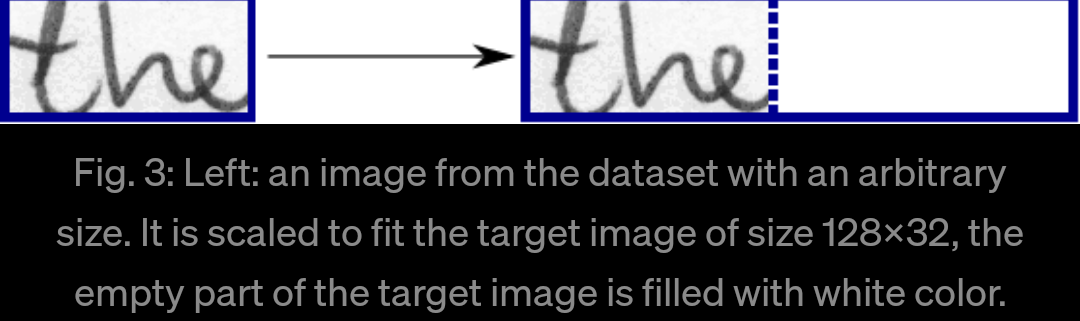


Fig. 3.3: Left: an image from the dataset with an arbitrary size. It is scaled to fit the target image of size 128×32, the empty part of the target image is filled with white color.

CNN output: Fig. 3.4 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).

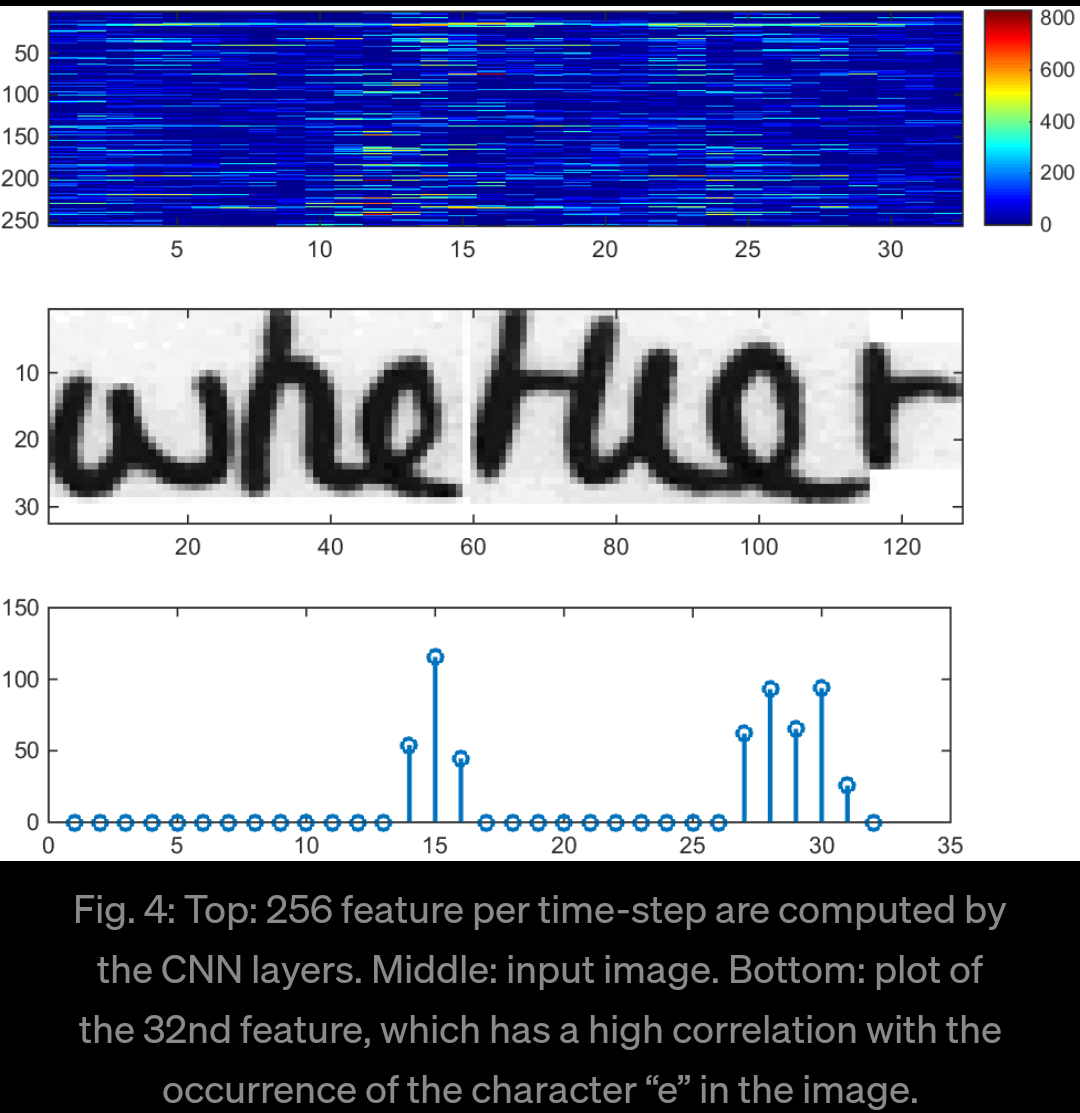


Fig. 3.4: Top: 256 feature per time-step are computed by the CNN layers. Middle: input image. Bottom: plot of the 32nd feature, which has a high correlation with the occurrence of the character “e” in the image.

RNN output: **Fig. 3.5** 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:;?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz”. 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 pa

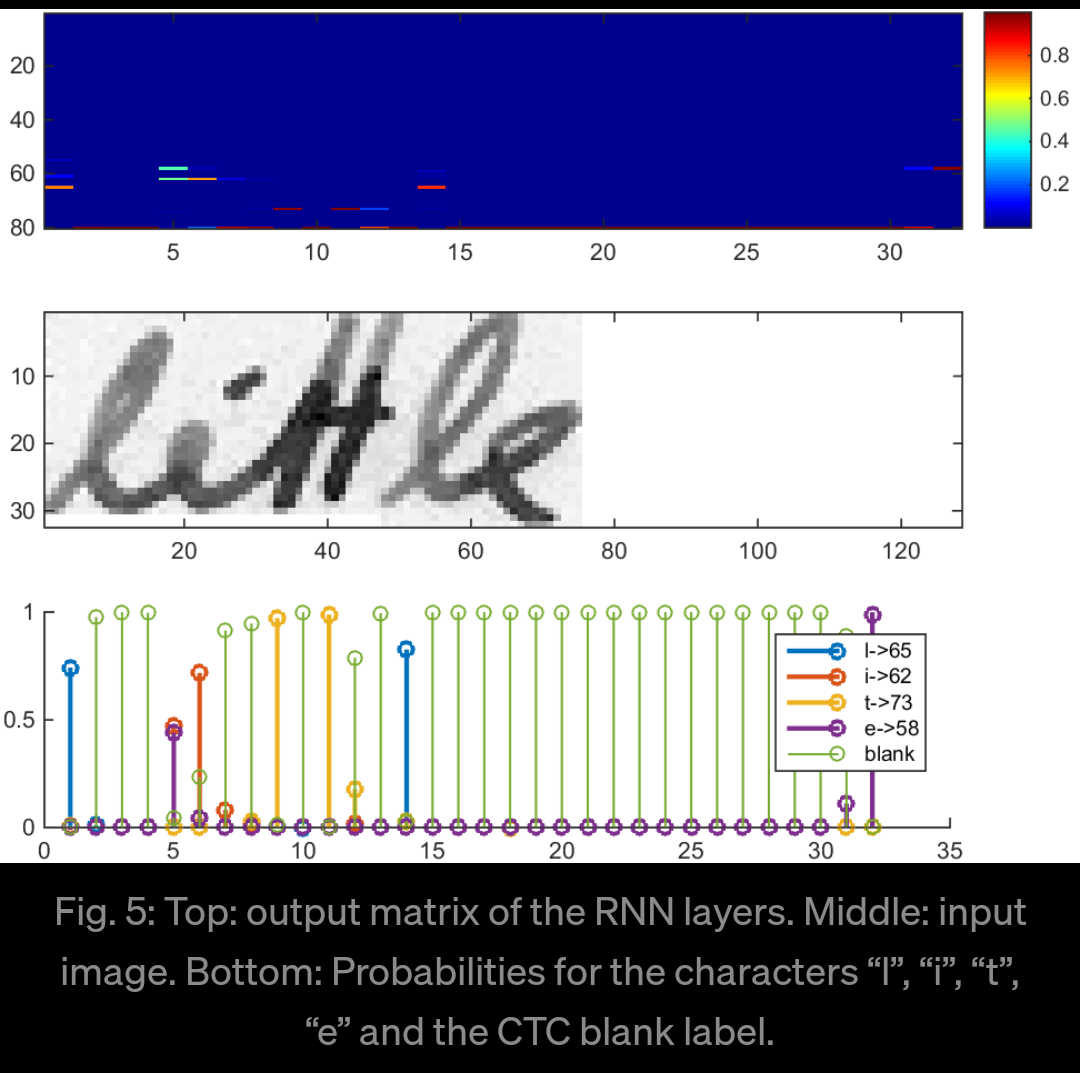


Fig. 3.5: Top: output matrix of the RNN layers. Middle: input image. Bottom: Probabilities for the characters “l”, “i”, “t”, “e” and the CTC blank label.

, then we throw away repeated characters and finally all blanks: “l---ii--t-t--l-…-e” → “l---i--t-t--l-…-e” → “little”.

**Implementation using TF**

The implementation consists of 4 modules:

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

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

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

main.py: puts all previously mentioned modules together

We only look at Model.py, as the other source files are concerned with basic file IO (DataLoader.py) and image processing (SamplePreprocessor.py).

**CNN**

For each CNN layer, create a kernel of size k×k to be used in the convolution operation.

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

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

**RNN**

Create and stack two RNN layers with 256 units each.

Then, create a bidirectional RNN 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.

**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.

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

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 RMSProp.

**Improving the model**

In case to feed complete text-lines instead of word-images, you have to increase the input size of the NN.

If you want to improve the recognition accuracy, you can follow one of these hints:

Data augmentation: increase dataset-size by applying further (random) transformations to the input images

Remove cursive writing style in the input images (see DeslantImg)

Increase input size (if input of NN is large enough, complete text-lines can be used)

Add more CNN layers

Replace LSTM by 2D-LSTM

Decoder: use token passing or word beam search decoding (see CTC Word Beam Search) to constrain the output to dictionary words

Text correction: if the recognized word is not contained in a dictionary, search for the most similar one

**3.2 Library Modules Used (Python)**

**3.2.1 Tensorflow**

The below diagram depicts the system architecture

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows users to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.

**Key features of TensorFlow**

Efficiently works with mathematical expressions involving multi-dimensional arrays

Good support of deep neural networks and machine learning concepts

GPU/CPU computing where the same code can be executed on both architectures

High scalability of computation across machines and huge data sets

Together, these features make TensorFlow the perfect framework for machine intelligence at a production scale.

**OVERVIEW**

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open source license on November 9, 2015. TensorFlow provides primitives for defining functions on tensors and automatically computing their derivatives.

**What is a Tensor ?**

TensorFlow programs use a tensor data structure to represent all data — only tensors are passed between operations in the computation graph. A Tensor can be considered as an n-dimensional array or list.

**Models and Training**

A model is the relationship between features and the label. A good machine learning approach determines the model for the user. The program will determine the relationship between different features and labels. Training is the stage of machine learning in which the model is gradually optimized (learned). In supervised machine learning, a model is trained from examples that contain labels. In unsupervised machine learning, the examples don't contain labels. Instead, the model typically finds patterns among the features.

**Tensor processing unit (TPU)** Tensor processing unit (TPU) is an ASIC built specifically for machine learning and tailored for TensorFlow. TPU is a programmable AI accelerator designed to provide high throughput of low-precision arithmetic (e.g., 8-bit), and oriented toward using or running models rather than training them.

**3.2.2 Keras**

Keras is an open source neural network library written in Python. It can run on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or MXNet. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

**FEATURES OF KERAS**

Keras contains numerous implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier. Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. It also allows use of distributed training of deep learning models on clusters of Graphics Processing Units (GPU).

**GUIDING PRINCIPLES OF KERAS**

Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

**Modularity**

A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible. Neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models.

**Easy extensibility**

New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.

**Work with Python**

Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

**WHY USE KERAS?**

**Keras prioritizes developer experience**

Keras is easy to learn and use. Keras integrates with lower-level deep learning languages (TensorFlow).In particular, as tf.keras, the Keras API integrates seamlessly with TensorFlow workflows.

**Keras has a broad adoption in the industry**

With over 200,000 individual users as of November 2017, Keras has stronger adoption in both the industry and the research community than any other deep learning framework except TensorFlow itself (and Keras is commonly used in conjunction with TensorFlow).

**3.3 Operations Done**

**3.3.1 CNN**

In machine learning, a convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analysing visual imagery.CNNs use a variation of multilayer perceptrons designed to require minimal pre-processing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

****

Fig 3.6 Basic CNN layout

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage. They have applications in image and video recognition, recommender systems and natural language processing. There are four main operations in the ConvNets:

**Convolution**

**Non-Linearity (ReLU)**

**Pooling or Sub Sampling**

**Classification (Fully Connected Layer)**

These operations are the basic building blocks of *every* Convolutional Neural Network, so understanding how these work is an important step to developing a sound understanding of ConvNets.

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.

.

**3.3.2 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), therefore there are 80 entries for each of the 32 time-steps.

**3.3.3 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.

Assistant is a virtual personal assistant developed by Google that is primarily available on mobile and smart home devices. Unlike Google Now, the Google Assistant can engage in two-way conversations. We use the vanilla beam search algorithm as a starting point. This algorithm iterates through the NN output and creates text candidates (called beams) which are scored. Fig. 2 shows an illustration of the evolution of beams: we start with the empty beam, then add all possible characters (we only have “a” and “b” in this example) to it in the first iteration and only keep the best scoring ones. The beam width controls the number of surviving beams. This is repeated until the complete NN output it processed.

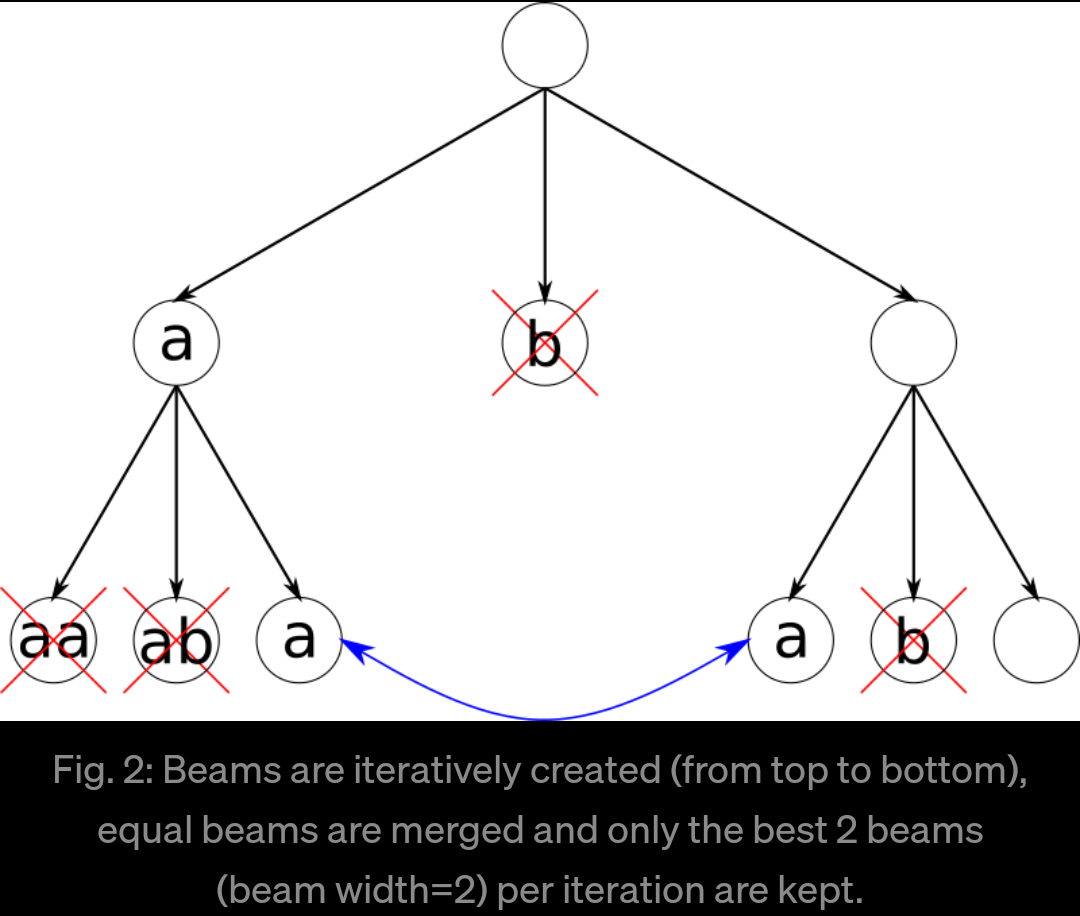


Fig. 3.5: Top: output matrix of the RNN layers. Middle: input image. Bottom: Probabilities for the characters “l”, “i”, “t”, “e” and the CTC blank label.

**3.3.4 LSTM**

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture[1] used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition,[2] speech recognition[3][4] and anomaly detection in network traffic or IDS's (intrusion detection systems).

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications.

**Architecture**

There are several architectures of LSTM units. A common architecture is composed of a cell (the memory part of the LSTM unit) and three "regulators", usually called gates, of the flow of information inside the LSTM unit: an input gate, an output gate and a forget gate. Some variations of the LSTM unit do not have one or more of these gates or maybe have other gates. For example, gated recurrent units (GRUs) do not have an output gate.

Intuitively, the cell is responsible for keeping track of the dependencies between the elements in the input sequence. The input gate controls the extent to which a new value flows into the cell, the forget gate controls the extent to which a value remains in the cell and the output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit. The activation function of the LSTM gates is often the logistic sigmoid function.

There are connections into and out of the LSTM gates, a few of which are recurrent. The weights of these connections, which need to be learned during training, determine how the gates operate.

### **3.4 Database**

**3.4.1 Kaggle - MNIST**

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

**MNIST**

Sample images from MNIST test dataset

The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset. The original creators of the database keep a list of some of the methods tested on it. In their original paper, they use a support-vector machine to get an error rate of 0.8%. An extended dataset similar to MNIST called EMNIST has been published in 2017,



Fig 3.8 MNIST Dataset

which contains 240,000 training images, and 40,000 testing images of handwritten digits and characters.

**Dataset**

The set of images in the MNIST database is a combination of two of NIST's databases: Special Database 1 and Special Database 3. Special Database 1 and Special Database 3 consist of digits written by high school students and employees of the United States Census Bureau, respectively.

Gogle Assistant is just another AI like Alexa, Siri or Bixby, and all Artificial Intelligent systems have the same basic working concept. Minor changes in the core are present in different AIs with its own custom properties.

**3.4.2 IAM**

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.

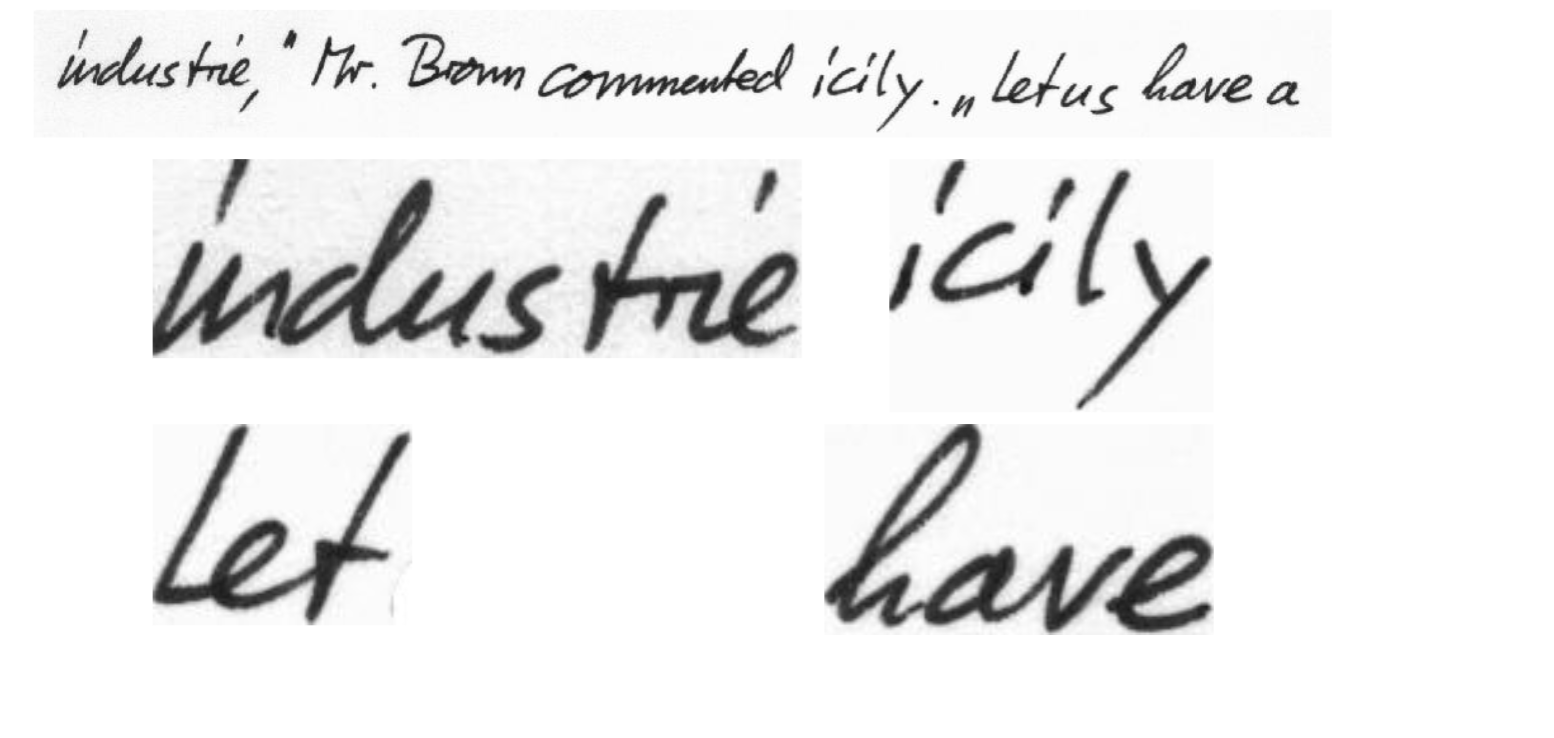


Fig 3.9 IAM Dataset

The database was first published in at the ICDAR 1999. Using this database an HMM based recognition system for handwritten sentences was developed and published in at the ICPR 2000. The segmentation scheme used in the second version of the database is documented in and has been published in the ICPR 2002. The IAM-database as of October 2002 is described in . We use the database extensively in our own research, see publications for further details.

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.

All forms and also all extracted text lines, words and sentences are available for download as PNG files, with corresponding XML meta-information included into the image files. All texts in the IAM database are built using sentences provided by the LOB Corpus .

**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

The words have been extracted from pages of scanned text using an automatic segmentation scheme and were verified manually. The segmentation scheme has been developed at our institute [3].

All form, line and word images are provided as PNG files and the corresponding form label files, including segmentation information and variety of estimated parameters (from the preprocessing steps described in [2]), are included in the image files as meta-information in XML format which is described in XML file and XML file format (DTD).

**3.5 Advancements Over Other Methods**

While exploiting the power of the computer systems, the curiosity of human, lead him to wonder, *“Can a machine think and behave like humans do?”*

Thus, the development of AI started with the intention of creating similar intelligence in machines that we find and regard high in humans.

**3.6Modifications**

**3.7 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 RMSProp.

**3.8 Testing**

**3.9 Improving** − It is the set of processes that enables us to provide basis for judgement, making decisions, and prediction. There are broadly two types.

**3.10 Output** It conducts specific observations to makes broad general statements. Even if all of the premises are true in a statement, inductive reasoning allows for the conclusion to be false.

**CHAPTER 4**

**SYSTEM DESIGN & IMPLEMENTATION**

## **4.1 SOFTWARE REQUIREMENTS**

This chapter deals with the overall flow and implementation of the project. The overall idea is implemented using Python – Jupyter Notebook and Spyder software.

**4.1.1 PYTHON** Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Python supports multiple programming paradigms, including object oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

Python interpreters are available for installation on many operating systems, allowing Python code execution on a wide variety of systems. Using third-party tools, such as Py2exe or Pyinstaller, Python code can be packaged into standalone executable programs for some of the most popular operating systems, allowing the distribution of Python-based software for use on those environments without requiring the installation of a Python interpreter. CPython, the reference implementation of Python, is free and open-source software and has a community-based development model, as do nearly all of its alternative implementations. CPython is managed by the non-profit Python Software Foundation.

**4.1.1.1 FEATURES**

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. An important feature of Python is dynamic name resolution (late binding), which binds method and variable names during program execution. Rather than requiring all desired functionality to be built into the language's core, Python was designed to be highly extensible.

**4.1.1.2 TYPING**

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Python allows programmers to define their own types using classes, which are most often used for object-oriented programming. New instances of classes are constructed by calling the class and the classes themselves are instances of the metaclass type (itself an instance of itself), allowing and reflection.

**4.1.1.3 LIBRARIES**

Modules for creating graphical user interfaces, connecting to relational databases, pseudorandom number generators, arithmetic with arbitrary precision decimals, manipulating regular expressions, and doing unit testing are also included. Python Package Index, contains more than 72,000 packages offering a wide range of functionality, including:

Graphical user interfaces, web frameworks, multimedia, databases, networking and communications

Test frameworks, automation and web scraping, documentation tools, system administration

## **Jupyter Notebook**

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter. Jupyter Notebooks are a spin-off project from the I-Python project, which used to have an I-Python Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the I-Python kernel, which allows you to write your programs in Python, but there are currently over 100 other kernel.

**Spyder**

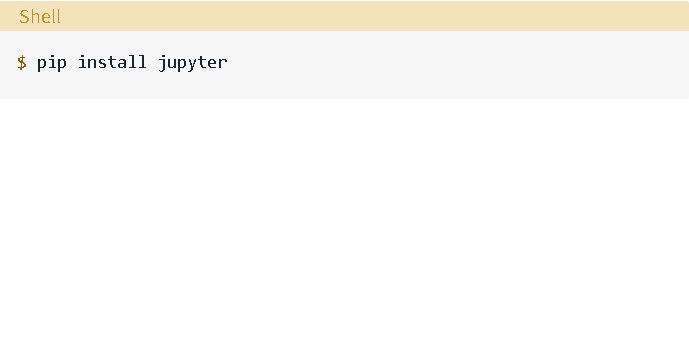
Spyder is a powerful scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. It features a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package. Furthermore, Spyder offers built-in integration with NumPy, SciPy, Pandas, IPython, QtConsole, Matplotlib, SymPy, and more. Beyond its many built-in features, Spyder’s abilities can be extended even further via its plugin system and API. Spyder can also be used as a PyQt5 extension library, allowing you to build upon its functionality and embeds it.

## **4.2 INSTALLATION AND SETUP**

## 

## The step by step process of how to install and setup the necessary software for the project will be explained in the following.

## **4.2.1 JUPYTER NOTEBOOK INSTALLATION**

We can use a handy tool that comes with Python called **pip** to install Jupyter Notebook like this. The next most popular distribution of Python is Anaconda. Anaconda has its own installer tool called **conda** that you could use for installing a third-party package. However, Anaconda comes with many scientific libraries preinstalled, including the Jupyter Notebook, so you don’tctually need to do anything other than install Anaconda itself.

## 

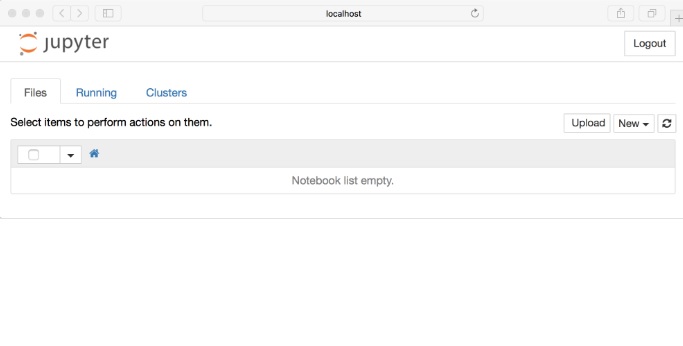
The following commands are typed in the terminals in order to bring up a new notebook.

## 

## 

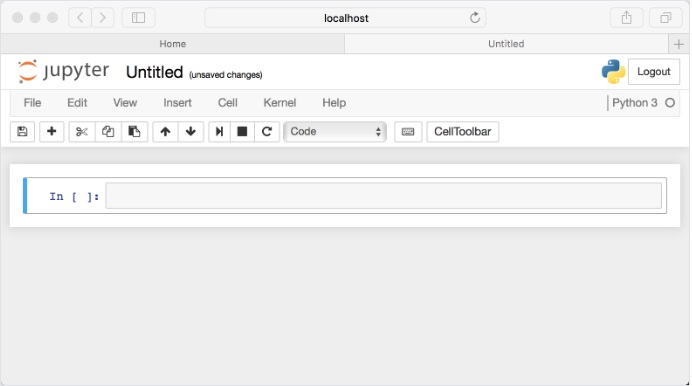
## Figure 4.2 Command to be typed to create new notebook

This will start up Jupyter and your default browser should start (or open a new tab) to the following URL: http://localhost:8888/tree



**Figure 4.3 New Notebook web page**

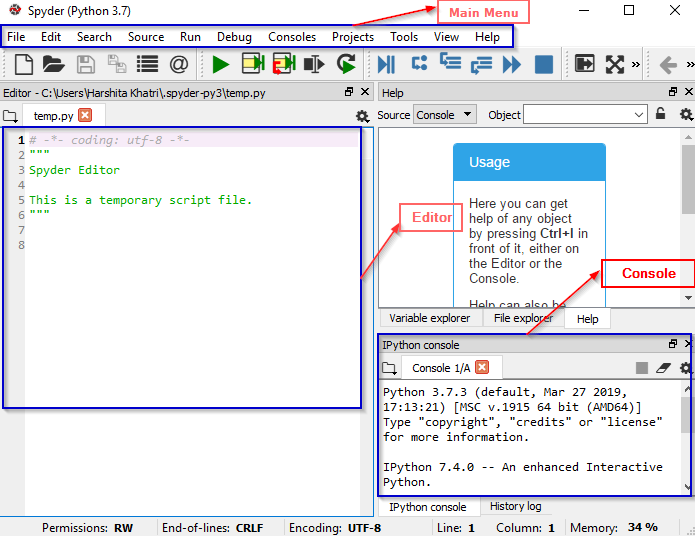
The above just shows how a server for the notebook has been opened, the following command will help in creating a new notebook. Click on the New button (upper right), and it will open up a list of choices. Click on Python 3.The webpage will look like this.



**Figure 4.4 New Notebook**

**4.2.2 SPYDER INSTALLATION**

Spyder is relatively easy to install on Windows, Linux and macOS. This section explains how to install the latest stable release of Spyder. To install with Anaconda Prompt, Spyder is included by default in Anaconda Python distribution, which comes with everything you need to get started in an all-in-one package. When it comes to Installing on a Windows based system Spyder is also included in the Win Python scientific Python distribution, although it doesn’t include convenient conda package and environment manager like Anaconda. It can be used immediately after installing, just like with Anaconda.



**Figure 4.5 Spyder Interface after Installation**

**Integrated Word Beam Search Decoding**

Besides the two decoders shipped with TF, it is possible to use word beam search decoding. Using this decoder, words are constrained to those contained in a dictionary, but arbitrary non-word character strings (numbers, punctuation marks) can still be recognized. The following illustration shows a sample for which word beam search is able to recognize the correct text, while the other decoders fail.

**Figure 4.6 Word beam search**



Word beam search can now be enabled by setting the corresponding command line argument. The dictionary is created (in training and validation mode) by using all words contained in the IAM dataset (i.e. also including words from validation set) and is saved into the file data/corpus.txt. Further, the (manually created) list of word-characters can be found in the file model/wordCharList.txt. Beam width is set to 50 to conform to the beam width of vanilla beam search decoding. Using this configuration, a character error rate of 8% and a word accuracy of 84% is achieved.

**Training the Model**

Figure 4.7 Training of the model

The data-loader expects the IAM dataset (or any other dataset that is compatible with it) in the data/ directory. Follow these instructions to get the dataset:

Register for free at fki.inf.unibe.ch.

Download words/words.tgz.

Download ascii/words.txt.

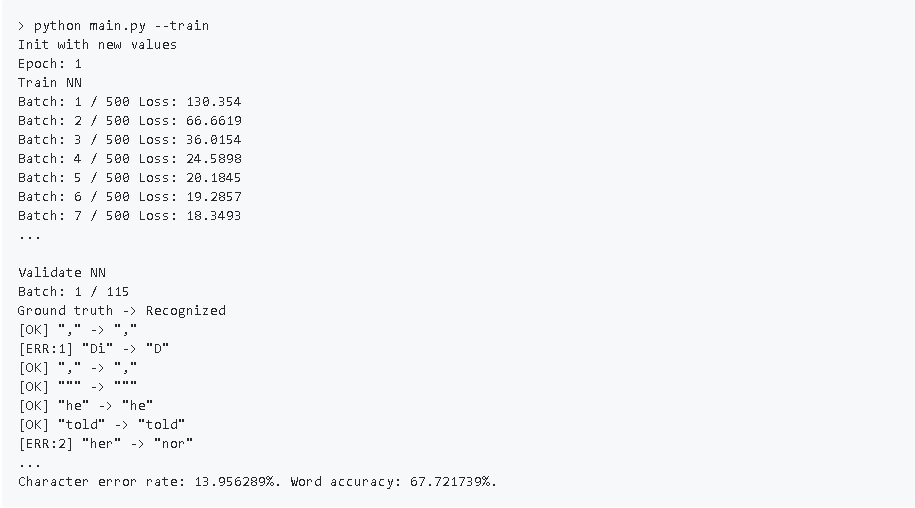
Put words.txt into the data/ directory.

Create the directory data/words/.

Put the content (directories a01, a02,) of words.tgz into data/words/.

Go to data/ and run python checkDirs.py for a rough check if everything is ok.

To train the model from scratch, delete the files contained in the model/ directory. Otherwise, the parameters are loaded from the last model-snapshot before training begins. Then, go to the src/ directory and execute python main.py --train. After each epoch of training, validation is done on a validation set (the dataset is split into 95% of the samples used for training and 5% for validation as defined in the class Data Loader). If you only want to do validation given a trained NN, execute python main.py --validate. Training on the CPU takes 18 hours on my system (VM, Ubuntu 16.04, 8GB of RAM and 4 cores running at 3.9GHz). The expected output is shown below.

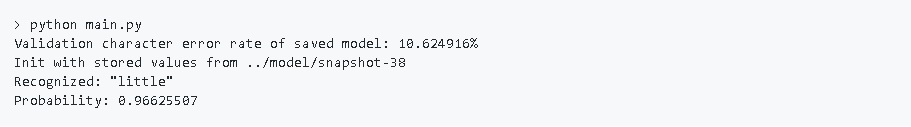


**Result**

Go to the model/ directory and unzip the file model.zip (pre-trained on the IAM dataset). Take care that the unzipped files are placed directly into the model/ directory and not some subdirectory created by the unzip-program. Afterwards, go to the src/ directory and run python main.py. The input image and the expected output is shown below.

****

## **Figure 4.8 Scanned Word**

****

## **Figure 4.9 Output of the scanned word**

## **CHAPTER 5**

## 

## **CONCLUSION AND FUTURE WORKS**

## **5.1 CONCLUSION**

As discussed a NN which is able to recognize text in images. The NN consists of 5 CNN and 2 RNN layers and outputs a character-probability matrix. This matrix is either used for CTC loss calculation or for CTC decoding. An implementation using TF is provided and some important parts of the code were presented. Finally, hints to improve the recognition accuracy were given.

## **5.2 FUTURE WORKS**

The Handwritten Text Recognition can be further enhanced by incorporating several other techniques such as Neural Networks, along with tensor flow. Putting more than one technique together instead of using a single technique will provide high Efficiency.

## REFERENCE

[1] Build a Handwritten Text Recognition System using Tensor Flow

[2] Scheidl - Handwritten Text Recognition in Historical Documents

[3] Shi - An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition

[4] Scheidl - Word Beam Search: A Connectionist Temporal Classification Decoding Algorithm

[5] Marti - The IAM-database: an English sentence database for offline handwriting recognition