Computational Modelling of an Optical Character Recognition System for *Yorùbá* Printed Text Images[⋆]

Ọlálékan Joseph ÒNÍ[a,*], Franklin Ọládiípọ̀ ASAHIAH[a]

[a] *Computing and Intelligent Systems Research Group, Department of Computer Science & Engineering, Ọbáfẹ́mi Awólọ́wọ̀ University, Ile-Ife, Nigeria. 220282*

Abstract

This study acquired a dataset of scanned images of Standard *Yorùbá* printed text and formulated a *Yorùbá* character image recognition model. The model formulated was implemented and the performance of the model evaluated to develop an Optical Character Recognition (OCR) model for *Yorùbá* printed text images.

The image dataset at 300 dots per inches (dpi) was acquired by generating image text-line from *Yorùbá* New Testament Bible (Bibeli Mimo) corpus using Unicode UTF8. The Long Short Term Memory (LSTM) model, a variant of Recurrent Neural Network (RNN) was used to formulate the Standard *Yorùbá* character image recognition model. The Python OCRopus framework was used to implement the model designed. The performance of the model designed was evaluated using character error rate based on Levenshtein Edit Distance algorithm.

The results show that the Character Error Rate (CER) of 3.138% for the font Times New Roman which gives better recognition than the other font style metric performance. The model achieved an OCR result of (7.435% CER) DejaVuSans font style image dataset, while for Ariel font image dataset, a result of 15.141% was achieved. The introduction of Language model-based Standard *Yorùbá* a spell-checker corrector show a reduction in the Character Error Rate.

[⋆]This document is a collaborative effort.

[*]Corresponding author

Email address: `olalekan.joseph@gmail.com` (Ọlálékan Joseph ÒNÍ)

The Times New Roman font recorded an error rate of 1.182%, the DejaVuSans font style at an error rate of 4.098% while the Ariel font at 5.87%.

The study concluded that the performance of the model shows that the farther away an image text font is from the font(s) used in training the network, the higher the character error rate of the recognition and that the inclusion of a post-processing stage shows a reduction in the Character Error Rates.

Keywords: Optical Character, *Yorùbá*, Orthography, Computational Modelling, Spell-Check Correction, OCRopus

1. Introduction

According to [1], *Yorùbá* language is one of the languages that are on the verge of "extinction". One of the ways of preserving the language is by creating a digital resource of *Yorùbá* books and literature. It is also equally important for researchers to have these documents in a digital format. This makes it possible to edit such documents, search for a word or phrase, store it more compactly, display or print a copy and apply techniques such as machine translation, text-to-speech and text mining on it [2].

Optical Character Recognition (OCR) is one of the most successful applications of automatic pattern recognition [3]. The task of producing a machine that could read text with the same proficiency as human do was the inspiration behind developing an OCR. The very early systems reported for OCR were intended to help the blind to read. Therefore, an Optical Character Recognition (OCR) systems aim at transforming a large number of documents, either printed or handwritten into machine-encoded text [4]. A system embedded with an optical recognition subsystem improve the speed of input operation and enable compact storage, fast retrieval and other file analysis and manipulation [5]. The range of applications includes postal code recognition, used in large administrative systems, banking, automatic cartography and reading devices for blind [6].

The *Yorùbá* language is spoken by more than 30 million people in places like the United States, United Kingdom, Benin Republic, and (principally) in south-western Nigeria [7]. The *Yorùbá* orthography can be categorized into phonemes and tonemes. The phonemes - which is an abstraction of the physical sound - can be further categorized into four major classes: Consonants, Oral Vowels, Syllabic Nasals, and Nasalised Vowels. In addition to the phonemes, *Yorùbá* uses tones. Tonemes are the representation of contrastive tone patterns in tone languages of which *Yorùbá* is one. In *Yorùbá* orthography, tones are marked on vowels and syllabic nasals using acute accent for high tones (´), grave accent for low tones (`), and the absence of accent for middle tones (¯) (the exception is on syllabic nasals, which are marked with a macron) [7]. The orthography has five nasalised vowels and two pure syllabic nasal vowels [8]. Equation 1 defines the alphabet of *Yorùbá* .

$$\alpha(|Yoruba|) = \begin{cases} Tone & \begin{cases} \{\text{acute tone mark, grave tone mark}, macron\} \end{cases} \\ Grapheme & \begin{cases} Upper-case & \{A, B, ...Y\} \\ Lower-case & \{a, b, ...y\} \end{cases} \end{cases} \tag{1}$$

The justification for this work is hinged on the importance of these diacritics to the interpretation of *Yorùbá* words and its frequency in typically written documents. [7] used a *Yorùbá* corpus with 129,317 word count and generated 239,840 syllables with an automatic syllabification program. The frequency analysis of these syllables showed that 89,824 (37.35%) of the syllables carried the acute accent for a high tone, while 77,369 (32.26%) carried the grave accent for a low tone and 72,647 (30.29%) were mid-tone with no mark or with a macron on a syllable. The frequency distribution indicated an almost even distribution of the tones, as the difference may not be considered statistically significant at a 90% confidence value.

1.1. *What is Computational Modelling?*

In the context of Computer Science and Engineering, Computational Modelling is concerned with the techniques and concepts employed in precisely representing the state and dynamics of any objects of human interest. A computational model, therefore, is a specification of a method for data representation and a mechanism that transforms those representations toward a desired outcome [9]. In other words, computational modelling is the creation and manipulation of symbols in the process of constructing a solution to a well-defined problem [10]. The **symbols** in the context of this work are Literals (the Yoruba orthography); the well-defined **problem** is the human recognition capability of *Yorùbá* characters; **construction** is the design of a *Yorùbá* OCR that can manipulate the symbols to mimic the recognition proficiency of human.

The other parts of this research work are outlined as follows: Section 2 discusses the literature reviewed, Section 3 described the methods used in developing and in evaluating the models, while Section 4 described the results and the conclusions are discussed in Section 5.

2. Related Works

The patent OCR work of [11] was based on template matching. However, unlike modern-day template matching algorithms for pattern recognition, templates were mechanical. A match is found when the light in the optoelectronic sensors fails to reach the detector. As technological advancement in computing became readily available, the template matching approach started appearing as software-based solutions. In this approach, characters are extracted one at a time and then compared against all possible patterns of the characters.

[12] reported the development of an OCR system for the recognition of *Yorùbá* based texts and the conversion of English numerals in the document to *Yorùbá* numerals. The system used correlation and template matching procedures to develop an OCR system for the recognition of *Yorùbá* based-texts and the conversion of English numerals in the document to *Yorùbá* numerals.

The system reported an accuracy of 86% for the typed text. It was also reported that all the English numbers were correctly recognized and converted to the *Yorùbá* numerals.

[13] presented a method for recognizing isolated diacritically-marked handwritten *Yorùbá* characters written in uppercase. The method comprises six steps: the first known as pre-processing stage is used to prepare the document for subsequent steps by basically removing noise from it; the next phase referred as segmentation stage was used for isolating the relevant Latin letter from the diacritical mark(s); the third steps referred to as feature extraction was used in computing the eight geometric properties of the Latin character; the fourth involves the introduction of a Bayesian classifier for the classification of the Latin character extracted from the last step; the penultimate step made use of a decision tree algorithm to recognized the diacritical marks; and finally a result fusion stage was used to combine the result of the two last stages into a single class label. Their performance evaluation shows a recognition rate of 91.18%.

[14] asked people from 10 different group aged between 15 and 50 years to write 200 handwriting English word samples. Part of these samples was written on white paper and others on a colored or a noisy background. He proposed vertical segmentation process in which the segmentation points are located after thinning the word image to get the stroke width of a single pixel. The method employed involves the calculation of the height and width of the word image for the analysis of the ligatures in an accurate manner. An accuracy of 83.5% was reported.

[15] in his thesis reported on the development of language independent OCR for a single script that is used by multiple languages. In other words, a multilingual (for complex modern scripts like Devanagari and Arabic) OCR framework for a printed document by using LSTM model. The testing shows that LSTM-based OCR can yield very low error rates.

2.1. The Model Adopted: Long Short-Term Memory (LSTM) networks

We present an equivalent formulation of the LSTM model conceived by [16] and improved on by [17]. Whilst there have been attempts at using Convolutional Neural Network (CNN) for sequence modelling problems like Speech Synthesis and Machine Translations [18], results shows that the use of LSTM - at context-awareness problems like OCR - are better at recognizing patterns occurring in time-series [19]. LSTM networks are a recent architecture of Recurrent Neural Networks (RNN) [16] proposed to change the basic unit of RNN, using a simple neuron with a computer memory-like cell, called LSTM cell. A simplified LSTM cell is shown in Figure 1.

The update of the memory cell state represented as "Cell State" is core to LSTM. The representation of this is shown in in Figure 1. To change this State, two key elements are important: the information that is no longer needed (so that the cell can neglect this information) and new information required for the cell state. The "Forget Gate Layer" makes the decision about the retention of the input. It is a sigmoid activation function given by:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2}$$

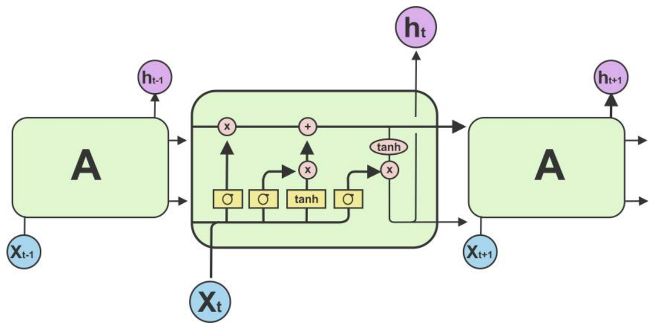


Figure 1: Long Short Term Memory (LSTM) Netwotk flow. (*Source*: 20)

The sigmoid activation function spans between '0' and '1' . The forget layer component makes the choice of inputs that are irrelevant to be retained.

Equation 3 depicts the process.

$$f_t = \sigma(W_{xf}.x_t + W_{hf}.h_{t-1} + b_f) \tag{3}$$

where,

f_t : forget gate's output,

W_{xf} : weight of the connection between the external input x_t and the forget gate,

W_{hf} : weight of the connection between the previous hidden state and the forget gate, and

b_f : bias of the forget gate.

Both the "Input Gate Layer" and a tanh layer (Equation 5) determine the relevant information to the cell state. The two components are combined by a multiplicative gate (denoted by $\bigotimes$).

$$i_t = \sigma(W_{xi}.x_t + W_{hi}.h_{t-1} + b_i) \tag{4}$$

where,

W_{xi} : weight between the external input and the input gate,

W_{hi} : weight between the previous hidden state and the input gate, and

b_i : bias of the unit.

$$v_t = \tanh(W_{xv}.x_t + W_{hv}.h_{t-1} + b_v) \tag{5}$$

where,

tanh is the tangent hyperbolic function defined as:

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{6}$$

Given the information from the previous two steps, the *cell state* C_t is then updated with the information (as shown in Equation 7).

$$C_t = f_t * C_{t-1} + i_t * v_t \tag{7}$$

where.

C_{t-1} is the previous cell state.

The last stage is to generate the cell output. The "Output Gate Layer" is another sigmoid activation function that picks cell inputs that are to be brought forward (Equation 8).

$$o_t = \sigma(W_{x0}.x_t + W_{h0}.h_{t-1} + b_0) \tag{8}$$

where,

W_{x0} : strength of the weight between the external input x_t and the output gate,

W_{h0} : weight of connection between the previously hidden state and the output gate, and

b_0 : is the bias of this unit.

The cell state (h_t) is then passed through a tanh activation filter to push the values between the range -1 and +1. The cell output is finally calculated (Equation 9).

$$h_t = o_t * \tanh(C_t) \tag{9}$$

3. Methodology

The main feature in an OCR post-processing stage is the correction of the text extracted from the image dataset. The goal of correcting this text results is to produce a more error-free digital equivalent of printed materials. The proposed model built on the existing model by introducing some changes. These consist of the adaptation of the character set to handle letters of the *Yorùbá* alphabet and the introduction of a language model for automatic correction of detected OCR error from the recognition output. The proposed model as shown in Figure 2 includes a language model subsystem for correction of outputs from the OCR system.

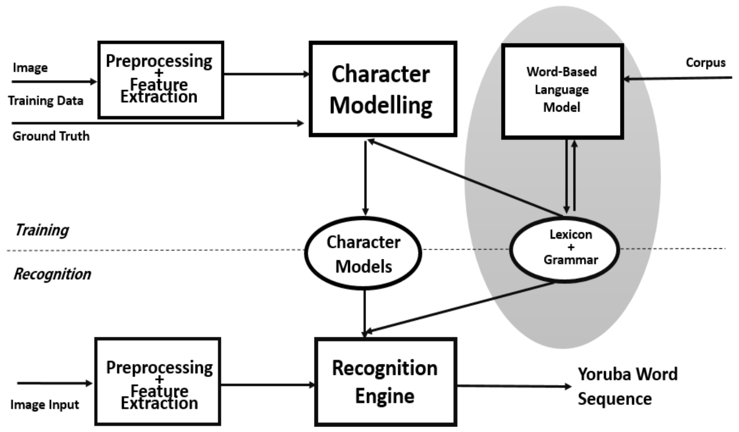


Figure 2: The proposed conceptual model for OCR of *Yorùbá* printed text images.

3.1. Corpus Analytics

We used the *Yorùbá* Bíbélì Mímọ́ which is free and readily available on the Internet to generate the printed text image dataset. The corpus has a word count of 233,845 and 7,534 sentences. The minimum sentence has a word count of 1, while the maximum has a word count of 1481. Table 1 shows the descriptive statistics which summarize the central tendency, dispersion, and shape of the corpus distribution. We decided not to use very long or short sentences because the training did not converge over many iterations. Shorter sentences are less likely to include a tone-marks and dots-below (which are the focus of this work), and the text is more likely to be repetitive. Longer sentences, on the other hand, are more difficult to learn due to their length, increase in the training time, and the cost of computation. The corpus used in generating the image dataset was limited to sentences of minimum length 20 and maximum length 35. At the character level, there are 41 unique characters aside special characters with the total number at 761,336. The diacritics (tone-mark and dot-below characters) is 41.1% of the corpus characters. Table 1 - Table 3 shows these statistics.

Table 1: Statistical Distribution of the *Yorùbá* Corpus at Sentence Level

Descriptive Statistics	Count
Count	7534.000000
Mean	142.626493
Standard Deviation	98.693964
Minimum	1.000000
25%	77.000000
50%	120.000000
75%	183.000000
Maximum	77.000000

Table 2: Vowel Characters Distribution within Corpus

Vowel Group	Count
(à, á)	59235
(è, é)	46451
(ì, í)	82604
(ò, ó)	52674
(ù, ú)	30018
(ṣ,ẹ,ị,ọ)	42291
(a,e,i,o,u)	90410

Table 3: **Consonant Characters Distribution within Corpus**

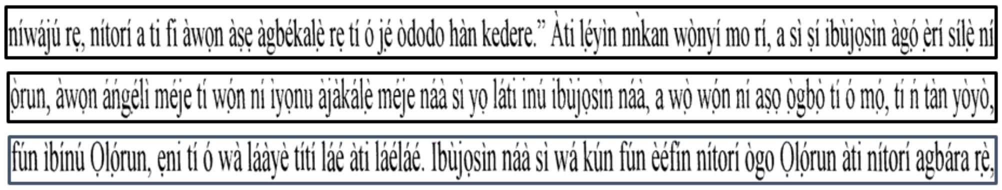
Vowel Group	Count
(k,l,r,w)	100466
(b,d,h)	41049
(t,f)	40780
(s)	19150
(g, j, p, y)	57197
(m, n)	98978
(c, q, x, z)	33

3.2. Data Collection: Yorùbá Image Dataset

There are existing works on *Yorùbá* OCR but there is no publicly available repository of *Yorùbá* text image dataset. The image was acquired across three different font styles. These font styles are Times New Roman, Ariel, and DejaVuSans. In developing the *Yorùbá* OCR sequence model, a total of 4,800 *Yorùbá* Machine-Printed line images were created. Overall, we used a random subset of 4,000 images in the training set, 400 for validation set, and 400 in the test set. The data is publicly available on this https://github.com/oniolalekan/yorubaOCR_Dataset Github repository. Table 2 shows the statistics of the dataset variables. The steps for creating of the line images is described in section 3.3.1. Each of the images contained a text-line with an average of 25 *Yorùbá* words. Times New Roman, Ariel, and DejaVuSans font styles were used in the training, testing, and validation of the system respectively. With each in multiple text sizes of 12, 18, 36, 48, 60 and 72. The quality of the image was at 300 dpi (dots per inches). Figure 3 shows a sample of the image dataset that was used.

Table 4: Description of the Dataset Variables

Input Variables	Quantity
Font Styles	3
Training Dataset	2000
Validation Dataset	2000


níwájú rẹ, nítorí a ti fi àwọn àṣẹ àgbékalẹ̀ rẹ tí ó jẹ́ òdodo hàn kedere.” Àti lẹ́yìn nǹkan wọ̀nyí mo rí, a sì ṣí ìbùjọsìn àgọ́ ẹ̀rí sílẹ̀ ní

ọ̀run, àwọn áńgẹ́lì méje tí wọ́n ní ìyọnu àjàkálẹ̀ méje náà sì yọ láti inú ìbùjọsìn náà, a wọ̀ wọ́n ní aṣọ ọ̀gbọ̀ tí ó mọ́, tí ń tàn yòyò,

fún ìbínú Ọlọ́run, ẹni tí ó wà láàyè títí láé àti láéláé. Ibùjọsìn náà sì wá kún fún èéfín nítorí ògo Ọlọ́run àti nítorí agbára rẹ̀,

Figure 3: Sample of the image dataset. The font type of the sample dataset is Ariel while the font size is 14 .

3.3. Steps of generating the Yorùbá image dataset

Three-step Process:

i We created a multi-line file (a text file containing 20 lines of *Yorùbá* words where each line is intended to produce an image file) of Yorùbá corpus from the larger dataset. Each file contains 20 rows of such text-lines.

ii We manually identified and copied the font style of the intended image.

iii We then wrote a Python script to generate 180 line images of varying sizes (12, 18, 36, 48, 60, 72) with their corresponding ground truth (the expected text in the *Yorùbá* image dataset).

The flowchart of the iterations is shown in Figure 4. At the end of an iteration where an image is generated, an update to next text-line is performed to generate the next image.

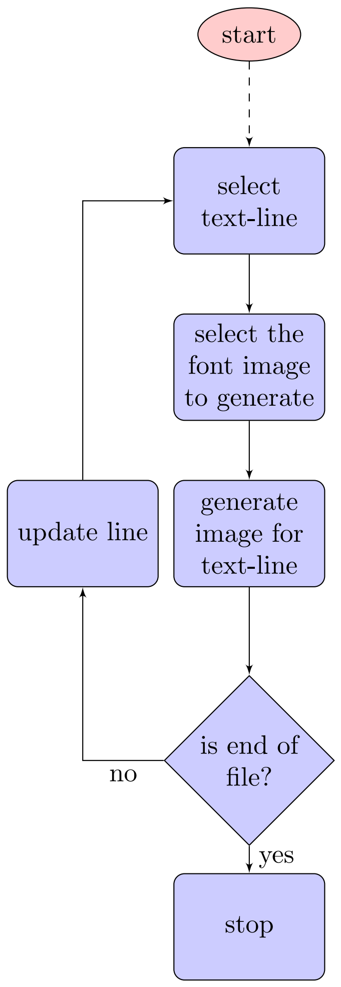

start
select text-line
select the font image to generate
update line
generate image for text-line
is end of file?
no
yes
stop

Figure 4: The flowchart of generating the *Yorùbá* image dataset

3.4. Preprocessing stages

This section discusses the steps involved in the preprocessing stages of the *Yorùbá* OCR system developed. The performance of these steps is key to the performance of the entire system.

Binarization is the first stage in the *Yorùbá* OCR methodology. The aim of this stage is to convert the input black and white or color document into a binary representation. We applied Adaptive thresholding as against using Gaussian adaptive because of its performance in situations where it is needed to correct the varying lighting conditions in different areas. The method is used in calculating the magnitude that must be exceeded for a small region of the image thereby resulting in different values for different regions of the same image.

Skew estimation was applied to attempt to horizontally restore any distorted *Yorùbá* image dataset. This is done by using Ocropy Python Toolkit to experiment with different angles. The algorithm produces an excellent result because at the point the image is properly aligned, there is a tremendous imbalance between rows with text and the blanks in between them. When the image is rotated, the divergence is blended.

Page segmentation: Page segmentation helps indicate the zones of interest to be recognized on the scanned page. The RAST (Recognition by Adaptive Subdivision of Transformation Space) algorithm was adopted because it is fast and robust compared to the other algorithms.

3.5. The Yorùbá OCR LSTM Architecture

A single LSTM network (also known as Vanilla LSTM) depicted in section 2.1 is adopted for the work. The columns of *Yorùbá* image data pixels are the inputs to the network. In a sequential manner from left to right, these columns are input into the network. Each possible letters are then scored, with the highest matching the letter. As the columns say letter $\grave{A}$ are fed into the network, then it is expected that there will be a spike from the $\grave{A}$ output.

Figure 5 shows the system architecture of the LSTM-based recognizer that was used for the Optical Character Recognition of *Yorùbá* printed text images.

Fully Connected (FC) layers connect every neuron in one layer to every neuron in the succeeding layer. The Rectified Linear Units (ReLUs) - which builds non-linearity in the network - when combined with FC, extracted the important features from the image dataset. The 1D-LSTM-based OCR system which is adapted for the *Yorùbá* OCR model uses a small number of tunable parameters. One important parameter is the number of LSTM cells in the hidden layer(s) and the number of hidden layers. In this work, only one hidden layer with 200 LSTM memory cells in each of right-to-left and left-to-right layers (corresponding to bidirectional mode). Other parameters are learning rate (10^{-4}) and the momentum (set to 0.9). The **softmax function** normalizes the outputs of each unit to be between 0 and 1, just like a sigmoid function. It also used to divide each output such that the total sum of the outputs is equal to 1. The output of the softmax function is equivalent to a categorical probability distribution, which tells the probability that any of the *Yorùbá* character set are true.

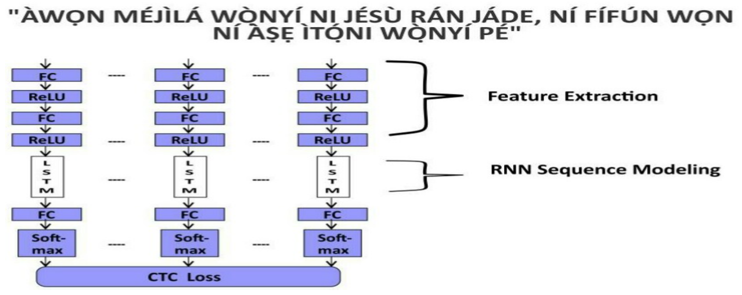

"ÀWỌN MÉJÌLÁ WỌ̀NYÍ NI JÉSÙ RÁN JÁDE, NÍ FÍFÚN WỌN NÍ ÀṢẸ ÌTỌ́NI WỌ̀NYÍ PÉ"
FC
ReLU
FC
ReLU
LSTM
FC
Soft-max
CTC Loss
Feature Extraction
RNN Sequence Modeling

Figure 5: The network architecture of the *Yorùbá* sequence model

The network was trained for 23,000 iterations and the intermediate models were saved after every 1,000 iterations. This creates a total of 23 models. The training of the model is done by learning from its mistakes. It transcribes the text in a line, then adjusts the weights in the Neural Net to compensate for the errors. Then it does this again for the next line, and the next, and so on. When it gets to the last line of labeled data, it starts over again. As it loops through the training data over and over again, the model gets better and better. The training errors for these iterations are computed, and then the network with the least error was selected as the best network, which is then used to OCR the *Yorùbá* test image dataset.

3.6. Unigram language modelling

In an effort to find and improve misspelled *Yorùbá* words in the model developed for the OCR, a Unigram error correction methodology, also known as dictionary-based error correction was conceived. Using this approach, a lexicon or a lookup dictionary from a *Yorùbá* Bible corpus was used to spell-check OCR recognized words and correct the words that are misspelled.

This method employed the use of a spell-checker to corrects single-error misspellings which are the predominant source of erroneous recognition in the *Yorùbá* OCR recognition system. This system rates the candidates for correction according to their correctness probability and then picks the most likely between the original *Yorùbá* word and the highest-rated candidate.

Given a word w, and the task of finding the correction c, the method "correction(w)" tries to choose the most likely spelling correction for w out of all possible candidate corrections. The approach uses probabilities to rank the possible candidates that maximize the probability that c is the intended correction. The candidate with the highest combined probability is chosen using the arguments of the maximal function.

3.7. Implementation Environment

The model was trained with the open-source OCRopus framework, a free document analysis and optical character recognition library [21]. It transcribes the image in a line, then adjusts the weights in the Neural Net to compensate for the errors. Then it does this again for the next line, and the next, and so on. When it gets to the last line of labeled data, it starts over again. As it loops through the training data over and over again, the model gets better and better.

3.8. Performance Metrics

The *Yorùbá* OCR accuracy is measured as "Character Error Rate (CER) (%)", using the *Levenshtein Edit Distance* algorithm. This algorithm is also commonly known as the Edit Distance. It is the ratio between *insertion*, *deletion* and *substitution* errors and the total number of characters. The formula is as shown in Equation 10. Algorithm 3.1 shows the algorithm used in evaluating the system.

$$\text{Character Error Rate (CER)} = \frac{Insertion + Deletion + Substitution}{\text{Total Character}} \times 100 \tag{10}$$

Algorithm 3.1: The Algorithm to calculate the Character Error Rate

Algorithm 1 Calculation of Character Error Rate (CER) with Levenshtein Edit Distance

```
1: function EDIT DISTANCE(Reference r, Hypothesis h)
2:     $int[|r|+1][|h|+1]D$                                  ▷ Initialisation
3:     for ($i = 0; i \leq |r|; i++$) do
4:         for ($j = 0; j \leq |h|; j++$) do
5:             if i==0 then
6:                 $D[0][j]j$
7:             else if j==0 then
8:                 $D[i][0]i$
9:             end if
10:        end for
11:    end for
12:    for ($i = 1; i \leq |r|; i++$) do                    ▷ Calculation
13:        for ($j = 1; j \leq |h|; j++$) do
14:            if r[i-1] == h[j-1] then
15:                D[i][j] ←D[i-1][j-1]
16:            else
17:                sub ←D[i-1][j-1]+1
18:                ins ←D[i][j-1]+1
19:                del ←D[i-1][j]+1
20:                D[i][j] ←min(sub,ins,del)
21:            end if
22:        end for
23:    end for
24:    return $D[|r|][|h|]$
25: end function
```

The algorithm that is used in computing the Character Error Rate is based on dynamic programming, a technique used in breaking down a complex problem into sub-problem to avoid computing multiple time the same sub-problem. It basically involves using past knowledge to make solving a future problem easier. The main challenge in this approach is whether the cheapest way to convert one string into another involves a final add, remove, or change. Hence, the evaluation of all the three possibilities and the return of the minimum distance from among the three.

In each case, the algorithm recursively computes the distance (number of adds, removes, and changes) required to "set up" a final add, a final remove, or a final change. Then, add one to the "add distance" and the "remove distance" to account for the final add or remove. For the "change distance", one is added only if the final characters in the strings are different (if not, no final change is required).

3.9. Training the model

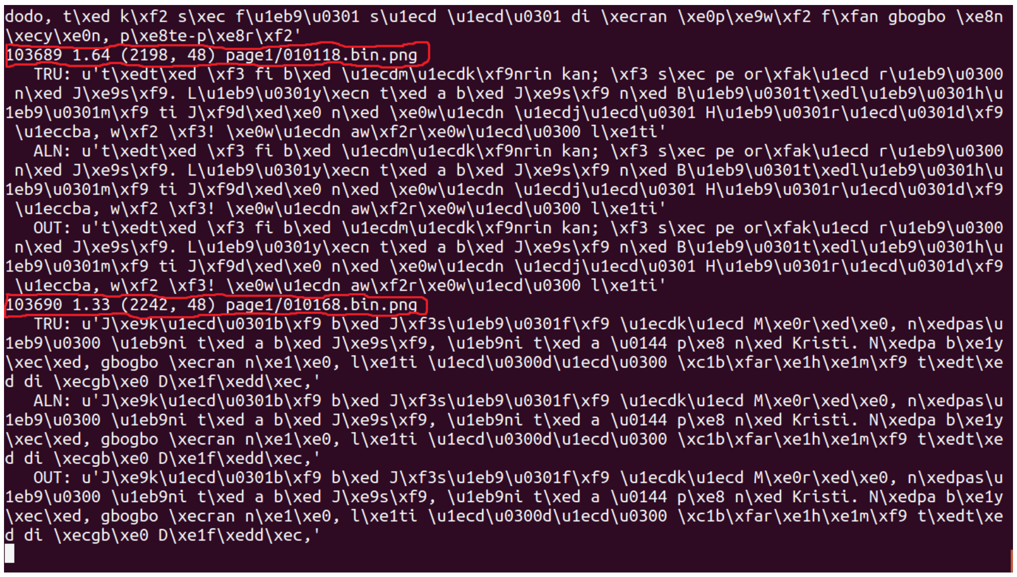
The LSTM networks model developed for optical character recognition of *Yorùbá* printed images make use of supervised learning where the set of sample pairs containing input *Yorùbá* text-line images and output ground truth transcription of the image are shown to the network. Through the backpropagation algorithm, the resulting output predictions are then compared with the ground truth. The mean square error is calculated over the entire dataset and the intermediate weights are adjusted using learning rate of 10^{-5} until the error decayed sufficiently. At 313 and 314 number of iterations, the number of characters in error through the network stands at 226 and 221 respectively as shown in Figure 6.

The training of the model is achieved by learning from its past mistakes. It extracts the text in the image, then re-adjusts the weights in the LSTM network to nullify for the errors. Then the iteration continues for the next line, and the next, and so on until it gets to the last line of labeled data and then starts again. As it iterates through the training data, the network gets better. The model shows the character error rate of 1.64% and 1.33% at $103,689$ and $103,690$ number of iterations respectively which is an improvement from the earlier reported iterations.

4. Results

4.1. Convergence error rate as training progresses

2000 labeled images were used as training data and the other 2000 was held out for test and validation dataset. The models were saved at every 1000th loop and this helps in evaluating the performance of the model as it learns. Figure



```
dodo, t\xed k\xf2 s\xec f\u1eb9\u0301 s\u1ecd \u1ecd\u0301 di \xecran \xe0p\xe9w\xf2 f\xfan gbogbo \xe8n
\xecy\xe0n, p\xe8te-p\xe8r\xf2'
103689 1.64 (2198, 48) page1/010118.bin.png
   TRU: u't\xedt\xed \xf3 fi b\xed \u1ecdm\u1ecdk\xf9nrin kan; \xf3 s\xec pe or\xfak\u1ecd r\u1eb9\u0300
 n\xed J\xe9s\xf9. L\u1eb9\u0301y\xecn t\xed a b\xed J\xe9s\xf9 n\xed B\u1eb9\u0301t\xedl\u1eb9\u0301h\u
1eb9\u0301m\xf9 ti J\xf9d\xed\xe0 n\xed \xe0w\u1ecdn \u1ecdj\u1ecd\u0301 H\u1eb9\u0301r\u1ecd\u0301d\xf9
 \u1eccba, w\xf2 \xf3! \xe0w\u1ecdn aw\xf2r\xe0w\u1ecd\u0300 l\xe1ti'
   ALN: u't\xedt\xed \xf3 fi b\xed \u1ecdm\u1ecdk\xf9nrin kan; \xf3 s\xec pe or\xfak\u1ecd r\u1eb9\u0300
 n\xed J\xe9s\xf9. L\u1eb9\u0301y\xecn t\xed a b\xed J\xe9s\xf9 n\xed B\u1eb9\u0301t\xedl\u1eb9\u0301h\u
1eb9\u0301m\xf9 ti J\xf9d\xed\xe0 n\xed \xe0w\u1ecdn \u1ecdj\u1ecd\u0301 H\u1eb9\u0301r\u1ecd\u0301d\xf9
 \u1eccba, w\xf2 \xf3! \xe0w\u1ecdn aw\xf2r\xe0w\u1ecd\u0300 l\xe1ti'
   OUT: u't\xedt\xed \xf3 fi b\xed \u1ecdm\u1ecdk\xf9nrin kan; \xf3 s\xec pe or\xfak\u1ecd r\u1eb9\u0300
 n\xed J\xe9s\xf9. L\u1eb9\u0301y\xecn t\xed a b\xed J\xe9s\xf9 n\xed B\u1eb9\u0301t\xedl\u1eb9\u0301h\u
1eb9\u0301m\xf9 ti J\xf9d\xed\xe0 n\xed \xe0w\u1ecdn \u1ecdj\u1ecd\u0301 H\u1eb9\u0301r\u1ecd\u0301d\xf9
 \u1eccba, w\xf2 \xf3! \xe0w\u1ecdn aw\xf2r\xe0w\u1ecd\u0300 l\xe1ti'
103690 1.33 (2242, 48) page1/010168.bin.png
   TRU: u'J\xe9k\u1ecd\u0301b\xf9 b\xed J\xf3s\u1eb9\u0301f\xf9 \u1ecdk\u1ecd M\xe0r\xed\xe0, n\xedpas\u
1eb9\u0300 \u1eb9ni t\xed a b\xed J\xe9s\xf9, \u1eb9ni t\xed a \u0144 p\xe8 n\xed Kristi. N\xedpa b\xe1y
\xec\xed, gbogbo \xecran n\xe1\xe0, l\xe1ti \u1ecd\u0300d\u1ecd\u0300 \xc1b\xfar\xe1h\xe1m\xf9 t\xedt\xe
d di \xecgb\xe0 D\xe1f\xedd\xec,'
   ALN: u'J\xe9k\u1ecd\u0301b\xf9 b\xed J\xf3s\u1eb9\u0301f\xf9 \u1ecdk\u1ecd M\xe0r\xed\xe0, n\xedpas\u
1eb9\u0300 \u1eb9ni t\xed a b\xed J\xe9s\xf9, \u1eb9ni t\xed a \u0144 p\xe8 n\xed Kristi. N\xedpa b\xe1y
\xec\xed, gbogbo \xecran n\xe1\xe0, l\xe1ti \u1ecd\u0300d\u1ecd\u0300 \xc1b\xfar\xe1h\xe1m\xf9 t\xedt\xe
d di \xecgb\xe0 D\xe1f\xedd\xec,'
   OUT: u'J\xe9k\u1ecd\u0301b\xf9 b\xed J\xf3s\u1eb9\u0301f\xf9 \u1ecdk\u1ecd M\xe0r\xed\xe0, n\xedpas\u
1eb9\u0300 \u1eb9ni t\xed a b\xed J\xe9s\xf9, \u1eb9ni t\xed a \u0144 p\xe8 n\xed Kristi. N\xedpa b\xe1y
\xec\xed, gbogbo \xecran n\xe1\xe0, l\xe1ti \u1ecd\u0300d\u1ecd\u0300 \xc1b\xfar\xe1h\xe1m\xf9 t\xedt\xe
d di \xecgb\xe0 D\xe1f\xedd\xec,'
```

Figure 6: A screen-shot of the model training.

7 shows the plot of the number of iteration to the character error rate. The convergence error rate starts at a high value (above 20%) but quickly reduces to about 7.1% after 6,700 iterations, then spikes to over 3% at 7,100. It eventually converges to a minimum of 1.33% at 16,000 iterations.

The network first learns to recognize spaces, followed by Latin characters, then special characters and

nally *Yorùbá* letters with diacritics. Although it learns the spaces and Latin characters contained in the image quickly, it continues to make mistakes again later during training. This happens as it starts picking up special characters, and *Yorùbá* letters with diacritics, some of which are confusable with the lower case Latin letters. The transition from learning spaces and Latin letters to learning *Yorùbá* letters explains the spike from 2.5% at 6,700 iterations to 3% at 7,100 iterations.

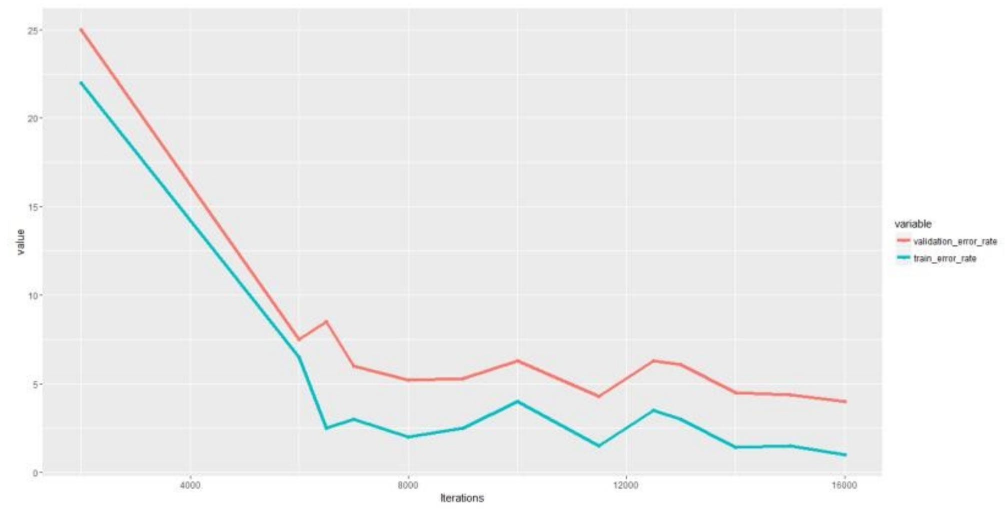


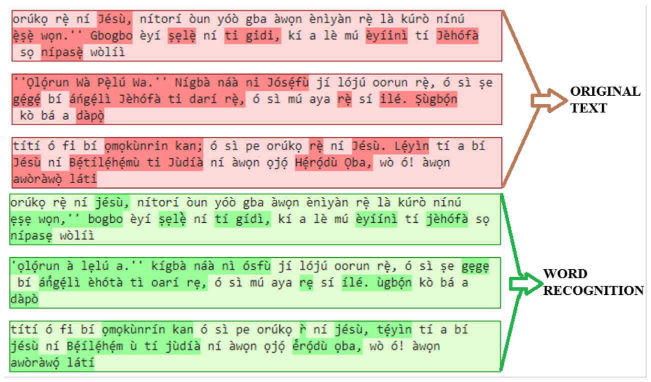
Figure 7: Character Error Rate for Training Dataset and Validation Dataset vs number of iterations during training.

4.2. Recognition output

The recognition output of the *Yorùbá* OCR system without the spell-check correction engine is shown in Figure 8. The upper section designated as "original text" shows the ground-truth of the image file sent to the *Yorùbá* OCR system while the lower section designated as "word recognition" shows the corresponding output from the *Yorùbá* OCR system. The grayed out part of both section indicates words wherein some or all characters where altered. In the same vein, Figure 9 shows the recognition output of the *Yorùbá* OCR system with the spell-checking correction engine. The upper section designated as "original text" shows the ground-truth of the image file sent to the *Yorùbá* OCR system while the lower section designated as "language modelling" shows the corresponding output from the *Yorùbá* OCR system. The grayed out part of both section indicates words wherein some or all characters where altered.

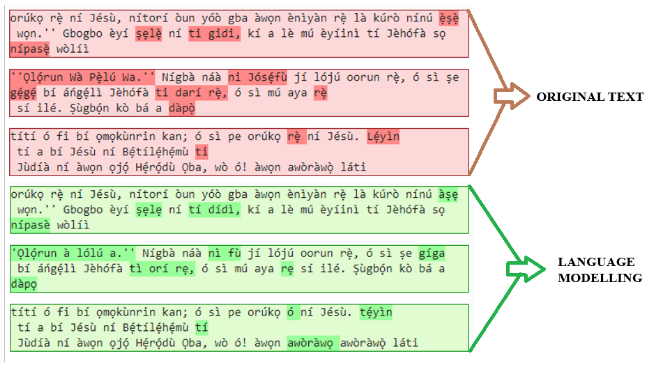
4.3. Performance evaluation results

There are two test datasets used in the evaluation process. The first dataset consists of 30% of images generated from the Times New Roman font of the



```
orúkọ rẹ̀ ní Jésù, nítorí òun yóò gba àwọn ènìyàn rẹ̀ là kúrò nínú
ẹ̀ṣẹ̀ wọn.'' Gbogbo èyí ṣẹlẹ̀ ní ti gidi, kí a lè mú èyíinì tí Jèhófà
 sọ nípasẹ̀ wòlíì

''Ọlọ́run Wà Pẹ̀lú Wa.'' Nígbà náà ni Jóṣẹ́fù jí lójú oorun rẹ̀, ó sì ṣe
gẹ́gẹ́ bí áńgẹ́lì Jèhófà ti darí rẹ̀, ó sì mú aya rẹ̀ sí ilé. Ṣùgbọ́n
 kò bá a dàpọ̀

títí ó fi bí ọmọkùnrin kan; ó sì pe orúkọ rẹ̀ ní Jésù. Lẹ́yìn tí a bí
Jésù ní Bẹ́tílẹ́hẹ́mù ti Jùdíà ní àwọn ọjọ́ Hẹ́rọ́dù Ọba, wò ó! àwọn
awòràwọ̀ láti

orúkọ rẹ̀ ní jésù, nítorí òun yóò gba àwọn ènìyàn rẹ̀ là kúrò nínú
ẹṣẹ wọn,'' bogbo èyí ṣẹlẹ̀ ní tí gídì, kí a lè mú èyíínì tí jèhófà sọ
nípasẹ wòlíì

'ọlọ́run à lẹlú a.'' kígbà náà nì ósfù jí lójú oorun rẹ̀, ó sì ṣe gẹgẹ
 bí áńgẹ́lì èhótà tì oarí rẹ, ó sì mú aya rẹ sí ílé. ùgbọ́n kò bá a
dàpò

títí ó fi bí ọmọkùnrín kan ó sì pe orúkọ r̀ ní jésù, tẹ́yìn tí a bí
jésù ní Bẹ́ílẹ́hẹ́m ù tí jùdíà ní àwọn ọjọ́ ẹ́rọ́dù ọba, wò ó! àwọn
awòràwọ́ láti
```

Figure 8: The recognition output of the Yorùbá OCR system without the spell-check model.



```
orúkọ rẹ̀ ní Jésù, nítorí òun yóò gba àwọn ènìyàn rẹ̀ là kúrò nínú ẹ̀ṣẹ̀
 wọn.'' Gbogbo èyí ṣẹlẹ̀ ní ti gidi, kí a lè mú èyíinì tí Jèhófà sọ
nípasẹ̀ wòlíì

''Ọlọ́run Wà Pẹ̀lú Wa.'' Nígbà náà ni Jóṣẹ́fù jí lójú oorun rẹ̀, ó sì ṣe
gẹ́gẹ́ bí áńgẹ́lì Jèhófà ti darí rẹ̀, ó sì mú aya rẹ̀
 sí ilé. Ṣùgbọ́n kò bá a dàpọ̀

títí ó fi bí ọmọkùnrin kan; ó sì pe orúkọ rẹ̀ ní Jésù. Lẹ́yìn
 tí a bí Jésù ní Bẹ́tílẹ́hẹ́mù ti
 Jùdíà ní àwọn ọjọ́ Hẹ́rọ́dù Ọba, wò ó! àwọn awòràwọ̀ láti

orúkọ rẹ̀ ní Jésù, nítorí òun yóò gba àwọn ènìyàn rẹ̀ là kúrò nínú àṣẹ
 wọn.'' Gbogbo èyí ṣẹlẹ ní tí dídì, kí a lè mú èyíinì tí Jèhófà sọ
nípasè wòlíì

'Ọlọ́run à lólú a.'' Nígbà náà nì fù jí lójú oorun rẹ̀, ó sì ṣe gíga
 bí áńgẹ́lì Jèhófà tì orí rẹ, ó sì mú aya rẹ sí ilé. Ṣùgbọ́n kò bá a
dàpọ

títí ó fi bí ọmọkùnrin kan; ó sì pe orúkọ ó ní Jésù. tẹ́yìn
 tí a bí Jésù ní Bẹ́tílẹ́hẹ́mù tí
 Jùdíà ní àwọn ọjọ́ Hẹ́rọ́dù Ọba, wò ó! àwọn awòràwọ awòràwọ̀ láti
```

Figure 9: The recognition output of the Yorùbá OCR system with the spelling checker model

Table 5: The Performance of the Model on Different Fonts

Font	Errors	Total	CER	Top Character
Times Roman	642	20462	3.138%	(38, 'i', 'ì')
DejaVuSans	1560	20982	7.435%	(262, 'í', 'i')
Ariel	3266	21570	15.141%	(161, 'í', 'i')

Yorùbá Bible that was not used for training the *Yorùbá* OCR systems. The total number of text-line images are 740 and the total number of characters is 20462. The second dataset consists 758 text-line images from DejaVuSans font style and 779 text-line images of Ariel font style both generated from the *Yorùbá* Bible. The results from the model as shown in Table 5, show character error rate of 3.138% for Times New Roman font, 7.435% for DejaVuSans font and 15.141% for Ariel font. The top confusions in applying each of the three font styles are also shown in Table 5. Although modern OCR systems claim above 97 % character accuracy for popular scripts like Arabic, for example. However, for the same data with low-resolution images or uncommon character set, this claim can drop to lower than 70 %. It is therefore difficult to directly use the results generated from OCR system for other tasks (such as indexing for textual search and machine translation) without expensive editing. This Standard *Yorùbá* OCR model is also not an exception.

The evaluation of the system after the Spell-checking corrector has been introduced shows a considerable drop in the character error rates. The results as shown in Table 6 shows a character error rate of 3.138% for Times New Roman font, 7.435% for DejaVuSans font and 15.141% for Ariel font. The word "Ṣùgbọ́n" was recognised as "ùgbọ́n" by the *Yorùbá* OCR model but was corrected with the introduction of the corrector. Another example is the word "gẹ́gẹ́" which was transcribed as "gẹgẹ" but corrected by the spell-checker.

When compared with the work of [22] which use similar architecture - bidirectional LSTM networks - to the problem of machine-printed Latin and Fraktur

Table 6: The Performance of the Model plus Introduction of Spelling checker on Different Fonts

Font	Errors	Total	CER
Times Roman	242	20462	1.182%
DejaVuSans	860	20982	4.098%
Ariel	1266	21570	5.87%

recogntion, we observed that this model compete favorably. Though their result achieved a better recognition error rate of 0.6% on English test-set, the performance will be grossly inadequate if we are to use it for *Yorùbá* machine-printed images which is a tone language.

5. Conclusion and Future Scope of Work

The performance of the model shows that the farther away an image text font is from the font(s) used in training the network, the higher the character error rate of the recognition.

There are numerous ways in which the work reported can be further extended. The performance of the *Yorùbá* OCR system can be improved by developing a more robust language model which can serve as an error correction subsystem towards a drive for a better output. Furthermore, the LSTM-based OCR reported for *Yorùbá* can be extended further to documents with multiple font styles and sizes and to other languages with similar characterization.

References

References

[1] F. A. Fabunmi, A. S. Salawu, Is yorùbá an endangered language?, Nordic Journal of African Studies 14 (3) (2005) 391–408.

[2] N. A. M. Isheawy, H. Hasan, Optical character recognition (ocr) system, IOSR Journal of Computer Engineering 17 (2) (2015) 22–26.

[3] Ø. D. Trier, A. K. Jain, T. Taxt, Feature extraction methods for character recognition-a survey, Pattern recognition 29 (4) (1996) 641–662.

[4] E. Kavallieratou, N. Fakotakis, G. Kokkinakis, Skew angle estimation for printed and handwritten documents using the wigner–ville distribution, Image and Vision Computing 20 (11) (2002) 813–824.

[5] P. Shah, S. Karamchandani, T. Nadkar, N. Gulechha, K. Koli, K. Lad, Ocr-based chassis-number recognition using artificial neural networks, in: Vehicular Electronics and Safety (ICVES), 2009 IEEE International Conference on, IEEE, 2009, pp. 31–34.

[6] N. Mani, B. Srinivasan, Application of artificial neural network model for optical character recognition, in: Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on, Vol. 3, IEEE, 1997, pp. 2517–2520.

[7] F. O. Asahiah, O. A. Ọdẹ́jọbí, E. R. Adagunodo, Restoring tone-marks in standard yorùbá electronic text: Improved model, Computer Science 18 (3) (2017) 301–315.

[8] A. Bamgbose, A Grammar of Yorùbá, West African language monograph series, Cambridge University Press, 2000.
URL https://books.google.com.ng/books?id=q20vW6-CmHAC

[9] P. J. Denning, Closing statement: What have we said about computation?, The Computer Journal 55 (7) (2012) 863–865.

[10] O. D. Ninan, O. A. Odéjobí, Theoretical Issues in the Computational Modelling of Yorùbá Narratives, in: M. A. Finlayson, B. Fisseni, B. Löwe, J. C. Meister (Eds.), 2013 Workshop on Computational Models of Narrative, Vol. 32 of OpenAccess Series in Informatics (OASIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2013, pp. 153–157. doi:10.4230/OASIcs.CMN.2013.153.
URL http://drops.dagstuhl.de/opus/volltexte/2013/4142

[11] G. Tauschek, Reading machine, uS Patent 2,115,563. Accessed on January 28, 2017 from https://patents.google.com/patent/US2115563A/en (Apr. 26 1938).

[12] O. O. Oladayo, Yoruba language and numerals' offline interpreter using morphological and template matching, Indonesian Journal of Electrical Engineering and Computer Science 13 (1) (2015) 166–173.

[13] A. O. Ibraheem, O. A. Ọdẹ́jọbí, Towards reliable handwritten character recognition amid diacritic chaos., accessed on June 10, 2017 from https://pdfs.semanticscholar.org/238c/2f223f8ee03a1c049aa03887147c6b2d44c9.pdf (2016).

[14] A. Choudhary, R. Rishi, S. Ahlawat, A new character segmentation approach for off-line cursive handwritten words, Procedia Computer Science 17 (2013) 88–95.

[15] A. Ul-Hasan, Generic Text Recognition using Long Short-Term Memory Networks, *Thesis, Technical University of Kaiserslautern, Germany* (2016).

[16] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.

[17] F. A. Gers, N. N. Schraudolph, J. Schmidhuber, Learning precise timing with lstm recurrent networks, Journal of machine learning research 3 (Aug) (2002) 115–143.

[18] S. Bai, J. Z. Kolter, V. Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, arXiv preprint arXiv:1803.01271.

[19] A. W. Senior, Off-line cursive handwriting recognition using recurrent neural networks.

[20] C. Olah, Undestanding lstm networks, accessed on January 08, 2018 from http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (2015).

[21] T. M. Breuel, The ocropus open source ocr system, in: Document Recognition and Retrieval XV, Vol. 6815, International Society for Optics and Photonics, 2008, p. 68150F.

[22] T. M. Breuel, A. Ul-Hasan, M. A. Al-Azawi, F. Shafait, High-performance ocr for printed english and fraktur using lstm networks, in: Document Analysis and Recognition (ICDAR), 2013 12th International Conference on, IEEE, 2013, pp. 683–687.