VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JNANA SANGAMA", BELGAUM – 590014

A Project Report on

"An OCR System for Printed Kannada Text"

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

Bachelor of Engineering in Electronics and Communication Engineering

Submitted by:

Niraj S Prasad	1PI14EC043
Pradyumna Mukunda	1PI14EC045
Santhosh DM	1PI14EC060

Under the guidance of

Dr. Mamatha HR

(Professor, ISE, PESIT)

January 2018 – May 2018

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

PES INSTITUTE OF TECHNOLOGY

100 Feet Ring Road, BSK 3rd Stage, Bengaluru – 560085

PES INSTITUTE OF TECHNOLOGY

100 Feet Ring Road, BSK 3rd Stage, Bengaluru – 560085

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the project work entitled "An OCR System for Printed Kannada Text," carried out by:

Niraj S Prasad	**1PI14EC043**
Pradyumna Mukunda	**1PI14EC045**
Santhosh DM	**1PI14EC060**

in partial fulfilment for the award of degree of **BACHELOR OF ENGINEERING IN ELECTRONICS AND COMMUNICATION ENGINEERING** of **Visvesvaraya Technological University, Belgaum** during the year **January-May 2018.** It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the above said degree.

Dr. Mamatha HR	**Dr. Anuradha M**	**Dr. K.S. Sridhar**
Professor, ISE, PESIT	Professor and Head, ECE	Principal

External Viva

Name of the Examiners	**Signature with Date**
1._____	_____
2._____	_____

DECLARATION

We hereby declare that the project entitled **"An OCR System for Printed Kannada Text"** has been carried out by us and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Engineering** in **Electronics and Communication Engineering** of **Visvesvaraya Technological University, Belagavi** during the academic semester January – May 2018. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

1PI14EC043 Niraj S Prasad _____

1PI14EC045 Pradyumna Mukunda _____

1PI14EC060 Santhosh DM _____

ACKNOWLEDGEMENT

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped us in carrying out this project work. We would like to take this opportunity to thank them all.

We would like to thank **Prof. Jawahar D**, CEO, PES Institutions and **Dr. K S Sridhar**, Principal**,** PES Institute of Technology, Bengaluru, for their timely advice on the academics and regular assistance throughout our first semester.

We would like to express my sincere gratitude to **Dr**. **Anuradha M**, Professor and Head of Department, ECE, PESIT, Bengaluru whose contribution to the success of the project has been immense.

We are deeply appreciative of the valuable suggestions given us at various stages during the course of the effects by our guide and project co-ordinator, **Dr. Mamatha H.R**, Professor, Department of ISE, PESIT, Bengaluru. We are grateful for his help in the preparation of this manuscript. We also express our thanks to the examining committee for their valuable suggestions.

We also extend my sense of gratitude and sincere thanks to all the faculty members of Department of ISE, and the Department of ECE, PESIT, Bengaluru, for their constant encouragement and support.

Last but not least we extend our gratitude to our parents for being a constant motivation and providing impetus throughout the project.

-

Niraj S Prasad

Pradyumna Mukunda

Santhosh DM

ABSTRACT

‘Optical Character Recognition’, or OCR, is basically converting an image containing text to an editable text format. The image could either be a scanned document, or a simple newspaper cut-out. Using Supervised Learning in the form of Neural Networks will make the system produce the output required with a much larger accuracy.

Unlike English, Kannada Language has a greater number of characters since it includes kaagunithas, vattaksharas, etc. This makes recognition of the characters much complex. The project mainly concentrates on OCR for the Kannada Text. With little or no systems available for this language, we plan to develop a system that is font and size independent.

In OCR, the image undergoes thresholding to convert it into a black and white image and then processed. This gives an advantage for segmentation of the characters. Characters can be extracted from the documents using various Segmentation methods. And the type of method used can vary the accuracy of the system. The vattaksharas are extracted/differentiated from the words by using base-line technique. When the characters are recognized, they are compared with Unicodes available on the system and then printed.
In the above method, CNN plays a pivotal role in reading the character and comparing it with the Unicode look up table values to print the output.

This system has been tested on a single dataset with varying fonts. A total number of 4 sample documents is used for experimentation. The system has been developed for only printed Kannada Text as of now.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

1. OCR…… Optical Character Recognition
2. CNN…… Convolutional Neural Network
3. ICR ……. Intelligent Character Recognition
4. KNN…… K-nearest neighbour
5. API ……. Application Program Interface
6. VATT ……. Vattaksharas
7. KAG……. Kaagunithas

CHAPTER 1: INTRODUCTION

1.1 AN INTRODUCTION TO OCR

The use of OCR dates back to 1914 when Emanuel Goldberg developed an intelligent machine to read a list of characters and convert them into telegraph code.

Optical Character Recognition is basically an electronic or mechanical conversion of image text into editable text format. The image could be hand-written, a scanned document, a photograph with some content, or any printed text. Digitising printed text is essential in editing and storing valuable information on a software platform.

Demand for a stable regional language OCR system has grown over the past few years. With the advance in technology, a smartphone can now read any text on an image captured.

Using OCR systems, organisations can now edit documents more efficiently. Digital text data can be stored and saved without much risk of loss, unlike a physical document. Traffic police
now accept digitally scanned documents. In case of a change in information, scanning and editing a document is currently practiced in a lot of places. Scanning and editing a file is one of the best approaches one can find.

1.2 PROBLEM DEFINITION

Besides all the advantages of a stable OCR system, accuracy is the major issue. With just 26 alphabets to deal with, developing an OCR system for the English language is less complicated and straightforward compared to other languages. Languages like Kannada, Hindi, Tamil, Telugu, etc., include compound consonants like vattaksharas, dheergas apart from the basic characters. This makes it more complicated to develop a similar system for these languages.

Living in Karnataka for a very long time, the necessity of a reliable Kannada OCR system is clearly noticeable. With some research, it has been found that there only a few Kannada OCR systems available on the market. These systems, however, aren't reliable and accurate enough to provide the best results. As per our research, using Convolutional Neural Networks, or CNN's, provides the most accurate results. Machine Learning is vital when it comes to dealing with OCR. Unless the system is trained by appropriate methods, the run time will be much more than what it should be. With the use of CNN, the run time decreases rapidly and an agile system with good accuracy is functional.

1.3 OBJECTIVE OF THE PROJECT

Due to lack of OCR Systems for Regional Languages, people find it hard to get a text document from an image. The Main objective of our project is to provide a convenient and reliable OCR system for the Kannada Language.

There is a high demand for storing information on to a digital storage device from the data available in printed or handwritten documents or images to later re-utilize this information by means of computers. And one way to store the information to a computer system from these printed documents could be to scan the documents or files. But to re-utilize this information, it would be very challenging to read or modify text or other information from these image files. Therefore, a method to automatically retrieve and store information, in the form of text, from image files is needed. This is where Optical Character Recognition comes into play.
Optical character recognition is an active research area that attempts to develop a digital system which can extract and process text from images without human intervention.
The objective of an OCR is to achieve modification or conversion of any form of text or text-containing documents such as handwritten text, printed or scanned text images, into an editable digital format for further processing.

After looking into the available resources, it has been found that the requirement for a Kannada OCR system is quite high. These systems are required in various fields for purposes including textual versions of printed document, assistive technology for blind and visually impaired users, helping users who have little or no knowledge of the language, etc.

Thus we aim at providing a system that is entirely automatic and highly accurate, providing the best possible results for the users.

CHAPTER 2: LITERATURE SURVEY

2.1 BRIEF HISTORY

Optical Character Recognition is the conversion of an image containing some text into an editable text format. It is a field of research in the fields of pattern recognition, computer vision, and artificial intelligence.
The earliest known version of the OCR dates back to 1914 when Emanuel Goldberg developed a system that recognizes characters and converts them to standard telegraph code. A more advanced version of Goldberg's system was created by Edmund d'Albe. It was called the Optophone, which is a hand-held scanner that should be moved around on printed text. The device produced tones that corresponded to each letter and character. Later in the 1930s, Goldberg developed the 'Statistical Machine' for finding microfilm archives using an OCR system. He won the patent for this invention.

2.1.1 Visually Impaired Users

In the year 1974, Ray Kurzweil started Kurzweil Computer Products Incorporated. A much more advanced version of the Optophone was created such that the OCR system could recognize text printed in any given font. Kurzweil wanted to develop a system to help the visually impaired users to read out from another script than Braille. He developed a system such that when a visually impaired person held a printed sheet with text over a computer, it would read it out loud. This involved the invention of two technologies – the text to speech converter and synthesizer, and the CCD flatbed scanner. In 1976, commercial sales of the product started. A company called LexisNexis purchased the product and added memory-based functions, so it read out newspaper scripts and could upload legal documents. Xerox then brought of Kurzweil's company to make more changes and advances in the paper to computer text field. Having merged with Nuance Communications, a research group headed by A G Ramakrishnan of the Indian Institute of Science developed the PrintToBraille tool, a GUI front-ended open source application. Scanned images of any text book could now use OCR to print Braille books. Various commercial Open Source OCR systems are available in different scripts which include Chinese, Japanese, Korean, Devanagari, Tamil, Bengali, Latin, Arabic, Indic, Hebrew, and Cyrillic.

2.2 TYPES OF OCR SYSTEMS

- OCR which targets typed text one character at a time.
- OWR or Optical Word Recognition which focuses on typed text one word at a time. This is possible only for languages that use a word-divider or a space.

- ICR or Intelligent Character Recognition that involves Machine Learning, ICR targets one sentence at a time.
- IWR or Intelligent Word Recognition that involves Machine Learning. IWR works for handwritten print scripts really well. Cursive text and those glyphs that are not separated in cursive works really well with IWR.

OCR usually analyses a static document; hence it is labelled 'offline.' Handwriting moving analysis can be used for hand-written text to generate the output in real-time. This process is called 'Dynamic Character Recognition' or 'ICR', or 'Real-time character Recognition.'

2.3 TECHNIQUES

2.3.1 Pre-processing

OCR systems might have to make some changes in the image before running the algorithm in order to get much higher accuracy and hit rate. To get a successful output, the following techniques might have to be considered –

1. *De-Skew*: The image might be tilted, or the text image might not be aligned perfectly as the system wants to it be. So some de-skewing needs to be done before OCR takes place. The document may be tilted either clockwise or counter-clockwise to align it perfectly.
2. *De-speckle*: The system smoothens the image and removes the negative and positive spots of the picture. This ensures greater accuracy.
3. *Binarisation*: In case the input is a color image, the system might not be able to differentiate between different colors. So is converts the image to greyscale and then performs OCR.
4. *Line and Word Detection*: It separates the words if necessary. It also establishes a baseline for words.
5. *Script Recognition*: In the same document, there might be different scripts. The system must be able to realize which script it is working with.
6. *Normalizing* the scale and the aspect-ratio.
7. *Character isolation and segmentation*: this is useful in Character OCR. Words are broken down into many characters, hence making reading smooth and accurate.

2.3.2 Character Recognition

There are currently two types of the core OCR algorithm.

Matrix Matching is a technique wherein a stored glyph is compared with the image. It is also called pattern recognition and image correlation. This technique works best with typewritten text. It does not work well when a different font is encountered.

Feature Matching breaks down the glyphs into lines, lops, line-intersections, and line directions. These features are compared with the abstract vector-like representation of the character. Feature detection in Computer Vision is applicable to this type of OCR System. This method uses Machine Learning and is the most modern OCR Software. KNN algorithm is a popular method. In our project, we have used an entirely different technique in the form of CNN.

Specific software such as Tesseract and Coneiform uses a two pass approach to character recognition. This is useful when the font is distorted, blurred, or faded.

2.3.3 Post Processing

A dictionary can be added to check the accuracy of the word, and it can be processed with maximum efficiency possible. The output stream can be a plain text file of characters. Sophisticated OCR systems also preserve the layout and font perfectly. In order to further optimize results from an OCR API in post-processing, the Levenshtein Distance algorithm can be used.

2.4 UNICODE

Unicode is defined by Wikipedia as “a computing industry standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems.” Unicode was developed when 8-bit encoding systems such as ASCII were still popular. Since ASCII could hold only 256 characters, only Roman characters were represented.

Many countries had developed their own versions of ASCII for their native languages. For example India developed ISCII. Alternatively, early Kannada writing software such as Baraha used customized ASCII fonts that merely rendered their own Kannada glyphs in place of the correct ASCII glyphs. While this solution is good for printing Kannada text on paper it is not suitable for applications such as transmitting Kannada text online or displaying Kannada text in web pages or on mobile devices. A universal encoding standard is needed. Unicode uses 16 bits (specifically UTF-16 uses 16 bits), which is way more than enough to represent characters in all of the world’s living languages, as well as historic scripts such as Brahmi.

UTF-16 assigns each of its characters with a unique 16-bit identification number known as a code point, and leaves the rendering of the character to the software. The code points for Kannada characters are in the range of 0x0C82 to 0x0CF2. This range of code points is reserved exclusively for Kannada characters, unlike in ISCII where the same character in different Indian languages is assigned the same code point.

Kannada[1][2]

Official Unicode Consortium code chart (PDF)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0C8x	□	◌ಁ	◌ಂ	◌ಃ		ಅ	ಆ	ಇ	ಈ	ಉ	ಊ	ಋ	ಌ		ಎ	ಏ
U+0C9x	ಐ		ಒ	ಓ	ಔ	ಕ	ಖ	ಗ	ಘ	ಙ	ಚ	ಛ	ಜ	ಝ	ಞ	ಟ
U+0CAx	ಠ	ಡ	ಢ	ಣ	ತ	ಥ	ದ	ಧ	ನ		ಪ	ಫ	ಬ	ಭ	ಮ	ಯ
U+0CBx	ರ	ಱ	ಲ	ಳ		ವ	ಶ	ಷ	ಸ	ಹ			◌಼	ಽ	◌ಾ	◌ಿ
U+0CCx	◌ೀ	◌ು	◌ೂ	◌ೃ	◌ೄ		◌ೆ	◌ೇ	◌ೈ		◌ೊ	◌ೋ	◌ೌ	◌್		
U+0CDx						◌ೕ	◌ೖ								ೞ	
U+0CEx	ೠ	ೡ	◌ೢ	◌ೣ			೦	೧	೨	೩	೪	೫	೬	೭	೮	೯
U+0CFx		◌ೱ	◌ೲ													

Figure 2.1: Unicodes for Kannada characters (Source: Wikipedia)

2.5 APPLICATIONS OF OCR

- Data Entry for passports, invoices, and other business documents.
- Automatic Number Plate Recognition.
- Key information extraction from insurance documents.
- Making electronic images of printed documents searchable.
- Giving instructions to computers by writing.
- Assistive technology for the blind and visually impaired users.
- Convert printed text-books into editable e-Books easily.

2.6 EXISTING OCR SYSTEMS FOR THE KANNADA LANGUAGE

There are only a few notable OCR systems for the Kannada Language. KanScan is an app that converts a Kannada text image into editable text format. It contains a lot of flaws and gives a lot of errors when it runs. Another system is the i2OCR that offers an accuracy of around 60% with a run-time of almost 1 whole minute for a 200-word article. The links for the same are given below.

- https://play.google.com/store/apps/details?id=com.kaleidosoftware.kanscan.free
- www.i2ocr.com/free-online-kannada-ocr

Hence, it can be seen that the need for a top-notch Kannada OCR system is essential for the masses. The OCR systems can be used for correction of Question papers for PUC and SSLC. It can also be used in number plate recognition on Government vehicles. Use of CNN gets us the highest possible accuracy in this system.

CHAPTER 3: METHODOLOGY

After a brief amount of research, it was found that Template Matching and Convolutional Neural Networks (CNN) are the methods that can be used for OCR. The CNN methodology was adopted to make progress with the project.

3.1 Block Diagram

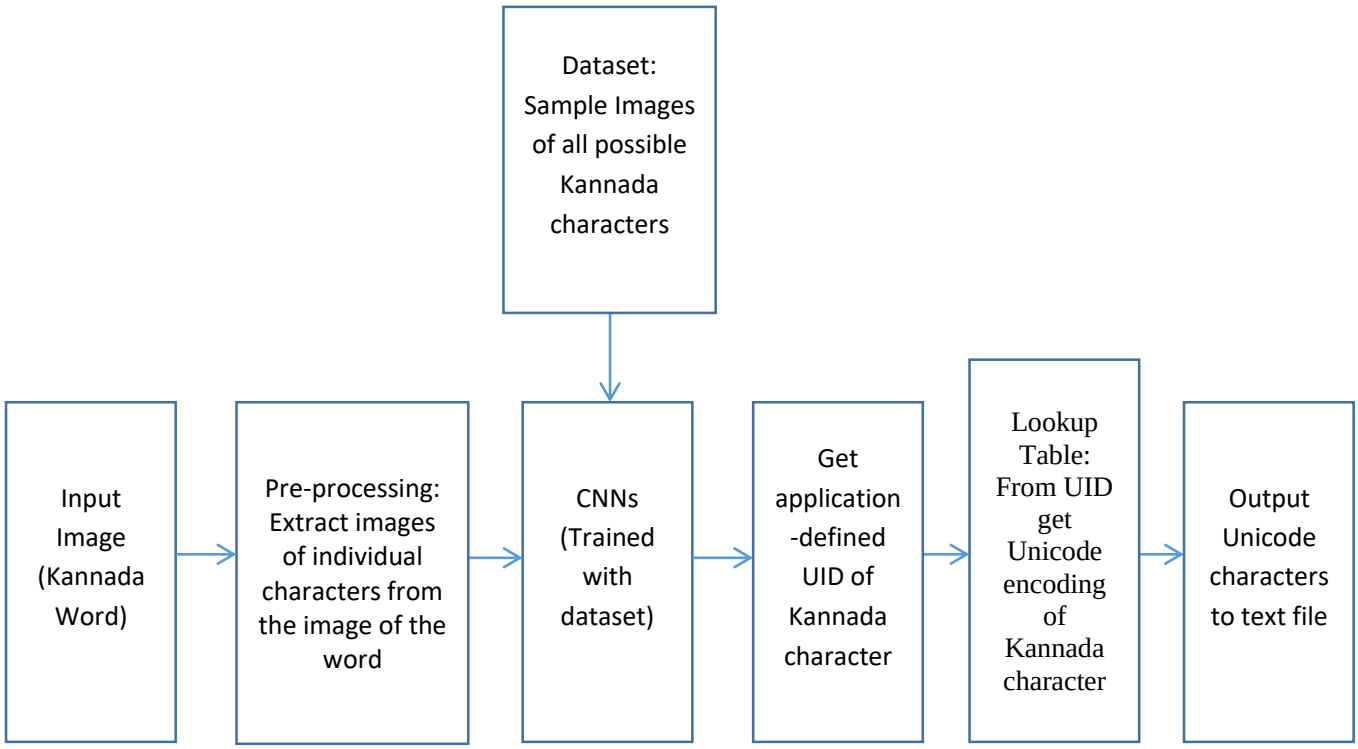


Figure 3.1: System Block Diagram

The simplest form of the system architecture is shown above. The sample image is the printed Kannada text given as the input to the system. The Pre-Processing block helps with the segmentation of words and characters. Size-based classification too occurs. 4 CNN's are trained with varied input sizes of the image. The dataset consists of twenty fonts overall. A UID table is then constructed to ensure the Unicode is matched. The output is then compared by the system with the UID table to get the editable text file.
Each block is explained in brief in the coming sections.

3.1.1 Pre – Processing

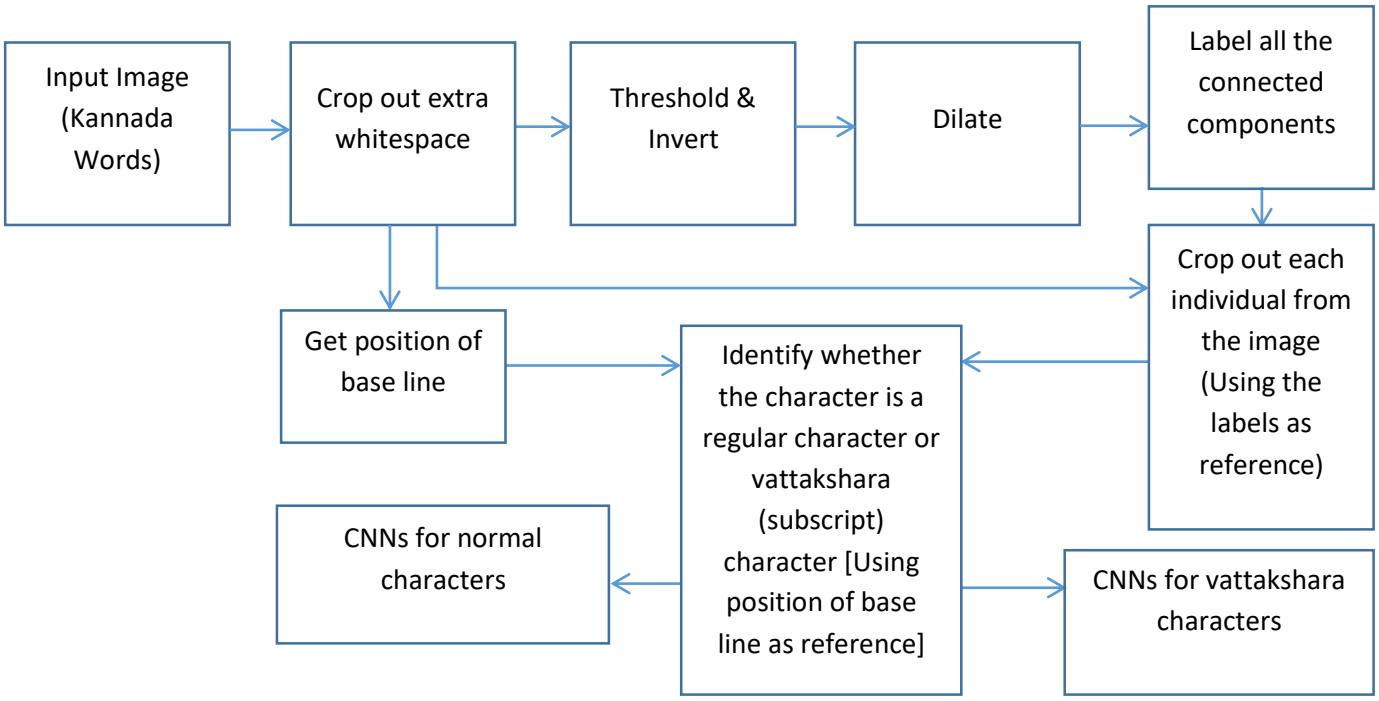


Figure 3.2: Pre-processing Block Diagram

The pre-processing block consists of segmentation of characters. First we trim out the extra image pixels around the word. Next we identify the position of the base line, which we will be using later on in the process. Then we perform thresholding to differentiate foreground pixels and background pixels. Next we have to label the connected components. Some characters are a single connected component while some of them consist of multiple components. Hence we first dilate the image and then label the connected components. The labels are not in the same order as the letters in the word; hence we have to sort them. Using the labels as reference, we crop out our characters. We identify them as regular or vattakshara characters using the position of the base line, then we feed them to the appropriate set of CNNs for identification.

A demonstration of the segmentation is shown below:

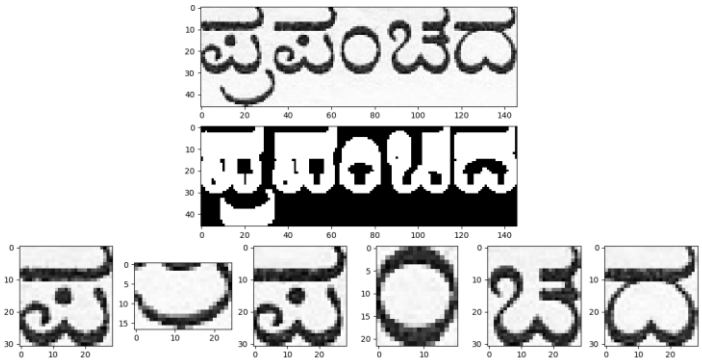


Figure 3.3: An Example for Segmentation

Finding the base line:

To find the position of the base line, we first take an input image of a word and then apply thresholding followed by Sobel edge detection to get all the **lower edges** in the image. We then take the horizontal sum of edge detection result and find the maximum value in the lower half. It is demonstrated for the image below. The base line has been identified to be at position 29 on the H (height) axis.

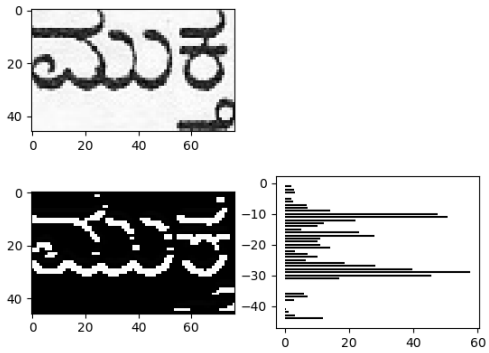


Figure 3.4: An Example for showing BaseLine identification

3.1.2 CNN Block Diagram

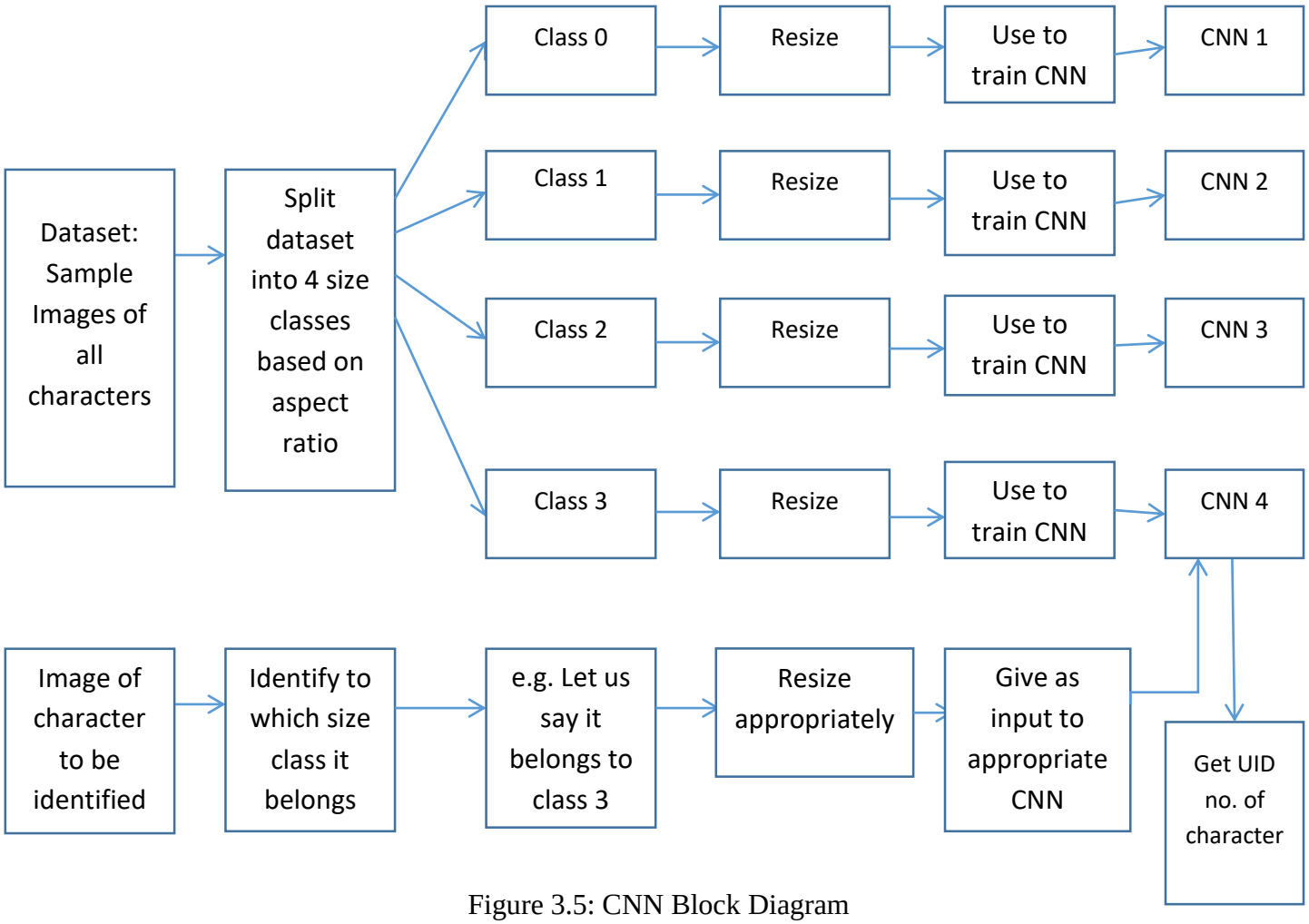


Figure 3.5: CNN Block Diagram

The block will be explained after a brief introduction to neural networks which is in the next section.

3.2 Neural Networks

Artificial Neural Networks or ANNs are computing systems modelled after and the biological neural networks in animal brains and designed to mimic them. They consist of an input layer and an output layer with multiple hidden layers in between. Each layer consists of a number of processing units called as neurons.

A neuron, the basic unit of a neural network, takes multiple inputs and returns a single output. In a feedforward neural network, the outputs of the neurons in one layer are fed as inputs to the neurons in the next layer. The input layer is the first layer in this chain which takes its input signals from the user and the output layer is the last layer in the chain which returns its outputs to the user. A neural network may have multiple inputs and multiple outputs.

The attractive feature of these systems is that they do not require task-specific programming and instead they are designed to "learn" to perform the tasks themselves by considering examples. For example, in our application i.e. Optical Character Recognition, if we want to configure an ANN to recognize the character 'A' in an image, we just have to train it with a number of images labelled 'A' and 'not A' and then subsequently if an unknown image is presented the system can successfully recognize it as 'A' or 'not A'. It is just like how we teach children to read the alphabet. In a non-learning system the programmer would have to define the rules to recognize the letter 'A' in the program itself, which would take hours of coding. In this way using Neural Networks for our application i.e. OCR would greatly simplify the design of the program.

In most neural networks a neuron is implemented as a unit which takes the weighted sum of its multiple inputs, applies an activation function to it and returns the result as output. Each input to a neuron has its own weight. When we "train" the neural network to "learn" something, what we are actually doing is adjusting these weights so as to get a desired output 'y' for a given input 'x'. Algorithms such as Gradient Descent Algorithm are used for this.

Use of Regular Neural Networks for Image Classification:

In a fully connected neural network, every neuron takes inputs from each and every neuron in the previous layer and feeds its output to each and every neuron in the next layer. In other words, every possible connection between two neurons in two different layers is made.

This architecture of neural network does not scale well for images. Firstly, the images would require a lot of manually engineered pre-processing if we want to obtain an accurate result. Secondly, it is because the size of the input is very large. Let us say we are giving a 200x200x3 RGB image as input to our network, by making each color value of each pixel as a separate input. Therefore we have 200x200x3 = 120,000 inputs. This means every neuron in the first layer takes 120,000 inputs and correspondingly has 120,000 weights, one for each input.

For one neuron alone we need to store and process 120,000 variables. And one neuron is barely enough for a decent neural network. The complexity of the system increases greatly when we add more neurons and layers to the network.

The solution to this problem can be found by studying the visual cortex of a biological eye. In the visual cortex individual neurons respond to visual stimuli only in a restricted region of the visual field and not the entire visual field. The conclusion we can draw from this is that we have to create a network where each neuron processes data from only a restricted area of the image known as the neuron's receptive field, and not the entire image.

3.2.1 Convolutional Neural Networks (CNNs)

In convolutional neural networks, we use three new types of layers, convolution layers, ReLU layers and pooling layers

Convolution layers: Convolution layers consist of a number of filters (say 'n'). The convolution layer takes an input image as is and performs 2D convolution operation on it with each of its 'n' filters, and returns 'n' output images known as feature maps. The filters replace neurons and the filter coefficients are just like weights in the sense that they are trainable.

ReLU layers: ReLU layer is just an activation function layer; it performs ReLU activation function on each pixel in the input image to return an output image of same size. ReLU is an abbreviation for rectified linear unit, it is the activation function defined by $f(x) = x$ for $x >= 0$ and $f(x) = 0$ for $x < 0$.

Pooling layers: Pooling is an operation where we downsample the input image by combining every m×n group of adjacent pixels together into a single pixel. In the most common type of pooling, max pooling, the max value in every group is retained while the other values are discarded.

These three layers are used along with the traditional fully connected layers to form a neural network. CNNs are classified as deep learning networks because they usually contain a large number of hidden layers.
Using CNNs greatly reduces the amount of manual pre-processing needed for the image. In fact, since we use trainable filters we can say that the CNN learns the pre-processing itself.

3.2.2 Software used in Development of CNN

TensorFlow:

TensorFlow is an open source machine learning framework developed by the Google Brain team. It is a symbolic math library that is used for machine learning applications like neural networks. It is available for PCs and servers running Windows, MacOS or Linux and even for mobile Android or iOS devices. It provides APIs in Python, C++, Java and many more programming languages. TensorFlow is used in Google applications such as RankBrain and DeepDream.

Keras:

Keras is an even higher level library that runs on top of TensorFlow, which simplifies the process of implementing deep neural networks even more. It is an open source neural network library written in Python and can operate on TensorFlow, Theano, Microsoft Cognitive Toolkit (CNTK) or Apache MXNet. If it takes 11 lines of code to implement a CNN in Keras the same would take 42 lines of code in TensorFlow.

3.3 Creating the Dataset

To create our dataset, we first list out all possible Kannada glyphs in a Microsoft Word document, like so:

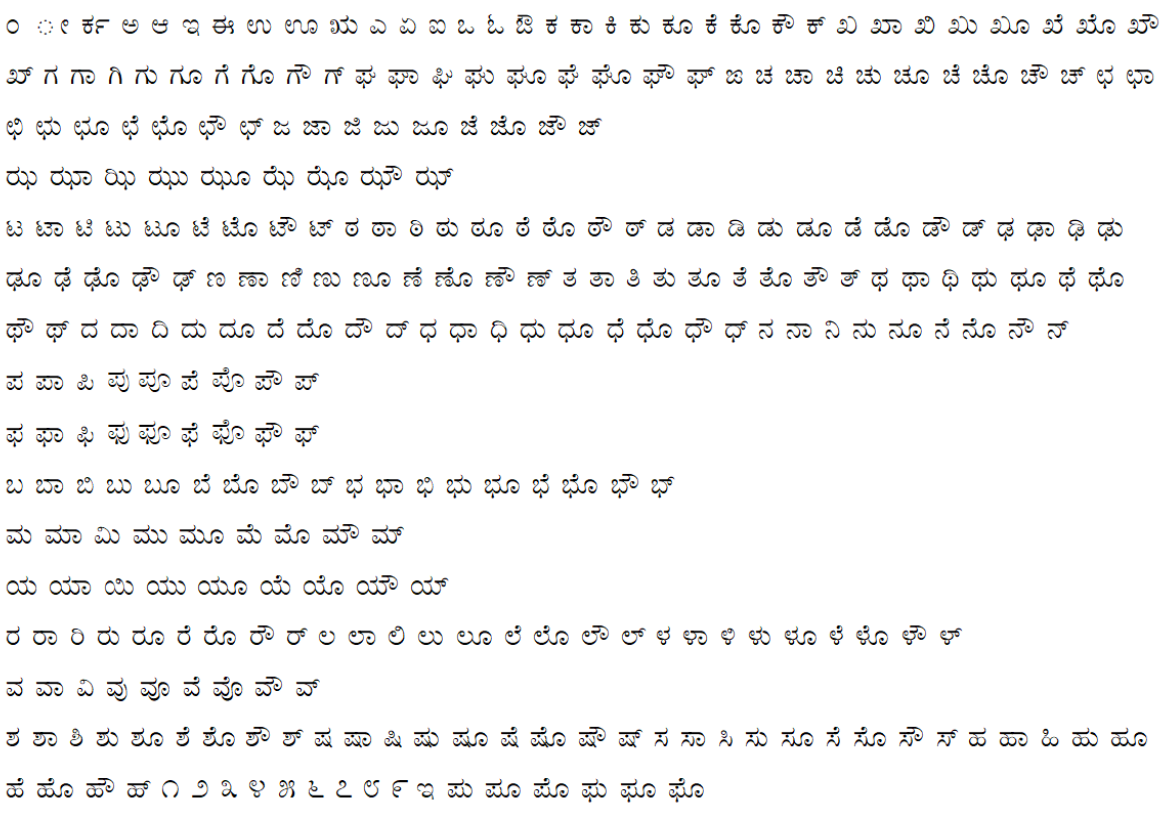


Figure 3.6: The Kannada Alphabet

We do this for 10 Kannada fonts, in normal text and in bold text for each font. The fonts used are given below:

Figure 3.7: An image showcasing different fonts

Next, we convert each page in the Word document into a PNG image, and then we run the image through a segmentation code written by us in GNU Octave. The code separates the individual characters and saves all of them into separate images. The results are shown below:

Figure 3.8: Character Extraction using Segmentation

The code not only separates the characters but also labels each character in order with a 3-digit number from 0 to 339, which will become the UID of the character. Any images with the same 3-digit number as a suffix now contain the same character, we can demonstrate this using file search. Any inconsistencies are corrected manually:

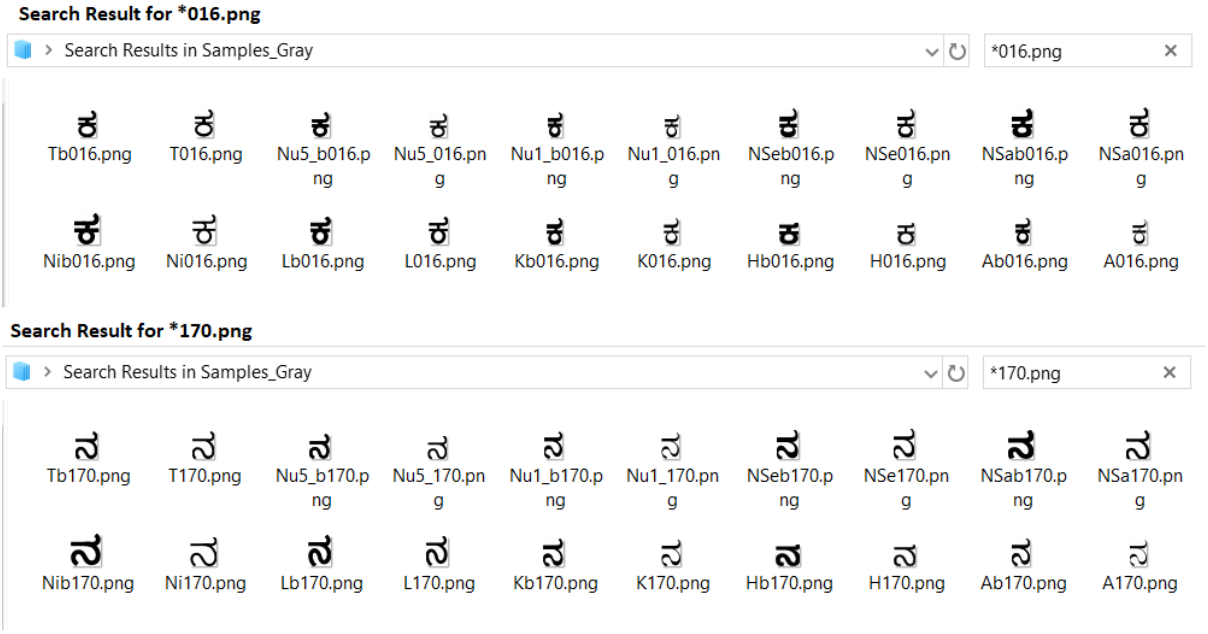


Figure 3.9: Storing the Samples

Now we split the entire dataset into 4 groups as follows:

We resize each image to a height of 20 pixels keeping the aspect ratio constant, then we study the width of each image.

- Images of width 22 or less are resized to 15x20 and saved into a folder “Samples15”
- Images of width 18 to 27 are resized to 25x20 and saved into a folder “Samples25”
- Images of width 23 to 37 are resized to 30x20 and saved into a folder “Samples30”
- Images of width 33 or greater are resized to 40x20 and saved into a folder “Samples40”

Note that the same sample may appear in two folders with different sizes.

Why is the dataset divided into four classes?

A CNN takes a constant input size, however the sizes and aspect ratios of our characters are variable. All characters must be resized to the constant input size before being input to the CNN. We could use a single CNN for all the characters ignoring the difference in aspect ratios, but this would give very inaccurate results. The solution is to divide the dataset into 4 classes based on the aspect ratios, then use these datasets to train 4 CNNs of different input sizes [i.e. 15x20, 25x20, 30x20 and 40x20.] In this way we try to roughly preserve the original aspect ratio of the characters so they can be identified properly.

Figure 3.10: An example showing characters with varying aspect ratios

The results of dividing and resizing samples are shown below:

Samples resized to 15x20 and stored in “Samples15”:

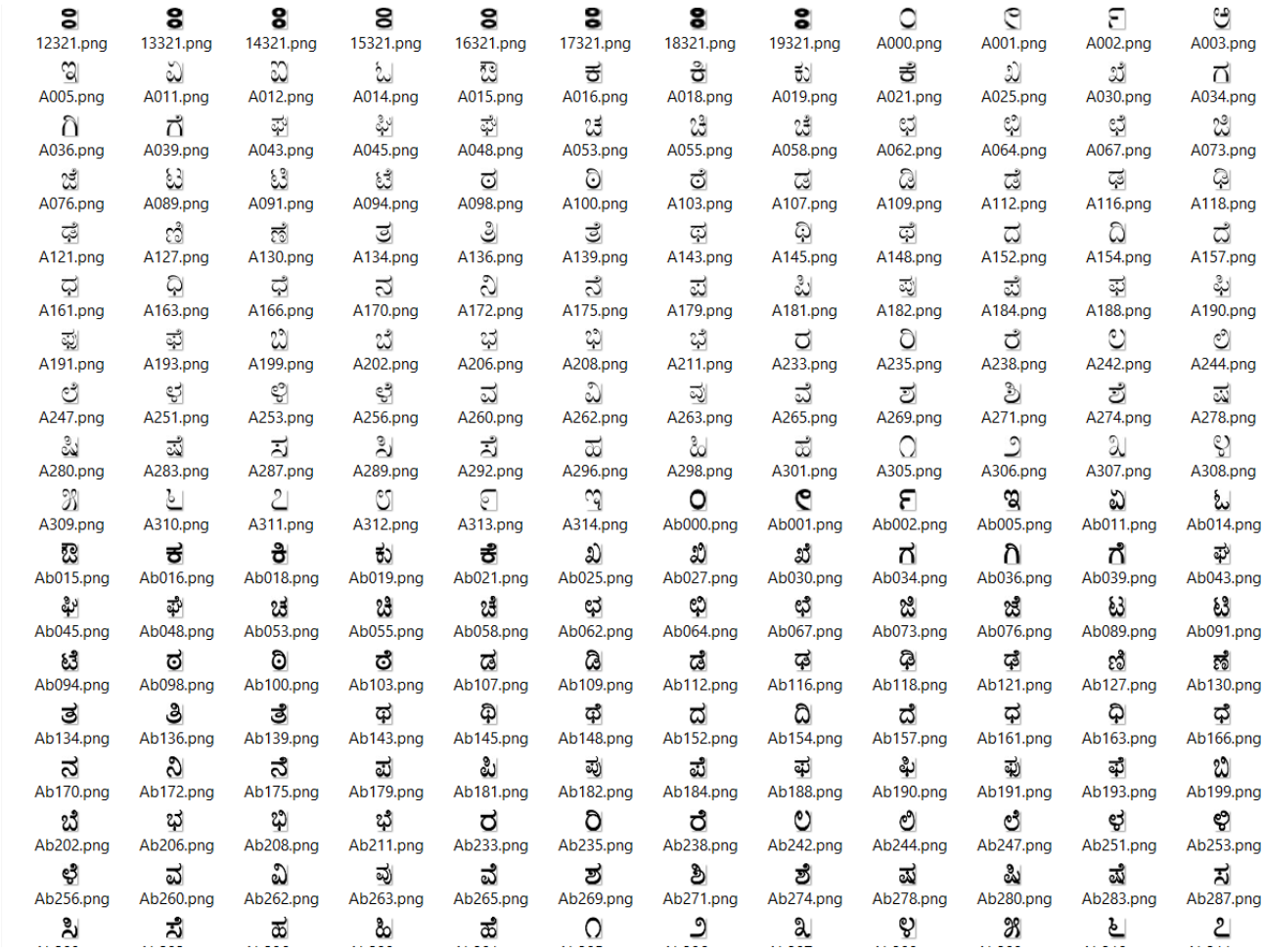


Figure 3.11: Samples of size 15 x 20

Samples resized to 40x20 and stored in “Samples40”:

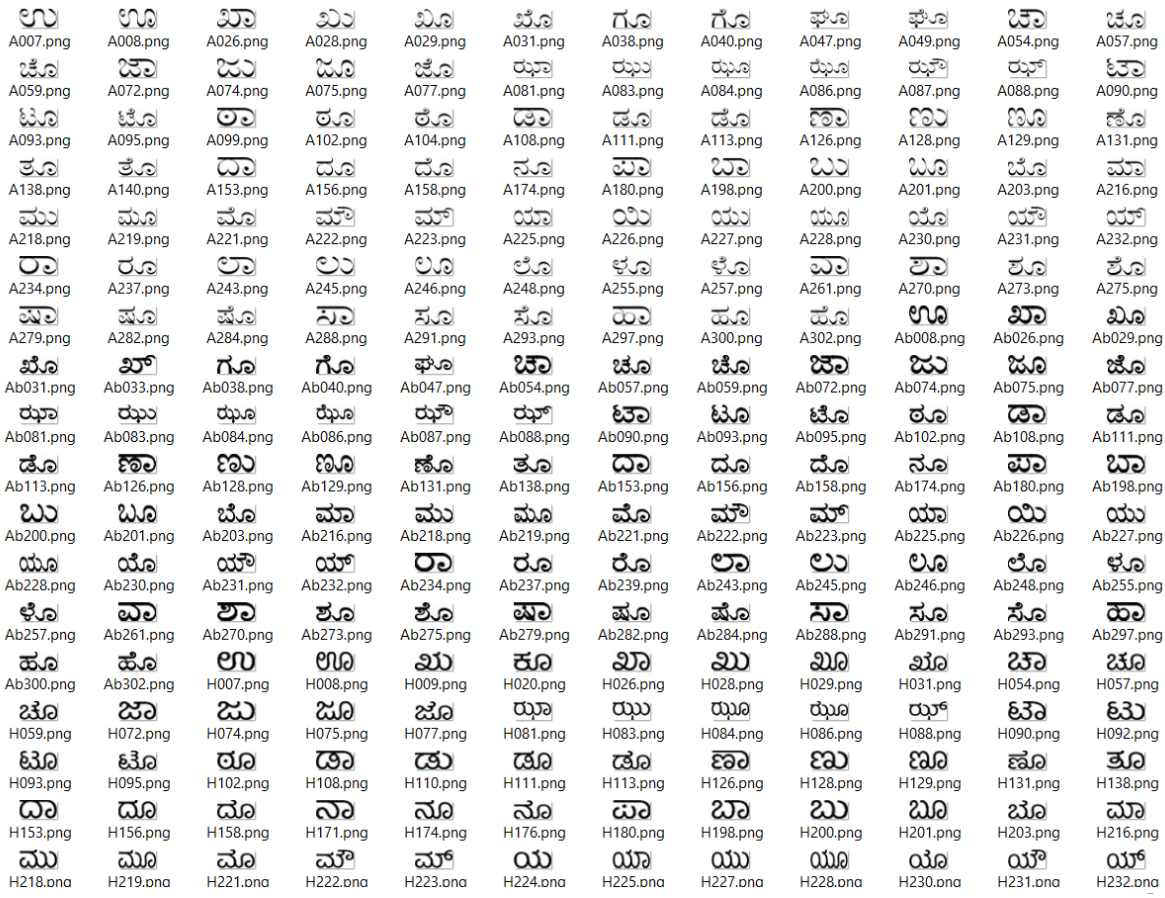


Figure 3.12: Samples of size 40 x 20

Now the dataset is ready in order to train the CNN’s.

CHAPTER 4: IMPLEMENTATION

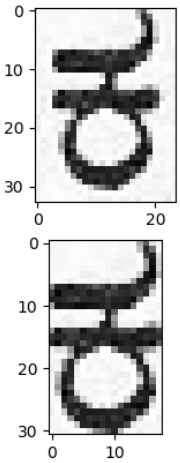
4.1 imgProc_util.py

This code contains some pre-processing functions which we will be using later on in our code:

```
import numpy as np
import cv2 as cv
import sys
pwd = sys.path[0]

def selfCrop(img,thresh=128):
    retval,imgBW = cv.threshold(img,thresh,255,cv.THRESH_BINARY)
    (H,W) = np.where(imgBW == 0)
    imgCrop = img[min(H):max(H)+1,min(W):max(W)+1]
    return imgCrop

def getBase(imgCrop):
    retval,imgCrop_BW = cv.threshold(imgCrop,128,255,cv.THRESH_BINARY)
    img_HorzEdge = np.uint8(cv.Sobel(imgCrop_BW,cv.CV_64F,0,1,ksize=3))
    hpp = np.sum(img_HorzEdge,1)/255;
    Hhalf = int(hpp.shape[0]/2)
    base = Hhalf + np.argmax(hpp[Hhalf:])
    return base

def splitChar(imgCrop,base):
    imgU = imgCrop[0:base+1,:]
    imgL = imgCrop[base+1:,:]
    if list(imgL) == []:
        return None

    retval,imgU_BW = cv.threshold(imgU,128,255,cv.THRESH_BINARY)
    (Hu,Wu) = np.where(imgU_BW == 0)
    U_Wpos = int((min(Wu)+max(Wu))/2)
    w,h = imgL.shape[::-1]
    retval,imgL_BW = cv.threshold(imgL,128,255,cv.THRESH_BINARY)
    imgL_BW[0:int(h/5),:] = 255
    (Hl,Wl) = np.where(imgL_BW == 0)
    if (list(Wl) != []):
        L_Wpos = max(Wl)
        L_Before_U = L_Wpos < U_Wpos
    else:
        L_Before_U = False

    if (L_Before_U):
        imgU = imgU[:,max(Wl)+1:]
    else:
        imgU_VertEdge = np.uint8(cv.Sobel(imgU_BW,cv.CV_64F,1,0,ksize=3))
        vppU = np.sum(imgU_VertEdge,0)/255;
        Whalf = int(vppU.shape[0]/2)
        rightEdge = Whalf + np.argmax(vppU[Whalf:])
        imgU = imgU[:,0:rightEdge+1]

    if (list(Wl) == []):
        imgL = None
    else:
        imgL = imgL[:,min(Wl):max(Wl)+1]

    return (imgU,imgL,L_Before_U)
```

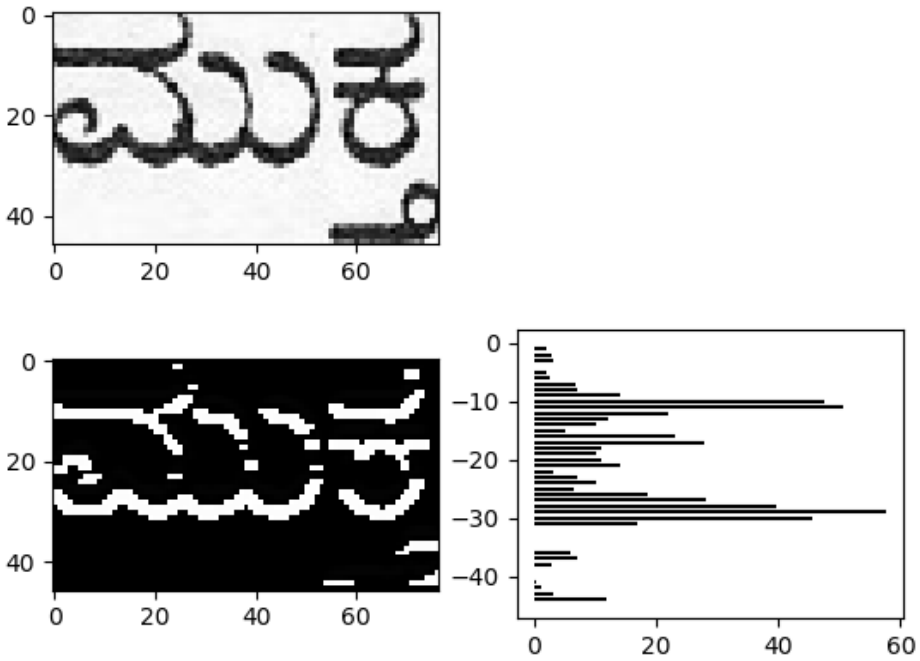
4.1 Code for imgProc_util.py

The function **selfCrop** takes an image of a word or character and trims out any extra white pixels surrounding the object in focus. A demonstration is shown below:



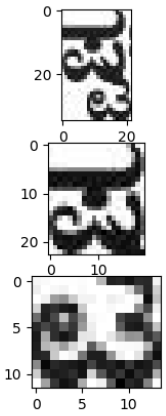
4.2 Depiction of the selfCrop code

The function **getBase** is used for identifying the base line. It works by taking an input image of a word, then thresholding and using Sobel edge detection to get all the **lower edges** in the image. It takes the horizontal sum of edge detection result and finds the maximum value in the lower half. It is demonstrated for the image below. The base line has been identified to be at position 29 on the H (height) axis.



4.3 Depiction of BaseLine Identification

The function **splitChar** is used when a character and a vattakshara are joined and we want to separate them. The function takes an input image and splits it into two images along the base line.



4.4 Depiction of splitChar

4.2 cnn_train.py

This code is used to build the CNN model, train it with the dataset and save it to a .h5 file.

```
import keras
from keras.layers import Dense, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.models import Sequential
from loadingRoutines import loadMainTrainData
import numpy as np
import matplotlib.pylab as plt
import sys
pwd = sys.path[0]
import time

batch_size = 32
num_classes = 340
epochs = 15

for i in range(4):
    # load the data set
    (x_train, y_train,img_w,img_h) = loadMainTrainData(i)
    x_test = x_train
    y_test = y_train

    #reshape the data into a 4D tensor - (sample_number, x_img_size, y_img_size, num_channels)
    x_train = x_train.reshape(x_train.shape[0], img_w, img_h, 1)
    x_test = x_test.reshape(x_test.shape[0], img_w, img_h, 1)
    input_shape = (img_w, img_h, 1)

    x_train = x_train.astype('float32')
    x_test = x_test.astype('float32')
    x_train /= 255
    x_test /= 255
    print('x_train shape:', x_train.shape)
    print(x_train.shape[0], 'train samples')
    print(x_test.shape[0], 'test samples')

    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)
```

4.5 Code for cnn_train.py

Description:

The batch_size variable represents the number of samples per update. The num_classes variable represents the number of classes, in our case the number of possible Kannada characters.

Epochs is the number of iterations over the entire dataset. Higher number of epochs means we train the network for longer duration of time, which gives higher accuracy.

Here we are training not one but 4 CNNs, with input sizes 15x20, 25x20, 30x30 and 40x20.
The function loadMainTrainData(i) returns the training data (x_train,y_train) and the input size (img_w,img_h) for the CNN labelled i. This function is defined by us in another file.

Initially we have to perform a number of transformations on the training data so as to express it in correct form (i.e. reshaping matrices, re-mapping values and such.) Then we move on to create and train the CNNs.

```
model = Sequential()
model.add(Conv2D(64, kernel_size=(5, 5), strides=(1, 1),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Flatten())
model.add(Dense(1000, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adam(),metrics=['accuracy'])

model.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1)

score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

filename = "%s/NewCNN%sx%s.h5" % (pwd,img_w,img_h)
model.save(filename)
del model
```

4.6 Code for Training the CNN's

Now we use the Keras library to create a Sequential model. In the Sequential model the layers are stacked one after another in the order of input to output.

We are implementing a CNN that begins with one convolutional layer (Conv2D) with 64 5x5 filters. This layer takes a single input image and returns 64 feature maps, on which we apply ReLU activation function. This is followed by a 2x2 max-pooling layer (MaxPooling2D) which reduces the size of the maps. Next we convert everything into a single dimension (Flatten). This becomes an input for a fully connected (Dense) layer of 1000 hidden neurons with ReLU activation function and finally we have a fully connected output layer of 340 neurons with softmax activation function.

Now we compile the model using the compile() function. We use categorical cross-entropy as our loss function. Loss function is the parameter we need to minimize using our optimization algorithm when we train our network. The optimization algorithm used is the Adam optimizer, explained in detail in the paper *Adam: A Method for Stochastic Optimization*. Also we want to monitor the accuracy at each stage so we configure the network so.

Next we do the training using the fit() function. This is the part of the code that takes the longest time to excecute. In the end we use the evaluate() function to evaluate the test loss and the test accuracy of the final trained model. Finally, we save the trained model in a .h5 file on the secondary storage, then we delete it from the RAM.

A demonstration on the console is shown below:

```
Using TensorFlow backend.
x_train shape: (2188, 15, 20, 1)
2188 train samples
2188 test samples
Epoch 1/15
2018-04-27 16:15:36.723768: I T:\src\github\tensorflow\tensorflow\core\platform\cpu_fea
2188/2188 [==============================] - 6s 3ms/step - loss: 4.6342 - acc: 0.1120
Epoch 2/15
2188/2188 [==============================] - 6s 3ms/step - loss: 2.0110 - acc: 0.5603
Epoch 3/15
2188/2188 [==============================] - 6s 3ms/step - loss: 0.8301 - acc: 0.7797
Epoch 4/15
2188/2188 [==============================] - 6s 3ms/step - loss: 0.4604 - acc: 0.8853
Epoch 5/15
2188/2188 [==============================] - 6s 3ms/step - loss: 0.2770 - acc: 0.9250
Epoch 6/15
2188/2188 [==============================] - 6s 3ms/step - loss: 0.1610 - acc: 0.9575
Epoch 7/15
2188/2188 [==============================] - 6s 3ms/step - loss: 0.1043 - acc: 0.9712
Epoch 8/15
2188/2188 [==============================] - 6s 3ms/step - loss: 0.0722 - acc: 0.9799
Epoch 9/15
2188/2188 [==============================] - 6s 3ms/step - loss: 0.0698 - acc: 0.9803
Epoch 10/15
2188/2188 [==============================] - 6s 3ms/step - loss: 0.0586 - acc: 0.9858
Epoch 11/15
2188/2188 [==============================] - 6s 3ms/step - loss: 0.0709 - acc: 0.9831
Epoch 12/15
2188/2188 [==============================] - 6s 3ms/step - loss: 0.0400 - acc: 0.9909
Epoch 13/15
2188/2188 [==============================] - 6s 3ms/step - loss: 0.0300 - acc: 0.9936
Epoch 14/15
2188/2188 [==============================] - 6s 3ms/step - loss: 0.0256 - acc: 0.9922
Epoch 15/15
2188/2188 [==============================] - 6s 3ms/step - loss: 0.0432 - acc: 0.9913
Test loss: 0.01144077842569888
Test accuracy: 0.9990859232175503
>>>
```

4.7 Console Demonstration showing successful CNN Creation

4.3 cnn_predict function

This function is called when we want to query one of our trained CNNs with an input. It takes an input image and a reference variable to a CNN which has already been loaded into the RAM by the external program that calls this function, and returns the UID of the character, obtained by taking the index of the output neuron that outputs the maximum value. Along with this we give the max. value itself as another output, to use as an accuracy metric.

```python
import numpy as np

def cnn_predict(img,model,input_shape):
    (img_w,img_h) = input_shape
    x_test = img.reshape(1,img_w, img_h, 1)
    x_test = x_test.astype('float32')
    x_test /= 255

    y_pred = model.predict(x_test)
    maxVal = np.max(y_pred,1)
    idx = np.argmax(y_pred,1)
    y_pred[0,idx] = 0
    nextMaxVal = np.max(y_pred,1)
    return idx[0], maxVal[0]
```

4.8 Code for cnn_predict function

4.4 getChar function

This function is used for identifying characters

```python
import numpy as np
import cv2 as cv
import sys
pwd = sys.path[0]
from cnn_predict import cnn_predict
import loadingRoutines


inputShapes = {0:(15,20),1:(25,20),2:(30,20),3:(40,20)}
vattInputShapes = {0:(15,15),1:(20,15),2:(30,15)}
tableM = loadingRoutines.loadMainTable()
tableV = loadingRoutines.loadVattTable()

def getChar(img,CNN):
    w,h = img.shape[::-1]
    ar = w/h
    w20 = ar*20
    if w20 <= 20:
        sizeClass = 0
    elif w20 <= 25:
        sizeClass = 1
    elif w20 < 35:
        sizeClass = 2
    else:
        sizeClass = 3
    inputShape = inputShapes[sizeClass]
    img = cv.resize(img,inputShape,interpolation = cv.INTER_CUBIC)
    (img_w,img_h) = inputShape
    (idx,metric) = cnn_predict(img,CNN[sizeClass],inputShape)
    dirga = (idx == 1)
    arka = (idx == 2)
    extraHeight = (idx >= 43 and idx <= 51) #gha
    extraHeight = extraHeight or (idx >= 62 and idx <= 70) #chha
    extraHeight = extraHeight or (idx >= 80 and idx <= 88) #jha
    extraHeight = extraHeight or (idx >= 116 and idx <= 123) #ddha
    extraHeight = extraHeight or (idx >= 143 and idx <= 151) #tha
    extraHeight = extraHeight or (idx >= 161 and idx <= 169) #dha
    extraHeight = extraHeight or (idx >= 188 and idx <= 196) #pha
    extraHeight = extraHeight or (idx >= 206 and idx <= 214) #bha
    extraHeight = extraHeight or (idx == 182 or idx == 183 or idx == 185)
    extraHeight = extraHeight or (idx == 263 or idx == 264 or idx == 266)
    extraHeight = extraHeight or (idx >= 318 and idx <= 320)
    extraHeight = extraHeight or (idx == 326)
    charseq = tableM[idx]
    if charseq[1] == 0:
        kag = False;
        charSize = 1
        char = chr(charseq[0])
    else:
        kag = True;
        charSize = 2
        char = chr(charseq[0]) + chr(charseq[1])
    flags = (kag,dirga,arka,extraHeight)
    return char,charSize,metric,flags
```

	A	B	C	D	E	F
1	Kannada	UID	Hex0	Hex1	Char0	Char1
2	ಂ	0	C82	0	3202	0
3	ಀ	1	CD5	0	3285	0
4	ಕ್	2	CB0	CCD	3248	3277
5	ಆ	3	C85	0	3205	0
6	ಆ	4	C86	0	3206	0
7	ಇ	5	C87	0	3207	0
8	ಈ	6	C88	0	3208	0
9	ಉ	7	C89	0	3209	0
10	ಊ	8	C8A	0	3210	0
11	ಋ	9	C8B	0	3211	0
12	ಎ	10	C8E	0	3214	0
13	ಏ	11	C8F	0	3215	0
14	ಐ	12	C90	0	3216	0
15	ಒ	13	C92	0	3218	0
16	ಓ	14	C93	0	3219	0
17	ಔ	15	C94	0	3220	0
18	ಕ	16	C95	0	3221	0
19	ಕಾ	17	C95	CBE	3221	3262
20	ಕಿ	18	C95	CBF	3221	3263
21	ಕು	19	C95	CC1	3221	3265
22	ಕೂ	20	C95	CC2	3221	3266
23	ಕೆ	21	C95	CC6	3221	3270
24	ಕೊ	22	C95	CCA	3221	3274
25	ಕೋ	23	C95	CCC	3221	3276
26	ಕ್	24	C95	CCD	3221	3277
27	ಖ	25	C96	0	3222	0
28	ಖಾ	26	C96	CBE	3222	3262
29	ಖಿ	27	C96	CBF	3222	3263

4.9 The getChar function code with UID Samples

This function takes an input image and a reference variable to a list of CNNs which has already been loaded into the RAM by the external program that calls this function. It classifies the image into one of 4 size classes

based on aspect ratio, then resizes and feeds this image to the respective CNN (using the cnn_predict function). The function returns the application-defined UID of the character identified which we query into a table (given above) to get the Unicode encoding (in decimal) of the aforementioned character. A Kannada character may be a single Unicode character or may actually be a combination of two Unicode characters. The output of this function is the Unicode string of the identified character, along with the size of the character, a value from 0 to 1 indicating the accuracy of the identification, and a number of flags.

4.5 getVatt function

This is used for identifying vattaksharas. It is similar to the getChar function but with only 3 size classes, and different flags.

```
def getVatt(img,CNN):
    w,h = img.shape[::-1]
    ar = w/h
    w15 = ar*15
    if w15 <= 20:
        sizeClass = 0
    elif w15 <= 25:
        sizeClass = 1
    elif w15 > 25:
        sizeClass = 2
    inputShape = vattInputShapes[sizeClass]
    img = cv.resize(img,inputShape,interpolation = cv.INTER_CUBIC)
    (img_w,img_h) = inputShape
    (idx,metric) = cnn_predict(img,CNN[sizeClass],inputShape)
    printLast = (idx == 1 or idx == 30 or idx == 31)
    charseq = tableV[idx]
    if charseq[1] == 0:
        charSize = 1
        char = chr(charseq[0])
    else:
        charSize = 2
        char = chr(charseq[0]) + chr(charseq[1])
    return char,charSize,metric,printLast
```

4.10 Code for getVatt function

4.6 getWord function

This function is used to identify an entire Kannada word from an image.
It first draws a bounding box around the word and crops out the extra whitespace around it using the selfCrop function (defined in imgProc_util.py). Then it gets the position of the base line using getBase function (defined in imgProc_util.py). Next it performs thresholding and inverting followed by dilation and then labels the

connected components i.e. characters. Any component whose height is less than one-fifth of the image height is discarded. Then it has to rearrange the labels in the correct order.

For every label in the reordered list of labels, we isolate the corresponding character from the image, then we identify it as a regular character or vattakshara character based on whether it is above or below the base line. We then use the getChar function or the getVatt function respectively.

After the character is identified it is usually appended to the end of the output string ‘word’. But for vattaksharas and some other special characters we have to insert the new character in between the word. Occasionally we get a regular character joined with a vattakshara. In this case what we do is split the image into two parts along the base line, then apply getChar function to the upper part and getVatt function to the lower part, and then prints them both, usually character first and vattakshara next. Sometimes the vattakshara is joined with the next character and we may have to print the vattakshara before the character. We implement all of these special cases in our program, with separate printing procedures for each.

Finally, after all the characters are identified we return the string ‘word’ as the output of this function.

```
import sys
pwd = sys.path[0]
import numpy as np
import cv2 as cv
from getChar import getChar,getVatt
import imgProc_util

def getWord(img,mainCNN,vattCNN,base=None,thresh=128):
    imgCrop = imgProc_util.selfCrop(img)
    if base == None:
        base = imgProc_util.getBase(imgCrop)
    imgW,imgH = imgCrop.shape[::-1]
    hThresh = imgH/5

    retval,img2 = cv.threshold(imgCrop,thresh,255,cv.THRESH_BINARY_INV)
    strel = cv.getStructuringElement(cv.MORPH_RECT,(1,int(imgH/5)))
    img2[0:int(0.9*base),:] = cv.morphologyEx(img2[0:int(0.9*base),:],cv.MORPH_DILATE,strel)
    img2[int(base+0.2*(imgH-base)):,:] = cv.morphologyEx(img2[int(base+0.2*(imgH-base)):,:],cv.MORPH_DILATE,strel)
    N,img2L = cv.connectedComponents(img2,connectivity=8)                    ## Identifies where the characters are, but labels them in wrong order
    word = ""
    nativeOrder = []
    prevSize = 0
    kag = False;
    prevKag = kag
    printLast = False
    for lbl in range(1,N):                                                   ## Routine to get position of character on w-axis for each label
        (H,W) = np.where(img2L == lbl)
        if (max(H)-min(H) < hThresh):
            continue
        Hhalf = (min(H)+max(H))/2
        Whalf = int((min(W)+max(W))/2)
        if(Hhalf > base):                                                    ## Force ordering of vattaksharas after main characters at same w-position
            nativeOrder.append((lbl,Whalf))
        else:
            nativeOrder.append((lbl,min(W)))
    reordered = sorted(nativeOrder,key = lambda x: x[1])                     ## Sort the characters in proper order, keeping labels intact
    for letter in reordered:
        lbl = letter[0]
        (H,W) = np.where(img2L == lbl)                                       ## List all the pixels constituting a certain character
        Hhalf = (min(H)+max(H))/2                                            ## Get h-position of character
        if(Hhalf > base):                                                    ## For characters below base line (vattakshara)
            mask = 255*np.uint8(img2L == lbl)                                ## Use masking to create a new image with all other characters removed
            img3 = np.bitwise_and(np.invert(imgCrop),mask)
            img3 = np.invert(img3)
            img3 = imgProc_util.selfCrop(img3)
            (vatt,size,acc,printLast) = getVatt(img3,vattCNN)                ## Get vattakshara from image
            if (acc < 0.5):
                vatt = ''
                size = 0
            if (~printLast & prevKag):                                       ## Procedure for printing vattakshara in Unicode
                prevKagChar = word[-1:]
                word = word[:-1] + vatt + prevKagChar
            else:
                word = word + vatt
            prevSize = prevSize + size
        else:
            Hmax = max(H)
            img3 = imgCrop[min(H):max(H)+1,min(W):max(W)+1]                  ## Crop out a single character from word image
            img3 = imgProc_util.selfCrop(img3)
            (char,size,acc,flags) = getChar(img3,mainCNN)                    ## Get character from image
            (kag,dirga,arka,extraHeight) = flags
            if(imgH-base > 9 and Hmax > (base+(imgH-base)/2) and ~extraHeight):   ## Test for joined vatt and character
                img3 = imgCrop[:,min(W):max(W)+1]
                res = imgProc_util.splitChar(img3,base)                      ## Split image
                if res is not None:
                    (imgU,imgL,L_Before_U) = res
                    if imgL is not None:
                        (char2,size2,acc2,flags) = getChar(imgU,mainCNN)     ## Get character from upper part of image
                        kag2 = flags[0]
                        if(acc2 > acc):
                            if(acc2 < 0.5):
                                continue
                            char = char2                                     ## Replace original with new
                            size = size2
                            kag = kag2
                            (vatt,sizeV,acc,printLast) = getVatt(imgL,vattCNN)   ## Get vattakshara from lower part of image
                            if(acc < 0.5):
                                vatt = ''
                                sizeV = 0
                            if(L_Before_U):                                  ## Procedure to print if character was connected to PREVIOUS vattakshara
                                if(~printLast & prevKag):
                                    prevKagChar = word[-1:]
                                    word = word[:-1] + vatt + prevKagChar + char
                                else:
                                    word = word + vatt + char
                                prevSize = size
                            else:                                            ## Prodecure to print if character was connected to NEXT vattakshara
                                if(~printLast & kag):
                                    kagChar = char[-1:]
                                    aksChar = char[:-1]
                                    word = word + aksChar + vatt + kagChar
                                else:
                                    word = word + char + vatt
                                prevSize = size + sizeV
                            prevKag = kag
                            continue                                         ## If character is identifiable and is not arkavattu
            if(acc < 0.5):
                continue
            if (arka):                                                       ## If character is arkavattu
                prevChar = word[-prevSize:]                                  ## Procedure for printing arkavattu in Unicode
                word = word[:-prevSize] + char + prevChar
            else:
                word = word + char                                           ## Print character normally
            if(dirga):
                prevSize = prevSize + size
            else:
                prevSize = size
            prevKag = kag
    return word
```

4.11 Code for getWord function

4.7 paragraph.py

This is the main program from which we invoke all the functions. We have not implemented a segmentation code in Python yet, so we have to first segment words from an input image using a GNU Octave program we wrote and store those images in a folder. Then we execute this code. The output is written in a text file called "paragraph.txt"

```
import keras
import sys
pwd = sys.path[0]
from time import time
import getWord
from loadingRoutines import loadCNNs
import numpy as np
import cv2 as cv
import os
import fnmatch

def output(filepath,text):
    with codecs.open(filepath,"w+","utf-16") as file:
        file.write(text)

mainCNN,vattCNN = loadCNNs()
dir = "%s/Words2" % pwd
allFiles = os.listdir(dir)
lines = list()
prefix = "*.png"
someFiles = fnmatch.filter(allFiles,prefix)

for file in someFiles:
    path = "%s/%s" % (dir,file)
    print(file)
    img = cv.imread(path,cv.IMREAD_GRAYSCALE)
    w,h = img.shape[::-1]
    if h <= 5 or w <= 5:
        continue
    word = getWord.getWord(img,mainCNN,vattCNN,thresh=100)
    lines.append(word)
space = '\x20'
text = space.join(lines)

file = "%s/paragraph.txt" % pwd
output(file,text)
```

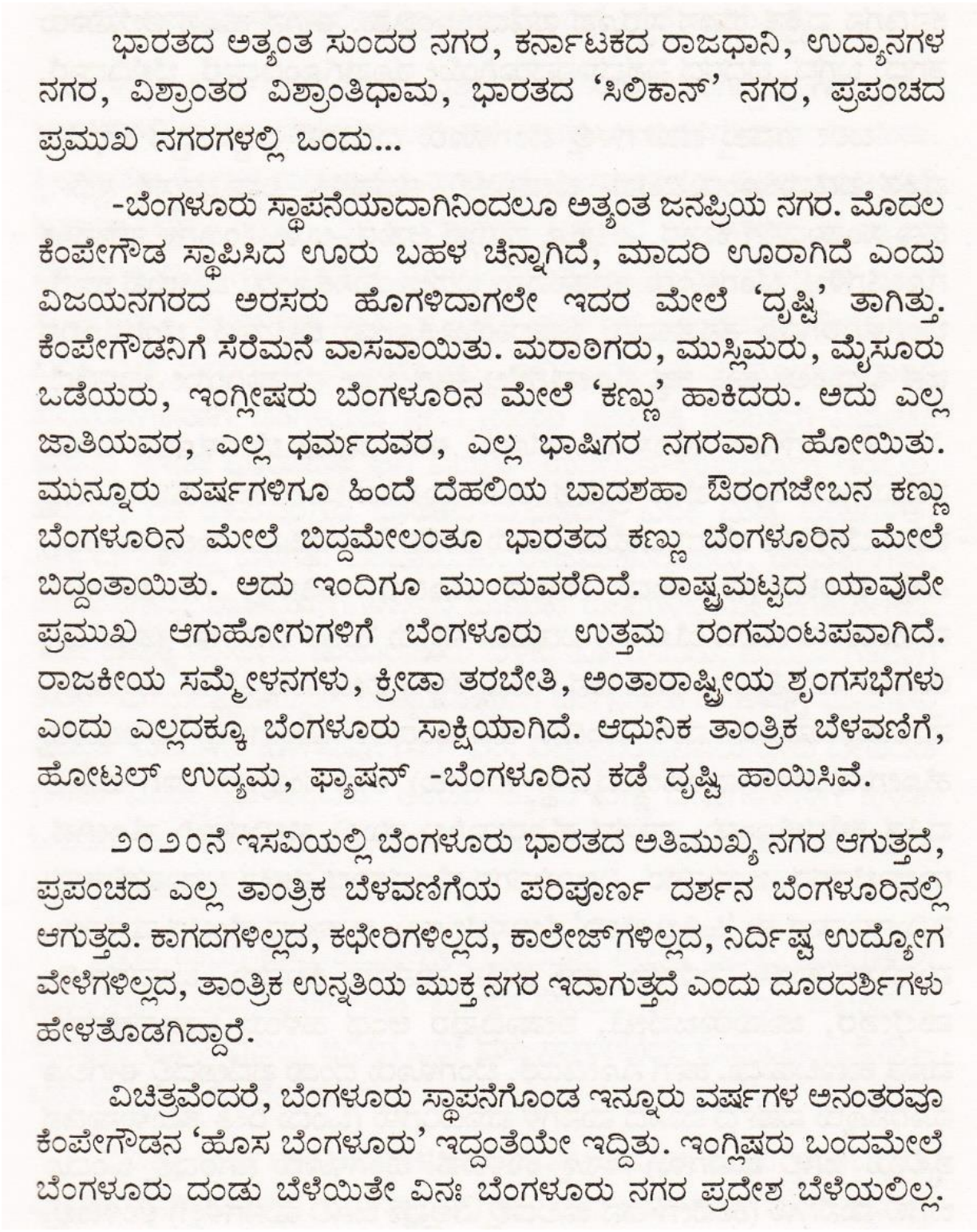
4.12 The Main Function

CHAPTER 5: RESULTS AND DISCUSSION

5.1 Sample and Output

5.1.1 Input Image

The sample image which was given as the input for the system is shown below:



ಭಾರತದ ಅತ್ಯಂತ ಸುಂದರ ನಗರ, ಕರ್ನಾಟಕದ ರಾಜಧಾನಿ, ಉದ್ಯಾನಗಳ ನಗರ, ವಿಶ್ರಾಂತರ ವಿಶ್ರಾಂತಿಧಾಮ, ಭಾರತದ 'ಸಿಲಿಕಾನ್' ನಗರ, ಪ್ರಪಂಚದ ಪ್ರಮುಖ ನಗರಗಳಲ್ಲಿ ಒಂದು...

-ಬೆಂಗಳೂರು ಸ್ಥಾಪನೆಯಾದಾಗಿನಿಂದಲೂ ಅತ್ಯಂತ ಜನಪ್ರಿಯ ನಗರ. ಮೊದಲ ಕೆಂಪೇಗೌಡ ಸ್ಥಾಪಿಸಿದ ಊರು ಬಹಳ ಚೆನ್ನಾಗಿದೆ, ಮಾದರಿ ಊರಾಗಿದೆ ಎಂದು ವಿಜಯನಗರದ ಅರಸರು ಹೊಗಳಿದಾಗಲೇ ಇದರ ಮೇಲೆ 'ದೃಷ್ಟಿ' ತಾಗಿತ್ತು. ಕೆಂಪೇಗೌಡನಿಗೆ ಸೆರೆಮನೆ ವಾಸವಾಯಿತು. ಮರಾಠಿಗರು, ಮುಸ್ಲಿಮರು, ಮೈಸೂರು ಒಡೆಯರು, ಇಂಗ್ಲೀಷರು ಬೆಂಗಳೂರಿನ ಮೇಲೆ 'ಕಣ್ಣು' ಹಾಕಿದರು. ಅದು ಎಲ್ಲ ಜಾತಿಯವರ, ಎಲ್ಲ ಧರ್ಮದವರ, ಎಲ್ಲ ಭಾಷಿಗರ ನಗರವಾಗಿ ಹೋಯಿತು. ಮುನ್ನೂರು ವರ್ಷಗಳಿಗೂ ಹಿಂದೆ ದೆಹಲಿಯ ಬಾದಶಹಾ ಔರಂಗಜೇಬನ ಕಣ್ಣು ಬೆಂಗಳೂರಿನ ಮೇಲೆ ಬಿದ್ದಮೇಲಂತೂ ಭಾರತದ ಕಣ್ಣು ಬೆಂಗಳೂರಿನ ಮೇಲೆ ಬಿದ್ದಂತಾಯಿತು. ಅದು ಇಂದಿಗೂ ಮುಂದುವರೆದಿದೆ. ರಾಷ್ಟ್ರಮಟ್ಟದ ಯಾವುದೇ ಪ್ರಮುಖ ಆಗುಹೋಗುಗಳಿಗೆ ಬೆಂಗಳೂರು ಉತ್ತಮ ರಂಗಮಂಟಪವಾಗಿದೆ. ರಾಜಕೀಯ ಸಮ್ಮೇಳನಗಳು, ಕ್ರೀಡಾ ತರಬೇತಿ, ಅಂತಾರಾಷ್ಟ್ರೀಯ ಶೃಂಗಸಭೆಗಳು ಎಂದು ಎಲ್ಲದಕ್ಕೂ ಬೆಂಗಳೂರು ಸಾಕ್ಷಿಯಾಗಿದೆ. ಆಧುನಿಕ ತಾಂತ್ರಿಕ ಬೆಳವಣಿಗೆ, ಹೋಟಲ್ ಉದ್ಯಮ, ಫ್ಯಾಷನ್ -ಬೆಂಗಳೂರಿನ ಕಡೆ ದೃಷ್ಟಿ ಹಾಯಿಸಿವೆ.

೨೦೨೦ನೆ ಇಸವಿಯಲ್ಲಿ ಬೆಂಗಳೂರು ಭಾರತದ ಅತಿಮುಖ್ಯ ನಗರ ಆಗುತ್ತದೆ, ಪ್ರಪಂಚದ ಎಲ್ಲ ತಾಂತ್ರಿಕ ಬೆಳವಣಿಗೆಯ ಪರಿಪೂರ್ಣ ದರ್ಶನ ಬೆಂಗಳೂರಿನಲ್ಲಿ ಆಗುತ್ತದೆ. ಕಾಗದಗಳಿಲ್ಲದ, ಕಛೇರಿಗಳಿಲ್ಲದ, ಕಾಲೇಜ್‌ಗಳಿಲ್ಲದ, ನಿರ್ದಿಷ್ಟ ಉದ್ಯೋಗ ವೇಳೆಗಳಿಲ್ಲದ, ತಾಂತ್ರಿಕ ಉನ್ನತಿಯ ಮುಕ್ತ ನಗರ ಇದಾಗುತ್ತದೆ ಎಂದು ದೂರದರ್ಶಿಗಳು ಹೇಳತೊಡಗಿದ್ದಾರೆ.

ವಿಚಿತ್ರವೆಂದರೆ, ಬೆಂಗಳೂರು ಸ್ಥಾಪನೆಗೊಂಡ ಇನ್ನೂರು ವರ್ಷಗಳ ಅನಂತರವೂ ಕೆಂಪೇಗೌಡನ 'ಹೊಸ ಬೆಂಗಳೂರು' ಇದ್ದಂತೆಯೇ ಇದ್ದಿತು. ಇಂಗ್ಲಿಷರು ಬಂದಮೇಲೆ ಬೆಂಗಳೂರು ದಂಡು ಬೆಳೆಯಿತೇ ವಿನಃ ಬೆಂಗಳೂರು ನಗರ ಪ್ರದೇಶ ಬೆಳೆಯಲಿಲ್ಲ.

Figure 5.1: Sample Input Image

5.1.2 Output Text

The Editable Output Text is as shown below. The errors have been highlighted manually.

ಭಾರತದ ಅತ್ಯಂತ ಸುಂದರ ನಗರ , ಕರ್ನಾಟಕದ ರಾಜಧಾನಿ, ಉದ್ಯಾನಗಳ ನಗರ, ವಿಶ್ರಾಂತರ ವಿಶ್ರಾಂತಿಧಾಮ, ಭಾರತದ ಳಸಿಲಿಕಾನ್, ನಗರ, ಪ್ರಪಂಚದ ಪ್ರಮುಖ ನಗರಗಳಲ್ಲಿ ಒಂದು. ಬೆಂಗಳೂರು ಸ್ಥಾಪನೆಯಾದಾಗಿನಿಂದಲೂ ಅತ್ಯಂತ ಜನಪ್ರಿಯ ನಗರ. ಮೊದಲ ಕೆಂಪೇಗೌಡ ಸ್ಥಾಪಿಸಿದ ಊರು ಬಹಳ ಚೆನ್ನಾಗಿದೆ, ಮಾದರಿ ಊರಾಗಿದೆ ಎಂದು ವಿಜಯನಗರದ ಅರಸರು ಹೊಗಳಿದಾಗಲೇ ಇದರ ಮೇಲೆ **ದಷ್ಟಿ** , ತಾಗಿತ್ತು. ಒಡೆಯರು, ಇಂಗ್ಲೀಷರು ಬೆಂಗಳೂರಿನ ಮೇಲೆ **ಹೆಕಣ್ಣು**, ಹಾಕಿದರು. ಅದು ಎಲ್ಲ ಜಾತಿಯವರ, ಎಲ್ಲ ಧರ್ಮದವರ, ಎಲ್ಲ ಭಾಷಿಗರ ನಗರವಾಗಿ ಹೋಯಿತು. **ಮುವ್ನೂರು** ವರ್ಷಗಳಿಗೂ ಹಿಂದೆ ದೆಹಲಿಯ ಬಾದಶಹಾ ಔರಂಗಜೇಬನ ಕಣ್ಣು ಬೆಂಗಳೂರಿನ ಮೇಲೆ ಬಿದ್ದಮೇಲಂತೂ ಭಾರತದ ಕಣ್ಣು ಬೆಂಗಳೂರಿನ ಮೇಲೆ ಬಿದ್ದಂತಾಯಿತು ಅದು ಇಂದಿಗೂ ಮುಂದುವರೆದಿದೆ **ರಾಷ್ಟ್ರಮಟ್ಟದ** ಯಾವುದೇ ಪ್ರಮುಖ ಆಗುಹೋಗುಗಳಿಗೆ ಬೆಂಗಳೂರು ಉತ್ತಮ ರಂಗಮಂಟಪವಾಗಿದೆ ರಾಜಕೀಯ ಸಮ್ಮೇಳನಗಳು, ಕ್ರೀಡಾ ತರಬೇತಿ, ಅಂತಾರಾಷ್ಟ್ರೀಯ ಶೃಂಗಸಭೆಗಳು ಎಂದು ಎಲ್ಲದಕ್ಕೂ ಬೆಂಗಳೂರು ಸಾಕ್ಷಿಯಾಗಿದೆ ಆಧುನಿಕ ತಾಂತ್ರಿಕ ಬೆಳವಣಿಗೆ, ಹೋಟಲ್ ಉದ್ಯಮ, **ಪ್ಯಷನ್** -ಬೆಂಗಳೂರಿನ ಕಡೆ ದೃಷ್ಟಿ ಹಾಯಿಸಿವೆ

೨ಂ ೨ಂನೆ ಇಸವಿಯಲ್ಲಿ ಬೆಂಗಳೂರು ಭಾರತದ ಅತಿಮುಖ್ಯ ನಗರ ಆಗುತ್ತದೆ, **ಪಪಂದ** ಎಲ್ಲ ತಾಂತ್ರಿಕ ಬೆಳವಣಿಗೆಯ ಪರಿಪೂರ್ಣ ದರ್ಶನ ಬೆಂಗಳೂರಿನಲ್ಲಿ ಆಗುತ್ತದೆ ಕಾಗದಗಳಿಲ್ಲದ, ಕಛೇರಿಗಳಿಲ್ಲದ, ಕಾಲೇಜ್ಗಳಿಲ್ಲದ, **ನಿರ್ದಿ** ಉದ್ಯೋಗ ವೇಳೆಗಳಿಲ್ಲದ, ತಾಂತ್ರಿಕ ಉನ್ನತಿಯ ಮುಕ್ತ ನಗರ ಇದಾಗುತ್ತದೆ ಎಂದು ದೂರದರ್ಶಿಗಳು ಹೇಳತೊಡಗಿದಾರೆ
ವಿಚಿತವೆಂದರೆ, ಬೆಂಗಳೂರು ಸ್ಥಾಪನೆಗೊಂಡ ಇನ್ನೂರು ವರ್ಷಗಳ ಅನಂತರವೂ ಕೆಂಪೇಗೌಡನ **ಳಹೊಸ** ಬೆಂಗಳೂರು, ಇದ್ದಂತೆಯೇ ಇದ್ದಿತು ಇಂಗ್ಲಿಷರು ಬಂದಮೇಲೆ ಕೆಂಪೇಗೌಡನಿಗೆ ಸೆರೆಮನೆ ವಾಸವಾಯಿತು. ಮರಾಠಿಗರು, ಮುಸ್ಲಿಮರು, ಮೈಸೂರು **ಂಗಳೂರು** ದಂಡು ಬೆಳೆಯಿತೇ ವಿನಃ ಬೆಂಗಳೂರು ನಗರ ಪ್ರದೇಶ ಬೆಳೆಯಲಿಲ್ಲ

5.2 Accuracy and Classification of Words

The words were classified as Easy, Medium, and Hard.

Easy words were simple words like ನಗರ. Easy words consist of only base characters.

Medium words include Kaagunithas and Arkavatthu.

The hard words consisted of the Vattaksharas and other punctuations.

The accuracy was then calculated for each category, and then as a whole.

The Accuracy Table for each category is shown below.

5.2.1 Difficulty level - Easy

1	ನಗರ,	Easy		y
2	ಒಂದು.	Easy		y
3	ಮೊದಲ	Easy		y
4	ಊರು	Easy		y
5	ಬಹಳ	Easy		y
6	ಎಂದು	Easy		y
7	ವಿಜಯನಗರದ	Easy		y
8	ಅರಸರು	Easy		y
9	ಇದರ	Easy		y
10	ಮೇಲೆ	Easy		y
11	ಅದು	Easy		y
12	ಅದು	Easy		y
13	೨೦೦	Easy		y
14	೨೦೦ನೆ	Easy		y
15	ಎಂದು	Easy		y
16	ಳಹೊಸ	Easy	ಹೊಸ	**N**
17	ವಿನಃ	Easy		y
	Accuracy =	***94.11%***		**16/17**

The easy word category gave an accuracy of ***94.11 %***.

5.2.2 Difficulty level – Medium

1	ಭಾರತದ	Medium		Y	25	ಬೆಂಗಳೂರಿನ	Medium	y	49	ದರ್ಶನ	Medium		y
2	ಸುಂದರ	Medium		y	26	ಮೇಲೆ	Medium	y	50	ಬೆಂಗಳೂರಿನಲ್ಲಿ	Medium		y
3	ಕರ್ನಾಟಕದ	Medium		y	27	ಬಿದ್ದಮೇಲಂತೂ	Medium	y	51	ನಿರ್ದಿ	Medium	ನಿರ್ದಿಷ್ಟ	N
4	ರಾಜಧಾನಿ,	Medium		y	28	ಭಾರತದ	Medium	y	52	ದೂರದರ್ಶಿಗಳು	Medium		y
5	ಳಸಿಲಿಕಾನ್,	Medium	ಸಿಲಿಕಾನ್	N	29	ಬೆಂಗಳೂರಿನ	Medium	y	53	ಹೇಳತೊಡಗಿದಾರೆ	Medium		y
6	ಬೆಂಗಳೂರು	Medium		y	30	ಮೇಲೆ	Medium	y	54	ವರ್ಷಗಳ	Medium		y
7	ಕೆಂಪೇಗೌಡ	Medium		y	31	ಇಂದಿಗೂ	Medium	y	55	ಅನಂತರವೂ	Medium		y
8	ಮಾದರಿ	Medium		y	32	ಮುಂದುವರೆದಿದೆ	Medium	y	56	ಕೆಂಪೇಗೌಡನ	Medium		y
9	ಊರಾಗಿದೆ	Medium		y	33	ಯಾವುದೇ	Medium	y	57	ಬೆಂಗಳೂರು,	Medium		y
10	ಹೊಗಳಿದಾಗಲೇ	Medium		y	34	ಆಗುಹೋಗುಗಳಿಗೆ	Medium	y	58	ಬಂದಮೇಲೆ	Medium		y
11	ಒಡೆಯರು,	Medium		y	35	ಬೆಂಗಳೂರು	Medium	y	59	ಕೆಂಪೇಗೌಡನಿಗೆ	Medium		y
12	ಬೆಂಗಳೂರಿನ	Medium		y	36	ರಂಗಮಂಟಪವಾಗಿದೆ	Medium	y	60	ಸೆರೆಮನೆ	Medium		y
13	ಮೇಲೆ	Medium		y	37	ರಾಜಕೀಯ	Medium	y	61	ವಾಸವಾಯಿತು.	Medium		y
14	ಹಾಕಿದರು.	Medium		Y	38	ತರಬೇತಿ,	Medium	y	62	ಮರಾಠಿಗರು,	Medium		y
15	ಜಾತಿಯವರ,	Medium		y	39	ಎಂದು	Medium	y	63	ದಂಡು	Medium		y
16	ಧರ್ಮದವರ,	Medium		y	40	ಆಧುನಿಕ	Medium	y	64	ಬೆಳೆಯಿತೇ	Medium		y
17	ಭಾಷಿಗರ	Medium		y	41	ಬೆಳವಣಿಗೆ,	Medium	y					
18	ನಗರವಾಗಿ	Medium		y	42	ಹೋಟಲ್	Medium	y	**Accuracy**	***96.87%***			**62/64**
19	ಹೋಯಿತು.	Medium		y	43	-ಬೆಂಗಳೂರಿನ	Medium	y					
20	ವರ್ಷಗಳಿಗೂ	Medium		y	44	ಕಡೆ	Medium	y					
21	ಹಿಂದೆ	Medium		y	45	ಹಾಯಿಸಿವೆ	Medium	y					
22	ದೆಹಲಿಯ	Medium		y	46	ಬೆಂಗಳೂರು	Medium	y					
23	ಬಾದಶಹಾ	Medium		y	47	ಭಾರತದ	Medium	y					
24	ಔರಂಗಜೇಬನ	Medium		y	48	ಬೆಳವಣಿಗೆಯ	Medium	y					

The medium word catergory gave a good accuracy of ***96.87 %.***

5.2.3 Difficulty level - Hard

1	ಅತ್ಯಂತ	Hard		y	23	ಪ್ರಮುಖ	Hard		y
2	ಉದ್ಯಾನಗಳ	Hard		y	24	ಸಮ್ಮೇಳನಗಳು,	Hard		y
3	ವಿಶ್ರಾಂತರ	Hard		y	25	ಕ್ರೀಡಾ	Hard		y
4	ವಿಶ್ರಾಂತಿಧಾಮ,	Hard		y	26	ಅಂತಾರಾಷ್ಟ್ರೀಯ	Hard		y
5	ಪ್ರಪಂಚದ	Hard		y	27	ಶೃಂಗಸಭೆಗಳು	Hard		y
6	ಪ್ರಮುಖ	Hard		y	28	ಎಲ್ಲದಕ್ಕೂ	Hard		y
7	ನಗರಗಳಲ್ಲಿ	Hard		y	29	ಸಾಕ್ಷಿಯಾಗಿದೆ	Hard		y
8	ಸ್ಥಾಪನೆಯಾದಾಗಿನಿಂದಲೂ	Hard		y	30	ತಾಂತ್ರಿಕ	Hard		y
9	ಅತ್ಯಂತ	Hard		y	31	ಉದ್ಯಮ,	Hard		y
10	ಜನಪ್ರಿಯ	Hard		y	32	ಪ್ರ್ಯಷನ್	Hard	ಫ್ಯಾಷನ್	N
11	ಸ್ಥಾಪಿಸಿದ	Hard		y	33	ದೃಷ್ಟಿ	Hard		y
12	ಚೆನ್ನಾಗಿದೆ,	Hard		y	34	ಇಸವಿಯಲ್ಲಿ	Hard		y
13	ದಷ್ಟಿ	Hard	ದೃಷ್ಟಿ	N	35	ಅತಿಮುಖ್ಯ	Hard		y
14	ತಾಗಿತ್ತು.	Hard		y	36	ಪರಿಪೂರ್ಣ	Hard		y
15	ಇಂಗ್ಲೀಷರು	Hard		y	37	ಆಗುತ್ತದೆ,	Hard		y
16	ಹೆಕಣ್ಣ,	Hard	ಕಣ್ಣ	N	38	ಪಪಂದ	Hard	ಪ್ರಪಂಚ	N
17	ಎಲ್ಲ	Hard		y	39	ಎಲ್ಲ	Hard		y
18	ಮುವ್ನೂರು	Hard	ಮುನ್ನೂರು	N	40	ತಾಂತ್ರಿಕ	Hard		y
19	ಕಣ್ಣ	Hard		y	41	ಆಗುತ್ತದೆ	Hard		y
20	ಬಿದ್ದಂತಾಯಿತು	Hard		y	42	ಕಾಗದಗಳಿಲ್ಲದ,	Hard		y
21	ರಾಷ್ಟ್ರಮಟ್ಟದ	Hard	ರಾಷ್ಟ್ರಮಟ್ಟದ	N	43	ಕಛೇರಿಗಳಿಲ್ಲದ,	Hard		y
22	ಉತ್ತಮ	Hard		y	44	ಕಾಲೇಜ್ಗಳಿಲ್ಲದ,	Hard		y
						Accuracy =	***86.36%***		**38/44**

The hard word catergory gave a relatively lower accuracy of ***86.36 %.***

5.2.4 Overall Accuracy

Hence, the overall accuracy when calculated turned out to be:

Difficulty Level	No. of Input Words	Correct Output	Incorrect Output	Accuracy
Easy	17	16	1	94.11 %
Medium	64	62	2	96.87 %
Hard	44	38	6	86.36 %
-	-	-	**Overall**	***92.45 %***

Table 5.1: Accuracy Table for given sample

5.3 Discussion

5.3.1 Results of Existing Kannada OCR Systems

The same sample image given to our system was given to the already existing OCR systems. The accuracy of each was calculated and compared.

The input image:

ಭಾರತದ ಅತ್ಯಂತ ಸುಂದರ ನಗರ, ಕರ್ನಾಟಕದ ರಾಜಧಾನಿ, ಉದ್ಯಾನಗಳ ನಗರ, ವಿಶ್ರಾಂತರ ವಿಶ್ರಾಂತಿಧಾಮ, ಭಾರತದ 'ಸಿಲಿಕಾನ್' ನಗರ, ಪ್ರಪಂಚದ ಪ್ರಮುಖ ನಗರಗಳಲ್ಲಿ ಒಂದು...

-ಬೆಂಗಳೂರು ಸ್ಥಾಪನೆಯಾದಾಗಿನಿಂದಲೂ ಅತ್ಯಂತ ಜನಪ್ರಿಯ ನಗರ. ಮೊದಲ ಕೆಂಪೇಗೌಡ ಸ್ಥಾಪಿಸಿದ ಊರು ಬಹಳ ಚಿನ್ನಾಗಿದೆ, ಮಾದರಿ ಊರಾಗಿದೆ ಎಂದು ವಿಜಯನಗರದ ಅರಸರು ಹೊಗಳಿದಾಗಲೇ ಇದರ ಮೇಲೆ 'ದೃಷ್ಟಿ' ತಾಗಿತ್ತು. ಕೆಂಪೇಗೌಡನಿಗೆ ಸೆರೆಮನೆ ವಾಸವಾಯಿತು. ಮರಾಠಿಗರು, ಮುಸ್ಲಿಮರು, ಮೈಸೂರು ಒಡೆಯರು, ಇಂಗ್ಲೀಷರು ಬೆಂಗಳೂರಿನ ಮೇಲೆ 'ಕಣ್ಣು' ಹಾಕಿದರು. ಅದು ಎಲ್ಲ ಜಾತಿಯವರ, ಎಲ್ಲ ಧರ್ಮದವರ, ಎಲ್ಲ ಭಾಷಿಗರ ನಗರವಾಗಿ ಹೋಯಿತು. ಮುನ್ನೂರು ವರ್ಷಗಳಿಗೂ ಹಿಂದೆ ದೆಹಲಿಯ ಬಾದಶಹಾ ಔರಂಗಜೇಬನ ಕಣ್ಣು ಬೆಂಗಳೂರಿನ ಮೇಲೆ ಬಿದ್ದಮೇಲಂತೂ ಭಾರತದ ಕಣ್ಣು ಬೆಂಗಳೂರಿನ ಮೇಲೆ ಬಿದ್ದಂತಾಯಿತು. ಅದು ಇಂದಿಗೂ ಮುಂದುವರೆದಿದೆ. ರಾಷ್ಟ್ರಮಟ್ಟದ ಯಾವುದೇ ಪ್ರಮುಖ ಆಗುಹೋಗುಗಳಿಗೆ ಬೆಂಗಳೂರು ಉತ್ತಮ ರಂಗಮಂಟಪವಾಗಿದೆ. ರಾಜಕೀಯ ಸಮ್ಮೇಳನಗಳು, ಕ್ರೀಡಾ ತರಬೇತಿ, ಅಂತಾರಾಷ್ಟ್ರೀಯ ಶೃಂಗಸಭೆಗಳು ಎಂದು ಎಲ್ಲದಕ್ಕೂ ಬೆಂಗಳೂರು ಸಾಕ್ಷಿಯಾಗಿದೆ. ಆಧುನಿಕ ತಾಂತ್ರಿಕ ಬೆಳವಣಿಗೆ, ಹೋಟಲ್ ಉದ್ಯಮ, ಫ್ಯಾಷನ್ -ಬೆಂಗಳೂರಿನ ಕಡೆ ದೃಷ್ಟಿ ಹಾಯಿಸಿವೆ.

೨೦೨೦ನೆ ಇಸವಿಯಲ್ಲಿ ಬೆಂಗಳೂರು ಭಾರತದ ಅತಿಮುಖ್ಯ ನಗರ ಆಗುತ್ತದೆ, ಪ್ರಪಂಚದ ಎಲ್ಲ ತಾಂತ್ರಿಕ ಬೆಳವಣಿಗೆಯ ಪರಿಪೂರ್ಣ ದರ್ಶನ ಬೆಂಗಳೂರಿನಲ್ಲಿ ಆಗುತ್ತದೆ. ಕಾಗದಗಳಿಲ್ಲದ, ಕಛೇರಿಗಳಿಲ್ಲದ, ಕಾಲೇಜ್‌ಗಳಿಲ್ಲದ, ನಿರ್ದಿಷ್ಟ ಉದ್ಯೋಗ ವೇಳೆಗಳಿಲ್ಲದ, ತಾಂತ್ರಿಕ ಉನ್ನತಿಯ ಮುಕ್ತ ನಗರ ಇದಾಗುತ್ತದೆ ಎಂದು ದೂರದರ್ಶಿಗಳು ಹೇಳತೊಡಗಿದ್ದಾರೆ.

ವಿಚಿತ್ರವೆಂದರೆ, ಬೆಂಗಳೂರು ಸ್ಥಾಪನೆಗೊಂಡ ಇನ್ನೂರು ವರ್ಷಗಳ ಅನಂತರವೂ ಕೆಂಪೇಗೌಡನ 'ಹೊಸ ಬೆಂಗಳೂರು' ಇದ್ದಂತೆಯೇ ಇದ್ದಿತು. ಇಂಗ್ಲಿಷರು ಬಂದಮೇಲೆ ಬೆಂಗಳೂರು ದಂಡು ಬೆಳೆಯಿತೇ ವಿನಃ ಬೆಂಗಳೂರು ನಗರ ಪ್ರದೇಶ ಬೆಳೆಯಲಿಲ್ಲ.

Result for KanScan:

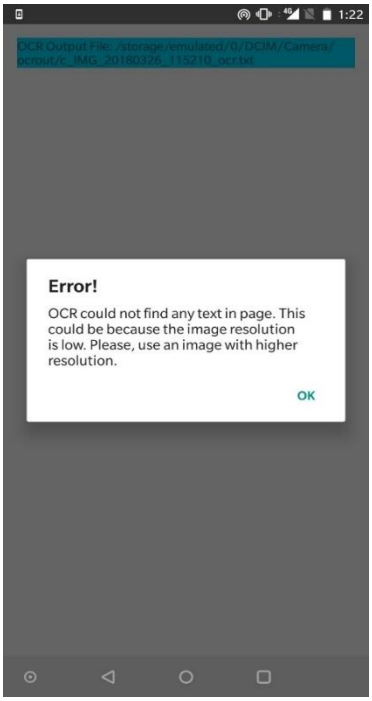


Figure 5.2: KanScan Output

Result for i2OCR:

ಭಾರತದ ಅತ್ಯಂತ **ಸುಲದರ** ನಗರ, **ಕನಾಣುಕದ** ರಾಜಧಾನಿ, **ಉಧ್ಯಾನಗಳ** ನಗರ, **ಏತ್ತಾಲತರ ಏಶ್ಯಾರಿತಿಧಾಮ**, ಭಾರತದ "ಸಿಲಿಕಾನ್ ನಗರ, **ಪ್ತಪಲಚದ** ಪ್ತಮುಖ ನಗರಗಳಲ್ಲಿ ಒಂದು...

=**ಬೆಂಗಳೂರು** ಸ್ಥಾಪನೆಯಾದಾಗಿನಿಂದಲೂ **ಅತ್ಯೆರಿತ ಜನಪ್ತಿಯ** ನಗರ. ಮೊದಲ **ಕೆಂವೇಗೌಡ** ಸ್ಥಾಪಿಸಿದ ಊರು ಬಹಳ ಚೆನ್ನಾಗಿದೆ, ಮಾದರಿ **ಲೂರಾಗಿದೆ ಎರಿದು** ವಿಜಯನಗರದ ಅರಸರು **ಡೂಗಳಿದಾಗಲೆಆ** ಇದರ **ಮೆಆಲೆ** "**ದ್ನಷ್ಟ** ತಾಗಿತ್ತು. **ಕೆರಿಪೆಆಗೌಡನಿಗೆ** ಸೆರೆಮನೆ ವಾಸವಾಯಿತು. ಮರಾಠಿಗರು, ಮುಸ್ಲಿಮರು, ಮೈಸೂರು ಒಡೆಯರು, **ಇಲಗ್ಲೀಷರು ಬೆರಿಗಳವಿರಿನ ಮೆಅಲೆ** "**ಕಣ್ಣಾ** ಹಾಕಿದರು. ಅದು **ಎಲಸೂ** ಜಾತಿಯೆವರ, ಎಲ್ಲ ಧರ್ಮದವರ, ಎಲ್ಲ ಭಾಷಿಗರ ನಗರವಾಗಿ **ಹೊಅಯಿತು**.

ಮುನ್ನೂರು ವರ್ಷಗಳಿಗೂ **ಹಿಲದೆ** ದೆಹಲಿಯ ಬಾದಶಹಾ ಔರಂಗಜೇಬನ ಕಣ್ಣು **ಬೆಲಗಳುಶಿರಿನ ಮೆಆಲೆ ಬಿದ್ದಮೆಆಲಂತೂ** ಭಾರತದ ಕಣ್ಣು **ಬೆರಿಗಳವಿರಿನ ಮೆಆಲೆ ಬಿದ್ದಲತಾಯಿತು**. ಅದು **ಇರಿದಿಗೊ ವಬಂದುವರೆದಿದೆ**. ರಾಷ್ಟ್ರಮಟ್ಟದ ಯಾವುದೇ ಪ್ತಮುಖ **ಆಗುಹುರಿರಿಗುಗಳಿಗೆ** ಬೆಂಗಳೂರು ಉತ್ತಮ ರಂಗಮಂಟಪವಾಗಿದೆ.

ರಾಜಕೀಯ **ಸಮೆಗೈಳನಗಳಎ**, ಕ್ರೀಡಾ ತರಬೇತಿ, **ಅರಿತಾರಾಷ್ಟ್ರಿಯೆ ಸ್ಪಂಗಸಭೆಗಳಎ ಎಲದು** ಎಲ್ಲದಕ್ಕೂ **ಬೆಲಗಳಣುರು** ಸಾಕ್ಷಿಯಾಗಿದೆ. ಆಧುನಿಕ **ತಾರಿತ್ತಿಕ** ಬೆಳವಣಿಗೆ, **ಹೊಳಟಲ್** ಉದ್ಯಮ, **ಫಾಶೆಷನ್** =ಬೆಂಗಳೂರಿನ ಕಡೆ **ವೃಷ್ಟಿ** ಹಾಯಿಸಿವೆ.

೨೦೨೦ನೆ ಇಸವಿಯಲ್ಲಿ 'ಬೆಂಗಳೂರು ಭಾರತದ ಅತಿಮುಖ್ಯ ನಗರ ಆಗುತ್ತೆದೆ,
ಪ್ರಪರಿಚದ ಎಲ್ಲ **ತಾಲತ್ತಿಕ** ಬೆಳವಣಿಗೆಯ **ಪರಿಪುರ್ನಿರ್ನಿ** ದರ್ಶನ **ದೆರಿಗಳಪುರಿನಲ್ಲಿ**
ಆಗುತ್ತದೆ. ಕಾಗದಗಳಿಲ್ಲದ, ಕಛೇರಿಗಳಿಲ್ಲದ, **ಕಾಲೆಳಜ್ಜಳಿಲ್ಲದ**, **ನಿರ್ದಿಷ್ಟ ಉದೊಶೆಳಗ**
ವೇಳೆಗಳಿಲ್ಲದ, **ತಾರಿತ್ತಿಕ** ಉನ್ನೆತಿಯ ಮುಕ್ತ ನಗರ **ಇದಾಗುತ್ತಂ ಎಲದು** ದೂರದರ್ಶಿಗಳು
ಹೆಆಳೆತೊಡಗಿದ್ದಾರೆ.
ಏಚಿತ್ತವೆಂದರೆ, **ಬೆಂಗಳೂರು ಸ್ಥಾಪನೆಗೆನಿಂಡ** ಇನ್ನೂರು ವರ್ಷಗಳ **ಅನಲತರವೊ**
ಕೆಂಪೇಗೌಡನ "ಹೊಸ **ಬೆಲಗಳೊರು**' **ಇದ್ದಂತೆಯಆ** ಇದ್ದಿತು. ಇಂಗ್ಲಿಷರು **ಬಲದಮೆಆಲೆ**
ಬೆಂಗಳೂರು ದಂಡು ಬೆಳೆಯಿತೇ **ಎನ**: **ಬೆಲಗಳುಎರು** ನಗರ ಪ್ರದೇಶ ಬೆಳೆಯಲಿಲ್ಲ.

5.3.2 Overall Accuracy Comparison Table

System	Accuracy
Mobile App – *KanScan*	**-no clear output-**
Website – i2OCR	**60 % approx.**
Our System	**92.45 %**

Table 5.2: Comparison of Accuracy of all Systems

5.4 More samples and accuracy comparison

Sample 1

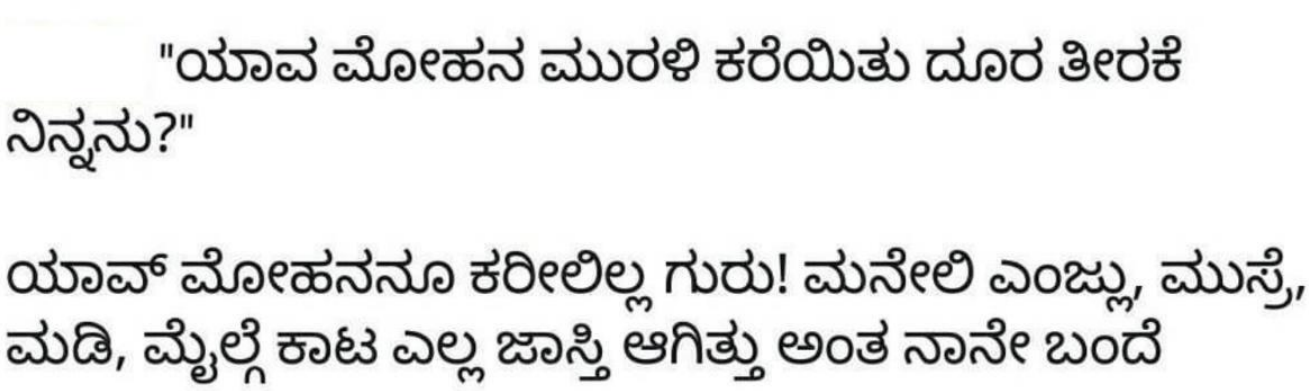


Figure 5.3: Sample Input Image 1

- i2OCR Editable Output (the bolded words are those with errors from the original sample) -

  "ಯಾವ **ಮೊಆಹನ** ಮುರಳಿ ಕರೆಯಿತು ದೂರ **ತಿಆರಕೆ**

  ನಿನ್ನೆನುಳ್

ಯಾವ್ **ಮೊಆಹನನೂ ಕರೀಲಿಲ್ಲ** ಗುರು! ಮನೇಲಿ **ಎರಿಜುಸ್ಸೂ**, **ಮುಸ್ಪೆ**,
ಮಡಿ, ಮೈಲ್ಗೆ ಕಾಟ ಎಲ್ಲ ಜಾಸ್ತಿ ಅಗಿತ್ತು **ಅರಿತ** ನಾನೇ **ಬರಿದೆ**

- Our Kannada OCR Output with errors bolded –
ಯಾವ ಮೋಹನ ಮುರಳಿ ಕೆರೆಯಿತು ದೂರ ತೀರಕೆ ನಿನ್ನನು?

ಯಾವ್ ಮೋಹನನೂ ಕೆರೀಲಿಲ್ಲ ಗುರು! ಮನೇಲಿ ಎಂಜ್ಜು. ಮುಸ್ಪೆ ಮಡಿ ಮೈಲ್ಗೆ ಕಾಟ ಎಲ್ಲ ಜಾಸ್ತಿ ಆಗಿತ್ತು ಅಂತ ನಾನೇ ಬಂದೆ

Sample 2:

ಒಂದು ಮಳೆಬಿಲ್ಲು... ಒಂದು ಮಳೆಮೋಡ....
ಹೇಗೊ ಜೊತೆಯಾಗಿ... ತುಂಬಾ ಸೊಗಸಾಗಿ...
ಏನನೋ ಮಾತಾಡಿವೆ...
ಭಾವನೆ...ಬಾಕಿ ಇದೆ...
ತೇಲಿ ನೂರಾರು ಮೈಲಿಯೂ...
ಸೇರಲು ಸನಿ-ಸನಿಹ...

ಒಬ್ಬ ಸಿಹಿ ಪೊಂಗಲ್ಲು... ಒಬ್ಬ ಬೊಂಬಾಯ್ ಬೋಂಡ...
ಹೀಗೆ ಜೊತೆಯಾಗಿ... ತುಂಬಾ ಮಡಿಯಾಗಿ...
ಮೂರೆಳೆಗಳು ಬಡೀಸೂತಿವೆ...
ಅದ್ರಲ್ಲೊಂದು ಕಾಳ್ ಹಾಕ್ತಿದೆ...
ಕಾದಿದಕೂ ಮೂರ್ನಾಲ್ಕು ಪಂಕ್ತಿಯೂ..
ಆಗಿದೆ ಸಾರ್-ಸಾರ್ಥಕ...

Figure 5.4: Sample Input Image 2

- i2OCR Editable Output (the bolded words are those with errors from the original sample) -

ಒಲದು ಮಳೆಬಿಲ್ಲು **ಒಲದು ಮಳೆಮೆರೀಡ**... ಹೇಗೂ ಜೊತೆಯಾಗಿ... ತುಂಬಾ **ಹೊಗಪಾಗಿ**...
ಏವೆಮೊ? ಮಾತಾಡಿವೆ...
ಭಾವೆನೆಬಂಚಾಕಿ ಇದೆ...
ತೆಆಲಿ ನೂರಾರು ಮೃಲಿಯೂ...
ಸೆಳಿರಲು ಹೆನಿಪನಿಹ...
ಒಬ್ಬ ಸಿಹಿ ಪೊಂಗಲ್ಲು ಒಬ್ಬ **ಬೊಲಬಾಯ್ ಬೊಲಾಡ**...
ಹಿಳಿಗೆ ಜೊತೆಯಾಗಿ... **ತುಲಬಾ** ಮಡಿಯಾಗಿ...

ಮೂರೆಳೆಗಳು ಬಡೀಸೂತಿವೆ...

ಅದ್ನಲೊಸ್ಸೂಲದು ಕಾಳ್ ಹಾಕ್ತಿದೆ...
ಕಾದಿದಕೂ **ಮೂರ್ನಾಲುಕಿ ಪಲಕ್ತಿಯೂ**..
ಆಗಿದೆ **ಷಾರ್ ಷಾರ್ಥೆಕೆಣು**

- Our Kannada OCR Output with errors bolded –
ಒಂದು ಮಳೆಬಿಲ್ಲು ಒಂದು ಮಳೆಮೋಡ ಹೇಗೊ ಜೊತೆಯಾಗಿ... ತುಂಬಾ ಸೊಗಸಾಗಿ...
ಏನನೋ ಮಾತಾಡಿವೆ... ಭಾವನೆ ಬಾಕಿ ಇದೆ ತೇಲಿ ನೂರಾರು ಮೈಲಿಯೂ
ಸೇರಲು ಸನಿ-ಸನಿಹ... ಒಬ್ಬ ಸಿಹಿ ಪೊಂಗಲ್ಲು ಒಬ್ಬ ಬೊಂಬಾಯ್ ಬೋಂಡ
ಹೀಗೆ ಜೊತೆಯಾಗಿ... ತುಂಬಾ ಮಡಿಯಾಗಿ... ಮೂರೆಳೆಗಳು ಬಡೀಸೂತಿವೆ..
ಅದ್ರಲ್ಲೊಂದು ಕಾಳ್ ಹಾಕ್ತಿದೆ ಕಾದಿದಕೂ ಮೂರ್ನಾಲ್ಕು
ಪಂಕ್ತಿಯೂ ಆಗಿದೆ ಸಾರ್-**ಸಾರ್ಥಕ**...

Sample 3:

ನೀನೆಂದರೆ ನನ್ನೊಳಗೆ, ಏನೋ ಒಂದು ಸಂಚಲನ
ನಾ ಬರೆಯದ ಕವಿತೆಗಳಾ, ನೀನೇ ಒಂದು ಸಂಕಲನ
ಓ ಜೀವವೇ ಹೇಳಿಬಿಡು, ನಿನಗೂ ಕೂಡ ಹೀಗೇನಾ?

ಓ ನಲ್ಮೆಯಾ ನಾವಿಕನೆ, ಎಂದು ನಿನ್ನ ಆಗಮನ?
ನೀ ಎಲ್ಲಿಯೇ ಅಡಗಿದರೂ, ಅಲ್ಲೇ ನನ್ನ ಈ ಗಮನ
ಈ ಪ್ರೀತಿಯ ಪರಿಣಾಮ, ನಿನಗೂ ಕೂಡ ಹೀಗೆನಾ?

ಓ ಪೂಜೆಯ ನೈವೇದ್ಯವೇ, ಎಂದು ನಿನ್ನ ಆಗಮನ?
ನೀ ಎಂದಿಗೆ ಬರಲಿಹೆಯೋ, ಅಲ್ಲೇ ನನ್ನ ಈ ಗಮನ
ಈ ಹಸಿವೆಯಾ ಪರಿಣಾಮ, ನಿಮಗೂ ಕೂಡ ಹೀಗೆನಾ?

Figure 5.5: Sample Input Image 3

- i2OCR Editable Output (the bolded words are those with errors from the original sample) -

ನಿ ಇನೆಲದರೆ ವೆನ್ನೊಳಗೆ, **ಏನೊ**? **ಒಲದು** ಸಂಚಲನ

ನಾ ಬರೆಯದ **ಕಏತೆಗಳಾ**, ನೀನೇ ಒಂದು ಸಂಕಲನ
ಓ **ಜಿಅವವೆಆ** ಹೇಳಿಬಿಡು, ನಿನಗೂ ಕೊಡ **ಹಿಳಿಗೆಆನಾ**?

ಓ **ವೆಲ್ವೆಯಾ ನಾಏಕಸೆ**, **ಬಂದು** ನಿನ್ನ ಆಗಮನ?
ನೀ **ಎಲ್ಲಿಯೆಳಿ** ಅಡಗಿದರೂ, **ಅಪ್ಲೇ** ನನ್ನ ಈ ಗಮನ
ಈ **ಪ್ತಿಅತಿಯ** ಪರಿಣಾಮ, ನಿನಗೂ **ಕೊಡ** ಹೀಗೆನಾ?

ಓ ಪೂಜೆಯ **ನೈವೇದಶೆವೇ**, **ಎಲದು** ನಿನ್ನ ಆಗಮನ?
ನೀ **ಎಲದಿಗೆ ಬರಲಿಹೆಯೊಆ**, ಅಲ್ಲೇ **ವೆನ್ನ** ಈ ಗಮನ
ಈ ಹಸಿವೆಯಾ ಪರಿಣಾಮ, ನಿಮಗೂ ಕೊಡ **ಹಿಳಿಗೆನಾ**?

- Our Kannada OCR Output with errors bolded –

ನೀನೆಂದರೆ ನನ್ನೊಳಗೆ. ಏನೋ ಒಂದು ಸಂಚಲನ
ನಾ ಬರೆಯದ ಕವಿತೆಗಳಾ ನೀನೇ ಒಂದು **ಸಂಕೆಲನ**
ಓ ಜೀವವೇ ಹೇಳಿಬಿಡು ನಿನಗೂ ಕೂಡ ಹೀಗೇನಾ?

ಓ ನಲ್ಮೆಯಾ ನಾವಿಕನೆ ಎಂದು ನಿನ್ನ ಆಗಮನ?
ನೀ ಎಲ್ಲಿಯೇ ಅಡಗಿದರೂ ಅಲ್ಲೇ ನನ್ನ ಈ ಗಮನ
ಈ ಪ್ರೀತಿಯ ಪರಿಣಾಮ ನಿನಗೂ ಕೂಡ ಹೀಗೆನಾ?

ಓ ಪೂಜೆಯ ನೈವೇದ್ಯವೇ. ಎಂದು ನಿನ್ನ ಆಗಮನ?
ನೀ ಎಂದಿಗೆ **ಬರಲಿಹೆಯೋ**! ಅಲ್ಲೇ ನನ್ನ ಈ ಗಮನ
ಈ ಹಸಿವೆಯಾ ಪರಿಣಾಮ ನಿಮಗೂ ಕೂಡ ಹೀಗೆನಾ?

Overall Accuracy for all the three samples have been formatted below for both the systems.

Sample Number	Number of Words	i2OCR Accuracy	Our OCR Accuracy
1	23	14/23 = **60.9%**	23/23 = **100%**
2	38	18/38 = **47.36%**	37/38 = **97.36%**
3	55	35/55 = **63.63%**	53/55 = **96.36%**

Overall Accuracy	**57.3 %**	**97.9%**

Table 5.3: Overall Accuracy for three samples

Observation from the above results:

There is an increase in accuracy of around **40** percent from the already existing popular OCR system to our Kannada OCR System.

CHAPTER 6: CONCLUSION AND FUTURE WORK

6.1 Conclusion

Using CNN (Convolutional Neural Networks) in OCR (Optical Character Recognition) systems is found to be a very reliable method of converting an image to a text document. Template matching turned out to be an obsolete practice in this case due to lack of accuracy and greater runtime.

At the end of this project, we were able to construct an OCR system for the Kannada language with an accuracy of over 90 %. The system is designed to work with only printed documents containing just Kannada characters. The runtime for an article of a minimum of 100 words was found to be less than two seconds, which is majorly lesser than the existing systems available in the market.

6.2 Future Work

The field of Character Recognition has still got a lot to offer. Many studies are still necessary to better understand and improve the performance of the OCR system. So far, we have been able to provide a system that can produce results of accuracy higher than the existing system.

There is a huge demand for image scanners with embedded OCR systems because of the increased demand for OCR in automation and publishing applications. As of 1996, the U.S market size for electronic imaging products was pegged at a whopping $3.2 billion. This market is observed to be growing at an annual rate of 16%. The price for OCR systems includes re-processing and post-processing of images which shows that the OCR systems include much more than pure recognology.

As of now, we have worked only on the Kannada language, but there exists an opportunity to work on other Regional languages as well.

Our future work on this project happens to develop an application software that is made available to the users at the tip of their hands. That was the core objective of our project, to make the service easy to use. All the user has to do is put in the image, let the system do its work, and view the editable text file.

CHAPTER 7: REFERENCES

[1] HR Mamatha, S Sucharitha, Srikanta Murthy, “Multi-font and Multi-size Kannada Character Recognition based on the Curvelets and Standard Deviation”, International Journal of Computer Applications, Foundation of Computer Science, New York, USA, 2011.

[2] R Prajna, VR Ramya, HR Mamatha “A study of different text line extraction techniques for multi-font and multi-size printed kannada documents”, International Journal of Computer Applications, Foundation of Computer Science, 2015.

[3] B.M.Sagar, Dr.Shobha G & Dr. Ramakanth kumar P, "OCR for printed kannada text to Machine editable format using Database approach", 9th WSEAS International Conference on AUTOMATION and INFORMATION(ICAI'08), Bucharest, Romania, June24- 26,2008.

[4] M.K Jindal, R. K. Sharma & G.S. Lehal, "Segmentation of Horizontally Overlapping Lines in Printed Indian Scripts", International Journal of Computational Intelligence Research. ISSN 0973-1873 Vol.3, No.4 (2007), pp. 277–286

[5] Vijaya Kumar Koppula & Atul Negi , "Using Fringe Maps for Text Line Segmentation in Printed or Handwritten Document Images", In the proceedings of 2010 Second Vaagdevi International Conference on Information Technology for Real World Problems,2010, pp83-88.

[6] Ashwin T.V and P.S Sastry, “A font and size independent OCR system for printed Kannada using SVM”, Sadhana, vol. 27, Part 1, February 2002, pp. 35–58.

[7] Anil. K. Jain, “Feature Extraction methods for Character Recognition – A survey”

[8] K. Indira, S. Sethu Selvi, “Kannada Character Recognition System: A Review”

[9] Netravati Belagali, Shanmukhappa A. Angadi, “OCR for Handwritten Kannada Language Script”

[10] Rafael C. Gonzalez, Richard E. Woods & Steven L. Eddins , (2009) "Digital Image Processing using MATLAB" , Indian Edition,2009,pp 348-361.

- **http://cs231n.github.io/convolutional-networks/#overview**
- **https://en.wikipedia.org/wiki/Convolutional_neural_network**
- **http://adventuresinmachinelearning.com/keras-tutorial-cnn-11-lines/**