Dissertation on

Digitization of Handwritten Document

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

Bachelor of Technology
In
Electronics & Communication Engineering

Submitted by:

Sharadhi Jaganath	**01FB14EEC178**
Sukanya Basu	**01FB14EEC204**
Sukruthi S Rao	**01FB14EEC205**

Under the guidance of

Prof SRIKANTH H.R.
Assistant Professor,
Department of Computer Science
PES University

January – May 2018

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

'Digitization of Handwritten Document'

Is a bonafide work carried out by

Sharadhi Jaganath	01FB14EEC178
Sukanya Basu	01FB14EEC204
Sukruthi S Rao	01FB14EEC205

In partial fulfilment for the completion of eighth semester project work in the Program of Study Bachelor of Technology in Electronics and Communication Engineering under rules and regulations of PES University, Bengaluru during the period Jan. 2018 – May. 2018. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 8th semester academic requirements in respect of project work.

Signature Prof. Srikanth H.R. Assistant Professor	Signature Dr.Shylaja S S Chairperson	Signature Dr.B.K.Keshavan Dean of Faculty

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

DECLARATION

We hereby declare that the project entitled **"Digitization of Handwritten Document"** has been carried out by us under the guidance of Prof. Srikanth H.R and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Electronics and Communication Engineering** of **PES University, Bengaluru** 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.

01FB14EEC178	**Sharadhi Jaganath**
01FB14EEC204	**Sukanya Basu**
01FB14EEC205	**Sukruthi S Rao**

ACKNOWLEDGEMENT

The satisfaction that accompanies the completion of any task would be incomplete without the mention of the people who made it possible, and whose constant guidance and encouragement helped us in completing the project successfully. We consider it a privilege to express our gratitude and respect to all those who guided us through the course of the project.

We extend our sincere thanks to Prof Srikanth HR, Assistant Professor, for his constant guidance, encouragement, support and invaluable advice, without which this project would not be where it is today.

We extend our gratitude to Prof. V. R. Badri Prasad, Associate Professor, for his overall coordination of our project and also for helping us in stages where we needed such guidance.

We would like to express our heartfelt thanks to Prof. M. R. Doreswamy, founder, Prof. D. Jawahar, CEO, Dr. K. N. B. Murthy, Vice Chancellor and Dr. K. S. Sridhar, Principal, for providing us with the congenial environment for presenting this project.

We would like to express our immense gratitude to our friends Tejas and Jainam for their continuous help and support.

ABSTRACT

The pen and paper mode of communication has been an integral part of our lives for centuries. However, with the invention of the typewriter and computer features like text documents and word editors have ensured that written information can be stored in another - more durable format. This mode of storing and transmitting information is much more convenient when compared to the primitive mode of data storage. Therefore, a need has arisen for the seamless conversion of information from it's written to digitised form.
This is what we intend to do in our project.

In this modern world, with it's ever- increasing pace of technology, the new magnate in the field appears to be Artificial Intelligence and Machine Learning.
These tools have come a long way in ensuring that our lives are simpler to live.
Therefore, the tools we have used to achieve our goal are - machine learning, computer vision and image processing. A handwritten document is first converted into an image (in either a .png or .jpeg format). This image undergoes a few steps of pre-processing to remove any external noise. The pre-processed image is then segmented into individual characters. These characters are sequentially sent into a recogniser module (consisting of a CNN) which recognises individual characters. These characters are stored in an output document.
This output document is then sent into a corrector module which uses sequence to sequence model (RNN) to correct any misspelled or incorrectly recognised words.
An abundance of novel approaches already exist which are aiming to achieve our exact goal. However, we are one of the few which incorporate a sequential corrector module after the recogniser module.

LIST OF TABLES

LIST OF FIGURES

9.4	Accuracy of the seq2seq model	30
9.5	Segmented Characters	
9.6	Erroneous test images	
10.1	Test result 1	
10.2	Test result 2	
10.3	Test result 3	
10.4	Test result 4	
11.1	UI 1 - Upload document	
11.2	UI 2 - Converting	
11.3	UI 3 - Digitized Doc	
11.4	UI 4 - Editing the Doc and save	
11.5	Pre Processing to make data...	
11.6	Segmentation Results	
11.7	Adding Own dataset to the training set	

CONTENTS

CHAPTER-1

INTRODUCTION

1.1 OVERVIEW & SCOPE

In this fast paced era of technology, everyone wants to have digital media at their fingertips at all times, and as the tech world understands the increasing potential of Artificial Intelligence, physical medium is losing its significance.

But there are some actions that are and will remain timeless, like writing on paper. A solid proof of this concept is that educational institutions all over the world still follow this. The aim of our project is to bridge the gap between these two parallels and make the transition from the physical world of writing to the digital world of text seamless. This is necessary for several reasons: there will soon come a time when a lot of this world will run on Artificial Intelligence, but everyone will have not have the resources to harness these benefits. If

however, simple tools are developed to with interfaces to convert one form of information to another, the society will benefit maximally.

Therefore, the scope of our project is as follows:

1. Input a handwritten document - with a few specific prerequisites
2. Segment the document into sentences, words and finally letters
3. Feed each letter sequentially, into the recognition block.
4. The recognition block will take in each letter and identify whether it belongs to one of twenty six classes (from a - z). Finally, the predicted letters will be stored in a text document.
5. This text document is finally sent into a corrector module which attempts to correct incorrectly spelt words.

1.2 MOTIVATION

All innovations are created with the motivation of the greater good. Every minor invention has made the lives of people easier. Today, Snap chat's Facial Recognition algorithm is so efficient that the millions of people using it today don't even realise the magic unfold every second they upload a snap. This is the magic of Artificial Intelligence in today's world - which it integrates so seamlessly into our everyday lives.

Every day, every action we take on the World Wide Web is monitored by the machine. And what more? It learns every move we make and it is literally driving us to buy things that we never thought we needed (by giving relevant advertisements).

This was our primary motivation for the project -to build a system that is so simple to use that you don't even realise how it is changing your life, but yet so complex that it can do amazing things.

Having said the above, our domain has to do with modes of human communication. The most convenient mode of communication is now digitised. However the most popular means to store information is still the pen and paper format. Therefore a need has arisen for a

seamless conversion of information from it's handwritten to it's digital format. However, the subtlety of handwriting is unique to every individual and therefore hard to decipher.

So, if a machine can efficiently learn and translate this text into a digitised format, it will result in the best of both worlds. Casual data can be scribbled, while it can also be stored in a more convenient manner (digitally) - which makes it easier to transmit and preserve.

1.3 CURRENT SYSTEMS

Existing systems in this field incorporate a similar architecture, and yet not all blocks have been incorporated together as ours has. The models that we have seen are complex, computationally heavy and hard to implement.

Current systems include heavy pre-processing and novel approaches that incorporate skew angles and ground truth. Image vectors are fed into a deep BLSTM network. For loss functions, some models use CTC loss functions.

The offline handwriting issue has been tended to by numerous specialists for a generous measure of time. Despite the fact that separated character recognition is en route to being solved, delivering incredible acknowledgment rates, analysts focusing on the recognition of manually written words can't brag a similar achievement.

Two fundamental methodologies for the previously mentioned issue have been recognized:

1. A worldwide approach

2. A division approach.

The primary approach involves the acknowledgment of the entire word by the utilization of distinguishing highlights.

The second approach requires that the word be first fragmented into letters. The letters are then perceived independently and can be utilized to coordinate against specific words.

As of late numerous scientists have been headed to create disconnected acknowledgment frameworks because of the testing logical nature of the issue and besides it's mechanical significance. The last emerges from various uses of penmanship acknowledgment frameworks. A portion of these include: postal address acknowledgment, machines for the visually impaired, handling physically rounded out tax documents and bank check acknowledgment.

For the word corrector module, existing systems used by the likes of Google and Microsoft are not open source and often use novel and expensive approaches. Open source models that do exist are primitive in nature and use brute force and probability models for computing mathematical distances between the input words and the correctly spelled words of the English lexicon to make a decision about the probable correct word. This approach needs huge computational power and is seen to take high computational time to correct even a mere four lettered word. The use of deep neural networks, especially sequential models is an area which has been very recently broached and currently being heavily researched. The implemented models target a very specific domain of words and certain errors which can be fixed.

CHAPTER-2

PROBLEM DEFINITION

A given handwritten document should be converted into digital text file given it is within the proposed constraints, followed by error correction of wrongly identified words and sentences.
First, a scanned image document should be pre-processed to remove noise.
After which, it is fed into the segmentation block, which segments the words and produces letters. This segmentation algorithm employs the concepts of MSER and convex hulls.
Now each individual letter must be passed sequentially into the CNN block for recognition.
This should yield a document with the digitised form of the handwritten data.
Next, the output text document is sent into a corrector module built using LSTMs to further improve its accuracy.

CHAPTER-3

LITERATURE SURVEY

[1] In this paper, the researchers emphasise that while optical character recognition (OCR) can be considered as a mature field with very high accuracy rates for printed documents, recognition of handwritten text remains a much harder challenge. The best performing teams of the ICDAR 2015 competition were able to achieve just 69.8% word level accuracy, and 84.5% character level accuracy rates. High variability of handwritten characters and limited accessibility of language models, especially for historical texts, are two big hurdles for existing OCR methods. While for a long time the main methods were based on carefully extracted features and probabilistic models, such as the Hidden Markov Models, recently, better results were achieved by using deep learning techniques, like the Convolution Neural Networks (CNN) and Recurrent Neural Networks (RNN) with Long Short Term Memory (LSTM).

[2] Gene Lewis talks about how they've found that there is a multiplicity of representations for languages, ranging from rule-based models that encode "hard" grammatical knowledge to stochastic models that learn a suitable representation from data; and that these representations range from simple n-gram models to highly complex probabilistic network representations. Hidden Markov Models have been shown to exhibit a strong ability to capture many of the high-level dynamics of natural English language however, such models make the rather strong assumption of conditional independence of the current word from all previous words given the immediately previous word, which prevents HMM's from modelling crucial long-range dependencies. Recurrent Neural Networks, on the other-hand, have been shown to be more adept at capturing these long-range dynamics.

[3] A general strategy for content limitation and acknowledgment in true pictures is exhibited. The proposed technique is novel, as it (i) leaves from a strict bolster forward pipeline and replaces it by a speculations confirmation system all the while preparing various content line theories, (ii) utilizes engineered textual styles to prepare the calculation disposing of the requirement for tedious obtaining and naming of certifiable preparing information and (iii) Uses Maximally Stable Extremal Regions (MSERs) which gives heartiness to geometric and brightening conditions. The execution of the strategy is assessed on two standard datasets. On the Chars74k dataset, an acknowledgment rate of 72% is accomplished, 18% higher than the best in class. The paper is first to report both content detection and acknowledgment comes about on the standard and rather difficult ICDAR 2003 dataset. The content confinement works for number of letters in order and the technique is effectively adjusted to the identification of different scripts, e.g. cyrillics.

[4] Here, the scholars talk about Robust Text Detection in Natural Scene Images - Content location in normal scene pictures is an essential for some, content-based picture examination undertakings. In this paper, they propose an exact and vigorous strategy for identifying writings in regular scene pictures. A quick and powerful pruning calculation is intended to separate Maximally Stable Extremal Regions (MSERs) as character competitors utilizing the technique of limiting regularized varieties. Character competitors are assembled into content applicants by the single-interface bunching calculation, where separate weights and bunching limit are found out consequently by a novel self-preparing separation metric learning calculation. The back probabilities of content hopefuls relating to non-content are assessed with a character classifier; content competitors with high non-content probabilities are dispensed with and writings are related to a content classifier. The proposed framework is assessed on the ICDAR 2011 Powerful Reading Competition database; the f-measure is more than 76%, much superior to the best in class execution of 71%.

[5] Profound learning strategies have as of late accomplished great execution in the region of visual and speech recognition. In this paper, the scholars propose a Handwriting identification strategy in light of Relaxation Convolutional neural network (R-CNN) and

alternately trained relaxation convolutional neural network (ATR-CNN). It is known that past strategies regularize CNN at full-associated layer or spatial-pooling layer, be that as it may, they centre on convolutional layer. The relaxation convolution layer received in our R-CNN, dissimilar to conventional convolutional layer, does not require neurons inside a component guide to have the same convolutional piece, supplying the neural system with more expressive power. They make use of ATR-CNN to regularize the neural system amid preparing methodology. Their past CNN assumed the first position in ICDAR'13 Chinese Handwriting Character Recognition Competition, while their most recent ATR-CNN beat their past one and accomplished the cutting edge exactness with a blunder rate of 3.94%, additionally narrowing the hole amongst machine and human onlookers (3.87%).

[6]How to write a spelling corrector by Peter Norvig, talks about a probabilistic model to choose from a list of probable correct words, which is most likely to be the correct word given a wrong word w. The call correction (w) tries to pick the most all likelihood spelling adjustment for w. There is no real way to know without a doubt, which recommends we utilize probabilities. We are attempting to discover the correct word c, out of all conceivable applicant remedies, that amplifies the likelihood that c is the planned adjustment, given the first word w. The four sections of this articulation are: Choice Mechanism: chooses the maximum parameter specified. Correctness module: The list of all correct words to choose from. Dialect Model: P(c) The likelihood that c shows up as an expression of English content. For instance, events of "the" make up around 7% of English content, so we ought to have P(the) = 0.07. Mistake Model: P(w|c): The likelihood that w would be composed in a content when the creator implied c. For instance, P(teh|the) is generally high, however P(theeexyz|the) would be low. The most probable correct word is chosen based on an aggregate of all the probabilities.

[7] Blog by Tal Weiss: “Rethinking Spelling Correction on the 21st Century”, discusses how crude methodologies utilizing the Levenshtein distance remove are not appropriate to this period, as he says, this is on the grounds that "The underlying driver of the horrifying

execution is that the speller is attempting to savage power its way to the correct arrangement". The outcome is a testing measure of calculation that requirements to happen, and which develops exponentially as for the length of the information string. He instead proposes a method which uses Deep Learning models. His approach is, build a seq2seq architecture where lengths of input and output can vary. This is fundamentally the same as models utilized for Machine Translation undertakings.

[8] Deep Neural Networks (DNNs) are capable models that have accomplished phenomenal execution on troublesome learning assignments. Despite the fact that DNNs function admirably at whatever substantial marked preparing sets are accessible, they can't be utilized to delineate to successions. In this paper, they introduce a general end-to-end way to deal with sequences discovering that influences insignificant suppositions on the grouping to structure. Their technique utilizes a multi layered Long Short-Term Memory (LSTM) to delineate information arrangement of input to a vector of a fixed dimensionality, and afterward another LSTM to unravel the objective grouping from the vector. Their primary outcome is that on an English to French interpretation assignment from the WMT'14 dataset, the interpretations created by the LSTM accomplish a BLEU score of 34.8 on the whole test set, where the LSTM's BLEU score was punished on out-of-vocabulary words. Furthermore, the LSTM did not experience issues on long sentences. The LSTM likewise learned sensible expression and sentence portrayals that are touchy to word arrange and are moderately invariant to the dynamic and the detached voice. At last, they found that reversing the order of the words in all source sentences enhanced the LSTM's execution uniquely.

CHAPTER-4

PROJECT REQUIREMENT SPECIFICATION

4.1 WORKFLOW DIAGRAM

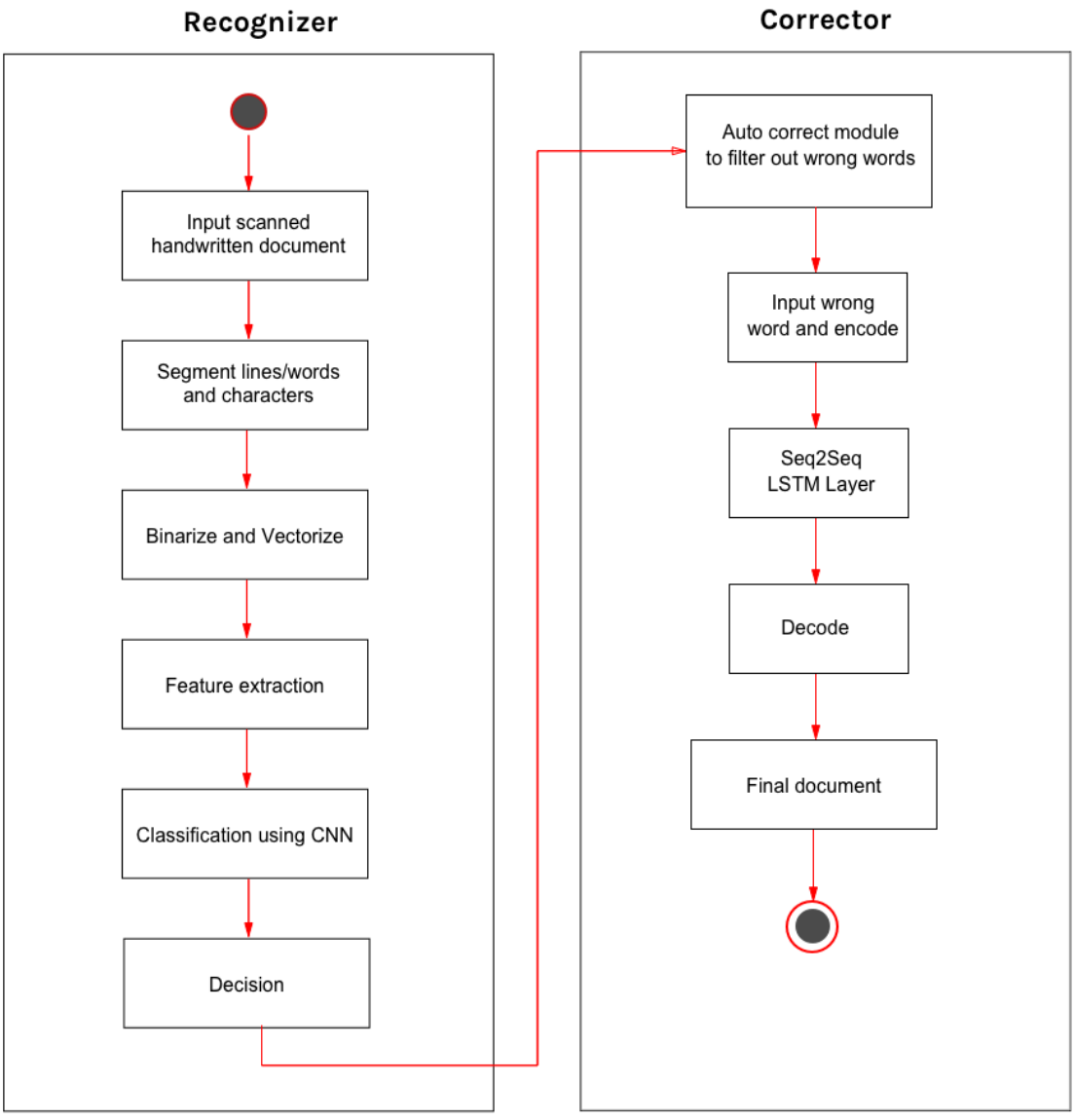


Fig 4: High Level Design Diagram

4.2 CONSTRAINTS

We are limited by the capabilities of the processing power of computers. Limitations also arise when the document is written illegibly due to either the handwriting itself, or other impediments like light ink and so on. Cursive handwriting is very subjective in nature. A letter can be written in several different ways and yet represent a single label. Even with a large training data set, it is impossible to achieve an accuracy beyond a certain value. Hence for this prototype, we avoid intense cursive writing. Another constraint includes a certain amount of line gap and word gap to be followed while writing.

For the spelling and grammatical corrections, we plan to use LSTM (Long short term memory). The word corpus for incorrect-correct word pairs is huge and this can introduce a highly probabilistic model. The correct word predicted is based on a high number of variables and accuracy decreases with the increase in the complexity and degree of error in the word. We anticipate some constraints in this prototype due to the above mentioned reasons.

It is imperative that the written data is both legible and appropriately spaced to ensure that the letters are segmented into the correct order. Any inconsistency in spacing between the lines or spacing between the words leads to incorrect implementation of our algorithm.

It must also be noted that our algorithm works significantly better with block letters when compared to cursive ones.

CHAPTER-5

SYSTEM REQUIREMENT SPECIFICATION

5.1 FUNCTIONAL REQUIREMENTS

Training model:

Requirement #	Requirement
F1	Keras has two broad models which have further methods. Sequential model has been chosen for our architecture
F2	TensorFlow uses a dataflow graph to represent computation in terms of the dependencies between individual operations. This has been customized to increase the accuracy of the model
F3	EMNIST is only available in the Unix Executable format. This function extracts the images to a png format of dimensions 28x28
F4	The model and final best weights need to be saved in a .json and .h5 file respectively for future use of the model

Testing module:

Reqmt #	Requirement
F5	Create a MSER object to detect regions of interest
F6	Use Dilation and Erosion on lines to make them clearly recognisable
F7	Sort the bounding rectangles around letters
F8	Remove unnecessary inner bounding rectangle for each letter

Corrector model:

Reqmt #	Requirement
F9	Keras Sequential model has been chosen for the LSTM architecture
F10	Extract a word from the Database and write to a text file in the format used for training the model
F11	Build One Hot Encoded 3D array of input and target sequences
F12	Decode the Sequence predicted by the LSTM Network

5.2 NON FUNCTIONAL REQUIREMENTS

5.2.1 Dependencies

We are limited by the capabilities of the processing power of computers. Limitations also arise when the document is written illegibly due to either the handwriting itself, or other impediments like light ink and so on. Cursive handwriting is very subjective in nature. A letter can be written in several different ways and yet represent a single label. Even with a large training data set, it is impossible to achieve an accuracy beyond a certain value. For the spelling corrector, the corpus of incorrect and correct word mapping pairs are quite literally infinite. The ways in which a word can be misspelled is large and also a single misspelling can be mapped to many different correct words, the rule by which it is done quite complicated at times. We anticipate some trouble due to the above mentioned reasons.

It must also be mentioned that our system does not work in real time. It does require some time to process the input sample to generate the output.

5.2.2 Assumptions

The assumptions to be made are as follows:

1. The sentences of the document must be of limited size (around 6).
2. The line gap and inter-word spacing must be significant. The thumb rule is to ensure that the spacing between two lines is equivalent to twice the size of the word-gap.
3. Line gap must approximately be 2 times the word gap
4. The handwriting of the individual must be clear and neat.
5. The ink of the writing must be bright.

5.2.3 User Requirements

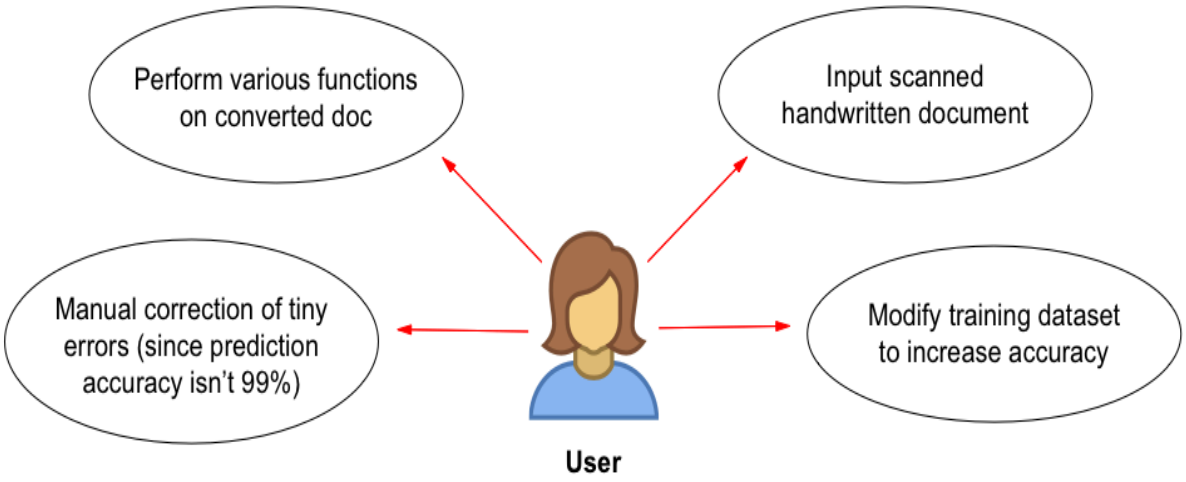


Fig 5: Use Case Diagram

The user's interaction with the system is expected to be writing the document which he/she wants to digitize. The user has to keep in mind the constraints of the system and write clearly (not extremely cursive) with sentences and words which are well spaced. The user then scans

the document with an appropriate tool specified for the case. The system then processes the image and outputs a text file. The gui then allows the user to edit the text in the output before saving it, giving an opportunity to the user to correct the errors in digitizing the document.

5.3 HARDWARE REQUIREMENTS

1. Operating System : Windows 7/8/10, OSX, Linux
2. Processor : Intel i3/i5/i7
3. RAM : 4/8/16 GB
4. Application Backend : TensorFlow

5.4 SOFTWARE REQUIREMENTS

1. User Interface : GUI built using Python Tkinter
2. Core Application : Python 3.6
3. Tools/Frameworks : TensorFlow, Keras
4. Application Backend : TensorFlow
5. Controller : Windows 7, OSX, Linux

CHAPTER-6

SYSTEM DESIGN

6.1 SYSTEM ARCHITECTURE

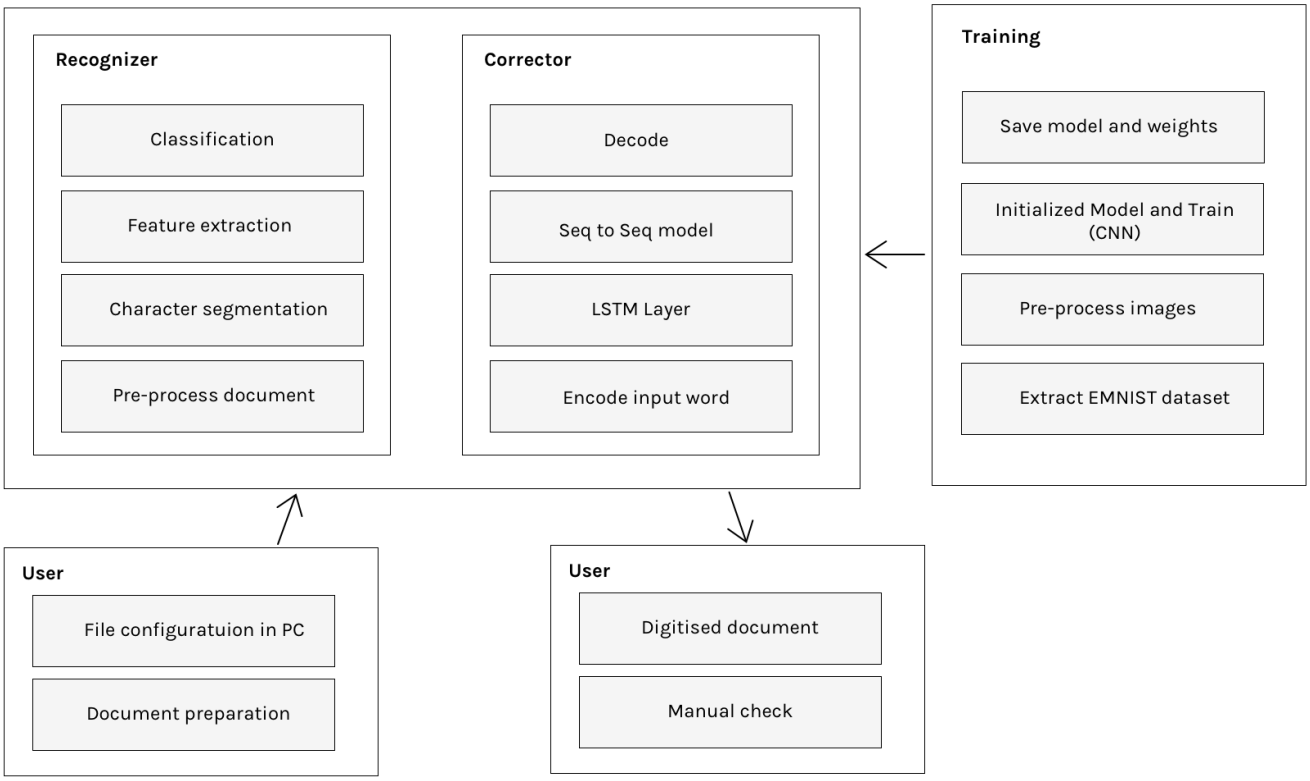


Fig 6.1: System architecture

6.2 MODULE DESCRIPTION

DATASET:

The MNIST dataset has become a standard benchmark for learning, classification and CV systems, leading which we have chosen this as our dataset as opposed to others like the IAM Dataset or Washington Database. Contributing to its widespread adoption are the intuitive nature of the task, it's relatively small size and storage requirements and the accessibility and ease-of-use of the database make it suitable for our purpose. The MNIST database was derived from a larger dataset known as the NIST Special Database 19 which contains digits, uppercase and lowercase handwritten letters. A variant of the full NIST dataset, is called Extended MNIST (EMNIST), which follows the same conversion paradigm used to create the MNIST dataset. The result is a set of datasets that consist a more challenging classification tasks involving letters and digits, and that shares the exact image structure and parameters as the original MNIST database, allowing for direct compatibility with all existing classifiers and systems. Benchmark results are gotten along with a validation of the conversion process through the comparison of the classification results on converted NIST digits and the MNIST digits.
[As stated in the Abstract of the Official documentation in *"EMNIST: an extension of MNIST to handwritten letters" by Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre van Schaik ´ The MARCS Institute for Brain, Behaviour and Development Western Sydney University Penrith, Australia 2751*]

The Flow of hierarchy for our system is:

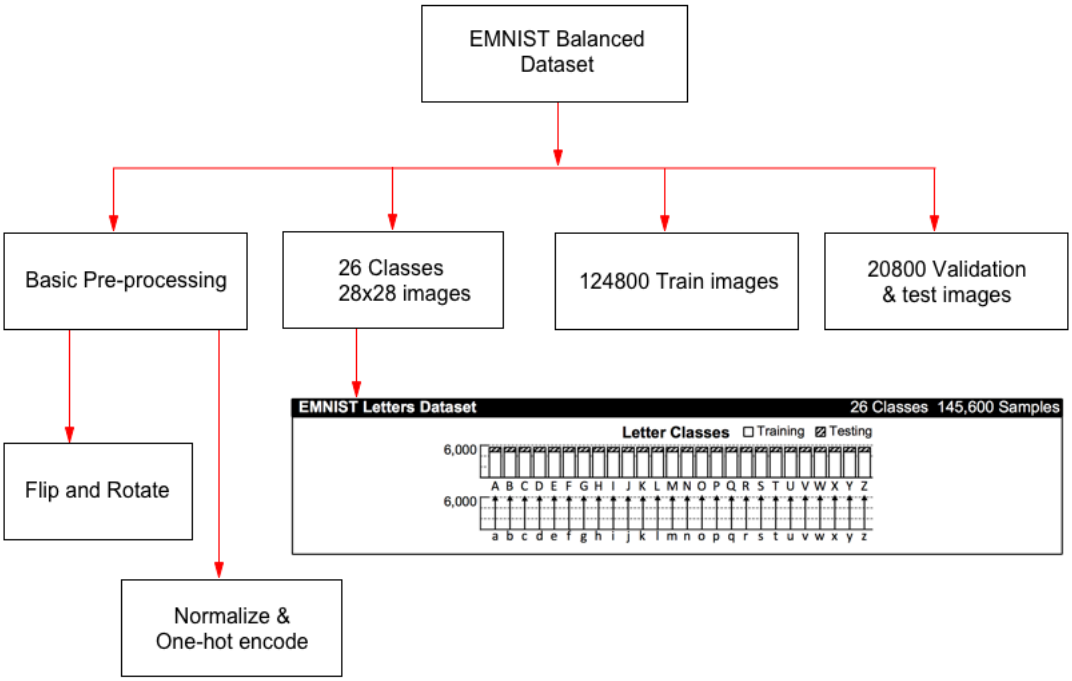


Fig 6.2: Dataset

CNN ARCHITECTURE:

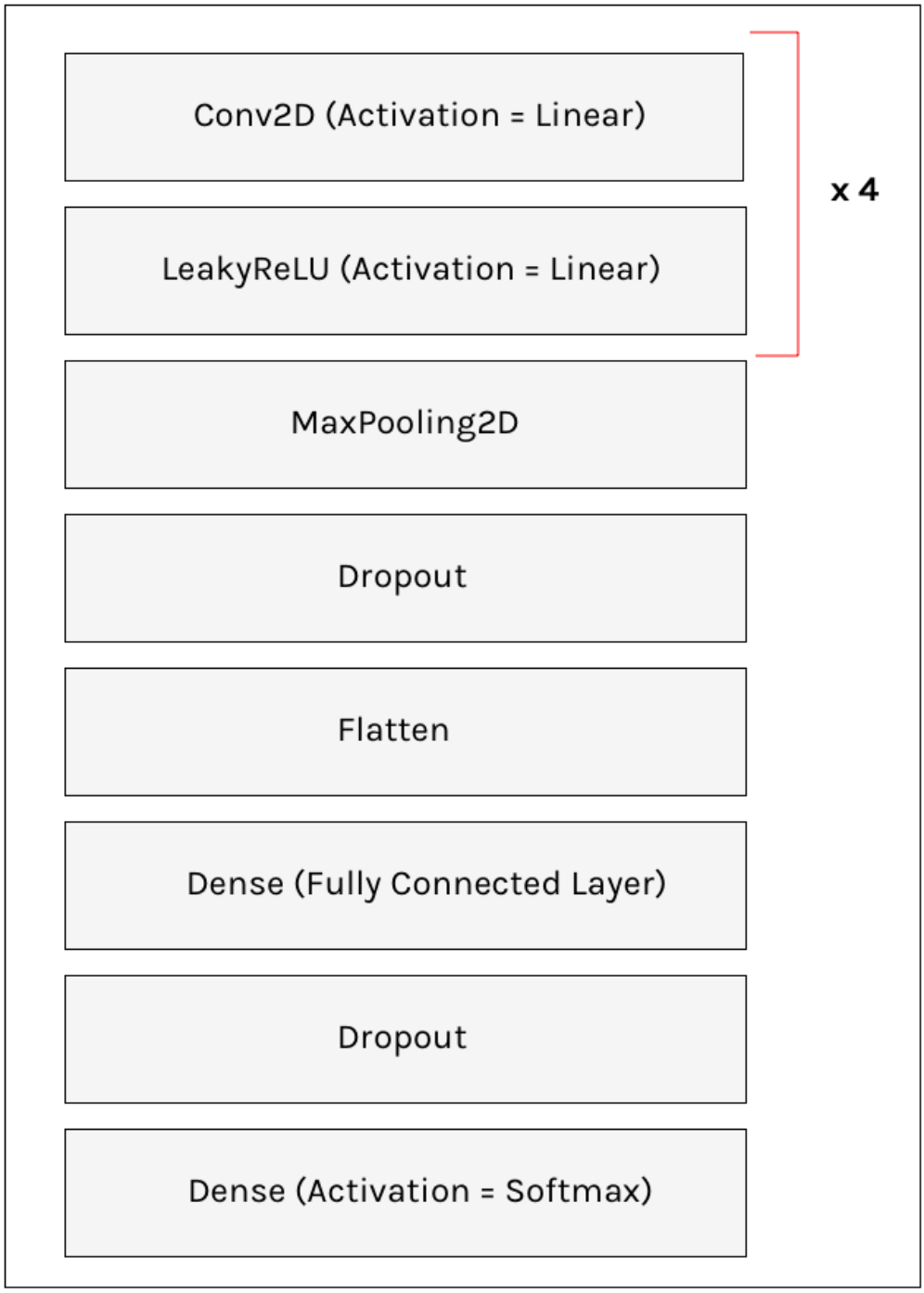


Fig 6.3: CNN Architecture

RECOGNITION MODULE:

- **MSER:**

Maximally Stable Extremal Regions are a Computer Vision technique applied to the input image for segmentation. MSER regions are connected areas that have been recognised by extrapolating pixels of similar intensity, surrounded by contrasting background. Each MSERs can be identified uniquely by (at least) one of its pixels x, as the connected component of the level set at some level which contains x. Such a pixel is called seed of the region. So in this case, for the image of the handwritten document, each pixel value is noted, and then the adjacent pixels are compared, if they are the same, it is added to the seed region.

In our project, The MSER algorithm has been used for text segmentation by combining MSER with Convex Hulls. MSER is first applied to the image in question to determine the character regions. To enhance the MSER regions all pixels are Dilated and Then Eroded to account for handwriting with light inks. This greatly increases the usability of MSER in the extraction of blurred text. An alternative use of MSER in text detection is the work by using a graph model. This method again applies MSER to the image to generate preliminary regions. These are then used to construct a graph model based on the position distance and colour distance between each MSER, which is treated as a node. Next the nodes are separated into foreground and background using cost functions. One cost function is to relate the distance from the node to the foreground and background. The other penalizes nodes for being significantly different from its neighbour. When these are minimized the graph is then cut to separate the text nodes from the non-text nodes. To enable text detection in a general scene, Scholars have used the MSER algorithm in a variety of projections. In addition to the grayscale intensity projection, they use the red, blue, and green colour channels to detect text regions that are colour distinct but not necessarily distinct in grayscale

intensity. This method allows for detection of more text than solely using the MSER plus MSER- functions.

We explored various other algorithms for segmentation like the "Watershed Algorithm": In the study of image processing, a watershed is a transformation defined on a grayscale image. So the test image is read in the mode = 'L', which reads the image in a grayscale before applying an Adaptive Threshold. The name Watershed refers metaphorically to a geological watershed, or drainage divide, which separates adjacent drainage basins. The watershed transformation treats the image (in this case the image of the handwritten document) it operates upon like a topographic map, meaning, with the brightness of each point representing its height, and finds the lines that run along the tops of ridges (in our case the contours).

There are different technical definitions of a watershed. In graphs, watershed lines may be defined on the nodes, on the edges, or hybrid lines on both nodes and edges. Watersheds may also be defined in the continuous domain. There are also many different algorithms to compute watersheds.

- **Adaptive Threshold**:

Adaptive thresholding usually takes a grayscale or colour image as input, in our case, a grayscale image of the handwritten document and, in the simplest implementation, outputs a binary image representing the segmentation. For each pixel in the image, a threshold is calculated. If the pixel value is below the threshold it is set to the background value, otherwise it assumes the foreground value. Which means, if most adjacent pixels are black (text) it is above the threshold, otherwise, it assumes the background pixel value, that is white.

There are two main approaches to finding the threshold: 1. the *Chow and Kaneko* approach and 2. *Local* thresholding. The assumption behind both these methods is that the smaller image regions are more likely to have approx. uniform illumination, thus being more suited for thresholding. Chow and Kaneko divide an image into an array of overlapping sub images and then find the most optimal threshold for each sub image by investigating its histogram. The threshold for every single pixel is found by

interpolating the outcomes of the sub images. But the drawback of this method is that it is computational expensive and, therefore, is not appropriate for our project-like real-time applications.

An alternative interesting approach to finding the local threshold is to statistically examine the intensity values of the local neighbourhood of each pixel. The metric which is most appropriate depends largely on the input image. Simple and fast functions include the *mean* of the *local* intensity distribution.

- **Convex Hulls:**

Here, cv2.convexHull() function checks a curve for convexity defects and corrects it. This is particularly efficient in Character segmentation because various handwritings have different curves and crevices. Generally speaking, convex curves are the curves which are always bulged out, or at-least flat. And if it is bulged inside, it is called convexity defects. This method was taught in our course of "image processing" and hence it's application here is significant. The convex hull is a ubiquitous structure in computational geometry. Even though it is a very useful tool in its own right, it is also helpful in constructing other structures like certain diagrams, and in applications like unsupervised image analysis. But in our case, we have used these various applications to segment out our characters in the most efficient way with less computational Overhead.

We can visualize what the convex hull looks like by a thought experiment. To understand it more, imagine points, let these points be nails sticking out of a plane, now take an elastic rubber band, stretch it around the nails and let it go. It will snap around the nails and take the shape that minimizes its length. The area enclosed by the rubber band is called the convex hull of it. This leads to an alternative definition of the convex hull of a finite set of points in the plane: it is the unique convex polygon whose vertices are points from that region and which contains all points of the nails. This takes us to the next step of drawing the Polylines for segmentation and saving of the characters.

- **PolyLines:**

A polyline is a connected sequence of line segments created as a single object. This is applicable for us to make some physical indication of all the above application for each character. The advantage of Poly Lines is that we can create straight line segments, arc segments, or a combination of the two.

Some reasons we chose to use polylines include the following:

- Vertices remain joined even after grip editing
- Absolute line width (as an alternative to relative line weight) that can be constant or tapered across a segment
- Easily create rectangles and polygons as single objects in combination with our Bounding Rectangles function.

The result thus until now becomes:

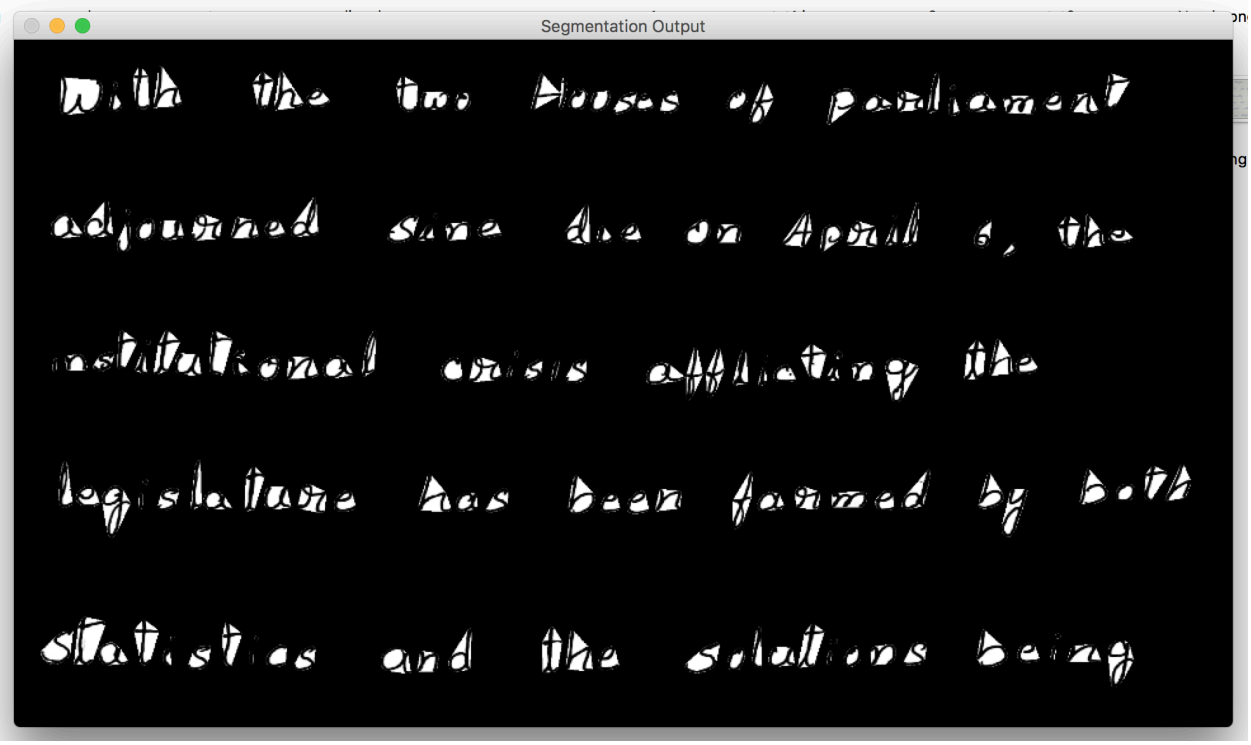


Fig 6.6: MSER + Convex Hulls

Seq2seq TRAINING ARCHITECTURE:

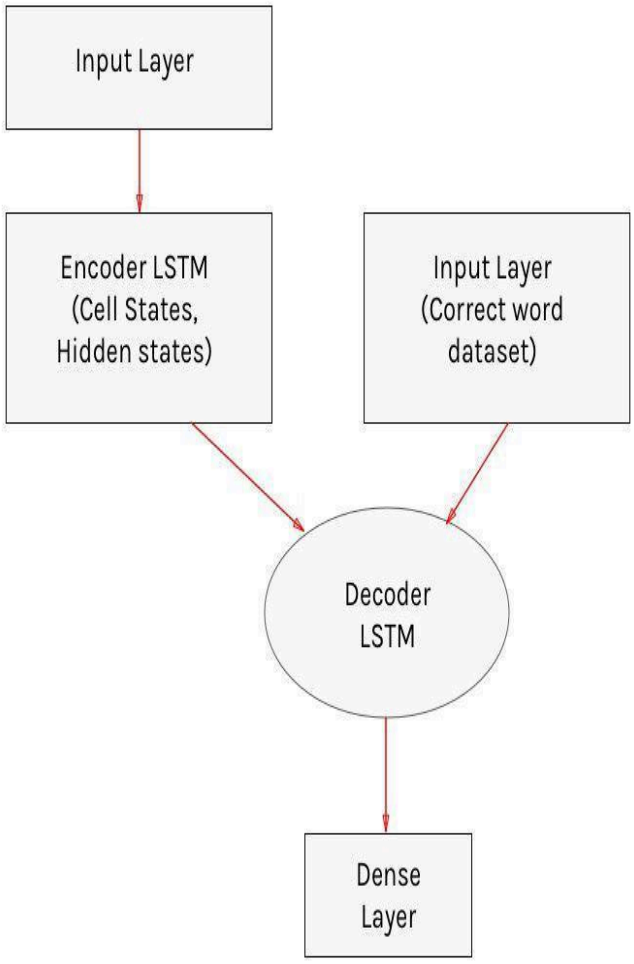


Fig 6.4: seq2seq training architecture

Seq2seq PREDICTION ARCHITECTURE:

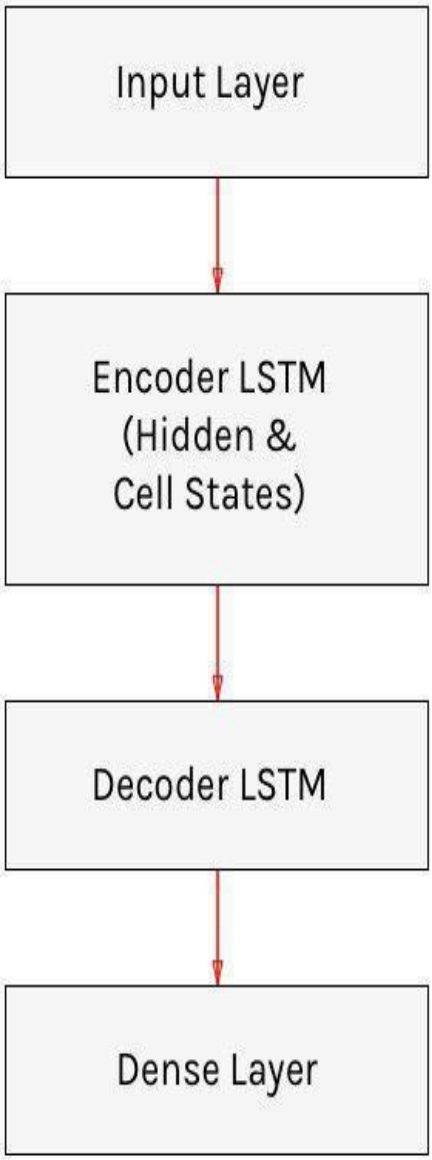


Fig 6.5: seq2seq prediction architecture

CORRECTOR MODULE:

- **Corrector dataset:**

- **Spell Corrector:**
- **Auto Correct:** An initial simple auto correct module is used to filter out the already correct words.

6.3 DATA FLOW DIAGRAM - LEVEL 0

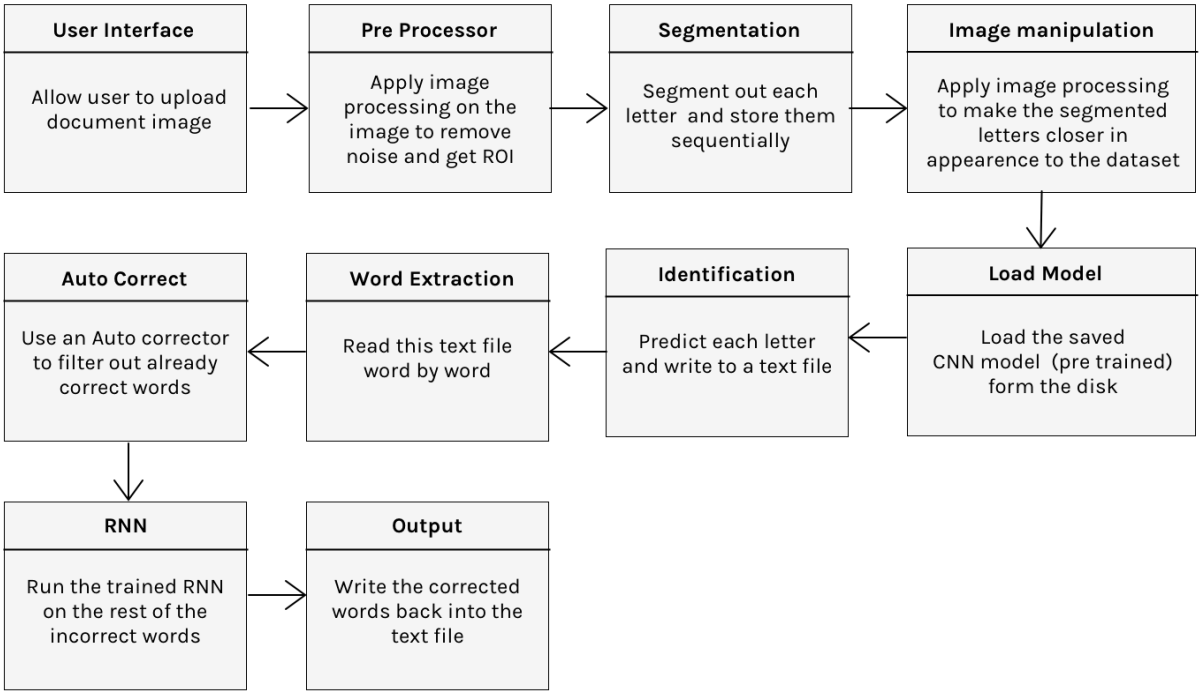


Fig 6.4: Data Flow Diagram Level 0

CHAPTER-7

DETAILED DESIGN

- **Module Name:** Training
- **Input:** EMNIST dataset
- **Description:** This module is responsible for training the Convolutional Neural Network to recognise the characters of the English alphabet. The EMNIST database is originally in the Unix Executable format, which is unpacked as images of letters each of 28x28 pixels. This data is then: 1. retyped to float, 2. Normalized, 3. Shaped to 3D, 4. One Hot Encoded. After which, the layers of the Deep Neural Network are stacked and trained on this data for a number of epochs until over fitting occurs.
- **Output:** 1. Model which is saved as a json file. 2. Weights saved as h5 file.

- **Module Name:** Recognizer
- **Input:** A handwritten document image of format .png or .jpg
- **Description:** This module is responsible for pre-processing of the image and character by character segmentation, following which, some image manipulations are done and saved into the local directory. Each of these are read back for prediction by our previously saved model of .JSON (for model) and .h5 (for weights) files. The recognised letters are read back into a text file with spaces wherever present.
- **Output:** A text file of the image document converted to digitized text

- **Module Name:** Autocorrect
- **Input:** A word which is output from the recognizer
- **Description:** Levenshtein distance between two words is based on the number of insertions, deletions and replacements of letters needed to transform one word into

another. The Levenshtein distance between a rightly spelled word and a word found in the dictionary corpus of the English lexicon will be zero. This method is used to identify the correctly spelled words output from the recognizer so that the dataset required to train the LSTM network to predict the same word given a correct word is significantly reduced.

- **Output:** A word which will be used to take decisions about the words to pass to the LSTM.

- **Module Name:** Corrector (Seq2seq model training)
- **Input:** A dataset of incorrect-correct word pairs.
- **Description:** The module uses teacher forcing method on a seq2seq module to train the system to predict the correct word given an incorrectly spelled word. The module has an encoder and a decoder unidirectional, single layer LSTM network. The encoder LSTM takes the incorrect word as input one letter at a time in each time step and manipulates the hidden state and the cell state of the LSTM node. The final states of the encoder is output and is used to initialise the states of the decoder. The target sequence in the training set is the correct word and it is input to the decoder one letter at a time and the system is expected to train such that it outputs the same sequence as the input to the decoder, but with one time step lag. BPTT (Back propagation through time) is used to tune the various weights.
- **Output:** A trained model saved as a .h5 file.

- **Module Name:** Corrector (Seq2seq model prediction)
- **Input:** An incorrect word.
- **Description:** The input word is encoded into a one hot vector, letter by letter. It is then fed into an already trained encoder whose output is the last hidden state and the cell state. After this is used to initialise the trained decoder, a start token is input to the decoder to seed the prediction. The decoder predicts letter after letter and the previous predicted letter is taken as the input to the next time step and ideally the entire predicted sequence should be the correctly spelled word

- **Output:** The correct word.

CHAPTER-8

IMPLEMENTATION/PSEUDO CODE

As in the case of all our subsections, this one consists of three major blocks.

They are:

1. The segmentation block
2. The CNN block
3. The seq2seq LSTM block

The Segmentation Block

The segmentation algorithm is based on the principles of MSER and convex hulls. Some introduction into the concepts:

MSER:

MSER regions are connected areas characterized by almost uniform intensity, surrounded by contrasting background.

• They are constructed through a process of trying multiple thresholds.

• The selected regions are those that maintain unchanged shapes over a large set of thresholds.

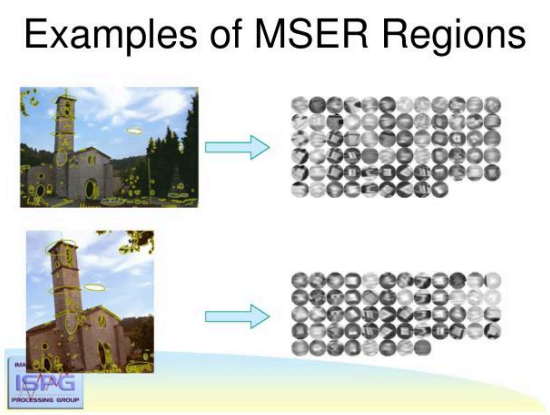


Fig 8.1: MSER Regions

Convex Hulls:

Given a set of points in the plane, the convex hull of the set is the smallest convex polygon that contains all the points of it.

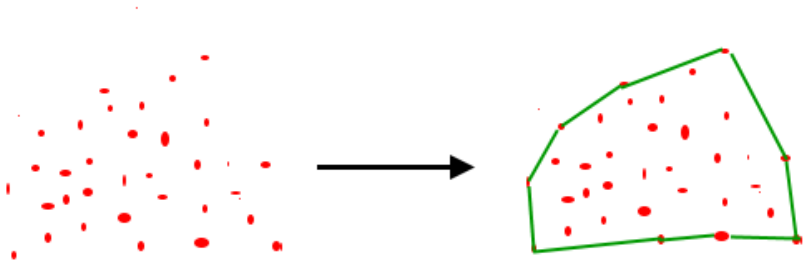


Fig 8.2: Convex Hulls Theory

Fig 8.3: Convex Hulls

Using a combination of the concepts of MSER and convex hulls (in opencv), the input image of the handwritten data is segmented into individual characters. These characters are taken in sequentially and recognised using a CNN whose architecture has been depicted in a previous section.

A simple description of a CNN is as follows:
Convolutional neural networks are deep artificial neural networks that are used primarily to classify images (e.g. name what they see), cluster them by similarity (photo search), and perform object recognition within scenes. They are algorithms that can identify faces, individuals, street signs, tumours, platypuses and many other aspects of visual data.
Convolutional networks perform optical character recognition (OCR) to digitize text and make natural-language processing possible on analog and handwritten documents, where the images are symbols to be transcribed. CNNs can also be applied to sound when it is represented visually as a spectrogram. More recently, convolutional networks have been applied directly to text analytics as well as graph data with graph convolutional networks.
The efficacy of convolutional nets (ConvNets or CNNs) in image recognition is one of the main reasons why the world has woken up to the efficacy of deep learning. They are powering major advances in computer vision (CV), which has obvious applications for self-driving cars, robotics, drones, security, medical diagnoses, and treatments for the visually impaired.
A few pre-processing techniques have been employed to the image before sending it into the CNN. It is first converted into grayscale and inverted.

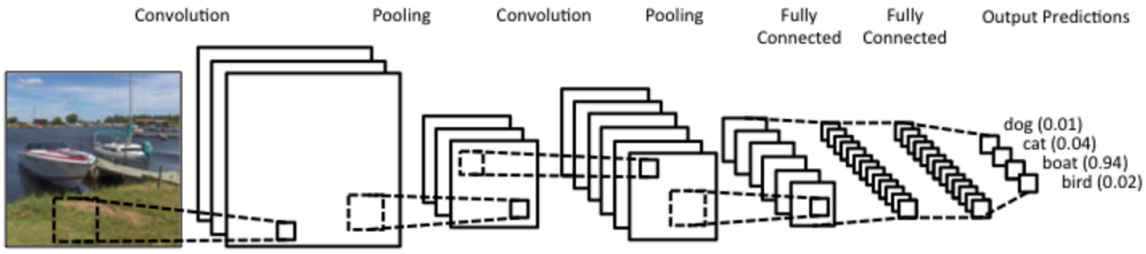


Fig 8.4: CNN Layers example

The output of the CNN (individual labels where each represents a letter) is stored into a text document labelled output.

This text document is taken as an input to RNN which is employed in word correction of misspelt or misidentified words.

The basic concept of an RNN is as follows:

The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks that's a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps (more on this later).

Here is what a typical RNN looks like:

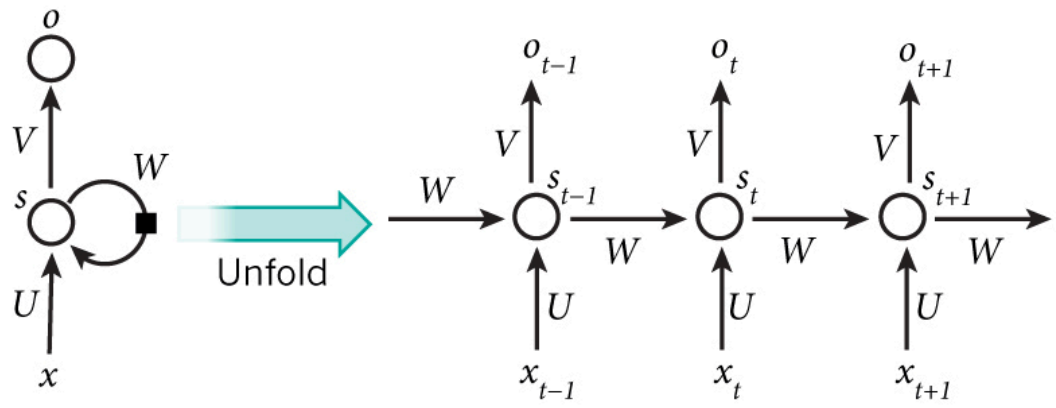


Fig 8.5: RNN

The limitation of the RNN to not "remember" (learn the inter dependencies) over long sequences due to the vanishing gradient problem is overcome by using LSTMs (Long Short Term Memory), a type of RNNs especially efficient at learning long term dependencies. The LSTM decides which long term information to forget, which information to add/remember and finally which form/part of the state to output. In the entire process, the LSTM cell outputs a cell state and a hidden state given the current input, previous hidden state and previous cell states. A mathematical representation of an LSTM cell looks as follows.

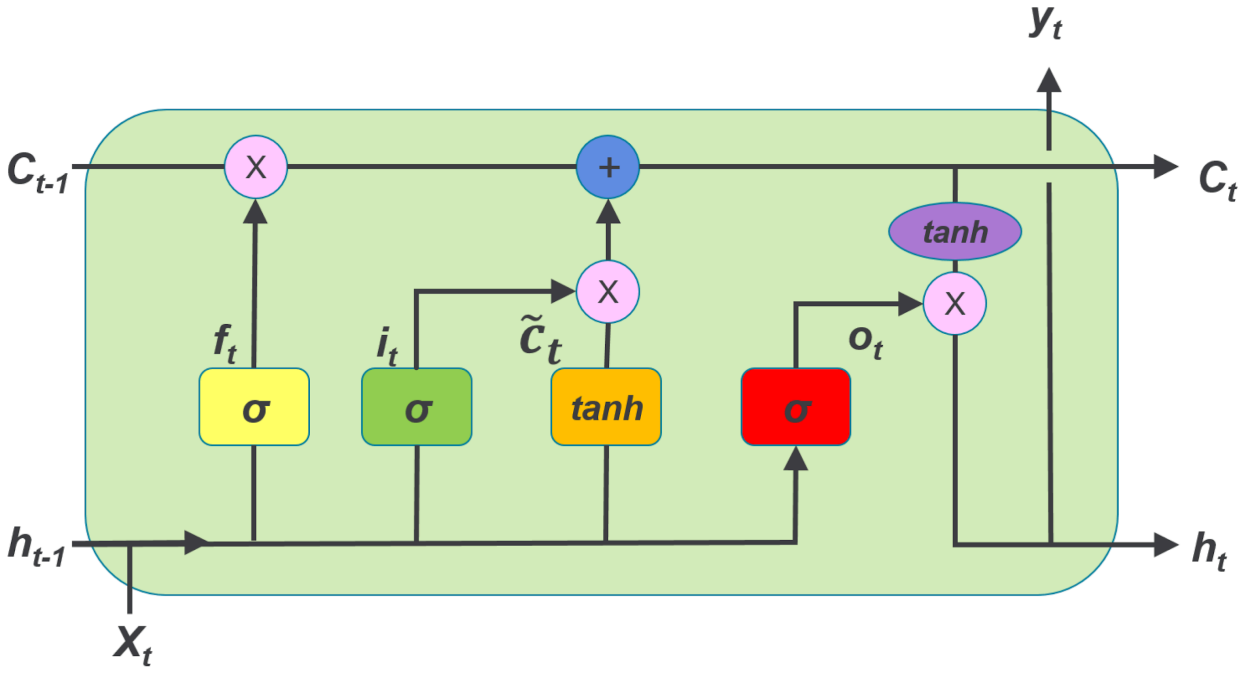


Fig 8.6: LSTM cell

A Sequence to Sequence model is chosen as the corrector module to utilise the entire information provided by the incorrect word and is trained using teacher forcing. The variable length input is converted into a fixed length vector (hence encoded). The decoder then processes this vector to predict a variable length output. The obvious choice for this sequential data is RNN. An RNN network is used as both the encoder and decoder. Different forms of RNN like LSTM, GRUs are commonly used. The networks can be either unidirectional or bidirectional, single layer or multi layered. The encoder usually takes in the input without producing any prediction. The decoder on the other hand, processes the vector, target sequence while predicting the next character. This is usually also referred to as encoder-decoder architecture.

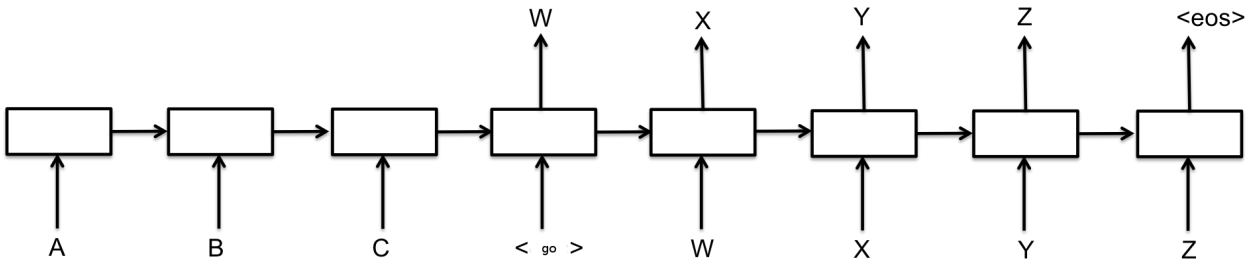


Fig 8.7: Seq2seq model

PSEUDO CODE:

Imports:

```
import tkinter as tk
from tkinter import *
from tkinter import ttk
from tkinter import filedialog
from scipy.misc import imread, imresize,imshow
import cv2
import numpy as np
import keras.models
from skimage.transform import resize
```

```
import imageio
import struct
import matplotlib
matplotlib.use('agg')
#import matplotlib.pyplot as plt
from keras.models import Sequential, load_model, Model
import os
from keras.models import model_from_json
from scipy.misc import imread, imresize,imshow
from PIL import Image
from search import srch
from autocorrect import spell
from corrector_lstm import decode
```

1. **Unpack images from Dataset:**

   Define function filename:

   Open filename as f:

   Datatype, Dimensions = Unpack structure

   shape_of_db = convert the unpacked structure into tuple

   Reshape shape_of_db and typecast to unit8

   Return shape_of_db

2. **Dataset pre-processing:**

   Retype data to float 32

   Set max_val to brightest pixel value, i.e 255

   Training_images = Training_images/255

   Testing_images = Testing_images/255

   Reshape the data to (1,28,28,1)

   One hot encoding of image labels:

```
train_label = keras.utils.to_categorical(train_label, Number_Classes )
test_label = keras.utils.to_categorical(test_label, Number_Classes)
```

3. **Initialize Model and Train:**

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='linear',
        input_shape=shape))
model.add(LeakyReLU(alpha = 0.001))
model.add(Conv2D(64, (3,3), activation='linear'))
model.add(LeakyReLU(alpha = 0.001))
model.add(Conv2D(64, (3,3), activation='linear'))
model.add(LeakyReLU(alpha = 0.001))
model.add(Conv2D(64, (3,3), activation='linear'))
model.add(LeakyReLU(alpha = 0.001))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(Number_Classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
        optimizer=keras.optimizers.Adadelta(),
        metrics=['accuracy'])
```

4. **Character Segmentation:**

Create an MSER Object

Read image in Grayscale (Img)

Img = Apply MedianBlur on the Image to remove noise

Img = Apply adaptive thresholding on the image to get extreme pixels

Detect the MSER regions on Img

Hulls = Detect the convex Hulls on the Regions of Interest, i.e MSER Regions

Draw the lines joining these Hull points

mask = create a list of zeros and subtract with original image

mask = Dilate the Black pixels

mask = Erode the Black pixels

FOR contour in Hulls:

Draw the contour

text_only = Bitwise And to remove the background

FOR i, contour in enumerate(hulls):

x,y,w,h= Get the coordinates of the Bounding Rectangle

Call Function to remove the unnecessary inner rectangles detected

Call Function to sort the rectangles in order

Call Function to insert spaces

Call Function to save each letter as an image in the filesystem

5. Dataset preparation for spell corrector

Read the correct words followed by list of corresponding incorrect words

Append all words into a list

Store correct word in temp variable

Write incorrect word into file followed by a tab and the correct word

Jump to next line

Type in incorrect-correct word pairs resulting from recognition, following above convention

6. Seq2seq Model Initialisation

Specify batch size, number of samples and epochs

Specify the latent dimensionality of the encoding space

Read the incorrect-correct word pairs from dataset

Split the dataset on the new line and store the lines in a list

For each line:

Extract input_seq , target_seq by splitting on tab

Prepare target_seq for training by adding a tab character prefix and new line character as suffix

Append the input_seq into a list to form the input for lstm network training

Append the target_seq into a list to form the target for lstm network training

Input_characters = set of all unique characters in the input sequences

Targert_characters=set of all unique characters in the target sequences

Determine the number of encoder tokens and target tokens, maximum length of the encoder input sequence and decoder sequence

Create dictionary of the form “input character: number” and “target character: number”

Create 3D null arrays of dimensions equal to the encoder_inputs, decoder_inputs and decoder_outputs

Create one hot encoded vectors of encoder_inputs,decoder_inputs and decoder_outputs

Define input layer as encoder inputs

Define the encoder LSTM layer

Use the return states=True function to capture the outputs, hidden states, cell states

Save the last encoder hidden state and cell state

Define an input layer for decoder network

Define the decoder LSTM and initialise it’s states with the saved encoder hidden state and cell state

Define a Dense layer

Define the model as taking encoder_inputs and decoder_inputs and predicting decoder_outputs

Compile the model

Fit the model

Save the weights of the model to an .h5 file

7. Seq2seq prediction model

Initialise the encoder_tokens, decoder_tokens, encoder_seq_length, decoder_seq_length, dictionary to convert input_tokens and target_tokens from characters to numbers

from the model trained

Define the same model as the training model and load the weights

Define the decoder states inputs

Define decoder LSTM layer which requires, the encoded output so far, the hidden and the cell states provided by the encoder initially and later the output of the decoder states as the model is called recursively

Define a dense layer to predict the characters, probabilistically

To correct an input sequence:

Encode the inputs into state vectors

Create an empty target sequence of length one

Initialise the first character of the sequence as the start character

Predict the letters by sampling current output as input to the next time step

Update the states

8. Word corrector module

Read the text file output from the recogniser

For each word in the file:

Pass through autocorrect module

If output is same as input, append to the list

Else pass through the trained seq2seq LSTM prediction model

and append the output of LSTM to the list

Write the list into a new file

CHAPTER-9

TESTING

Testing was performed on the various modules to see which implementation worked best. Firstly, the choice of model for training was crucial. We initially started with a simple 4 layer Neural Network, but eventually moved to Convolutional Neural Networks for better accuracy. Here is a stark comparison:

NN:

```
  matplotlib.use('agg')
Train on 124800 samples, validate on 20800 samples
Epoch 1/5
124800/124800 [==============================] - 1150s 9ms/step - loss: 0.8944 - acc: 0.7334 - val_loss:
0.5549 - val_acc: 0.8368
Epoch 2/5
124800/124800 [==============================] - 1179s 9ms/step - loss: 0.6199 - acc: 0.8115 - val_loss:
0.4720 - val_acc: 0.8588
Epoch 3/5
124800/124800 [==============================] - 1100s 9ms/step - loss: 0.5514 - acc: 0.8311 - val_loss:
0.4327 - val_acc: 0.8699
Epoch 4/5
124800/124800 [==============================] - 1190s 10ms/step - loss: 0.5187 - acc: 0.8410 - val_loss:
0.4176 - val_acc: 0.8740
Epoch 5/5
124800/124800 [==============================] - 1180s 9ms/step - loss: 0.4964 - acc: 0.8465 - val_loss:
0.4107 - val_acc: 0.8732
20800/20800 [==============================] - 25s 1ms/step
ok
Test loss: 0.4107382860751106
Test accuracy: 0.8731730769230769
```

Fig 9.1: NN Accuracy

CNN:

```
  matplotlib.use('agg')
Train on 124800 samples, validate on 20800 samples
Epoch 1/5
124800/124800 [==============================] - 492s 4ms/step - loss: 0.7165 - acc: 0.7829 -
val_loss: 0.2904 - val_acc: 0.9100
Epoch 2/5
124800/124800 [==============================] - 463s 4ms/step - loss: 0.3955 - acc: 0.8762 -
val_loss: 0.2528 - val_acc: 0.9216
Epoch 3/5
124800/124800 [==============================] - 477s 4ms/step - loss: 0.3375 - acc: 0.8945 -
val_loss: 0.2305 - val_acc: 0.9269
Epoch 4/5
124800/124800 [==============================] - 499s 4ms/step - loss: 0.3133 - acc: 0.9019 -
val_loss: 0.2228 - val_acc: 0.9280
Epoch 5/5
124800/124800 [==============================] - 481s 4ms/step - loss: 0.2985 - acc: 0.9066 -
val_loss: 0.2186 - val_acc: 0.9314
20800/20800 [==============================] - 27s 1ms/step
Saved trained model at /Users/parallellabs/Desktop/Project/keras_emnist.h5
Test loss: 0.21862426132062235
Test accuracy: 0.9313942307692308
```

Fig 9.2: CNN Accuracy

A dataset of incorrect-correct word pairs was collected and the LSTM network trained. The model was trained with around 30000 samples and validated on around 7000 samples (0.2 of the training dataset).

The results are as shown.

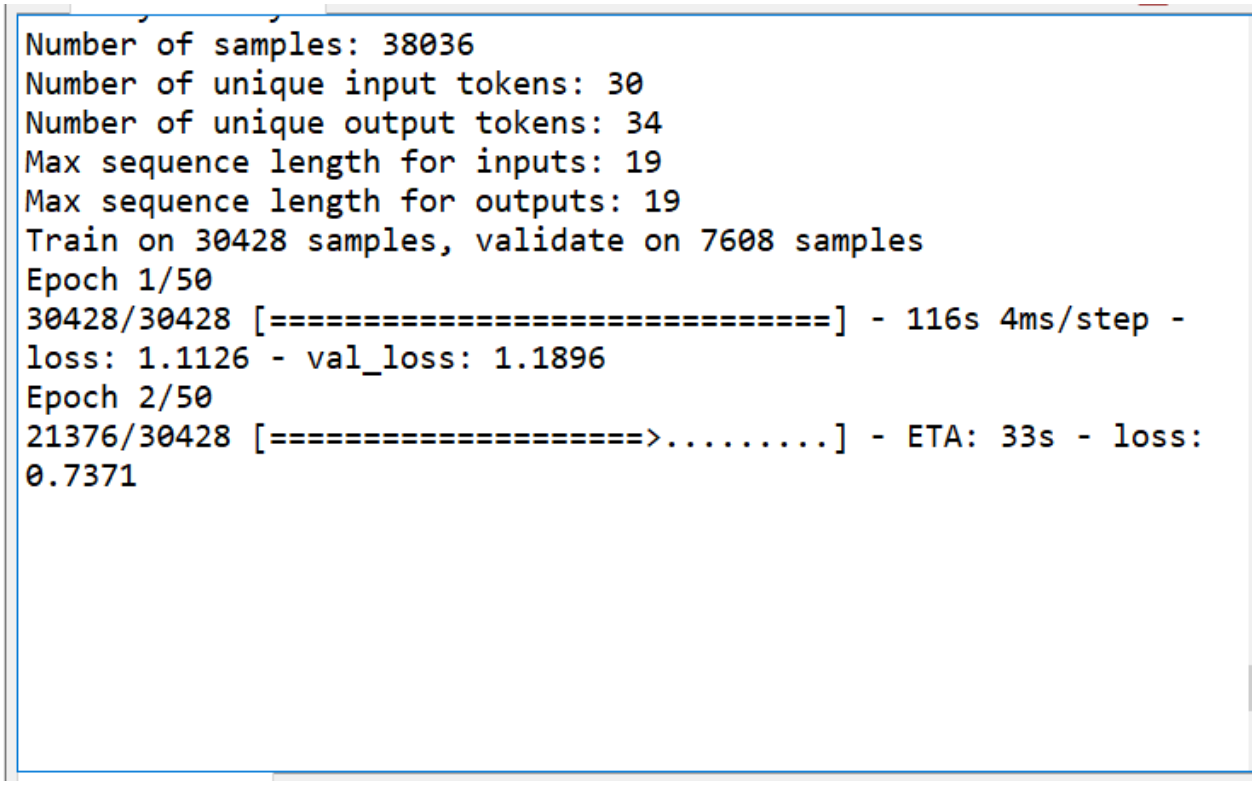
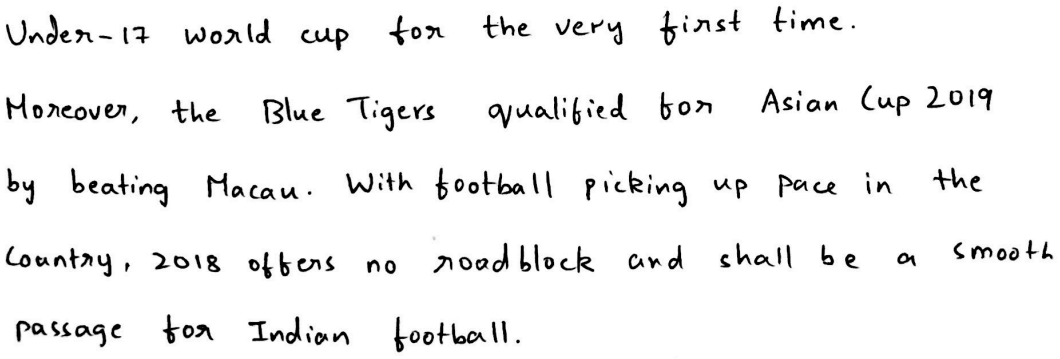


Fig 9.3: Statistics of the seq2seq model

```
Epoch 45/50
30428/30428 [==============================] - 105s 3ms/step -
loss: 0.0263 - val_loss: 1.6109
Epoch 46/50
30428/30428 [==============================] - 105s 3ms/step -
loss: 0.0259 - val_loss: 1.5815
Epoch 47/50
30428/30428 [==============================] - 105s 3ms/step -
loss: 0.0255 - val_loss: 1.5910==============>....] - ETA: 12s
- loss: 0.0250
Epoch 48/50
30428/30428 [==============================] - 107s 4ms/step -
loss: 0.0248 - val_loss: 1.6231
Epoch 49/50
30428/30428 [==============================] - 109s 4ms/step -
loss: 0.0245 - val_loss: 1.6268
Epoch 50/50
30428/30428 [==============================] - 108s 4ms/step -
loss: 0.0243 - val_loss: 1.6402
```

Fig 9.4: Accuracy of the seq2seq model

Testing was done on various sets of handwriting to get the most reliable model and output. A few handwriting testing samples that are within the constraints and work well with our algorithm are:



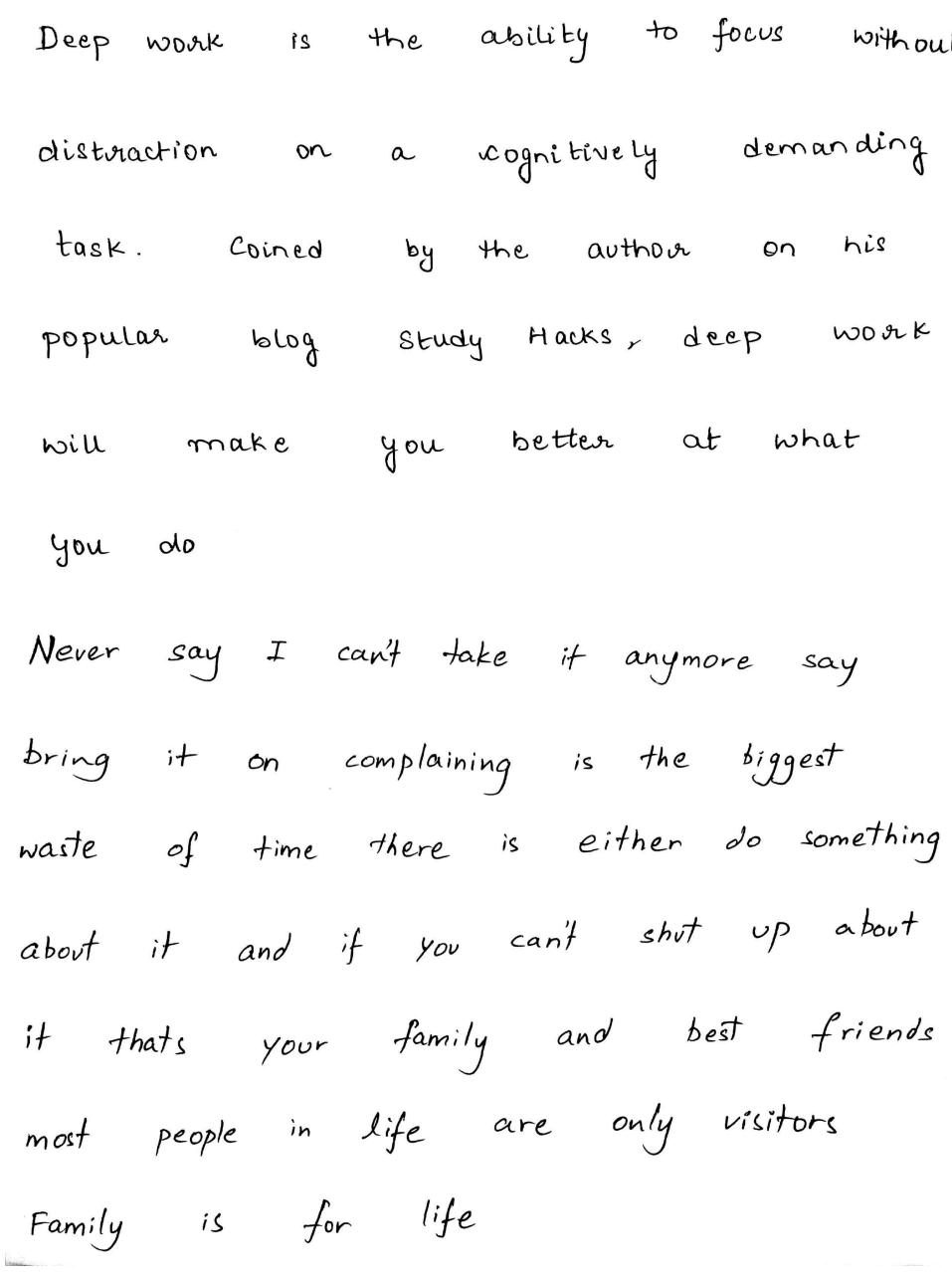


Fig 9.3: Test Samples

Testing on previous such samples, we found certain constraints such as word interspacing and line interspacing thresholds that our system needed to handle. Sample segmentation outputs looked like:

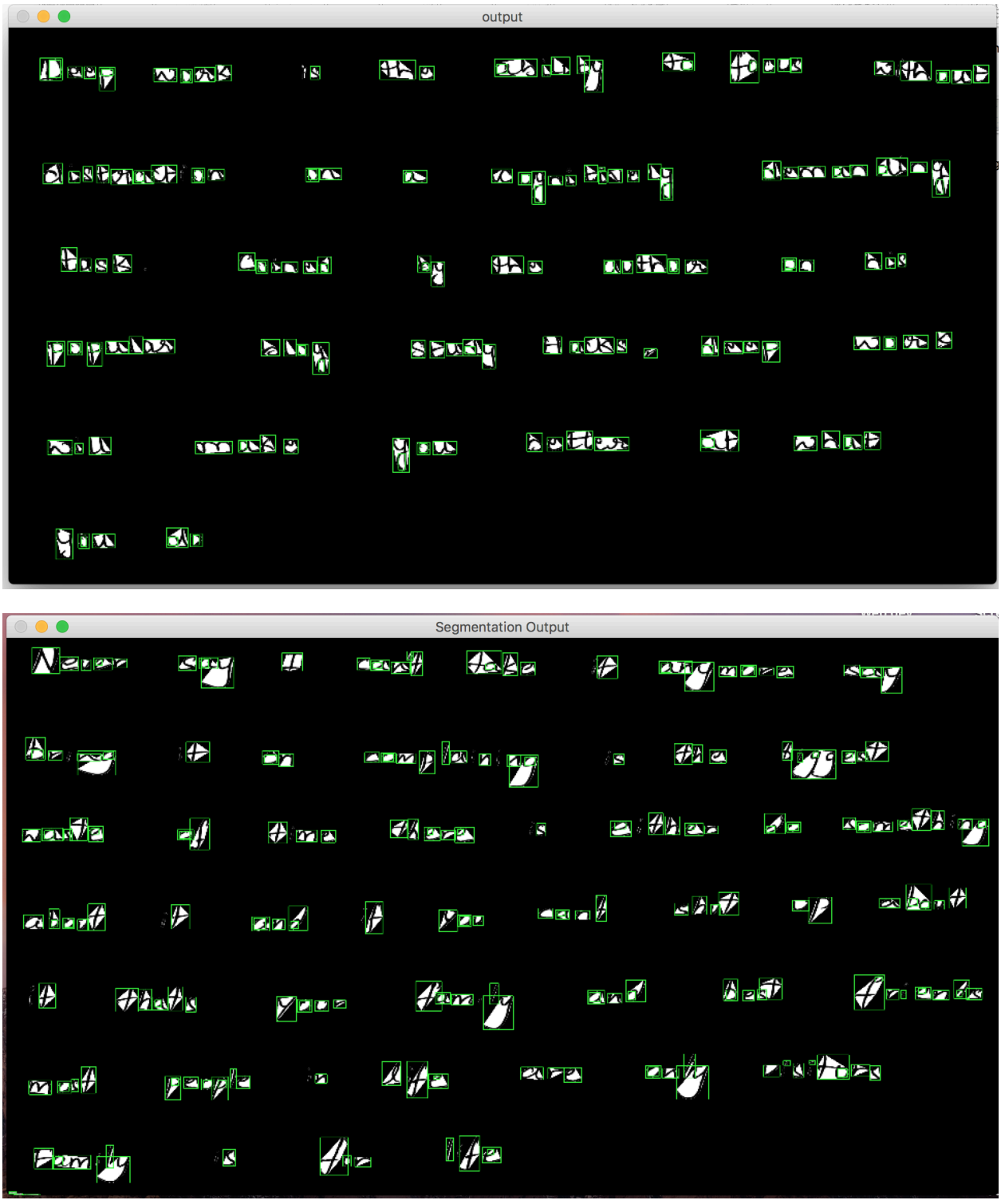


Fig 9.4: Segmentation Output

These segmented letters are resized to fit in a proportional square by adding a certain amount of pixel padding of White pixels on either side. For example:

Fig 9.5: Segmented characters

The document output by the recogniser when the segmented characters were sent in was found to be as follows.

A few samples that fail with our testing suite include:

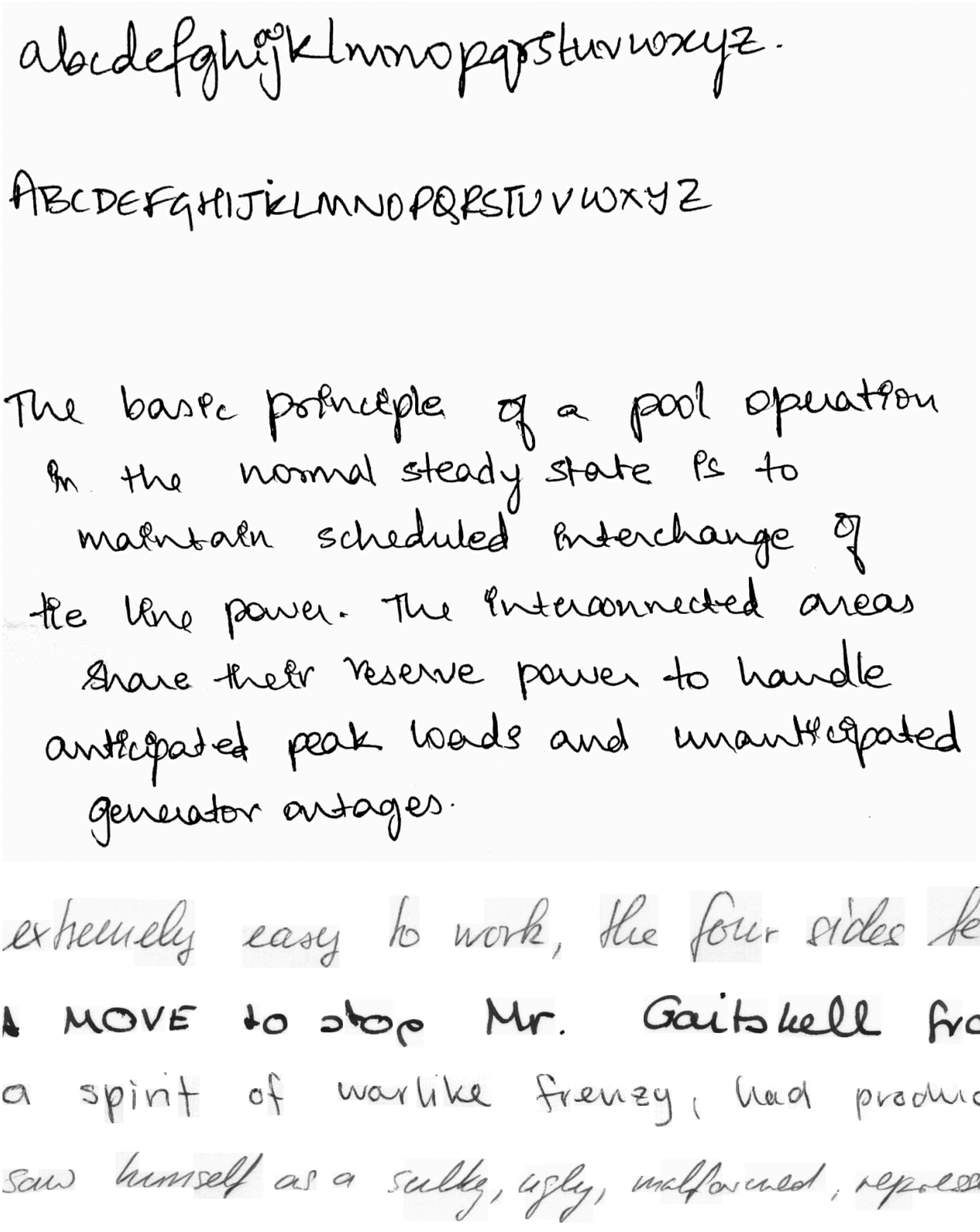


Fig 9.6: Erroneous test images

CHAPTER-10

RESULTS AND DISCUSSIONS

The test samples were first tested with a Neural Network model and accuracy was noted, following which a 2-layer Convolutional Neural Network was built, improving it further, we chose a 4-layer Convolutional Neural Network as our final model and the test cases showed decent results with a 60-65% accuracy rate with varied handwritings. Note that this was before using our Corrector Module for spelling correction. For example:

The serial run of test cases on a various kinds of handwritings took a considerable amount of work on our part. This was a colossal waste of time and it wasn't practical to run this for building a prototype. Which is why we put forth certain minimal constraints for the system to work as expected.

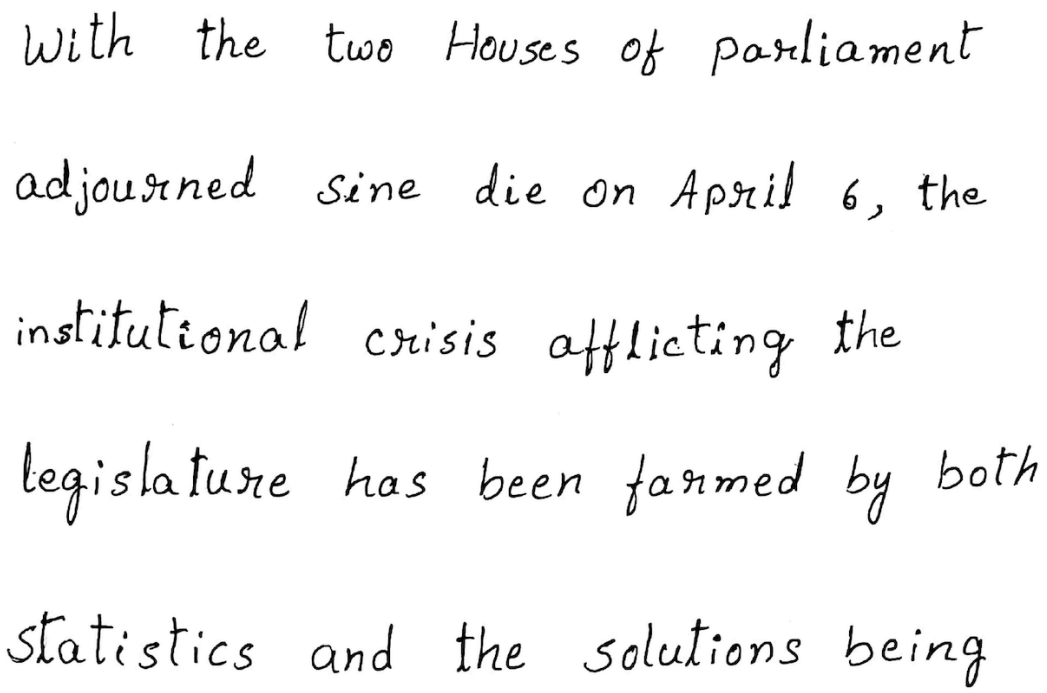
After writing the corrector module, a lot of these mistakes from the Recognizer module were corrected and an accuracy of about xx% was achieved. The result being:

This was a large improvement on the Recognizer module and proved to be extremely useful in everyday scenarios. All the user now had to do was to execute our program file and they would have to wait for just a few seconds to get the converted document which they could correct and save back into the text file.

With this in mind, further improvements can be made to our system which is explained in the following section.

Sample Input/output results:

1.



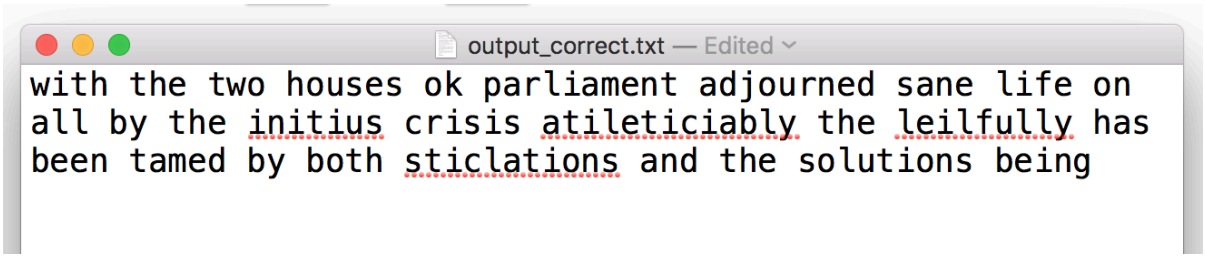
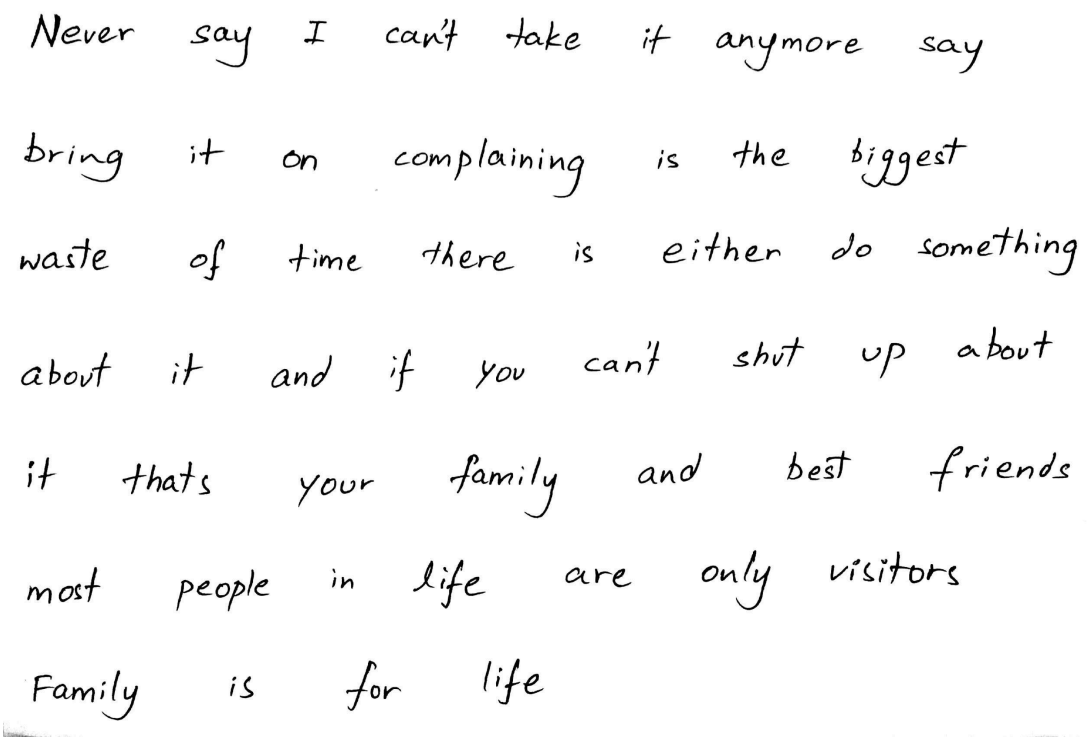


Fig 10.1: Test result 1

2.



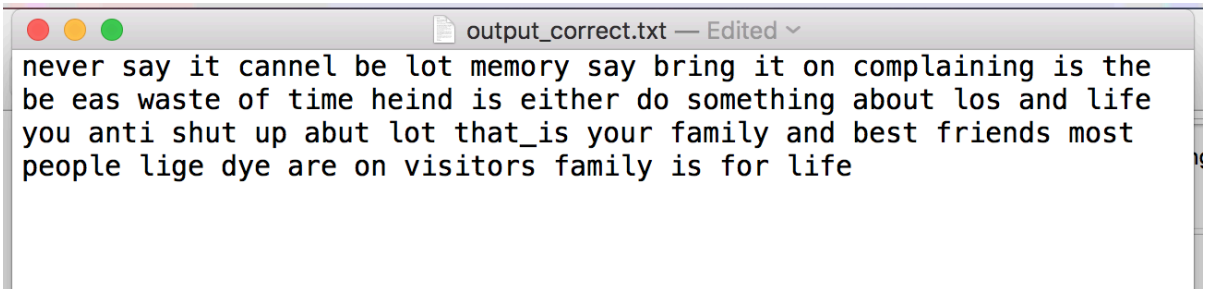
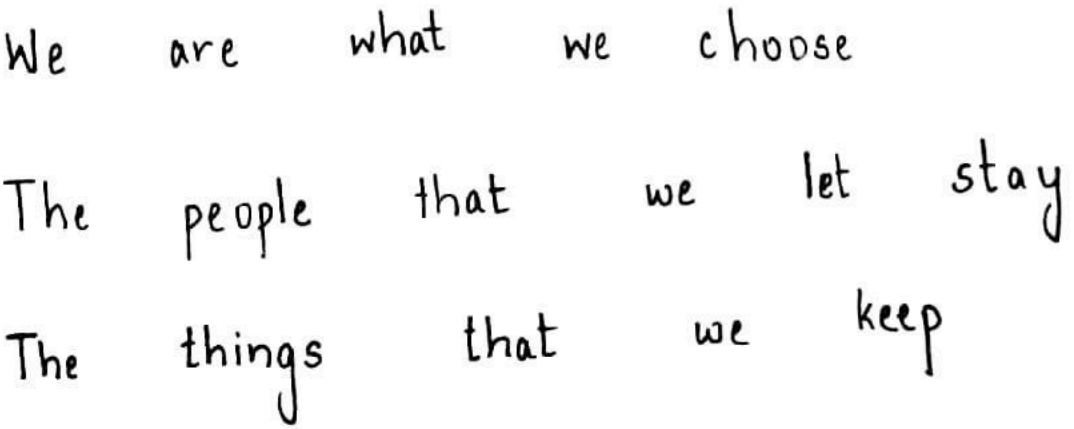


Fig 10.2: Test result 2

3.



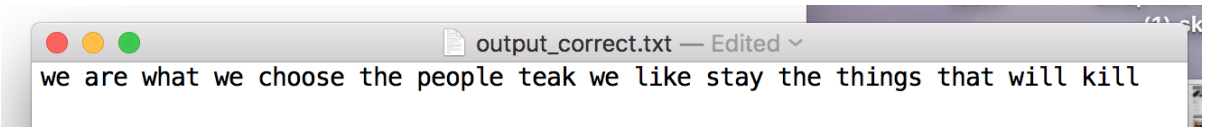


Fig 10.3: Test result 3

4.

Deep work is the ability to focus without distraction on a cognitively demanding task. Coined by the author on his popular blog Study Hacks, deep work will make you better at what you do

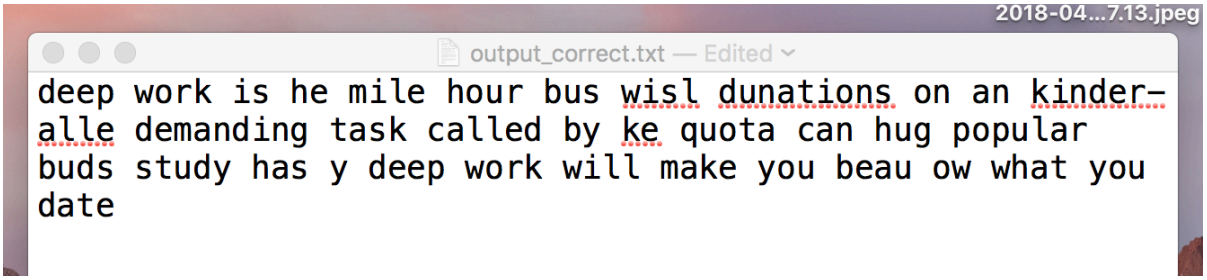


Fig 10.4: Test result 4

CHAPTER-11

SNAPSHOTS

User Interface:

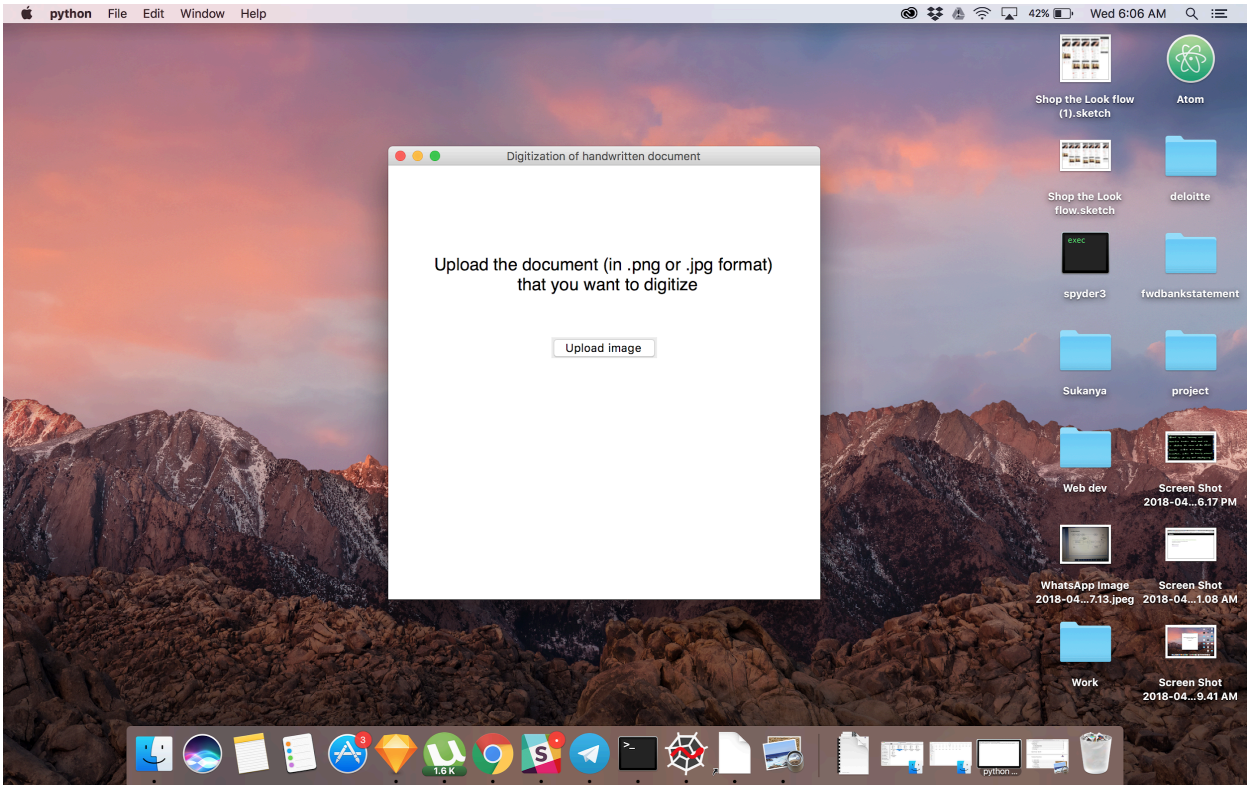


Fig 11.1: UI 1 - Upload document

Fig 11.2: UI 2 - Converting

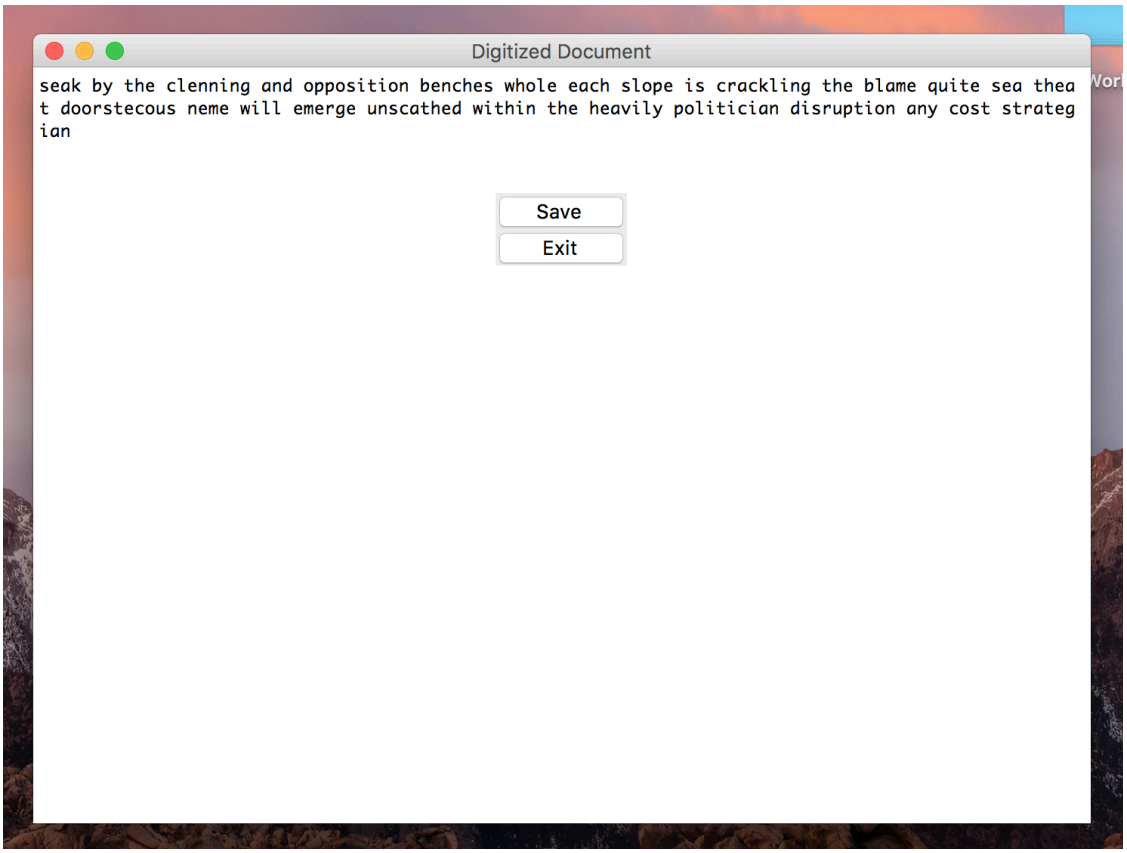


Fig 11.3: UI 3 - Digitized Doc

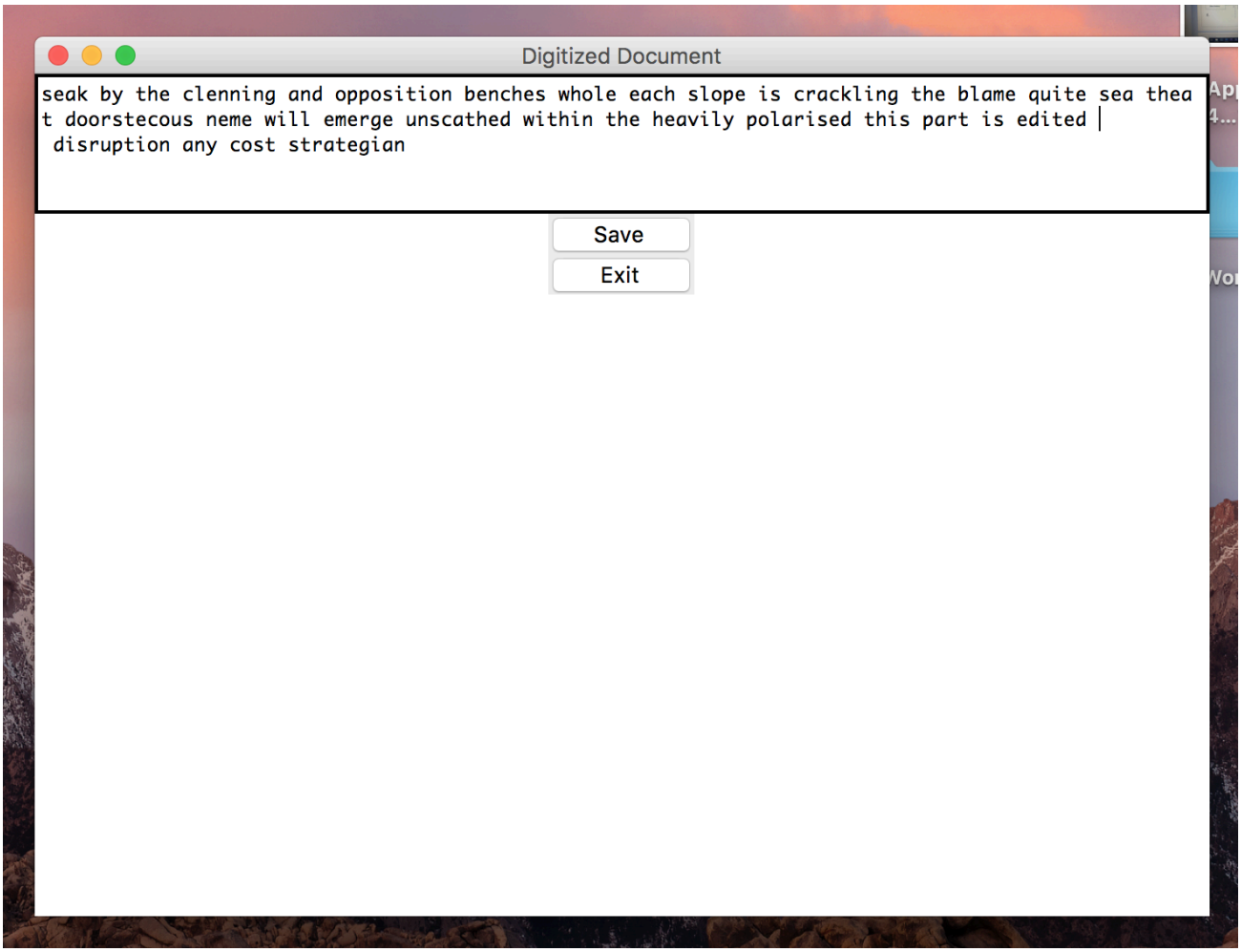


Fig 11.4: UI 4 - Editing the Doc and save

Fig 11.5: Pre Processing to make data close to dataset image

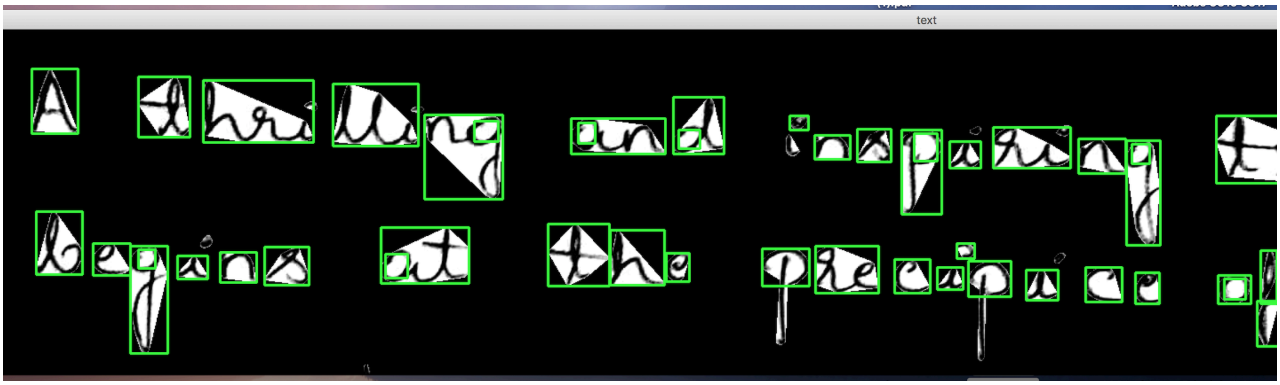


Fig 11.6: Segmentation Results

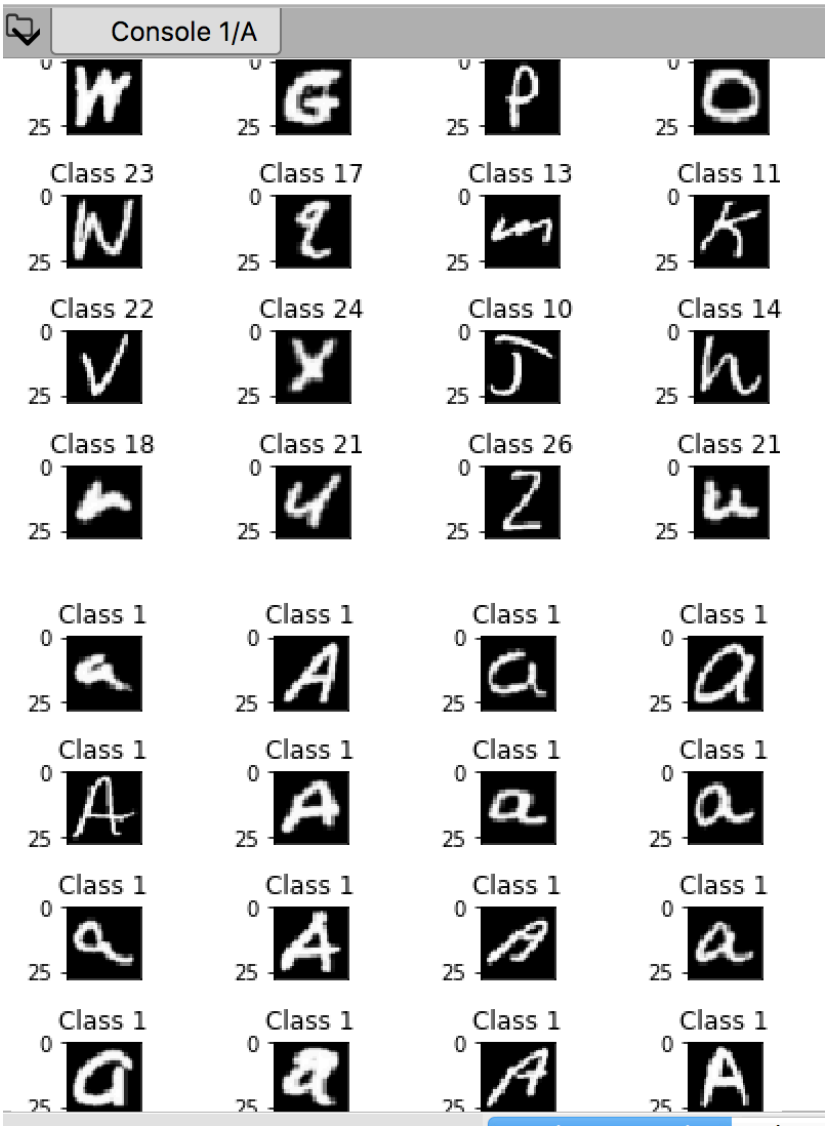


Fig 11.7: Adding Own dataset to the training set

CHAPTER-12

CONCLUSIONS

This project gave us significant insight not only in Techniques of Machine Learning and Natural Language Processing, but also in the application of software engineering principles in the industry. The experience enabled us, as a team to develop as a programmer, thinker and also a team player. Venturing into the space gives us an advantage to further our careers in this path.

This project was envisioned with a goal in mind: Build a system that is simplistic to use yet complex and in it's working, and that is what we have aimed to achieve through all our modules.

Creating an impact in the world is something we consider to be an achievement. The tool that we have developed can be used every day in a commoner's life and will make it a little, if not greatly, easier.

The project exposed us to the numerous technology stacks existing in this space. We also got to learn the impact of Artificial Intelligence in today's world. This enabled us to appreciate the various courses related to Image Processing, Pattern Recognition and ANN courses we did in college much more because we was able to see the concepts working in real life.

Although the project seemed a little difficult at first because we was not well equipped with the required knowledge of Machine Learning and those particular subjects in college, as the development went on, we was able to learn all the required technologies and implement everything. This project has further scope and will continue to be developed.

CHAPTER-13

FURTHER ENHANCEMENT

There is room for improvement in our project

- The real challenge that comes into play is recognition of cursive handwriting. The current accuracy of our model requires infinite improvement when a sample with cursive handwriting is used. Therefore, a more sophisticated dataset set will take things forward by leaps and bounds. Similarly a more robust segmentation algorithm will help segment cursive handwriting by a much greater extent.

- There are several constraints that our data sample must adhere to in order to generate an appropriate efficiency for our sample dataset. If the sentence is written off - kilter, the letters are not recognised in order, thereby rendering the process useless. Similarly, the spacing between individual lines and words need to be very conspicuous in order to correctly ensure that the sequence of letters are maintained.

- When RNN's are used, every word is sent into the corrector regardless of whether or not it is correctly or incorrectly misspelt. This can lead to the incorrect auto correction of correctly spelt words.

CHAPTER - 14

BIBLIOGRAPHY / REFERENCES

There are several sources used for the information needed to work on this project

They are as follows:

1. The various blog posts of Analytics Vidhya
2. The sources of the IEEE papers and blogs:

[1]: "A Novel Connectionist System for Unconstrained Handwriting Recognition" Alex Graves, Marcus Liwicki, Santiago Fernaʹndez Roman Bertolami, Horst Bunke, Juʈrgen Schmidhuber

[2]: "Sentence Correction using Recurrent Neural Networks" Gene Lewis, Department of Computer Science Stanford University Stanford, CA 94305

[3]: "A Method for Text Localization and Recognition in Real-World Images" Lukas Neumann, Jiri Matas. Part of the Lecture Notes in Computer Science book series (LNCS, volume 6494)

[4]: "Robust Text Detection in Natural Scene Images" Xu-Cheng Yin, Member, IEEE, Xuwang Yin, Kaizhu Huang, and Hong-Wei Hao

[5]: "Handwritten Character Recognition by Alternately Trained Relaxation Convolutional Neural Network" Chunpeng Wu, Wei Fan, Yuan He, Jun Sun, Satoshi Naoi Fujitsu Research & Development Center Co. Ltd., Beijing, 100025, China

[6]: "Deep Spelling - Rethinking spelling correction in the 21st century" Tal Weiss, Founder, Evature