**HAND-WRITTEN TEXT RECOGNITION**

**SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF**

**BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING**

**(2017-21)**

**GRAPHIC ERA DEEMED TO BE UNIVERSITY,DEHRADUN**

**UNDERTAKEN AT**

**GEOPIC**

**ONGC DEHRADUN**

**Submitted by:**

**Jagrati Bhatnagar**

**Under the Guidance of:**

**Mr. Ginminthang Buhril**

**Ch. Manager (Programming)**

**GEOPIC**

**ONGC,Dehradun**

# CERTIFICATE



This is to certify that **Jagrati Bhatnagar** has successfully completed the Report entitled by **“*Handwritten Text Recognition*”**, in partial fulfillment of the degree of Bachelor of Technology in Computer Science and Engineering from 3 June 2019 to 3 July 2019 at Geopic ONGC. This is an excellent report of her own work carried out by her. She has completed this report with excellent efforts and dedicated spirit. I truly appreciate her sincerity and skills she devoted to this report.

**Mr. Ginminthang Buhril**

**Ch. Manager (Programming) . GEOPIC, ONGC.**

# DECLARATION

I hereby declare that I have completed this project work under the guidance of my Mentor at GEOPIC ONGC Dehradun,Mr. Ginminthang Buhril. This report is being submitted in partial fulfillment for the requirement of the degree of Bachelor of Technology (CSE) (2017-21).

I hereby declare that all the statements made in this report are true, complete and correct to the best of my knowledge and belief.

Thanking you

**Jagrati Bhatnagar**

**ACKNOWLEDGEMENT**

We wish to expresser sincere gratitude to **Mr. Vijay Kumar Sharma**, Chief General Manager (Programming), GEOPIC SW, for providing us an opportunity to do our internship and project work in “**OIL AND NATURAL GAS CORPORATION LTD.**”

We sincerely thank Mr. **Ginminthang Buhril** for his guidance and encouragement in carrying out this project work. We also wish to express our gratitude to the officials and other staff members of ONGC LTD who rendered their help during the period of the project work.

We thank Oil and Natural Gas Corporation Limited, Dehradun for giving us an opportunity to have an industrial exposure under the guidance of the experts.

We also thank all those who have directly or indirectly helped us during the tenure of our training.

Sincerely thanking all of the above once again, we hope to continue to take the guide from the aforementioned in near future. It has been a great experience for all of us.

Thanking You,

Jagrati Bhatnagar

# ABOUT ONGC



Oil and Natural Gas Corporation Limited (ONGC) is an

Indian [multinational](https://en.wikipedia.org/wiki/Multinational_corporation) [oil](https://en.wikipedia.org/wiki/Petroleum) and [gas](https://en.wikipedia.org/wiki/Natural_gas) company headquartered in [Dehradun, India.](https://en.wikipedia.org/wiki/Dehradun,_India) It is one of the largest Asia-based oil and gas exploration and production companies, and produces around 72% of India's [crude oil](https://en.wikipedia.org/wiki/Petroleum) (equivalent to around 30% of the country's total demand) and around 48% of its [natural gas.](https://en.wikipedia.org/wiki/Natural_gas) It is one of the largest publicly traded companies by [market capitalization](https://en.wikipedia.org/wiki/Market_capitalization) in India. ONGC has been ranked 357th in the [Fortune Global 500](https://en.wikipedia.org/wiki/Fortune_Global_500) list of the world's biggest corporations for the year 2012. It is also among the Top 250 Global Energy Company by [Platts.](https://en.wikipedia.org/wiki/Platts)

ONGC was founded on 14 August 1956 by the Indian state, which currently holds a 69.23% equity stake. It is involved in exploring for and exploiting hydrocarbons in 26 sedimentary basins of India, and owns and operates over 11,000 kilometers of pipelines in the country. Its international subsidiary ONGC Videsh currently has projects in 15 countries. ONGC has discovered 6 of the 7 commercially-producing Indian Basins, in the last 50 years, adding over 7.1 billion tonnes of In-place Oil & Gas volume of hydrocarbons in Indian basins.

**ONGC in Dehradun**

The headquarters of Oil & Natural Gas Corporation LTD (ONGC) is situated in Dehradun along with the following offices.

* Tel Bhavan (Head Quarter)
* KeshavDevMalviya Institute of Petroleum Exploration (KDMIPE)
* Institute of Drilling Technology (IDT)
* ONGC Academy (formerly Institute of Management Development (IMD))
* Geo Data Processing and Interpretation Center (GEOPIC)
* Exploration & Development Directorate

**PROFILE OF THE ORGANIZATION**

**INTRODUCTION:**

Oil and Natural Gas Corporation Limited (ONGC) was incorporated on June 23, 1993. It is an Indian public sector petroleum company. It is a Fortune Global 500 company ranked 335th, and contributes 77% of India's crude oil production and 81% of India's natural gas production.

It is the highest profit-making corporation in India. It was set up as a commission on August 14, 1956. Indian government holds 74.14% equity stake in this company. ONGC is one of Asia's largest and most active companies involved in exploration and production of oil. It is involved in exploring for and exploiting hydrocarbons in 26 sedimentary basins of India. ONGC has produced more than 600 million metric tonnes of crude oil and supplied more than 200 billion cubic meters of gas since its inception, thus fuelling the increasing energy requirements of the Indian economy. Today, ONGC is the most valuable company in India, contributing 77 percent of India’s crude oil production and 81 per cent of India’s natural gas production. To sustain this growth, ONGC has drawn up ambitious strategic objectives, which include doubling the oil and gas reserves.

Having accreted six billion tonnes oil and oil equivalent reserves in its first 45 years of operation, ONGC now aims to double these reserves by 2020. The second strategic objective is to augment the global recovery factor from the existing 28 per cent to the global norm of 40 per cent in next 20 years

**MISSION AND VISION OF ONGC:**

To be a world-class Oil and Gas Company integrated in energy business with dominant Indian leadership and global presence.

**Contents**

1. Introduction

1.1 Objective of the Project

1.2 Approach

2. Machine Learning

2.1 Introduction

2.2 Creating and training of datasets

3. Image processing involved in character recognition

3.1 Pre-processing of sample image

3.2 Feature extraction

4. Operations

4.1 CNN

4.2 RNN

4.3 CTC

4.4Data

4.5 Output of the operations

5. GUI

4.1 About tkinter

4.2 Screenshots of GUI

6. Simulation and Results

6.1 Simulation

6.2 Results

7. Implementation and project snapshot

7.1 implementation

7.2 snapshots

8. Conclusion and future improvements

8.1 Conclusion

8.2 Future scope of this project

**ABSTRACT**

In todays’ world advancement in sophisticated scientific techniques is pushing further the limits of human outreach in various fields of technology. One such field is the field of character recognition commonly known as OCR (Optical Character Recognition).

In this fast paced world there is an immense urge for the digitalization of printed documents and documentation of information directly in digital form. And there is still some gap in this area even today. OCR techniques and their continuous improvisation from time to time is trying to fill this gap. This project is about devising an algorithm for recognition of hand written characters also known as HTR (Handwritten Text Recognition) leaving aside types of OCR that deals with recognition of computer or typewriter printed characters.

A novel technique is proposed for recognition English language characters using Machine Learning including the schemes of feature extraction of the characters and implemented. The persistency in recognition of characters by the ML was found to be more than 90% of times.

**CHAPTER 1**

**INTRODUCTION**

This project, ‘Handwritten Text Recognition’ is a software algorithm project to recognize any hand written character efficiently on computer with input is either an old optical handwritten text image or currently provided through camera.

Character recognition, usually abbreviated to optical character recognition or shortened OCR, is the mechanical or electronic translation of images of handwritten, typewritten or printed text (usually captured by a scanner) into machine-editable text. It is a field of research in pattern recognition, artificial intelligence and machine vision. Though academic research in the field continues, the focus on character recognition has shifted to implementation of proven techniques. Optical character recognition is a scheme which enables a computer to learn, understand, improvise and interpret the written or printed character in their own language, but present correspondingly as specified by the user. Optical Character Recognition uses the image processing technique to identify any character computer/typewriter printed or hand written. A lot of work has been done in this field. But a continuous improvisation of OCR techniques is being done based on the fact that algorithm must have higher accuracy of recognition, higher persistency in number of times of correct prediction and increased execution time.

The idea is to device efficient algorithms which get input in digital image format. After that it processes the image for better comparison. Then after the processed image is compared with already available set of font images. The last step gives a prediction of the character in percentage accuracy.

**1.1 OBJECTIVE OF THE PROJECT**

The objective of this project is to identify handwritten text with the use of machine learning. We have to construct suitable datasets and train it properly. The program should be able to extract the characters one by one and map the target output for training purpose. After automatic processing of the image, the training dataset has to be used to train “classification engine” for recognition purpose. The program code has to be written in python and supported with the usage of Graphical User Interface (GUI).

**1.2 APPROACH**

To solve the defined handwritten text recognition problem of classification we used Anaconda computation software. The computation code is divided into the next categories:

 Pre-processing of the image

 Feature extraction

 Creating a dataset

 Training & Testing of the network

 Recognition

**CHAPTER 2**

**Machine Learning**

**2.1 INTRODUCTION**

Lecun et. al focused on using gradient-based learning techniques using multi-module machine learning models, a precursor to some of the initial end-to-end modern deep learning models . The next major upgrade in producing high OCR accuracies was the use of a Hidden Markov Model for the task of OCR. This approach uses letters as a state, which then allows for the context of the character to be accounted for when determining the next hidden variable . This lead to higher accuracy compared to both feature extraction techniques and the Naive Bayes approach .

The main drawback was still the manual extraction features, which requires prior knowledge of the language and was not particularly robust to the diversity and complexity of handwriting. Ng et. al applied CNNs to the problem of taking text found in the wild (signs, written, etc) and identified text within the image by using a sliding window. The sliding window moves across the image to find a potential instance of a character being present. A CNN with two convolutional layers, two average pooling layers, and a fully connected layer was used to classify each character .

One of the most prominent papers for the task of handwritten text recognition is Scan, Attend, and Read: End -to-End Handwritten Paragraph Recognition with MDLSTM Attention . The approach was to take an LSTM layer for each scanning direction and encode the raw image data to a feature map. The model would then use attention to emphasize certain feature maps over others. After the attention map was constructed, it would be fed into the decoder which would predict the character given the current image summary and state.

This approach was quite novel because it did not decouple the segmentation and classification processes as it did both within the same model . The downside of this model is that it doesn’t incorporate a language model to generate the sequence of characters and words. It is completely dependent on the visual classification of each character without considering the context of the constructed word. We found a previous CS 231N project to be helpful in guiding us with our task as well. Yan uses the Faster RCNN model to identify individual characters within a word and for classification.

This uses a sliding window across the image to first determine whether an object exists within the boundaries. That bounded image is then classified to its corresponding character. Yan also implements edit distance which allows for making modifications to the classified word to determine if another classified word ismore likely to be correct (for instance xoo vs zoo).

**2.2 CREATING AND TRAINING DATASETS**

Our main resource for training our handwriting recognizer was the IAM Handwriting Dataset . This dataset contains handwritten text of over 1500 forms, where a form is a paper with lines of texts, from over 600 writers, contributing to 5500+ sentences and 11500+ words. The words were then segmented and manually verified; all associated form label metadata is provided in associated XML files.

The source text was based on the Lancaster-Oslo/Bergen (LOB) corpus, which contains texts of full English sentences with a total of over 1 million words. The database also includes 1,066 forms produced by approximately 400 different writers. This database given its breadth, depth, and quality tends to serve as the basis for many handwriting recognition tasks and for those reasons motivated our choice of the IAM Handwriting Dataset as the source of our training, validation, and test data for our models.

Last but not least, in deep learning large datasets–even with many pre-trained models–are very important and this dataset containing over 100K+ word instances met those requirements (deep learning model need at least 105 − 106 training examples in order to be in position to perform well, notwithstanding transfer learning).

**CHAPTER 3**

**Image Processing involved in Character Recognition**

**3.1 PRE-PROCESSING OF SAMPLE IMAGE**

Pre-processing of the sample image involves few steps that are mentioned as follows:

**Grey-scaling of RGB image**

Grey-scaling of an image is a process by which an RGB image is converted into a black and white image. This process is important for Binarization as after grey-scaling of the image, only shades of grey remains in the image, binarization of such image is efficient

**Binarization**

Binarisation of an image converts it into an image which only have pure black and pure white pixel values in it. Basically during binarization of a grey-scale image, pixels with intensity lower than half of the full intensity value gets a zero value converting them into black ones. And the remaining pixels get a full intensity value converting it into white pixels.

**Inversion**

Inversion is a process in which each pixel of the image gets a colour which is the inverted colour of the previous one. This process is the most important one because any character on a sample image can only be extracted efficiently if it contains only one colour which is distinct from the background colour. Note that it is only required if the objects we have to identify if of darker intensity on a lighter background.

The flow chart shown below illustrates the physical meaning of the processes that are mentioned above:

RGB => Grey-scaling => Binarization => Inversion

**3.2 FEATURE EXTRACTION**

Features of a character depicts the morphological and spatial characteristics in the image. Feature extraction is a method of extracting of features of characters from the sample image. There are basically two types of feature extraction:

 Statistical feature extraction

 Structural feature extraction

Statistical feature extraction In this type of extraction the extracted feature vector is the combination of all the features extracted from each character. The associated feature in feature vector of this type of extraction is due to the relative positions of features in character image matrix. Structural feature extraction This is a primitive method of feature extraction which extracts morphological features of a character from image matrix. It takes into account the edges, curvature, regions, etc. This method extracts the features of the way character are written on image matrix.

The different methods used for feature extraction are

1.Piecewise–linear regression

2.Curve-fitting

3. Zoning

4.Chain code, etc.

The functions that are used in feature extraction are:

* Indexing and labelling
* Boxing and Cropping
* Reshaping and Resizing

**CHAPTER 4**

**OPERATIONS**

**4.1 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.

**4.2 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.

**4.3 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.

**4.4 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.

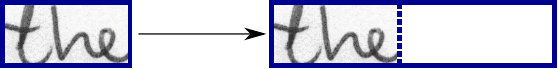


Fig. 3: Left: an image from the dataset with an arbitrary size. It is scaled to fit the target image of size 128×32, the empty part of the target image is filled with white color.

**4.5 Output of the operations**

**CNN output**: Fig. 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).

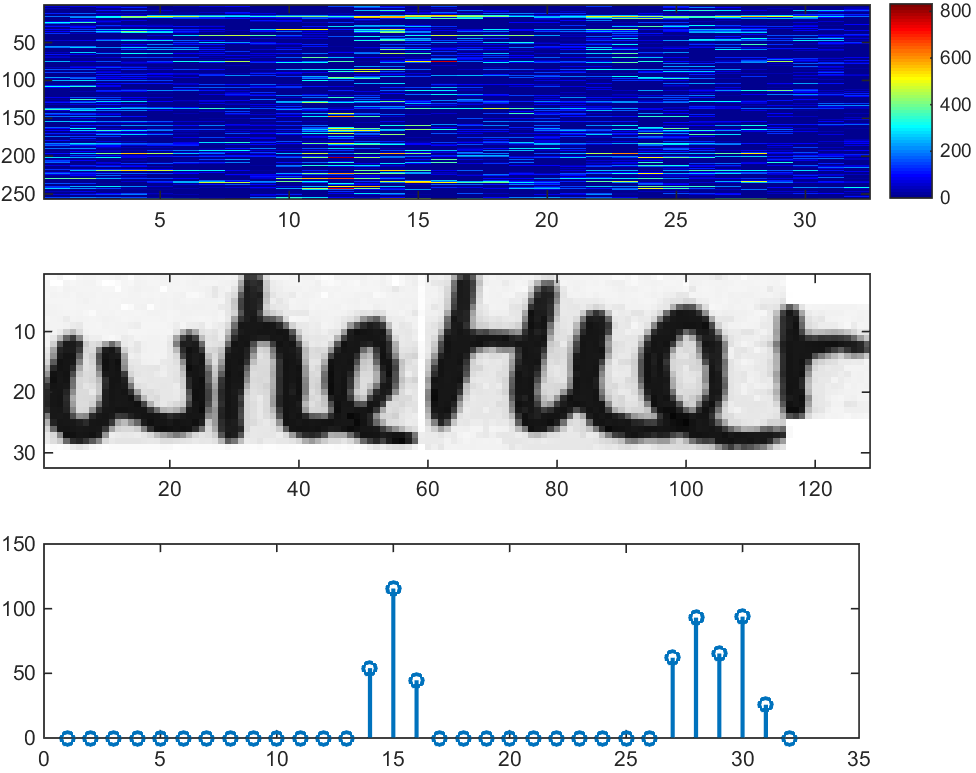


Fig. 4: Top: 256 feature per time-step are computed by the CNN layers. Middle: input image. Bottom: plot of the 32nd feature, which has a high correlation with the occurrence of the character “e” in the image.

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

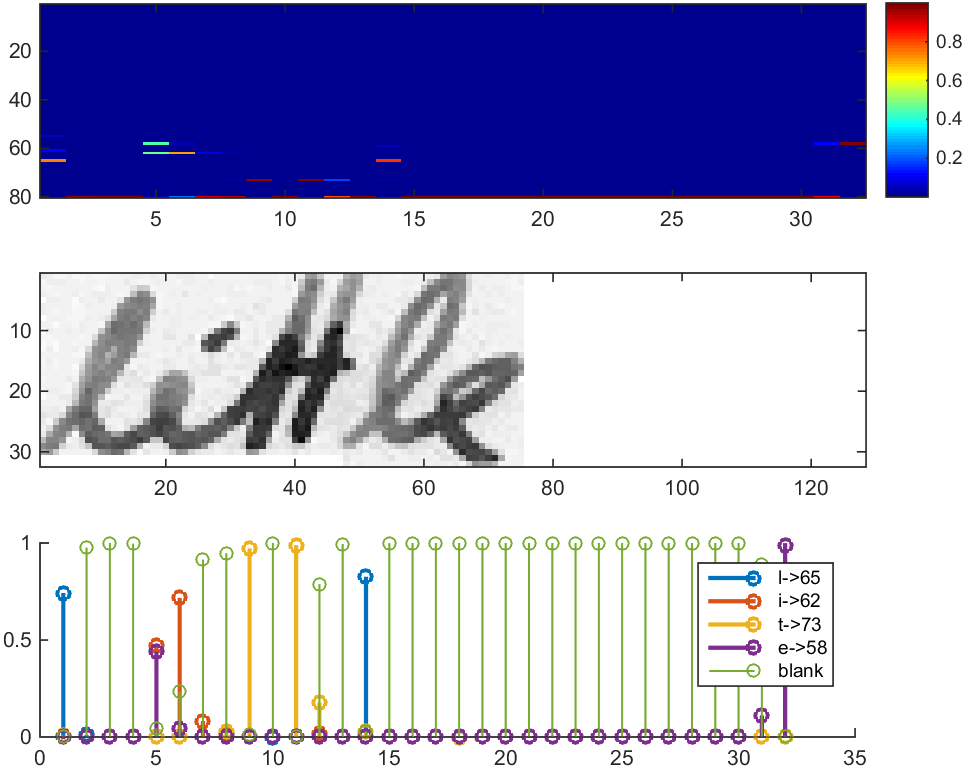


Fig. 5: Top: output matrix of the RNN layers. Middle: input image. Bottom: Probabilities for the characters “l”, “i”, “t”, “e” and the CTC blank label.

**CHAPTER 5**

**GUI**

**5.1 About tkinter**

**Tkinter** is a [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) [binding](https://en.wikipedia.org/wiki/Language_binding) to the [Tk](https://en.wikipedia.org/wiki/Tk_(software)) [GUI](https://en.wikipedia.org/wiki/Graphical_user_interface) toolkit. It is the standard Python interface to the Tk GUI toolkit,[[1]](https://en.wikipedia.org/wiki/Tkinter#cite_note-1) and is Python's [*de facto* standard](https://en.wikipedia.org/wiki/De_facto_standard) GUI. Tkinter is included with standard [Linux](https://en.wikipedia.org/wiki/Linux), [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows) and [Mac OS X](https://en.wikipedia.org/wiki/Mac_OS_X) installs of Python.

The name *Tkinter* comes from *Tk interface*. Tkinter was written by Fredrik Lundh.

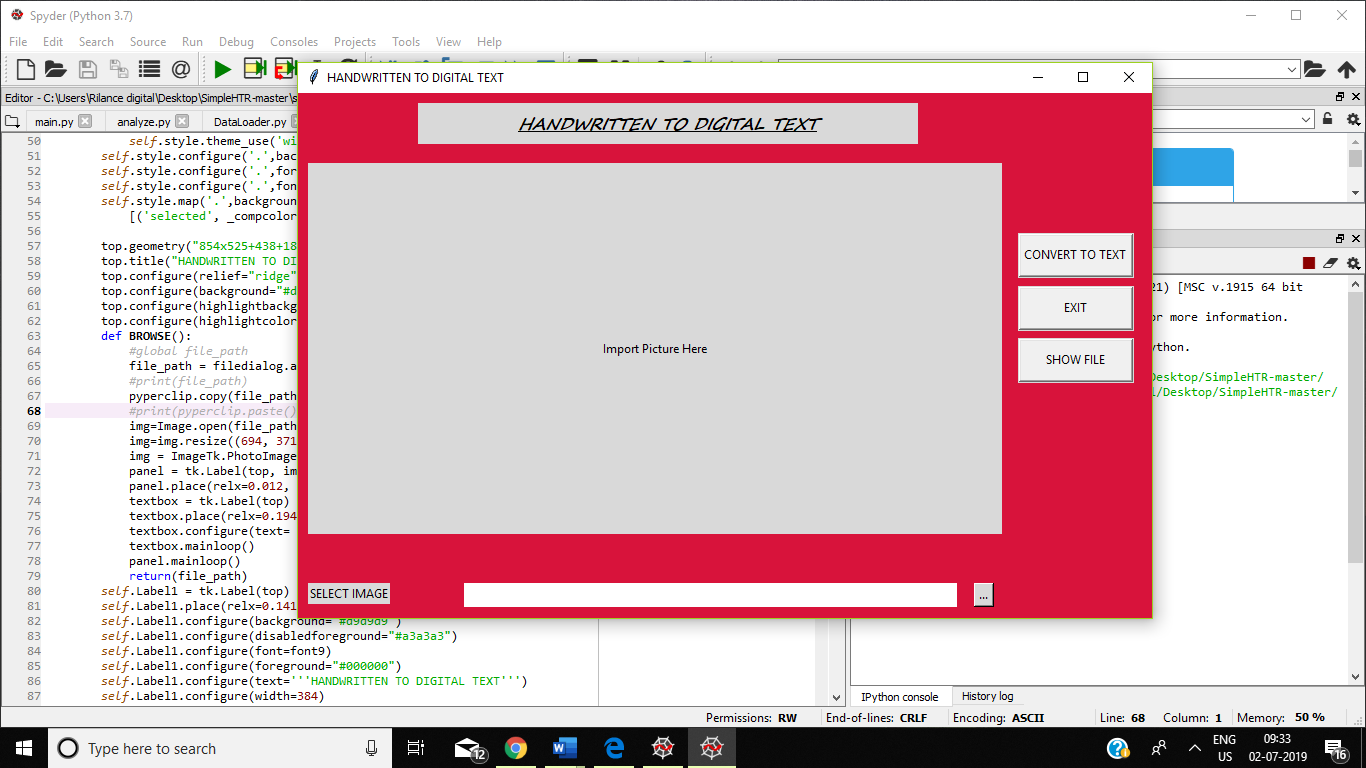
Tkinter is [free software](https://en.wikipedia.org/wiki/Free_software) released under a [Python license](https://en.wikipedia.org/wiki/Python_license).

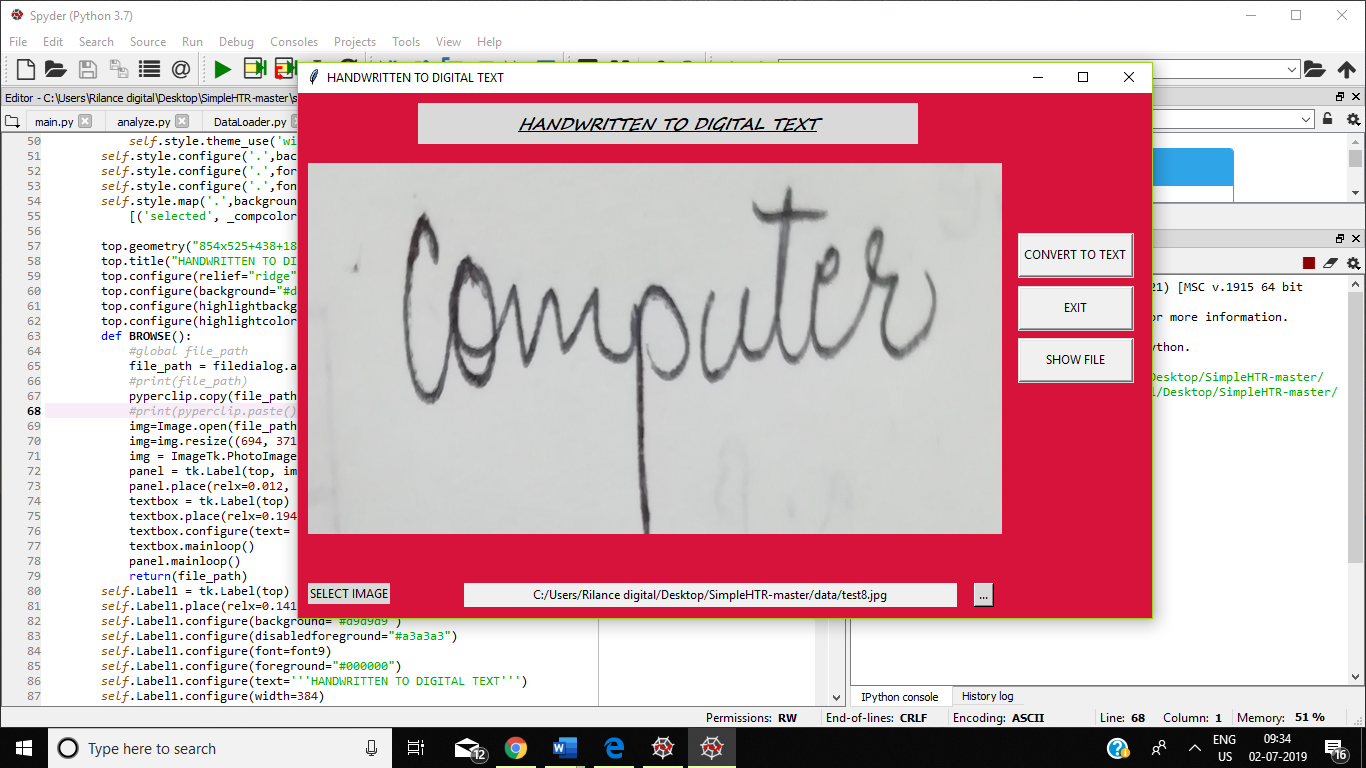
As with most other modern Tk bindings, Tkinter is implemented as a Python wrapper around a complete [Tcl](https://en.wikipedia.org/wiki/Tcl) interpreter embedded in the Python interpreter. Tkinter calls are translated into Tcl commands which are fed to this embedded interpreter, thus making it possible to mix Python and Tcl in a single application.

Python 2.7 and Python 3.1 incorporate the "themed Tk" ("ttk") functionality of Tk 8.5. This allows Tk widgets to be easily themed to look like the native desktop environment in which the application is running, thereby addressing a long-standing criticism of Tk (and hence of Tkinter).

There are several popular GUI library alternatives available, such as [wxPython](https://en.wikipedia.org/wiki/WxPython), [PyQt](https://en.wikipedia.org/wiki/PyQt) ([PySide](https://en.wikipedia.org/wiki/PySide)), [Pygame](https://en.wikipedia.org/wiki/Pygame), [Pyglet](https://en.wikipedia.org/wiki/Pyglet), and [PyGTK](https://en.wikipedia.org/wiki/PyGTK).

**5.2 GUI screenshots**

****

****

**CHAPTER 6**

**Simulation and Results**

**6.1 SIMULATION**

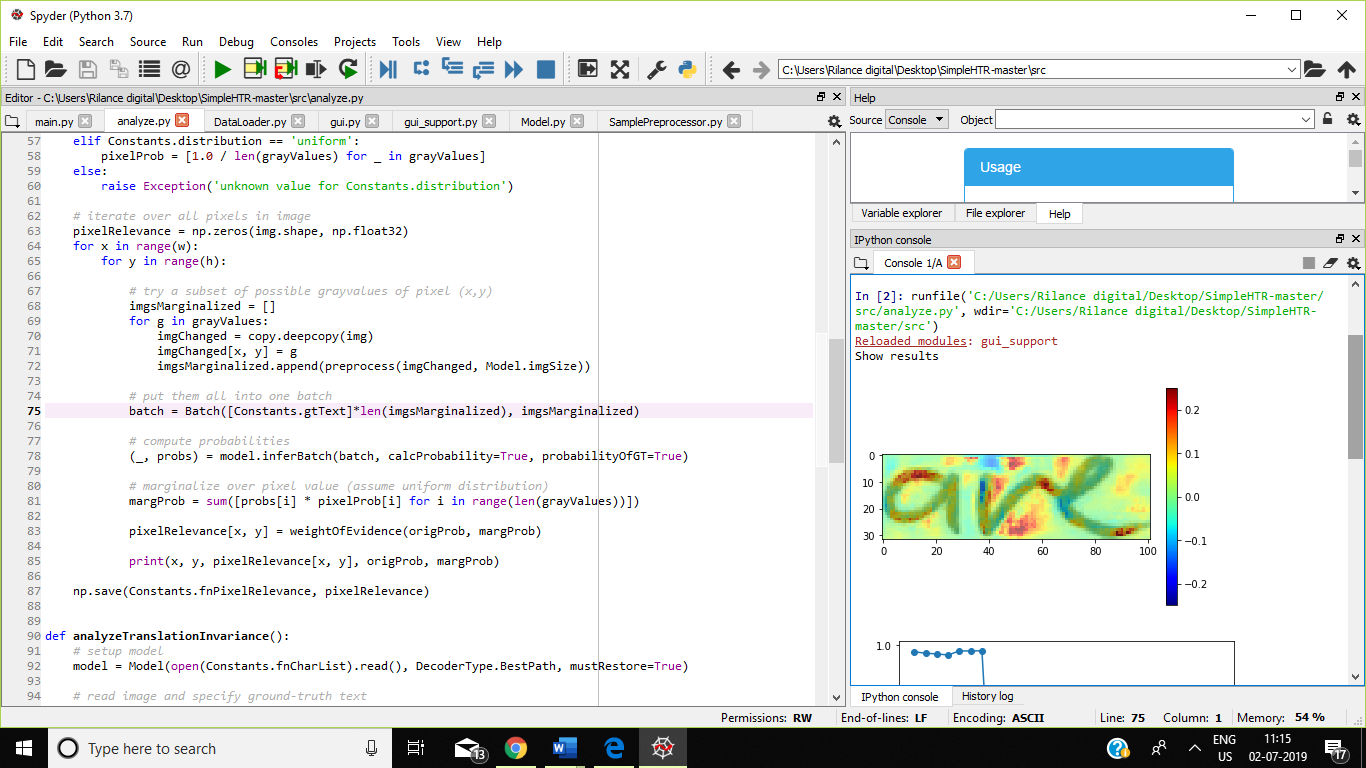
**6.1.1 Pre-processing of the image**

First of all the image on which the characters are written by hand is required. Below is the example of one case in which an image sample is taken.

1. **original handwritten image sample**

C:\Users\S D SINGH\Downloads\SimpleHTR-master\SimpleHTR-master\data\analyze.png

1. **Grey-scaling and Binarization of image**

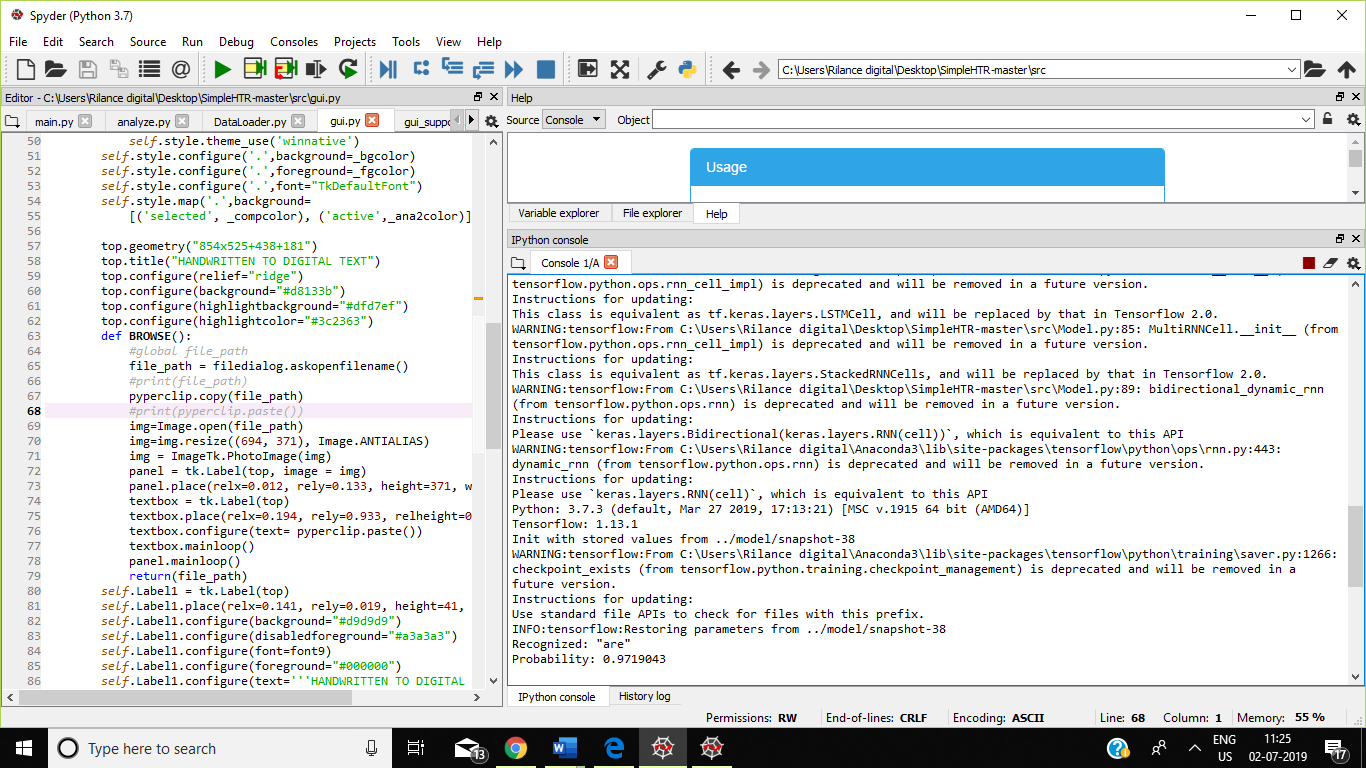
****

**6.2 RESULTS**

**6.2.1 Identification of characters**

After proper training and testing of the network, the pixelated 7 by 5 sized image of ‘are’ is fed to the network as input. Then the out we get is the resultant 2D matrix plot same as the word ‘are’ from the ideal dataset which was fed to the network as training dataset.

The character ‘are’ identified by network.



**CHAPTER 7**

**Implementation and project Snapshots**

**7.1 IMPLEMENTATION**

Implementation using TF

The implementation consists of 4 modules:

1. SamplePreprocessor.py: prepares the images from the IAM dataset for the NN
2. DataLoader.py: reads samples, puts them into batches and provides an iterator-interface to go through the data
3. Model.py: creates the model as described above, loads and saves models, manages the TF sessions and provides an interface for training and inference
4. main.py: puts all previously mentioned modules together

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.

*kernel = tf.Variable(tf.truncated\_normal([k, k, chIn, chOut], stddev=0.1))*

*conv = tf.nn.conv2d(inputTensor, kernel, padding='SAME', strides=(1, 1, 1, 1))*

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.

*relu = tf.nn.relu(conv)*

*pool = tf.nn.max\_pool(relu, (1, px, py, 1), (1, sx, sy, 1), 'VALID')*

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

**RNN**

Create and stack two RNN layers with 256 units each.

*cells = [tf.contrib.rnn.LSTMCell(num\_units=256, state\_is\_tuple=True) for \_ in range(2)]*

*stacked = tf.contrib.rnn.MultiRNNCell(cells, state\_is\_tuple=True)*

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.

*((fw, bw),\_) = tf.nn.bidirectional\_dynamic\_rnn(cell\_fw=stacked, cell\_bw=stacked, inputs=inputTensor, dtype=inputTensor.dtype)*

**CTC**

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

|  |  |
| --- | --- |
|  | *gtTexts = tf.SparseTensor(tf.placeholder(tf.int64, shape=[None, 2]), tf.placeholder(tf.int32, [None]), tf.placeholder(tf.int64, [2]))* |
|  | *seqLen = tf.placeholder(tf.int32, [None])* |

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

|  |  |
| --- | --- |
|  | *loss = tf.nn.ctc\_loss(labels=gtTexts, inputs=inputTensor, sequence\_length=seqLen, ctc\_merge\_repeated=True)* |
|  | *decoder = tf.nn.ctc\_greedy\_decoder(inputs=inputTensor, sequence\_length=seqLen)* |

**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.

*optimizer = tf.train.RMSPropOptimizer(0.001).minimize(loss)*

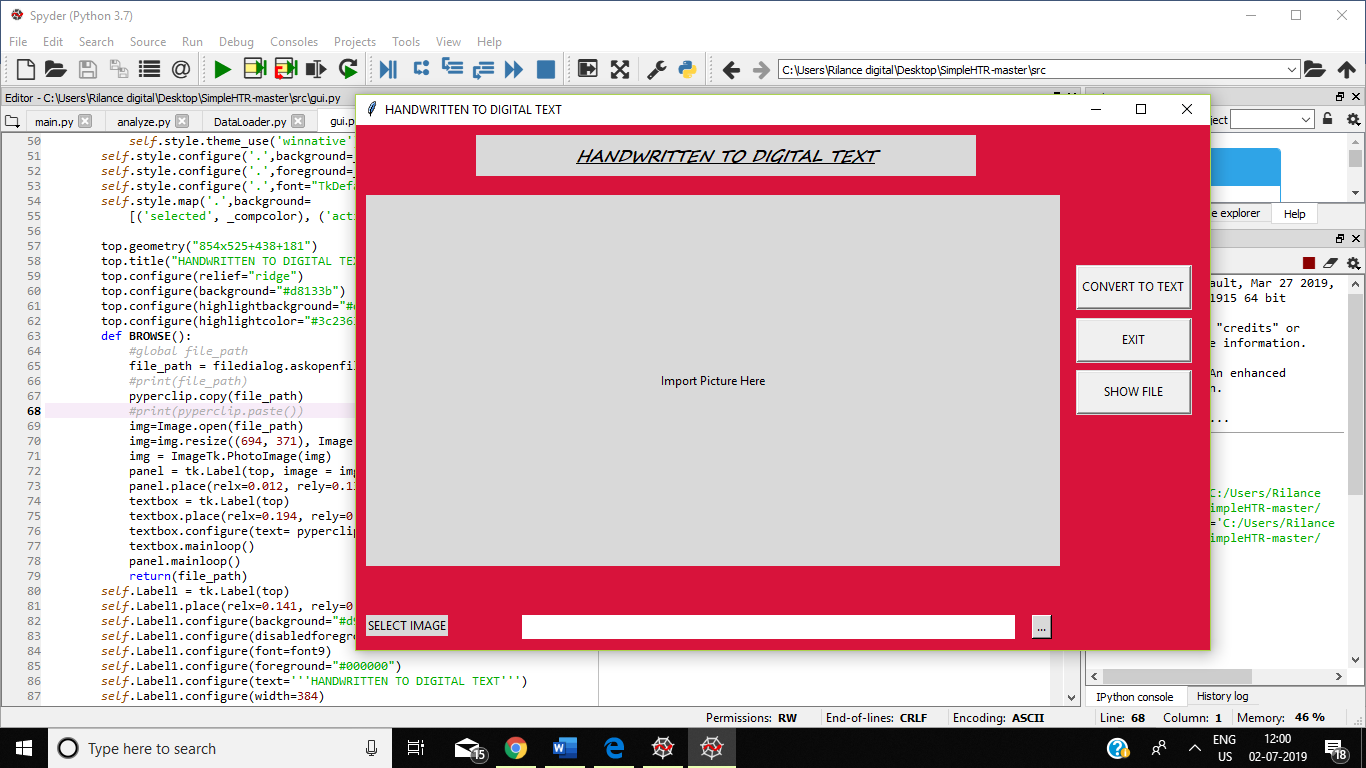
**Improving the model**

In case you want to feed complete text-lines as shown in Fig. 6 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:

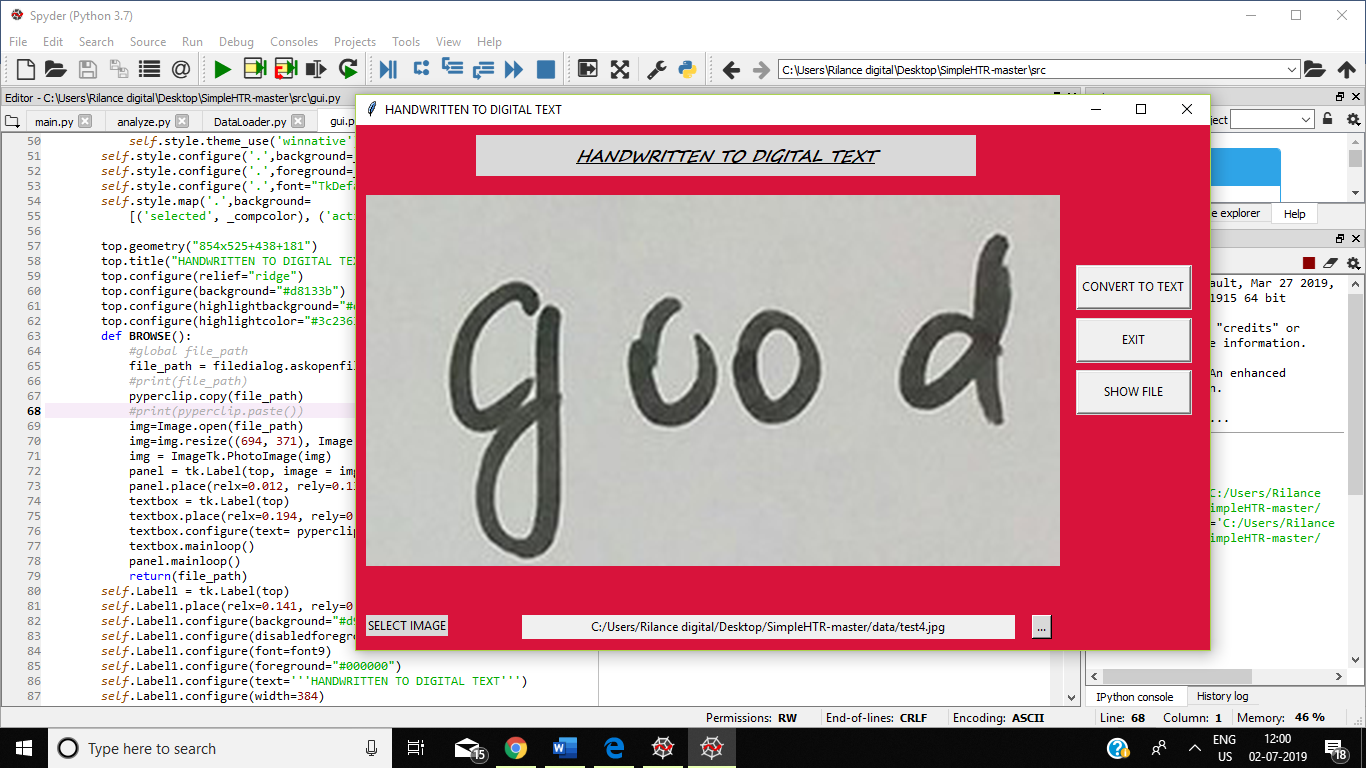
* Data augmentation: increase dataset-size by applying further (random) transformations to the input images
* Remove cursive writing style in the input images
* Increase input size (if input of NN is large enough, complete text-lines can be used)
* Add more CNN layers
* Replace LSTM by 2D-LSTM
* Decoder: use token passing or word beam search decoding 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

**7.2 Project snapshots**

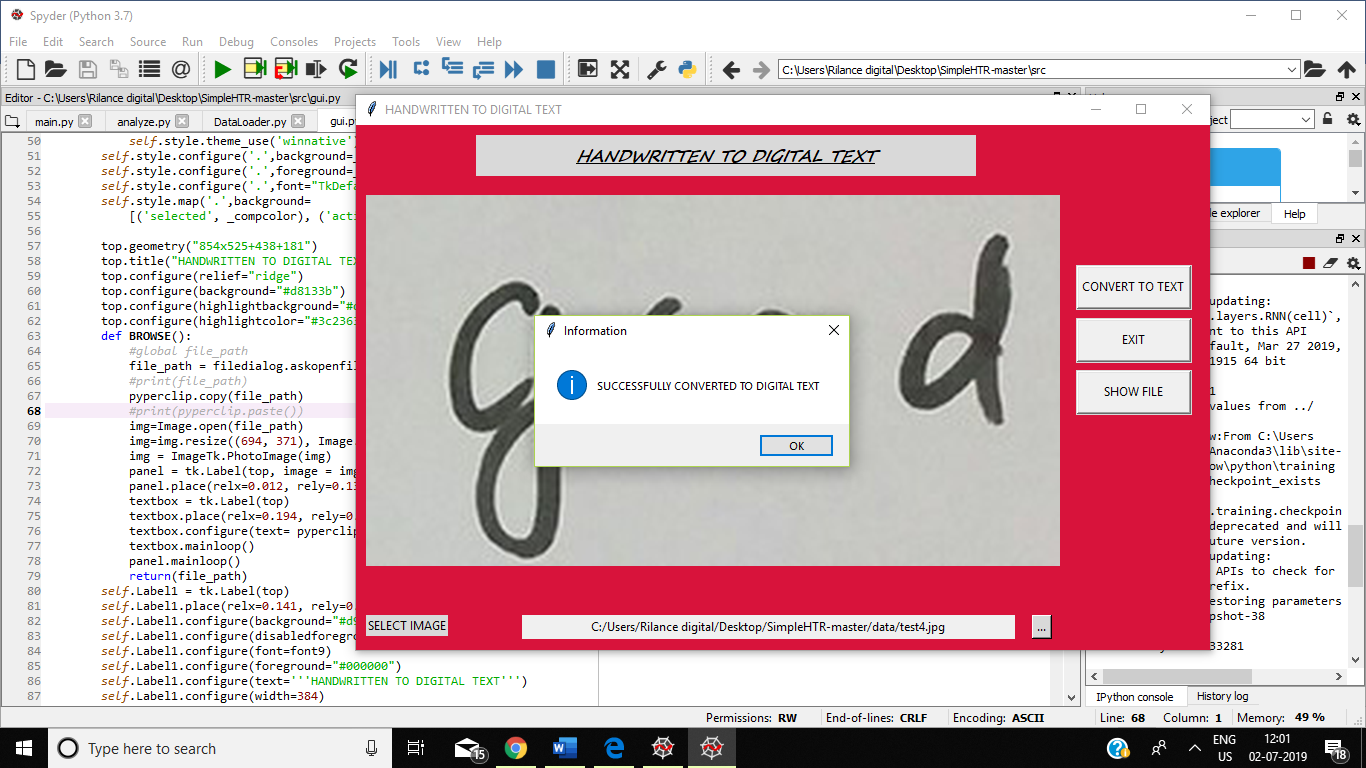
****

**7.2.1 Start the GUI**

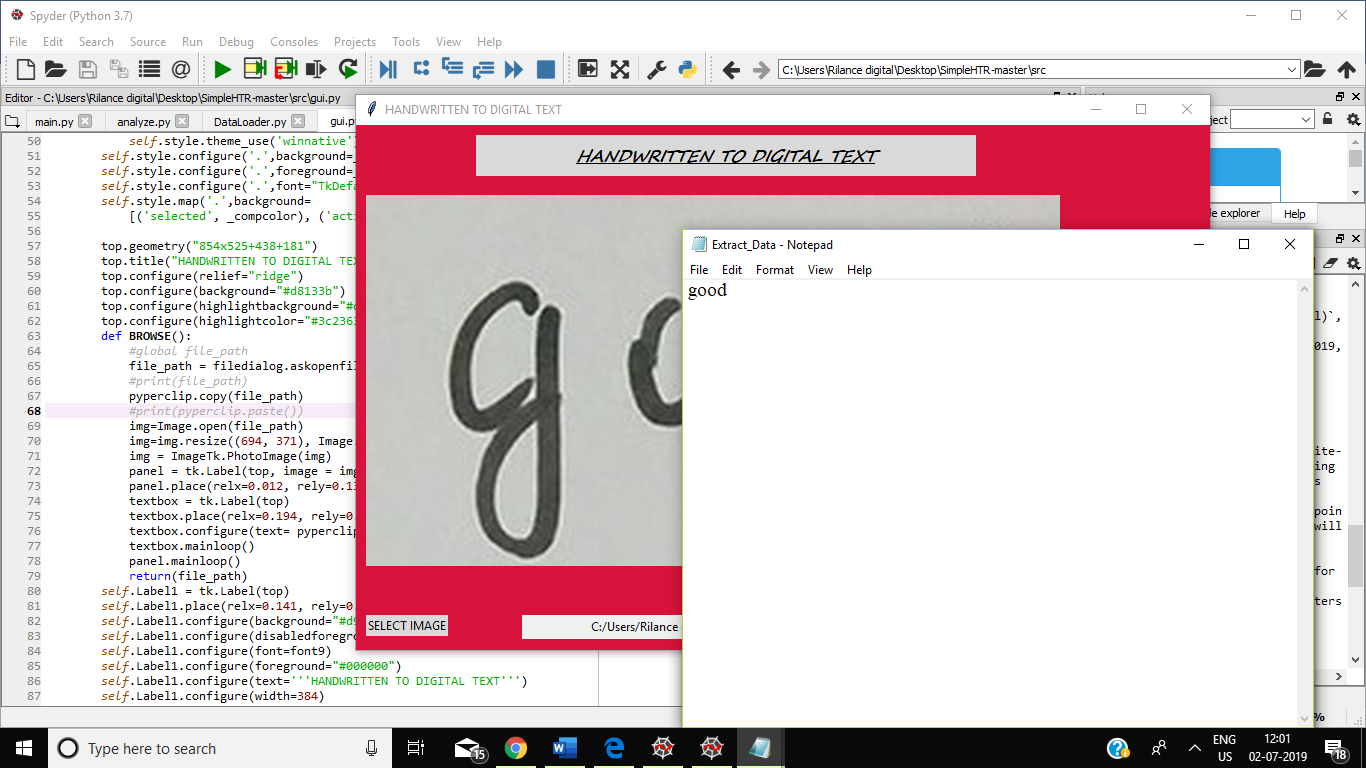
**7.2.2 Browse the image**

****

**7.2.3 Click on convert to text button**

****

**7.2.4 Click on show file button to view the converted text**

****

**CHAPTER 8**

**Conclusion and Future Improvements**

**8.1 CONCLUSION**

Classification of characters and learning of image processing techniques is done in this project.The result which was got was correct up to more than 90% of the cases, but it would be improved at the end. This work was basically focused on envisaging methods that can efficiently extract feature vectors from each individual character. The method I came up with gave efficient and effective result both for feature extraction as well as recognition. There are also different methods through which ‘handwritten text recognition’ is achieved.

**8.2 FUTURE SCOPE OF THIS PROJECT**

The application of this HTR algorithm is extensive. Now-a-days recent advancement in technologies has pushed the limits further for man to get rid of older equipment which posed inconvenience in using. In our case that equipment is a keyboard. There are many situations when using a keyboard is cumbersome like,

 We don’t get fluency with keyboard as real word writing

 When any key on keyboard is damaged

 Keyboard have scripts on its keys in only one language

 We have to find each character on keyboard which takes time

 In touch-enabled portable devices it is difficult to add a keyboard with much ease

On the other hand if we use an HTR software in any device, we can get benefits like,

 Multiple language support

 No keyboard required

 Real world writing style support

 Convenient for touch enabled devices

 Previously hand written record can be documented easily

 Extensive features can also be added to the software like,

1. Translation 2. Voice reading

**REFERENCES**

1.Convolutional Neural Network Benchmarks: <https://github.com/jcjohnson/cnn-benchmarks>

2. U. Marti and H. Bunke. The IAM-database: An English Sentence Database for Off-line Handwriting Recognition. Int. Journal on Document Analysis and Recognition, Volume 5, pages 39 - 46, 2002.

3. T. Wang, D. Wu, A. Coates, A. Ng. ”End-to-End Text Recognition with Convolutional Neural Networks” ICPR 2012

4.recurrent neural network:<https://www.guru99.com/rnn-tutorial.html>

5.tkinter: <https://www.geeksforgeeks.org/python-gui-tkinter/>

6.openCv with tkinter:<https://www.pyimagesearch.com/2016/05/23/opencv-with-tkinter/>