JIMMA UNIVERSITY

JIMMA INSTITUTE OF TECHNOLOGY

SCHOOL OF GRADUATE STUDIES

Offline Handwritten Amharic Word Recognition using Deep Learning

By: EYOB SISAY

Advisor: DR. GETACHEW ALEMU

Co-Advisor: MR. FETULHAK ABDURAHMAN

A thesis submitted to School of Graduate Studies, Jimma University in fulfillment of the requirements for the degree of Masters of Science

in the field of

Computer Engineering

June 18, 2021

Jimma, Ethiopia

JIMMA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

FACULTY OF ELECTRICAL AND COMPUTER ENGINEERING

MASTERS THESIS ON

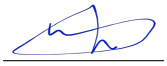
Offline Handwritten Amharic Word Recognition using Deep Learning

APPROVED BY THE BOARD OF EXAMINERS

Chairperson:	*Signed:*	*Date:*
MR. FETULHAK ABDURAHMAN	_____	__ / __ / ____
Internal Examiner:	*Signed:*	*Date:*
	_____	__ / __ / ____
External Examiner:	*Signed:*	*Date:*
	_____	__ / __ / ____

Declaration

I, EYOB SISAY, declare that this thesis titled, *" Offline Handwritten Amharic Word Recognition using Deep Learning "* and the work presented herein are my own, except where explicitly stated otherwise in the text, and with the guidance of my advisor. I confirm that the work has not previously been submitted for a degree or any other qualification at this or any other University or institution, this has been clearly stated. Moreover, all sources of materials that used in the thesis are acknowledged.

Student Name:	*Signed:*	*Date:*
EYOB SISAY SEYFU		__ / __ / ____

As research Adviser, I hereby certify that I have read and evaluated this thesis paper prepared under my guidance, by EYOB SISAY SEYFU entitled *" Offline Handwritten Amharic Word Recognition using Deep Learning "* and recommend and would be accepted as a fulfilling requirement for the Degree Masters of Science in the field of Computer Engineering.

Main Advisor:	*Signed:*	*Date:*
DR. GETACHEW ALEMU	_____	__ / __ / ____

Co-Advisor:	*Signed:*	*Date:*
MR. FETULHAK ABDURAHMAN	_____	__ / __ / ____

Abstract

Offline Handwritten Amharic Word Recognition using Deep Learning

The human brain and visual system works together when reading and writing texts typically for communication purpose. In today's digital technologies, handwriting recognition is involving artificial neural networks (ANNs) by replacing the machine learning algorithms. The use of ANNs for highly learned features from the data in the field of artificial intelligence (AI) brought deep learning systems. The data in this study is handwritten text, which is still a challenge in the research world. This is due to the fact that several language writing system use different alphabets and also handwriting calligraphy is personal. Our approaches for handwritten Amharic word recognition begin from preparing the dataset from scratch. To the authors knowledge, there was no any publicly available handwritten Amharic text dataset. We chose A* path-planning and scale space algorithms respectively for line and word level segmentation from the raw handwritten text image. We have collected 12,064 words and prepared 36,192 well augmented word-images for our dataset. Then in the main process, we have used CNN-RNN-CTC architecture. And in our model, Bayesian optimization algorithm used to set values for hyper-parameters such as input size, number of layers, filters size, etc. Thus, our best performing model takes $64 \times 256 \times 1$ word-image in 12 convolutional layers to extract 31×512 features, 2 stacked bi-directional GRUs for sequence modeling and CTC neural network output algorithm. We have finally got state-of-the-art results: 2.67% CER and 8.83% WER.

Keywords: CNN-RNN-CTC, Deep learning, HTR, handwritten Amharic words

Acknowledgements

I'm thankful to GOD in everything. To the Jimma University and specifically to Faculty of Electrical and Computer Engineering, which provided me with basic requirements of the research. For all the teachings. And, special thanks to my teachers and advisors, DR. GETACHEW A. and MR. FETULHAK A. without their help it couldn't have been possible.

Thanks to my aunt MRS. TEWABECH S. and my family, for having helped and supported in the different stages of my life, guiding me and believing in me at all times. I'm result of your efforts. To all my friends without you all it would have been harder and, above all, more boring.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	**A**rtificial **I**ntelligence
ANN	**A**rtificial **N**eural **N**etwork
AWR	**A**mharic **W**ord **R**ecognition
ASCII	**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
BERT	**B**idirectional **E**ncoder **R**epresentations from **T**ransformers
BGRU	**B**idirectional **G**ated **R**ecurrent **U**nit
BLSTM	**B**idirectional **L**ong **S**hort **T**erm **M**emory
BRNN	**B**idirectional **R**ecurrent **N**eural **N**etwork
CER	**C**haracter **E**rror **R**ate
CNN	**C**onvolutional **N**eural **N**etwork
CRF	**C**onditional **R**andom **F**ields
CTC	**C**onnectionist **T**emporal **C**lassification
DC	**D**istrict of **C**olumbia
ELMo	**E**mbeddings from **L**anguage **Mo**dels
GAN	**G**enerative **A**dversarial **N**etwork
G-CNN	**G**ated **C**onvolutional **N**eural **N**etwork
GPU	**G**raphical **P**rocessing **U**nit
GRU	**G**ated **R**ecurrent **U**nit
HAW-DB	**H**andwritten **A**mharic **W**ord **D**atabase
HMM	**H**idden **M**arkov's **M**odel
HTR	**H**andwritten **T**ext **R**ecognition
KNN	**K**-nearest **N**eighbors **N**etwork
LSTM	**L**ong **S**hort **T**erm **M**emory
MDLSTM	**M**ulti-**D**irectional **L**ong **S**hort **T**erm **M**emory
NLP/U	**N**atural **L**anguage **P**rocessing and **U**nderstanding
OCR	**O**ptical **C**haracter **R**ecognition
RAM	**R**andom **A**ccess **M**emory
RNN	**R**ecurrent **N**eural **N**etwork

SVM	**S**upport **V**ector **M**achine
WER	**W**ord **E**rror **R**ate
US	**U**nited **S**tates

Dedicated to my aunt
TEWABECH SEYFU

Chapter 1. Introduction

1.1 Background of the Study

1.1.1 Amharic Writing System

The *"Classic"* Ethiopic script was tailored for the classical language, Ge'ez, and so many new signs have been derived for modern languages. As a subgroup within these, Amharic (Amarñña: **አማርኛ**), is one of the Ethiopian Semitic languages using Ethiopic script. It is the official language of Ethiopia, which has more than 27 million speakers. It is spoken as a first language by the Amharas majorly in North Central Ethiopia. There are Amharic speakers in a number of other countries, particularly in Egypt, Israel, Sweden[1], also by many Ethiopians residing in some cities and towns of the US like: Washington, D. C. and New York City[2]. Moreover, Amharic is considered as a holy language by the Rastafarian religion followers worldwide. In general, Amharic is the second-most common language of Ethiopia (after Oromo) and the second-most commonly spoken Semitic language in the world (after Arabic).

Amharic alphabets, also called *"Fidel"* is a syllable writing system where consonants and vowels co-exist within each graphic symbol. It has 34 base characters from which other six or more families formed by changing its shape (see Appendix A). In addition, *"Fidel"* contain 7 derived characters such as: **ሸ** from **ሰ**, **ኸ** from **ከ**, **ጨ** from **ጠ**, etc. These result in similarity in shape which differs with a single stroke or mark.

Totally in a *"Fidel"*, there are 238 core characters and 27 labialized forms (which have two sounds such as: **ኗ, ቈ, ቿ**, etc.) [1]. Again, ten among the base alphabets don't have unique sound and can be used interchangeably in written documents (such as the pairs in a brace: **{ሀ,ሐ,ኀ,ኸ}, {ሠ,ሰ}, {አ,ዐ}, {ጸ,ፀ}**). Hence, we face multiple spelling variants for the same word (e.g. **አንበሳ, አምበሳ**). Amharic text is written unlike the majority of its Semitic scripts but like the Latin script from left to right, and separate words by blank space and top to bottom for newlines. But unlike the Latin, there is no lowercase and uppercase letters.

[1] Wikipedia: Amharic, 28. April 2019, Sunday, 9:06 AM.
[2] Wikipedia: Ethiopian Americans, 28. April 2019, Sunday, 9:41 AM.

1.1.2 Handwritten Amharic Text Recognition

Handwritten text recognition (HTR) system mainly combines Computer vision and Sequence learning. It often have an interpretation of handwritten text image by a vector of numerical values and normalization before the model, then convert into sequence of characters. Hence, the main goal herewith deals not who is the writer, but instead what is written out for the computer to represent by its symbols, Unicode or ASCII. Though, variations of the handwritten document is an issue in both scenarios.

Machine learning based HTR such as: advanced hidden Markov's model (HMM [2]) can automatically learn from the data, but ineffective as the data become too many and complex in variance. Therefore, systems including HTR are being shifted to Deep learning, which extracts and learns the specific features of the data each with respect to a sequence of labels hierarchically using layers of artificial neural network (ANN [3]) algorithms. Most of these, common in HTR are: convolutional neural networks (CNNs), recurrent neural networks (RNNs), long-short term memory (LSTM) or gated recurrent units (GRU). The state-of-the-art researches altered connectionist temporal classifications (CTC) that extract the final labeling from the model's output.

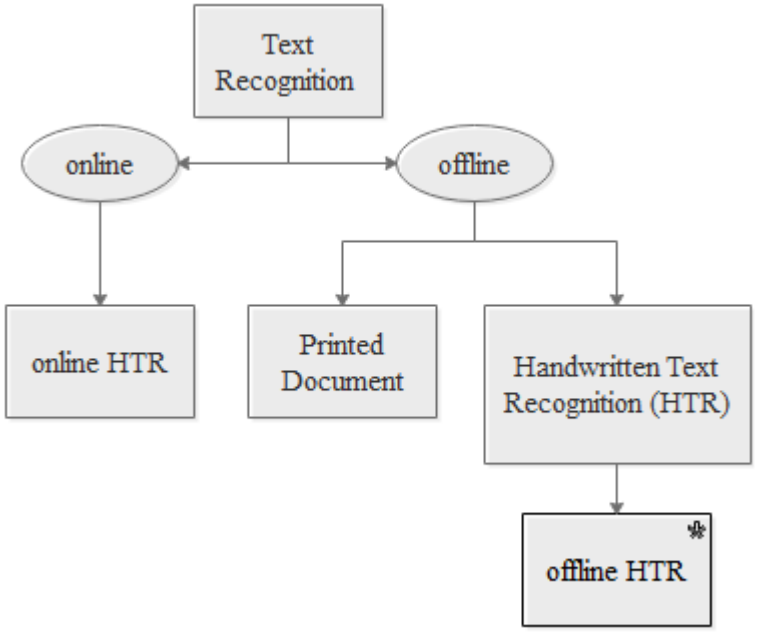


FIGURE 1.1: Text recognition

Also, the HTR systems can be designed as either online or offline [4]. Concisely, the online-HTR is done at the time of writing as the temporal features captured from both the pen trajectory and the resulting image, whereas the offline-HTR is done only

on the scanned image of handwritten document, which is after the writing is already completed. Thus, offline-HTR is more challenging than the online case.

Handwritten Amharic word recognition (AWR) is still active research area for the language, which transcribe handwritten word images into its respective digital forms. This is different from the one as character-level, line-level segmented, or as a page recognition mechanism in such a way handwritten word-level images serve as input to the system. As a sample text in figure 1.2 shows that we often face some cursive nature in handwriting of Amharic words that makes it difficult to segment individual characters. In order to avoid such difficulties it is better to segment words having a blank space between them and feed the recognition model with the word images.

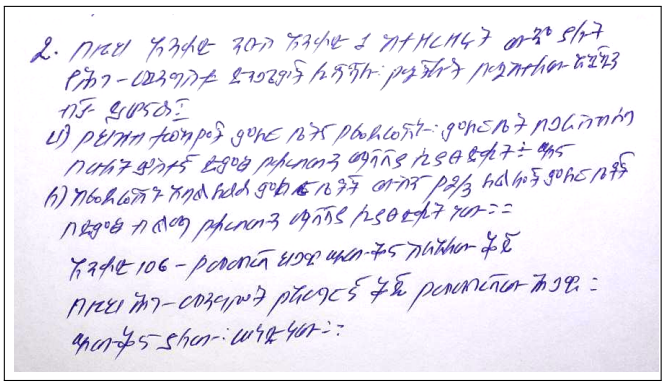


FIGURE 1.2: A sample handwritten Amharic text

1.2 Statement of the Problem

Historical documents and manuscripts are common in countries such as Ethiopia; these too old documents are all handwritten, and many of them are handwriting on tooled leather, then called “Brana”, all in ancient time. Now, the surviving “Brana” documents are placed somewhere by chance, many with missing pages and damage due to improper handling. It’s still a challenge to develop a system preserving them.

Another challenge, it is obvious that the calligraphy and nature of handwriting differs personally in the writers. Every individual has a different use of white spaces or lines, and style of writing and moreover, depending on the various underlying factors, even the style of a particular person also changes in different instances of writing time.

With the advent of neural network models such as deep learning and CTC-decoder transcribing handwritten documents in to digital forms without handcrafted features become achievable. Having these computational models, there are a lot of state-of-art research works but the issues that are not considered in these research efforts raised as the research-question in this thesis.

FIGURE 1.3: An old Brana poem in Amharic entitled Märgämä k br, "Condemnation of glory" (hence MärKL) [5].

These questions are:

1. What type of dataset of handwritten Amharic words can really represent most from different domains and of different appearances?
2. What are the best methodologies that can be used in handwritten Amharic text recognition?
3. Which score technique measures a performance of the system when validating the recognition over unseen part of the dataset?

1.3 Objectives of the Study

1.3.1 General Objective

The main objective of this study is to design and develop offline handwritten Amharic words recognition system using deep learning approaches such as: CNN, GRU/LSTM and CTC-decoder custom model.

1.3.2 Specific Objectives

The specific objectives of the study include:

- To develop the dataset of handwritten Amharic words for the language.
- To optimize a model that works well for AWR by using Bayesian hyper-tuning.
- To evaluate and measure the performance of AWR system using train-test set and k-fold cross validation.
- To compare the result with pre-designed architectures such as: VGG, ResNet, DenseNet and EfficientNet.

1.4 Significance of the Study

The purpose of this study is to expose the operation of Deep learning models on handwritten Amharic words, where it has been preferred to offer a robust theoretical foundation of the system, emphasizing from input data preparation to testing the designed model, pursuing high performance in the transcripts. Thus, the system is capable of taking an word image and return a word, in the form of a tag sequence.

The potential AWR applications are in preserving historical heritages by converting old documents, medicinal or magical books and manuscripts (a sample manuscript is shown in figure 1.3), bank check processing or used to read students and lectures handwritten notes in to digital format. These also aid in search engines, machine translations and other machine learning purposes: natural language processing and understanding (NLP/U) for analysis of the text syntactic and semantics on a computer or an underlying machine.

In the transcription of unconstrained handwritten Amharic text to the corresponding editable digital form, this thesis contributions are as follows:

1. Lack of public dataset for handwritten recognition for the Amharic language makes it difficult for researchers to make comparison of their work with others.

In this study, we release a new and the first to the authors knowledge Amharic Handwritten dataset called HAW-DB to alleviate the issues.

2. Bayesian hyper-parameter tuning, a model-based method for finding the minimum of a function, has been used to tune and select the hyper-parameters of a given well-performing model on a validation dataset.

3. K-fold (10-fold) cross validation is applied while training the model in order to make it a less biased.

4. Comparisons are made among alternatives to the backbone CNN, the feature extracting component, by switching the CNN-part in custom model we developed through Bayesian optimization.

1.5 Research Methodology

In AWR system, the handwritten word image dataset preparation is an earlier stage. There is no any free downloadable dataset online in the Internet. Hence we collected photo of handwritten papers afterwards conducted certain procedures for word segmentation. Subsequently, the handwritten word image is input to the CNN layers, which extract features. And then, the output is regarded as a sequence (it can be 1 or 2 Dimensional) of features and serves as input to the GRU or LSTM layers. And using the output, CTC calculates the error made with respect to the expected word, the value of the objective function for the given sample and the most probable output sequence, which we will take as the output of the system. Finally, evaluating the model using the cross validation metrics for HTR: the average character error rate (CER), word error rate (WER) and the respective accuracy rates. CER and WER refers to the amount of incorrectly transcribed text in a handwriting image.

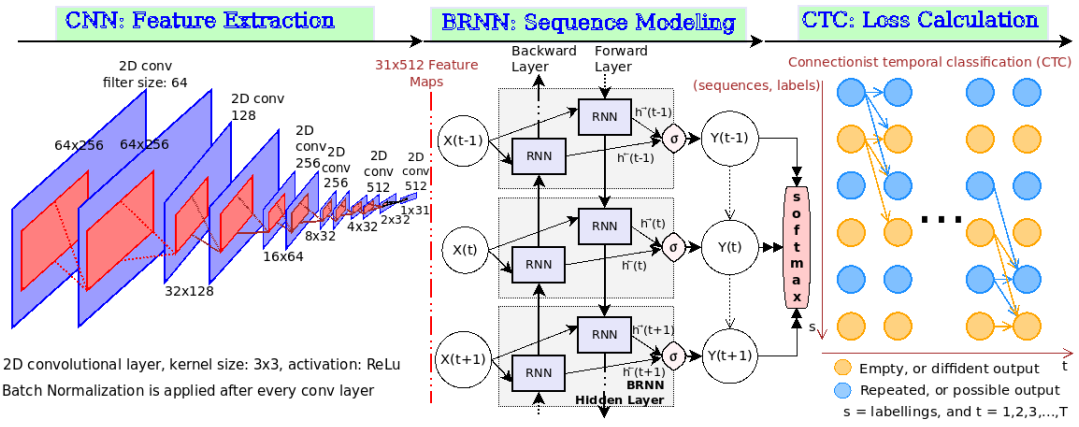


FIGURE 1.4: Major components of the system

1.6 Scope and Limitations of the Study

This thesis focuses on handwritten Amharic words dataset preparation methods using the input text images, and on identifying optimization and evaluation method for a model that results in better testing output sequence. Only offline scenario is considered and recognizes handwritten Amharic words after trained with the dataset prepared by ourselves. Offline methods involve recognizing text once it is written down usually on a paper. Thus, unlike online methods, there is no information to the strokes or directions involved during writing with some background noises and possible degradation of part of the text.

Eventually, the system do not handle any misspellings made in the source or by classifications in the system itself. In relation to that, some of Amharic alphabets have the same sound is still challenging to the system design; for instance, words in free handwriting varies with writers choice of characters to use even when writing the same word, which means there can be spelling variations for a single Amharic word. Because, the writers simply focuses on the message needed to be forwarded. The system fails to know any new character contained in word that's out of the set of characters sequence which is given to train the model.

Skew detection and removal is also an issue for the HTR system which has not been addressed perfectly yet and including the current work. Consequentially, this leads to incorrect spelling. But in the recognition system, other than using dictionary to replace the output with a right version, it can integrate automatic spelling correction. It's left as future work to correct spellings of words that may occur due to human mistakes, part of text degradation or addition of noise by using state-of-the-art deep neural networks, generative adversarial network (GAN).

1.7 Thesis Structure

This thesis work have been done and organized as follows:

Chapter 1: Introduction to handwritten Amharic word recognition, also the current chapter. In this chapter, we will give the background of the study, problem statement, objectives, significance and practical applications in the future.

Chapter 2: Review of some research related to the study that we can learn through two parts: Machine learning and Deep learning approaches, usage model and the achieved results with brief description on their strength and weakness.

Chapter 3: Data collection and Dataset preparation. Statistics of of the dataset and word segmentation approaches are presented.

Chapter 4: Presents the approach or procedure followed to address the problem: this chapter will go into details on the fundamental knowledge serving as the neural network in deep learning and custom model architectures of networks.

Chapter 5: The test results obtained carried out. Two tests have been performed on the system to evaluate its performance in the transcription of handwritten Amharic words: (1) the train-test split, and (2) a 10 fold cross-validation by using our own handwritten Amharic words dataset. In the following sections we show an extract of the results obtained in these tests.

Chapter 6: Summary of the thesis work done, recommendation and future works. We write the conclusions obtained and propose possible future related works, which either seek to solve the shortcomings and errors that throughout our approach we have been able perform, or propose alternative ways to tackle the same problem.

Chapter 2. Background Information and Theory

In this chapter, we briefly introduce the methods for HTR into two parts: machine learning and deep neural network based recognition systems.

2.1 Machine Learning Approaches

Machine learning can enable Artificial Intelligence (AI) systems to learn from data. Learning simply means *"a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E"* [6]. Based on the way that experience E is updated, machine learning algorithms are classified into three main categories: supervised, unsupervised, and reinforcement learning. The HTR algorithms are under supervised learning, thus they need labeling of every input data and they learn to predict the output from the data. Hidden Markov models (HMMs) and also conditional random fields (CRFs) that is a predominant sequence labeling framework [7] are among the common ones, for instance,.

2.1.1 Hidden Markov Model (HMM)

The HMM is a statistical model that was first proposed by Baum L.E. [8] uses a Markov process that contains hidden (unknown) events. In the case when we need to compute a probability for a sequence of observable events, Markov's chain is useful.

In general, there are some drawbacks that exist when using machine learning algorithms. For the case such as designing HMM state models, we need to have adequate task-oriented knowledge. The assumptions need also be with explicit dependency to make their inference tractable. For example, the assumption that observations in HMM should be independent. However, HMM has discriminative sequence labeling but the training is generative.

2.2 Deep Learning Approaches

2.2.1 Neural Network (NN)

The background of artificial neuron is the biological nervous system [9]. E.g. figure 2.1 shows an electrochemical process taking place to identify NaCl concentration.

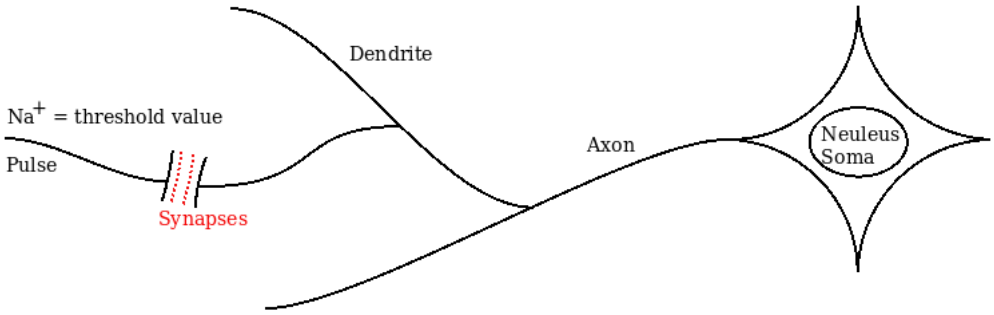


FIGURE 2.1: Biological neuron

In artificial neural network (ANN), a neuron is a node or a point at which certain mathematical operation is performed. A neuron to perform its task:

1. The function it is going to perform must be known, and
2. It has to have an input by which the function is executed.

Depending on the value of executed function the neuron gives out the output, or it can be blocked.

- Neuron gives output if the executed function is greater than certain threshold.
- Neuron is inhibited (blocked) if the executed function is below that threshold.

Generally, an artificial neuron has the following structure as shown in figure 2.2. Where, x is given input signals or vector and if f is already known neuron activation function, then $y = f(x)$.

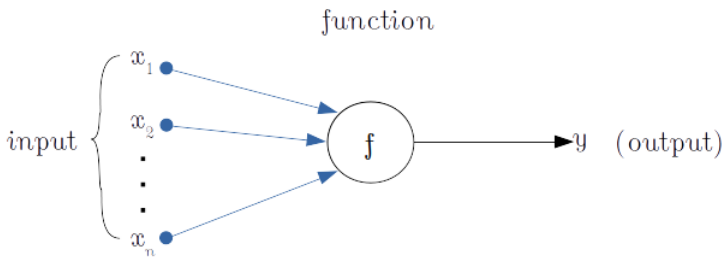


FIGURE 2.2: Artificial neuron model

2.2.2 Convolutional Neural Network (CNN)

Convolutional neural network (CNN) is one of ANN inspired by the primary visual cortex of the brain, specialized NN for processing visual information of static or moving objects and feature extraction. From a computational point of view, the main advantage of this type of NN lies in operating on the elements of an input tensor taking into account the value, position and neighborhoods of them. These operations are convolutions between the input tensor and the network's tensors, called filters.

$$C(i,j) = (I*K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.1}$$

The CNN is composed of a set of convolutional kernels where each neuron acts as a kernel. The convolutional layer has also weight sharing ability so that it exploits most relevant feature-maps by sliding kernel with the same set of weights on the image. Equation 2.1 gives the common 2D convolution operation used in deep learning for a 2-dimensional image I with a 2-dimensional kernel K. Thus in the HTR systems, CNN involves convolution of the input tensors in two dimensions to give the output.

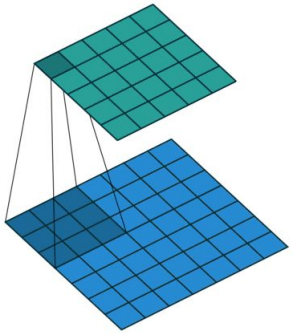


FIGURE 2.3: An illustration of convolutional neural net.

The output size of the CNN with given an input size N and receptive fields F is determined by the formula $(N-F)/strides+1$. For figure 2.3 with 3x3 connectivity and stride equals 1 is applied to 7x7 input image so that the output would be 5x5. Generally, the choice of kernel size, stride and zero padding along axis j, which forms the receptive fields affect the output size of axis j [10].

CNN usually ends with nonlinear activation functions like *ReLU* or *tanh*. This results in local connections, where each region of the input is connected to a neuron in the output. Every CNN layer can apply different filters and give well chained results.

2.2.3 Recurrent Neural Network (RNN)

When dealing with language data, it is very common to work with sequences, such as words (sequences of characters), sentences (sequences of words) and the so on. Recurrent neural network (RNN) is initially proposed by Elman in 1990 [11] and explored for use in HTR [12], [13]. RNN is a model in which the output is a function of not only the current input but also of the previous output, which is encoded into a hidden state h. I.e. RNN has also a memory of the previous timestep by which it can encode the information present in the sequence itself. Therefore, RNN are able to model multivariate time-series (e.g. sequence labeling in pattern recognition tasks like HTR) and output a class prediction by considering the whole temporal sequence. RNN updates its memory as indicated in equation 2.2.

$$h_t = g\left[Wx_t + Uh_{t-1}\right],\ a_t = Vh_t \tag{2.2}$$

Where h_t represents the hidden state at time-step t. The weight matrices W, U and V are input-to-hidden, hidden-to-hidden and hidden-to-output respectively. Finally, an activation a at time-step t given by the product of V and h_t.

Bidirectional Long Short Term Memory (BLSTM)

Bidirectional long short-term memory (BLSTM) is a kind of bidirectional recurrent NNs (BRNNs), which is to connect two hidden layers of opposite directions to the same output and first proposed in [14]. The output layer can get information from past (backwards) and future (forward) states simultaneously. Each LSTM consists of more sophisticated and recurrently connected sub-nets, known as "memory cells". The activation of each cell is controlled by three multiplicative gate units: the input gate, forget gate and output gate. These gates allow information to have long range inter-dependencies. Hence, this is adding new values on to the activation as it goes deeper through the layers, thereby avoiding the vanishing gradient problem that makes LSTM much similar with the residual neural networks, or ResNets [15]. Equation 2.3, 2.4, 2.5, 2.6 and 2.7 define general behavior of the LSTM units.

The input gate controls whether the input of the cell is integrated in the cell state.

$$a_l^t = \sum_{i=1}^{I} w_{il} x_i^t + \sum_{h=1}^{H} w_{hl} z_h^{t-1} + \sum_{c=1}^{C} w_{cl} s_c^{t-1} \tag{2.3}$$

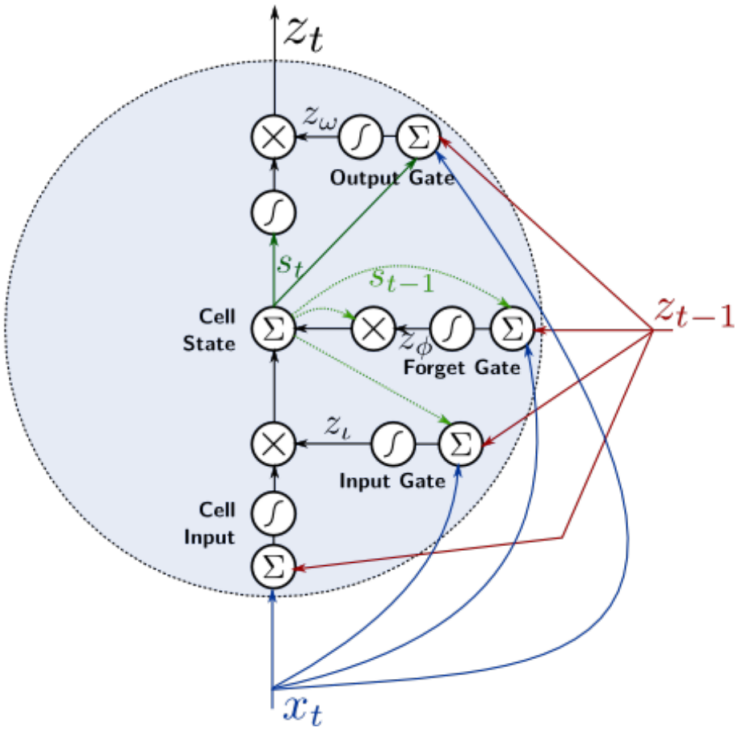


FIGURE 2.4: An illustration of long short-term memory unit

The forget gate controls if the previous state is integrated in the cell state, or if it is forgotten.

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} z_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1} \tag{2.4}$$

Hence, the outputs of these gates given by: $z_l^t = f(a_l^t)$ and $z_\phi^t = f(a_\phi^t)$ and having:

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} z_h^{t-1} \tag{2.5}$$

The cell state is the sum of the previous state, scaled by the forget gate, and of the cell input, scaled by the input gate: $s_c^t = z_\phi^t s_c^{t-1} + z_l^t g(a_c^t)$. Most of the time, an activation function f, g and h for the gates are sigmoid functions. The output gate controls whether the LSTM unit emits the activation $h(s_c^t)$.

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} z_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^t \tag{2.6}$$

Therefore, $z_\omega^t = f(a_\omega^t)$ and as depicted in figure 2.4 also, the cell output is computed by applying the activation function h to the cell state, scaled by the output gate.

$$z_c^t = z_\omega^t h(s_c^t) \tag{2.7}$$

Bidirectional Gated – Recurrent Unit (BGRU)

GRUs according to [16]. Let $x = (x_1, ..., x_t, ..., x_T)$ for $x_t \in R^n$ be a sequence of T observations and $y \in C$ where C is ground truth class label. A GRU cell receives x_t and outputs an activation $h_t \in R^m$ at every time-step t as calculated in equation 2.8.

$$h_t^j = (1-z_t^j)h_{t-1}^j + z_t^j \tilde{h}_t^j \tag{2.8}$$

While applying activation at previous time-step h_{t-1} and the update gate factor z_t is given by equation 2.9 for optimizable parameters $W_z \in R^{m\times n}$ and $U_z \in R^{m\times m}$ shared across all t and a sigmoid function σ that outputs values in an interval $[0, 1]$.

$$z_t^j = \sigma(W_z x_t + U_z h_{t-1})^j \tag{2.9}$$

Likewise the update gate, the reset gate r_t is provided by equation 2.10.

$$r_t^j = \sigma(W_z x_t + U_z h_{t-1})^j \tag{2.10}$$

The $\tilde{h}_t^j$, called a candidate activation, can be obtained by using equation 2.11. When the element-wise dot product $\bigodot$ is done between r_t and the previous time-step h_{t-1}.

$$\tilde{h}_t^j = \tanh(W x_t + U(r_t \bigodot h_{t-1}))^j \tag{2.11}$$

The log-it value z^i in a dense layer is given by equation 2.12 after the final GRU activation at time T is input to the dense layer with softmax activation function.

$$z^i = \sum_j w_s^{ij} h_T^j \tag{2.12}$$

For the softmax layer weights w_s^{ij} contained in W_s and for the stacked layers of bidirectional GRU, $\left[h_{fw,T}^j, h_{bw,0}^j\right]$ implies to concatenate the forward and backward GRUs activation. Thus, the log-it value is modified as expressed by equation 2.13.

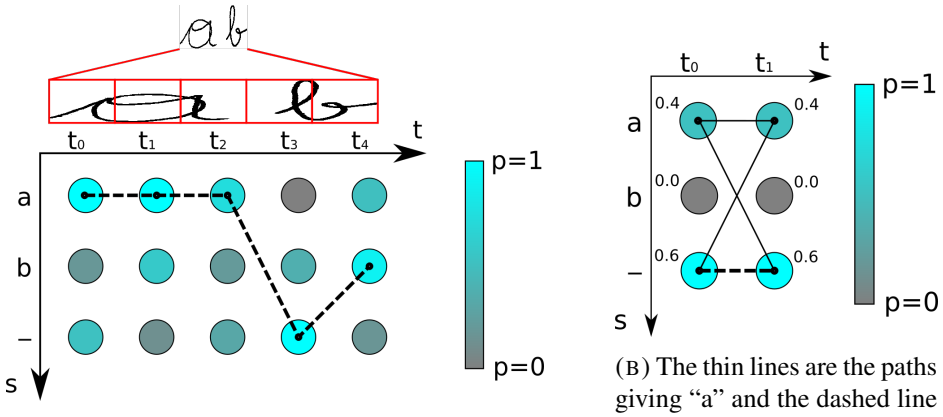
$$z^i = \sum_j w_s^{ij} \left[h_{fw,T}^j, h_{bw,0}^j\right] \tag{2.13}$$

The softmax activation realizes the final sequence labeling as given by equation 2.14.

$$\hat{y}^i = \frac{e^{z^i}}{\sum_i e^{z^i}} \tag{2.14}$$

2.2.4 Connectionist Temporal Classification (CTC)

The CTC eliminates duplicates in the output sequence, $e.g.\ aaabb$ becomes ab. But, a problem arises when the original word has some consecutively repeated letter, such as *"see"* or *"book"*, the system transcribes them as a single letter. To overcome this, a *blank* symbol $\emptyset$ is used to represent a *no label* observations.



(A) The path is (aaa_b) in this example.

(B) The thin lines are the paths giving "a" and the dashed line is the path giving labeling $\emptyset$.

FIGURE 2.5: An illustration on the example how CTC can fix an output sequence

The CTC corresponds to the task of labeling unsegmented sequence data with RNNs [17]. It applies the output of the NNs (the sequence of symbols of interest, such as a word or the sequence of characters) to an input sequence, and there is no need of one target at each timestep (unlike sliding window approach over each frame). Thus, it does not require pre-segmented data while training the network and the output is already sequence of characters that do not need any post-processing. Hence, let's assume the input sequence $x = (x_1, ..., x_t, ..., x_T)$, whose length is T, target space $\mathcal{Z} = L^*$ which is the set of all sequences over an alphabet of each label L. The target sequence $z = (z_1, ..., z_u, ..., z_U)$, and the length U is always less or equal to T $(i.e. |z| \leq |x|)$. A many-to-one mapping $\mathcal{B}$ is $L'^T \mapsto L^{\leq T}$, where $L^{\leq T}$ is a set of possible label sequences whereas the output alphabet that contains a blank: $L'^T = L \cup \{\emptyset\}$. Finally, by eliminating all blanks and repeated labels from the paths, for instance: $\mathcal{B}(\emptyset bbo\emptyset oo\emptyset\emptyset kk) = \mathcal{B}(book)$.

An RNN with m inputs, n outputs and weight vector w as a continuous map $\mathcal{N}_w$: $(R^m)^T \mapsto (R^n)^T$. We assume the network outputs do not have any connection for different timesteps, and $y = \mathcal{N}_w(x)$ as the sequence of the outputs, and denote by y_k^t which is the activation of output label k at time t.

$$p(\pi|x) = \prod_t y_{\pi_t}^t, \ \forall \pi \in L'^T \tag{2.15}$$

Equation 2.15 is the probability of observing π, defining a distribution over the set of sequences L'^T, given the input sequences x. Next, the many-to-one mapping $\mathcal{B}$ is used to compute the posterior probability of a label sequence l by simply summing up all one by one, as given in equation 2.16.

$$p\left(l|x\right) = \sum_{\pi \in \mathcal{B}^{-1}(l)} p\left(\pi|x\right) \ , \ \forall l \in L^{\leq T} \tag{2.16}$$

To train the network with unsegmented data $\mathcal{S} = \left\{(x, z), \forall z \in L^{\leq |x|}\right\}$ formulated by maximizing the correct labelling and by minimizing the CTC-loss in equation 2.17.

$$E_{CTC} = -\sum_{(x,z)\in\mathcal{S}} \log p\left(z|x\right) \tag{2.17}$$

The use of forward and backward algorithm in a CTC graph, which represents each of the possible label sequences, are defined as equation 2.18 and 2.19 respectively for $\pi_t = l'_s$ where $l' \in L'^T$ and $l \in L^{\leq T}$.

$$\alpha_t(\mathcal{S}) = p\left(\pi_{1:t} : \mathcal{B}(\pi_{1:t}) = l_{1:s/2}|x\right) = \sum_{\pi:\mathcal{B}(\pi_{1:t})=l_{1:s/2}} \left(\prod_{t'=1}^{t} y^{t'}_{\pi_{t'}}\right) \tag{2.18}$$

$$\beta_t(\mathcal{S}) = p\left(\pi_{t+1:T} : \mathcal{B}(\pi_{t+1:T}) = l_{s/2:|l|}|x\right) = \sum_{\pi:\mathcal{B}(\pi_{t:T})=l_{s/2:|l|}} \left(\prod_{t'=t+1}^{T} y^{t'}_{\pi_{t'}}\right) \tag{2.19}$$

The recurrence natures for the two variables are given by equation 2.20 and 2.21.

$$\alpha_t(\mathcal{S}) = \begin{cases} y^t_{l'_s} \sum_{n=0}^{1} \alpha_{t-1}(\mathcal{S}-n)\,, & if\ l'_s = \emptyset\ or\ l'_s = l'_{s-2} \\ y^t_{l'_s} \sum_{n=0}^{2} \alpha_{t-1}(\mathcal{S}-n)\,, & otherwise. \end{cases} \tag{2.20}$$

$$\beta_t(\mathcal{S}) = \begin{cases} y^{t+1}_{l'_{s+1}} \beta^{t+1}(\mathcal{S}+1) + y^{t+1}_{l'_s} \beta_{t+1}(\mathcal{S})\,, & if\ l'_s = \emptyset\ or\ l'_s = l'_{s-2} \\ y^{t+1}_{l'_{s+1}} \beta_{t+1}(\mathcal{S}+1) + y^{t+1}_{l'_s} \beta_{t+1}(\mathcal{S}) + y^{t+1}_{l'_{s+2}} \beta_{t+1}(\mathcal{S}+2)\,, & otherwise. \end{cases} \tag{2.21}$$

Equation 2.22 shows that a probability of l given by the product of α and β at time t.

$$p(l|x) = \sum_{t=1}^{T} \sum_{s=1}^{|l|} \frac{\alpha_t(\mathcal{S})\beta_t(\mathcal{S})}{y^t_{l_s}} = \sum_{s=1}^{|l'|} \alpha_t(\mathcal{S})\beta_t(\mathcal{S}) \tag{2.22}$$

If a set where label k is located be $lab(l, k) = \{s : z'_s = k\}$. Equation 2.23 is derivatives of the loss with respect to the activation a^t_k before the softmax.

$$\frac{\partial E_{CTC}}{\partial a^t_k} = y^t_k - \sum_{s\in lab(z,k)} \frac{\alpha_t(\mathcal{S})\beta_t(\mathcal{S})}{\sum_{s'=1}^{|z'|} \alpha_t(\mathcal{S}')\beta_t(\mathcal{S}')} \tag{2.23}$$

Most Deep learning model for the HTR system pipeline CNN, RNN and CTC. For the RNN component of the system, LSTMs with their very deep residual networks are notoriously difficult to train and still they have very long gradient paths even though there is propagating gradient from the end all the way transformation cells over the beginning. Whereas, the GRUs have internal gates by which they can reset all long temporal information. Hence, they are easier in order to train than LSTMs.

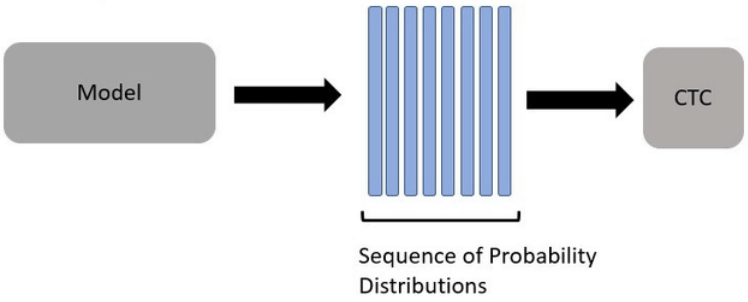


FIGURE 2.6: An illustration of the Deep learning model pipeline for the HTR system

2.3 Evaluation Metrics

In HTR model, the metrics should be measuring similarity or dissimilarity (distance) between two text strings. The character error rate (CER) and word error rate (WER) are therefore the standard numerical error measures. The CER is the number of edit operations to match the recognized text with the ground truth (GT), which essentially is the Levenshtein edit-distance, divided by the GT text length (see Equation 2.24).

$$CER = \frac{insertions + deletions + substitutions}{length(GT)} \tag{2.24}$$

For instance, when the GT is "$How\,are\,they$" but recognized as "$Hox\,are\,xhex$". The character edit-distance is 3, the GT length is 12, so that $CER = 3/12 = 25\%$. For WER, the text is split into a sequence of words so that the unique words are $w_1 =$ "How", $w_2 =$ "Hox", $w_3 =$ "are", $w_4 =$ "$they$", $w_5 =$ "$xhex$", put into a table with unique identifiers. Then, each word is replaced by its identifier from the table, so that the three words become "$w_1 w_3 w_4$" and "$w_2 w_3 w_5$". The word distance is 2, the ground truth length is 3, then $WER = 2/3 = 66.66\%$ [1]. If the number of edit operations exceeds the (GT) text length, then the value of CER and WER can be greater than 100%. For example, let's assume "$abcd$" is recognized but the ground truth is "yz", the CER becomes $4/2 = 200\%$.

[1] Digitization & Handwritten Text Recognition. By: D. Alvermann and E. Heigl on 14. May 2020.

2.4 Literature Reviews

The first handwriting recognition was in the 1995 which is character-level online-HTR based on Hidden Markov Models (HMMs) [18]. However, for a given character in a word HMMs inherently do not consider the previous or following characters. This ultimately limited the attainable accuracy and extension of such models [19].

Nowadays, usage of digital technologies arise in most sectors and many aspects of life to store and transfer information. People still communicate through handwriting, but they desire it's transcription into electronic text. Handwritten recognition for this purpose has emerged from the earlier works that use of support vector machine (SVM), k-nearest neighbors algorithm (K-NN), hidden Markov model (HMM) or any other machine learning approach to neural networks (NNs), and hybrid of NNs with machine learning algorithms [20] for implementing HTR.

HMMs–based HTR System

The reward of researches investigating the application of HMMs for handwriting recognition comes from the fact that HMMs have a strong theoretical and statistical foundation to cope with noise and variability of data. Consequently, the recognition of isolated handwritten characters has been done using HMMs by building a model for each character. The number of characters in a script is not large; therefore, it is possible to collect sufficient dataset for each class. As well as, the recognition of a small set of words in a specific application is achieved successfully. Because, it is again possible to collect sufficient dataset and built a model for each word. Here, an earlier handwritten word recognition can be done by concatenation of HMMs of characters constituting the word [21], [22]. Also, a local research corresponding to this has presented feature-level concatenation method. In the method some sample features of training words are generated by feature sets of constituent characters. The feature set stores a variety of sample features for each character reflecting different real-world writing styles. Finally, feature-level concatenation method outperformed over HMM-level concatenation across different document qualities and varying sizes of training and test data [1].

The HMMs-based HTR systems usually require data preprocessing steps such as slant correction, size normalization (or encodes the relative size of primitive strokes, without size normalization), input features extracting (by moving a sliding window in the image in such a way so as to give a sequence of observations, so features are extracted in each window frame using image transformations, such as cosine transform, Fourier transform, and KarhunenLoève transform), etc.

However, HMMs-based HTR systems are not enough to withstand many of the usual challenges especially that are associated with offline handwriting recognition: the input is a variable-sized 2D image, the output is a variable-sized series of character, with no direct relation to the input size, and the cursive nature of handwriting makes a prior segmentation into characters difficult.

DNNs–based HTR System

There are many emerging HTR architectures in the Deep learning that allow building models, which are prominent in handling the variable-sized 2D nature of the input image together with the sequential aspect of the prediction. These DNN models are providing the automatic transcription without requiring a further segmentation into the sets of constituent characters from unconstrained handwriting prior to the training. The CNN-RNN-CTC architecture is the most common state-of-the-art in the development of HTR systems. This is by modeling CNNs to extract features from raw images of the text, RNNs to learn and encode sequential relationship between features and CTC as a decoding algorithm.

Multi-Dimensional Long Short-Term Memory (MDLSTM) in recurrent neural nets (RNNs) studies used to generalize the LSTM architecture to multi-dimensional inputs. The image is presented to multi-layers of multi-dimensional LSTMs so that there is a layer to scan along each direction. Then, the inner state and output of the LSTM cell are obtained from the previous states and output in each direction. And, the MDLSTMs are joined with the CTC for yielding much lower error rates [23].

An architecture of CNN-RNN-CTC with an application of the attention mechanism can improve the recognition task [24]. Another study similarly appeared with an attention-based Sequence-to-Sequence architecture, which can be trained end-to-end on full line or word levels [25]. The model in the study has combined a convolutional feature extractor with a recurrent neural network to encode both visual information and temporal context in the input image, and use of an extra RNN to decode the actual character sequence. And also, this made an appropriate alignment between the input and output sequence after comparing positional encoding and attention mechanisms. The model in the study can be used for Keyword Spotting (KWS) without any prior indexing if the encoder conforms to the CTC framework, even though this was not necessary for the performance regarding HTR.

Another study using an encoder-decoder approach with a gated convolutional unit applied at intermediate feature maps of the encoder unit [26]. And in the study, the encoder with text line image dataset from seven languages is trained and the decoder is fine-tuned for specific languages in their model.

A comparison between recurrence-free gated convolutional networks (G-CNNs) and CNN+BLSTM baseline models favored as a new direction to the use of G-CNN in HTR task [27]. As well as, by using some online handwritten data to train the offline handwritten text recognition models a study can apply the two architectures: CNN-LSTM-CTC models and recurrence free models using one-dimensional gated recurrent convolutional layers as an alternative for handling sequence data [28].

A fully convolutional neural network approach is used along with inter-layer residual connections and attention gates are used to control inter-layer information flow [29]. Then, it is claimed that the architecture outperforms text recognition task on seven public benchmark dataset by achieving state of the art result.

2.4.1 Related Works

Unfortunately, only few literature have been made for handwritten Amharic text recognition. Most of the existing handwritten recognition for Amharic language are based on the transcription of isolated handwritten characters [30]–[33]. HMM model for recognition of handwritten Amharic words used a traditional hand-engineered feature to represent character symbols [1]. Hence, the segmentation process for each of the characters from the handwritten text is more challenging due to cursive nature of the free-handwriting. And, this is one of the reason for the less accuracy of such a character based HTR models.

Eventually, there are two lately researches that are published on Amharic optical character recognition(OCR) task [34], [35] and approached the task with the long short-term memory recurrent neural networks (LSTM-RNNs) in combination with CTC algorithms in an end-to-end system. This is done for printed and synthesized Amharic text-line images.

According to the baseline model used, there are also many other studies related and some of which are summarized in table 2.1. Most of these are using different datasets so that it does not allow us to compare to one another. Nevertheless, as far as we know, there is no research work particularly on offline handwritten Amharic word recognition using Deep learning. In this study, we proposed our own custom model with convolutional neural nets (CNNs) plus recurrent neural nets (RNNs) plus CTC loss function and RNN output sequence decoding algorithms, which is the state-of-the-art technique, for offline HTR systems. In addition, we have publicly provided a dataset for handwritten Amharic words consisting of 36,192 word-images.

TABLE 2.1: Summary of some related works.

Year	Authors	Target scripts/Database	Baseline models	Results/Accuracy/CER	Citation
2019	R. Ghosh, C. Vamshi and P. Kumar	Indian Devanagari and Bengali Characters	CNN + LSTM and BLSTM + CTC	99.50% for Devanagari and 95.24% for Bengali	[36]
2019	H.P. Tran, A. Smith, and E. Dimla	IAM Handwriting English Sentence Database [37]	CNN + BLSTM + CTC	88.18% on test data	[38]
2019	Xinfeng Zhang and Kunpeng Yan	offline Chinese handwritten text, a dataset collected from students	CNN + BRNN	83.0% on test set data	[39]
2020	José C. Aradillas, J.J. Murillo-Fuentes and Pablo M. Olmos	small English handwritting ICFHR 2018 competition database, Washington and Parzival	CNN + LSTM + CTC (with transfer learning)	CER (up to 6% in some cases) in the test set	[40]
2018	A. Granet, E. Morin, H. Mouchére, S. Quiniou and C. Viard-Gaudin	French and Italian language using Italian Comedy (CI) dataset from a pre-trained model on RIMES (RM), Los Esposalles (ESP) and Georges Washington (GW) dataset	CNN + BLSTM + CTC (with transfer learning)	this shows a possibility of transfer learning across languages, but it's model recognition rate is low.	[41]
2020	S. Bansal, P. Krishnan and C.V. Jawahar	a collection of synthetic text in the Hindi language	CRNN (uses Embed-Net, a pre-trained word image embedding net-work)	word recognition accuracy (WRA) of 85.364% at K textual transcriptions = 20	[42]
2019	R. Chamchong, W. Gao and M. D. McDonnell	Thai handwritten dataset collected from ancient archive documents	CNN + BLSTM + CTC		[43]
2018	R. Maalej and M. Kherallah	Arabic handwritten text IFN/ENIT database [44]	CNN + BLSTM + CTC		[45]

Chapter 3. Data Collection and Preparation

3.1 Data Collection

The photo image is taken using camera and scanner. A total of 60 writers, who are with different age groups and educational background, were asked to write the textual content of documents with their free hand-writing style. The document used as handwritten are the Aristotle monograph (by: Hailegiorgis Mamo, wisdom from Pilate), Astronomy newsletter (columnist: Zemene Yohannes, Zoskales), Ethiopian constitution, and "Thus Spake Zarathustra" (author: Friedrich Nietzsche; translated by: Esubalew Amare). As shown in Table 3.1, they are diversified data and inclusive to many words from different aspects of views. The dataset contains 4,309 unique words and the 2,733 out of them are single words, which are written only one time.

TABLE 3.1: Statistics of the dataset

Data Sources	Pages	Lines	Words
Aristotle monograph	7	178	1,389
Astronomy newsletter	7	188	1,476
Ethiopian constitution	50	1,074	7,817
Thus Spake Zarathustra	6	163	1,382
Total	**70**	**1,574**	**12,064**

3.2 Data Preparation

The data preparation mainly categorized into 3 processes: scanning and cropping handwritten text image, binarization, and segmentation.

3.2.1 Scanning and Image Cropping

This is a process of detecting the predominant contour in the image and segment by using four separate points of perspective transformation. Then, image cropping is helpful to resize and limit definitions in the image of handwriting.

3.2.2 Binarization

Binarization is one of the most common techniques in digital image processing (DIP) that is applied to the text image: reduce it to binary form (usually, 1's on a background of 0's) using a thresholding transformation function followed by a thinning process. Thus, it used to thin the characters until they become strings of binary.

Some types of binarization include:

- OTSU: A threshold selection method from gray-level histogram (1979).
- NIBLACK: An introduction to digital image processing (1986).
- SAUVOLA: Adaptive document binarization (1997).
- WOLF: Text localization, binarization and enhancement in multimedia document (2002).
- SU: Binarization of historical document images using local maximum and minimum (2010).

We have used some variance of cameras and few of them were with less resolutions. Therefore, we chose Wolf for most of our text images at this stage. However, Otsu finds an optimal threshold to apply it for every pixels in the text image. But, Otsu's and Sauvola's algorithms are ineffective on complicated background, or incapable to correctly delete the graphics that are present in some images. This results false lines detection and an error in the line localization and consequently to the entire method.

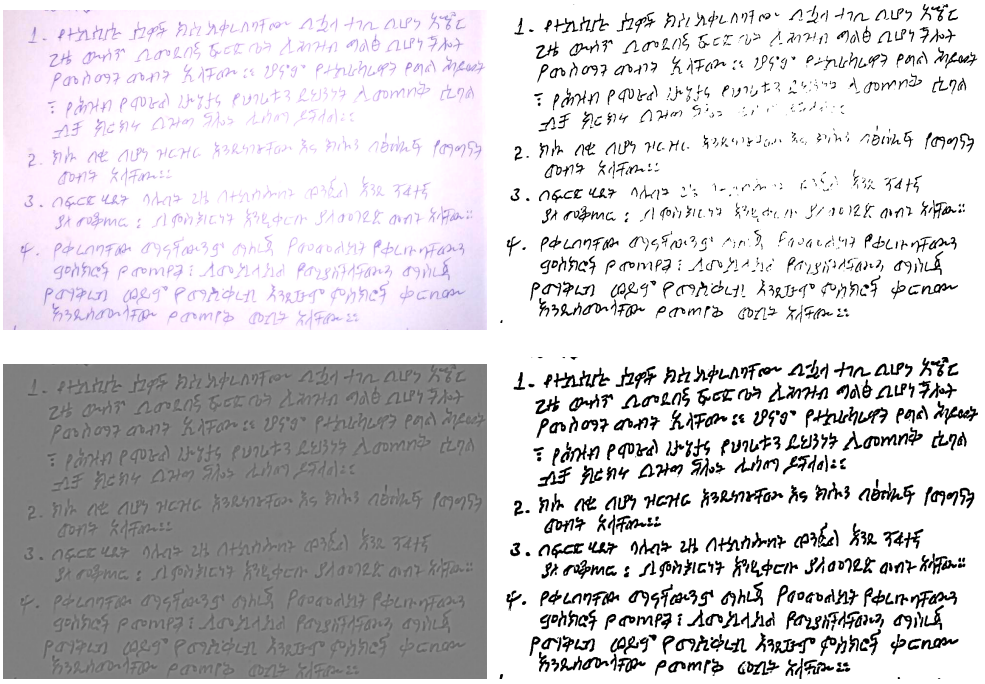


FIGURE 3.1: A challenge in binarization

In addition, some noise creates broken strings of characters with gaps of a few pixels in the binarization and thinning processes. One way to "repair" the gaps is applying an averaging mask over the binary image to blur it, so that bridges of nonzero pixels between the gaps. After bridging the gaps, it is desired to threshold the image in order to convert it back to binary form. In figure 3.1, on the left upper side image is taken in flash light, which results in loss of some black-pixels after the binarization process as shown on the right upper side. The potential solution to this problem is to use high-pass filter before the process as shown in followed pictures. We then chose an adaptive binarization the so called "Efficient illumination compensation techniques for text images," (2012) contributed for balancing image illumination to help with the process of adaptive binarization.

3.2.3 Segmentation

After binarization algorithm take place; According to [46] it is possible to segment text in levels of lines, words or characters. Most of the methodologies for segmenting include: pixel counting, histogram, smearing, water–flow, or stochastic approaches, Y histogram projection, false–line exclusion, text line separation and also text line region recovery. The projection profile analysis is used for automatically locating starting and ending points for each line of the handwritten document. A* path–planning algorithm [47] is performed to determine the path that separates each pair of lines; then, each line is segmented (see figure 3.2). In addition, *deslanting technique* applied for the removal of cursive and sloped handwriting styles from the line images [48]. Among the latest works in word segmentation, the use of scale space word partitioning algorithm [49] provides easy, fast and good results in the process. It's importance is scale selection, that is, finding the optimum scale at which blobs correspond to words. This is done by finding the maximum over scale of the extent or area of the blobs. Finally, this process terminates after it saves each image file containing only that one binarized word as PNG format.

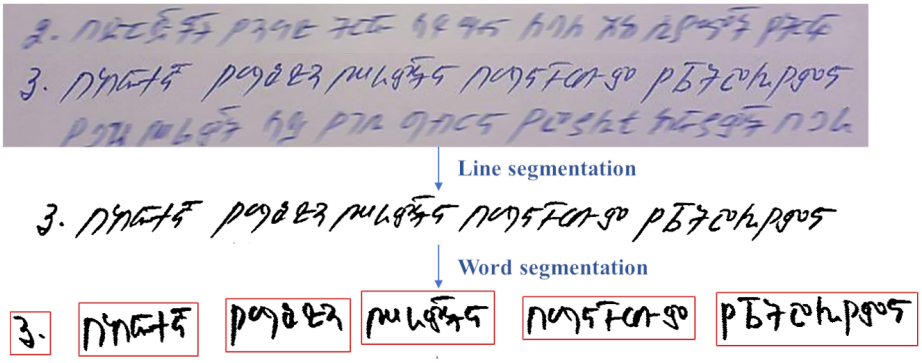


FIGURE 3.2: Text image segmentation

3.2.4 Word Image Labeling

Word image labeling process refers to the way by which every word image is mapped to its content in digital form. The word image filename and other basic details along with the ground truth used to be recorded row-by-row in a text file. These information are single-space separated and would be used later in the data pre-processing stage.

Figure 3.3 illustrates the general overview of how the dataset for handwritten Amharic words (HAW-DB) is organized. The word image's filename is in the first column, a status flag (ok/err) of the file is in the second, a threshold gray level is next, a pair of integers indicating an initial point or x, y–coordinates for the word's bonding box in a line image, also width and height of the word image are put into the subsequent columns, and lastly a grammatical form and the label are the final columns.

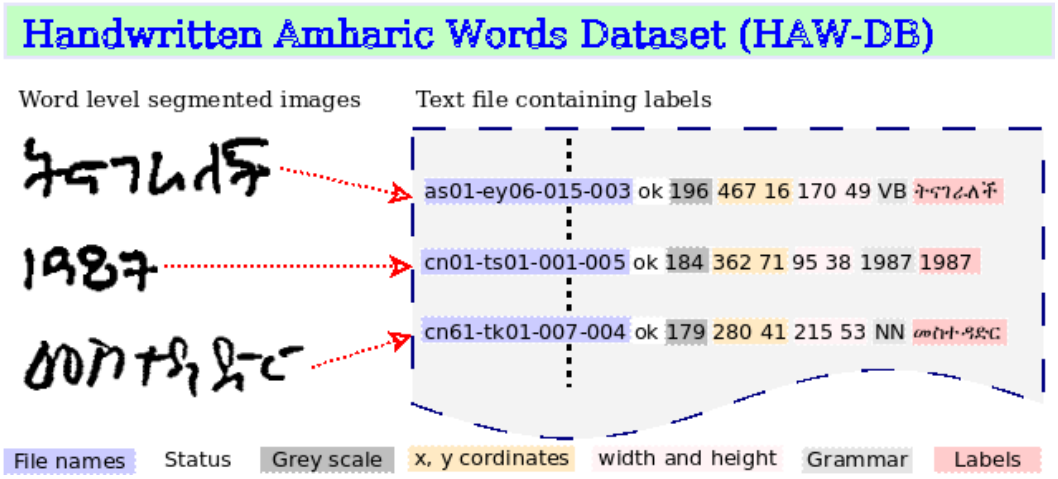


FIGURE 3.3: Illustration of word images and labels

3.3 Data Augmentation

The dataset contains 36,192 handwritten Amharic word images after augmentation. Separate pre-processing steps and model evaluations are followed during the comparison experiments on the two data. From the total 36,192 word-images in the dataset, 12,064 are the original handwritten word images while the remaining 24,128 are augmented ones generated by multiple random combination of augmentation functions (see figure 3.4) such as: rotation, shifting, shrinking, expanding, degrading, or altering Gaussian noises. Afterwards, we have randomly split the dataset with approximately 80% for training, 10% as validation and remaining 10% for testing.

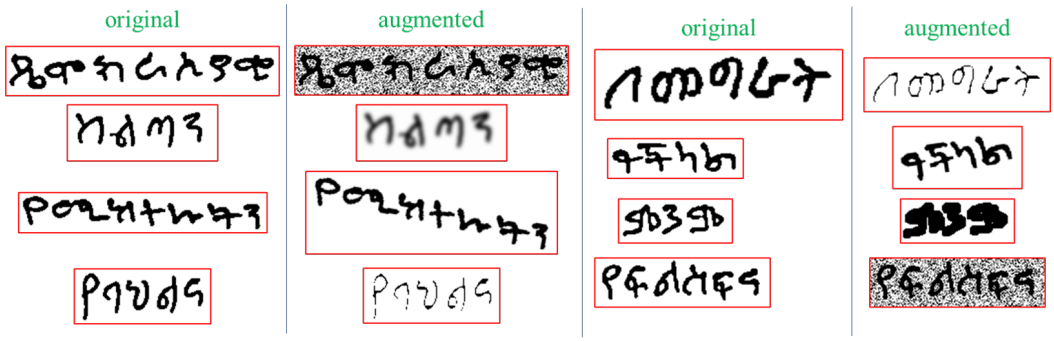


FIGURE 3.4: Samples of data augmentation

3.4 Data Pre–processing

Data pre-processing early stage was a preprocessing of the word-images. We better began from standardizing the format of the input data before proceeding to the model implementation in Keras with TensorFlow backend.

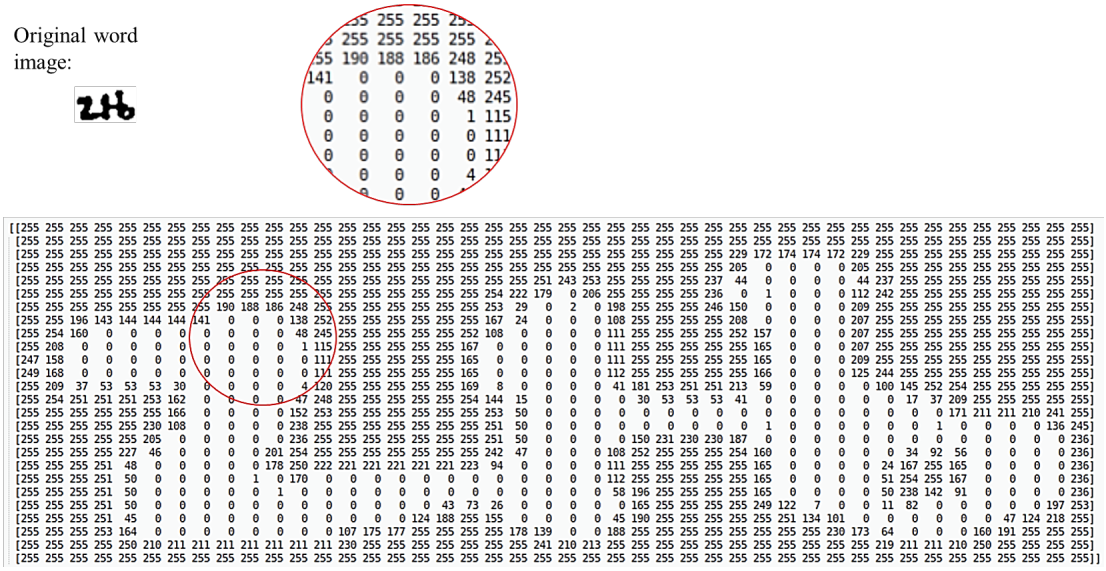


FIGURE 3.5: An example of gray-level handwritten Amharic word-image with its associated tensor for a word "ጊዜ"

Gray-scale image has channel one. And, we established the same scale for all images to allow the design of the system architecture knowing at all times the dimensions of the data flow through the different layers. This has been carried out by giving all of them a height of 64 pixels and adding padding until reaching a width of 256 pixels, maintaining their aspect ratio. However, the system could be designed for variable size images, but it would complicate the code without adding value to the project.

After image scaling, the gray-scale tones have been inverted, by transforming their associated tensor in the process. Thus, we achieved that the pixels corresponding to written parts of the image are represented with high values within the tensor.

Next, we normalized the word-images tensor so that every elements in it inclusively lies between 0 and 1. Then, transform every input sequences to Numpy array using `numpy.asarray` which is in-built function of Python Numpy library.

Appendix B, table B.1 shows HAW-DB's total number of unique Amharic alphabets, numbers and punctuation marks. Finally, labels are encoded and post-padded to same lengths. This label is denoted by `Y` and the input images by `X` in the algorithm 1.

Algorithm 1: Transforming to `numpy` array, and padding labels

$X \leftarrow images$ /* images are preprocessed word-images */
$Y \leftarrow labels$
$C \leftarrow length(char\_list)$ /* total number of characters */
/* finding the inputs' and labels' maximum length */
$X_{maxlen} \leftarrow$ **getMaxLength**(X_{length})
$Y_{maxlen} \leftarrow$ **getMaxLength**(Y_{length})
/* converting the inputs to numpy arrays */
$X \leftarrow$ **numpy.asarray**(X)
$X_{len} \leftarrow$ **numpy.asarray**(X_{maxlen})
$Y_{len} \leftarrow$ **numpy.asarray**(Y_{maxlen})
/* post-padding the labels */
$Y_{padded} \leftarrow$ **pad_sequences**$(Y, maxlen = Y_{maxlen}, padding = post, value = C)$

For instance, scaling to size of 4×8, inverting and normalizing the word-image depicted in figure 3.5 results as shown in figure 3.6.

```
[[[0.        ] [0.        ] [0.        ] [0.        ] [0.83137255] [0.        ] [0.        ] [0.        ]]
 [[1.        ] [1.        ] [0.29019608] [0.        ] [1.        ] [0.        ] [0.        ] [0.        ]]
 [[0.        ] [0.5372549 ] [0.        ] [0.51764706] [1.        ] [1.        ] [0.        ] [0.        ]]
 [[0.12941176] [1.        ] [0.03137255] [0.16078431] [0.41568627] [0.70196078] [0.        ] [0.        ]]]
```

FIGURE 3.6: Conversion of the word-image to shape (4, 8, 1) and normalization

In figure 3.6, the right two columns are filled with zeros to reach the pre-specified width (this case, 8). Eventually, when transforming it to the numpy array format, produced as it is depicted in figure 3.7.

```
array(
   [[[0.        ], [0.        ], [0.        ], [0.        ], [0.83137255], [0.        ], [0.        ], [0.        ]],
    [[1.        ], [1.        ], [0.29019608], [0.        ], [1.        ], [0.        ], [0.        ], [0.        ]],
    [[0.        ], [0.5372549 ], [0.        ], [0.51764706], [1.        ], [1.        ], [0.        ], [0.        ]],
    [[0.12941176], [1.        ], [0.03137255], [0.16078431], [0.41568627], [0.70196078], [0.        ], [0.        ]]])
```

FIGURE 3.7: Conversion of the normalized image to numpy array

The word label " ገዘ" would be encoded as $[206, 169]$ and let the maximum label length is three in words' set, then post-padded sequence would become $[206, 169, 300]$ by appending 300 for making all labels to the same length in that domain.

Chapter 4. System Design and Architecture

In the previous chapter, we have seen procedures on how the dataset is prepared. As output for the pre-processing and input for the AWR system, we have the dataset consist of a set of images, with their corresponding transcripts, in the standard format. This chapter provides a special emphasis has been placed on the main system layers: CNN, LSTM, GRU and CTC that make up the deep learning architecture.

4.1 Proposed End-to-End AWR Model

In procedure after the data preprocessing, the proposed HTR model for handwritten Amharic word recognition consists of a popular variation of feedforward neural networks (FNNs): convolutional NN (CNN) and recurrent NN (RNN) layers, and a final connectionist temporal classification (CTC) layer.

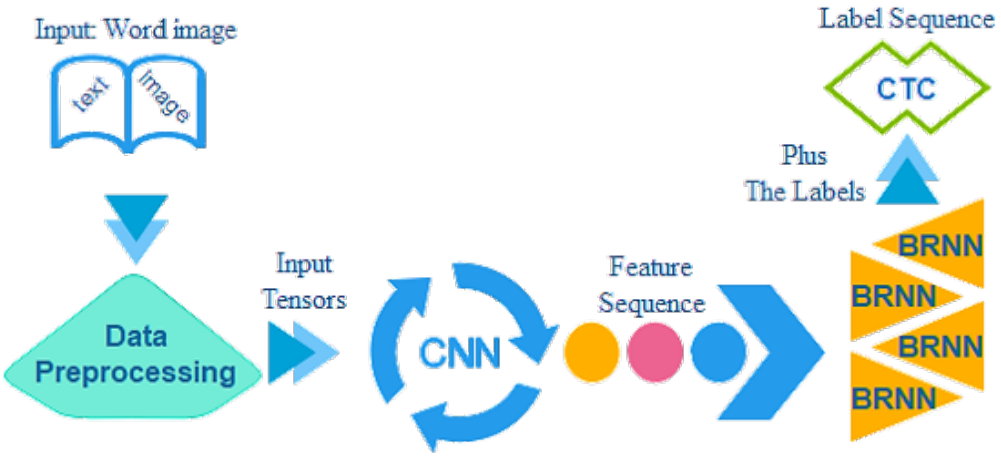


FIGURE 4.1: The end-to-end system components

As an overall diagram, the end-to-end system components of our model are shown in Figure 4.1. In general, the network architecture can be grouped in to three parts. The first is the CNN part, which is used to extract features from the word images. The second component is a pair of bi-directional recurrent nets (BRNN).

The BRNN part is used for sequence modeling by taking the features as input that are generated by using the CNN component. The last essential component is the CTC algorithm which is used for decoding the input feature sequence into the correct order of characters in the predicted labels.

4.1.1 Bayesian Optimization of Hyper-parameters

The Bayesian hyper-parameter tuning algorithm required to set hyper-parameters in our neural network (NN). In other words, this helps us to find a vector of hyper-parameters that works well. For instance: the number of layers, the number of filters, the number hidden units, learning rates, dropout rates and the so on. This algorithm works better than Naive grid search and sampling random combinations of values for hyper-parameters when it used to predict regions of hyper-parameter space that might yield best results. It computes the probability how well a new combination will do and models the uncertainty of that prediction. It has optimized explorations while it is predicting the result with a new hyper-parameter settings from knowing the different ones results so far (see algorithm 2, also appendix C).

Algorithm 2 shows invoking Bayesian optimization. This algorithm also requires to fix the total number of iterations and the number of random points. The random search guide the optimization process to begin on which iterations first time. Then, it creates models based on the points for each iteration and selects the more optimized one.

Algorithm 2: Bayesian hyper-parameters optimization algorithm

$inputShape \leftarrow trainImages$.**shape[**$1:$**]** `/* e.g. (32,128,1) */`
$numberOfClasses \leftarrow$ **len(**$charList$**)** $+ 1$ `/* characters and null */`
$maxEpoches \leftarrow 100$ `/* initializing epoches */`
$earlyStopEpoches \leftarrow 10$
$learningRateEpoches \leftarrow 5$

`/* parameters that must be saved for each iteration */`
$listEarlyStopEpoches \leftarrow \{\}$
$listValidationLosses \leftarrow \{\}$
$listSavedModelNames \leftarrow \{\}$

`/* begin the optimizer with hyperparameter bounds */`
$opt \leftarrow$ **BayesianOptimization**$(function \leftarrow model,$
$parameterBounds \leftarrow \{numCNN(2:12), numFilters(32:512),$
$numUnits(128:1024), kernelSize(2:4), batchSize(4:32),$
$rnnInputLen(15:63), dropout(0.05:05)\})$
opt.**maximize(**$initialPoints \leftarrow randomPoints, numIterations \leftarrow numIter$**)**

`/* displaying the best result from the set */`
print($"The\ best\ result:", opt.max$**)**

`/* saving the parameters for each iteration */`
for res **in** $opt.result$**:** `/* opt.result is a list parameters */`
$dataFrameForEpoches \leftarrow listEarlyStopEpoches\,[id]$
$dataFrameForLosses \leftarrow listValidationLosses\,[id]$
$dataFrameForModels \leftarrow listSavedModelNames\,[id]$
$dataFrameForBayesErrors \leftarrow res\,['target']$

4.1.2 CNN Based Feature Extraction

Convolutional Neural Networks (CNNs) are multilayered trainable Neural Networks architectures developed typically for feature extraction tasks. They are well known for their high performance in extracting relevant feature maps from the image tensors.

Figure 4.2 illustrates the custom CNN layers. Each of these layers, consist the types of operations such as: convolutional, batch normalization and pooling layers. The *convolutional layer*, which are major components of the CNNs, each has of a number of kernel matrices that perform convolution on their input and produce an output matrix of features where a bias value is added. The learning procedures aim to train the kernel weights and biases as shared neuron connection weights.

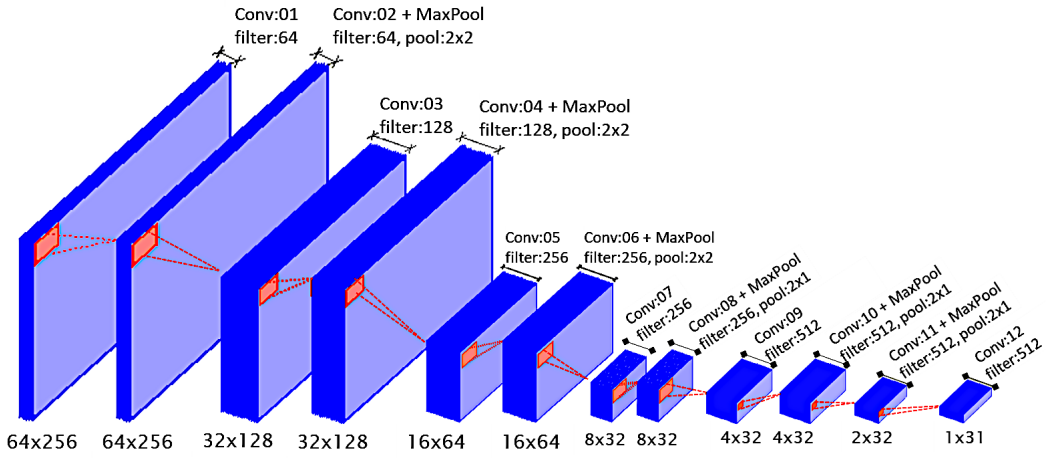


FIGURE 4.2: Feature extraction components

Succinctly, the pooling layer makes a subsampling to the input tensor combing neighboring elements, which reduces the dimensionality. Variants of pooling include: average-pooling, max-pooling, etc. We chose the max-pooling, which is a common pooling layer that takes the maximum among the local neighborhoods.

Batch normalization is the another layer, which used for normalizing the input layer and also the hidden layers by shifting their mean and scaling of the activation. Then because of normalizing the input value and scaling and shifting effects done by this additional layer in deep neural networks, the network can use higher learning rate without vanishing or exploding gradients.

The extracted features from the CNN component are given as an input for the BRNN. Table 4.1 shows all layers of our custom architecture which we are proceeding to the details as follows.

TABLE 4.1: AWR model architecture
Abbreviations: Conv (convolutional layer), MaxPool (max pooling layer) and BN (batch normalization layer)

Type	**Description**	**Output size**
Input	Gray-level word-image	$64 \times 256 \times 1$
Conv1 + ReLU + BN1	kernel 3×3	$64 \times 256 \times 64$
Conv2 + ReLU + MaxPool1	kernel 3×3, pool 2×2	$32 \times 128 \times 64$
Conv3 + ReLU + BN2	kernel 3×3	$32 \times 128 \times 128$
Conv4 + ReLU + MaxPool2	kernel 3×3, pool 2×2	$16 \times 64 \times 128$
Conv5 + ReLU + BN3	kernel 3×3	$16 \times 64 \times 256$
Conv6 + ReLU + MaxPool3	kernel 3×3, pool 2×2	$8 \times 32 \times 256$
Conv7 + ReLU + BN4	kernel 3×3	$8 \times 32 \times 256$
Conv8 + ReLU + MaxPool4	kernel 3×3, pool 2×1	$4 \times 32 \times 256$
Conv9 + ReLU + BN5	kernel 3×3	$4 \times 32 \times 512$
Conv10 + ReLU + BN6 + MaxPool5	kernel 3×3, pool 2×1	$2 \times 32 \times 512$
Conv11 + ReLU + BN7	kernel 3×3	$2 \times 32 \times 512$
Conv12 + ReLU	kernel 2×2	$1 \times 31 \times 512$
Map to sequence function	remove dimension	31×512
BGRU (or BLSTM) + dropout	each with 512 hidden cells	31×1024
BGRU (or BLSTM) + dropout	each with 512 hidden cells	31×1024
Dense layer + softmax (301 classes)	project onto 301 characters	31×301
CTC	decode or loss calculation	≤ 31

We have modified the CNN layers in such a way so as to finally result in a two-dimensional sequence, so an height of 1, a width of 31 and 512 features per element. We have a stack of 12 convolutional layers that map the input image of size 64×256 onto a sequence of length 31 with 512 features per element. For the two-layered BGRU (or BLSTM), the CNN layers gives the mapping onto (vector-valued) similar one-dimensional RNN input sequence. The output of the CNN layers is squeezed along the width dimension. This is because, the CTC layer needs a one-dimensional sequence. However, each sequence element itself is a vector that consists of the corresponding character probabilities. Afterwards, two layers of BGRU each with hidden unit size of 512 take place and train two GRU on the input sequence at a time. Then, linear projection is applied to map the output of the BGRU onto the character distribution for each time-step, and we have set a total of 300 character sequences and one extra character for representing the blank. At last, the CTC decoder outputs a sequence (a word) which can have 31 characters at most.

4.1.3 BRNN Based Sequence Modeling

The bidirectional networks of the RNN cells allow to consider the opposite direction, from left to right as well as from right to left. Thus, BRNN, which is a full gradient version of RNN, outperform unidirectional ones and uses older features (through forward states) and the newer (by the backward) for a particular time frame.

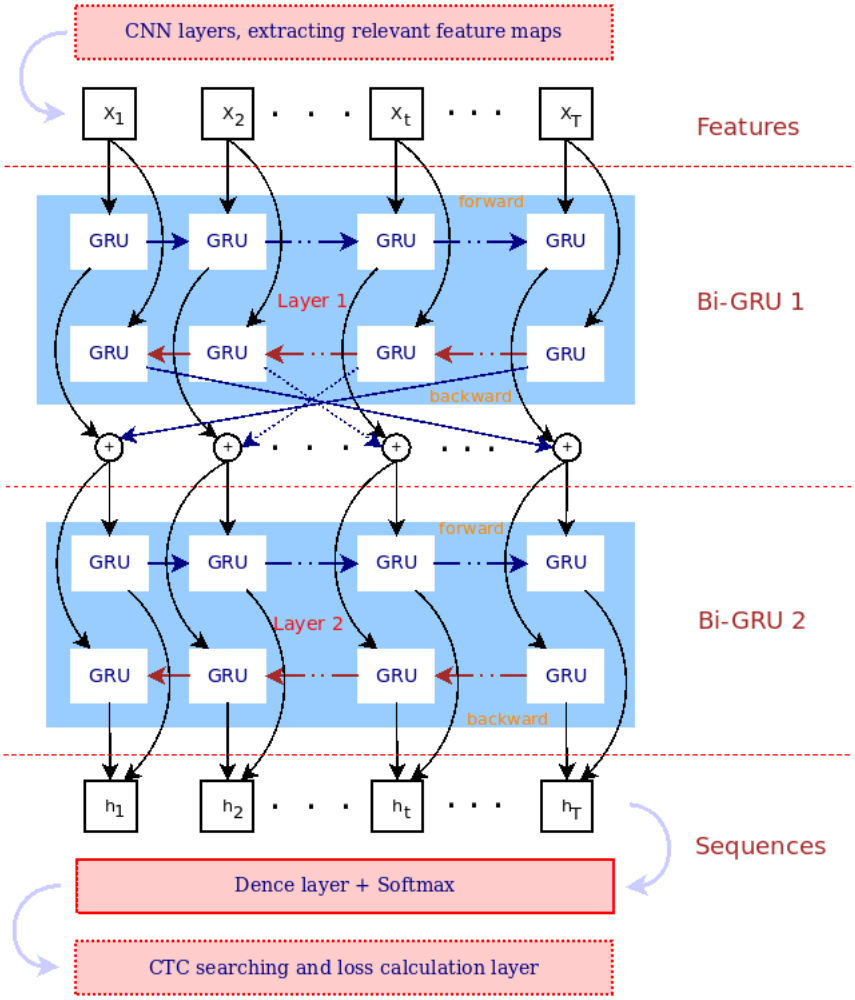


FIGURE 4.3: Gated-recurrent units

We have experimentally compared two of the BRNN frameworks abundantly used in the state-of-the-art of handwriting recognition, specifically: bidirectional gated-recurrent units (BGRU) and the BLSTM based architectures with the same size of hidden unit and number of layers. Then, it is much faster learning algorithm and better in terms of accuracy when using BGRU, since our dataset is not very large. Hence, we chose a BRNN with BGRU cells. These cells can reset all long-temporal information using internal gates and parameters are able to be optimized. BGRU is also often chosen over BLSTM because hidden states are totally exposed and so that they are easier to interpret.

4.1.4 CTC Based Transcription

For an activation function, softmax is adopted with slightly better results than other candidates. And, for conversion of the softmax layer output into correct label sequence we use the CTC algorithm using greedy search as well as beam search techniques applied for comparison. Before, we already mentioned that an output sequence allowing BGRU (or BLSTM) to be used for CTC.

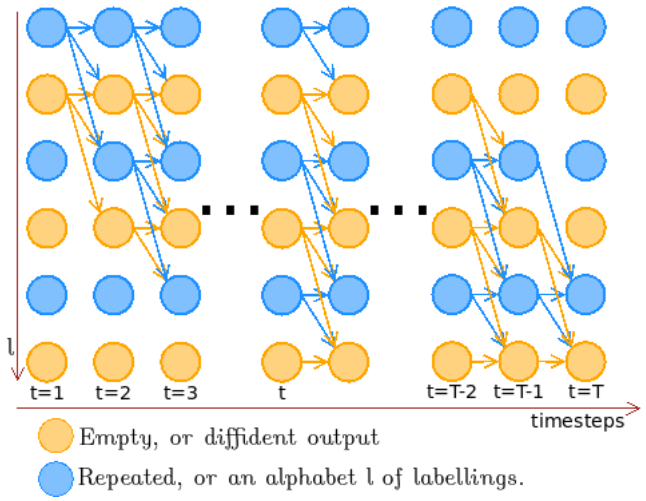


FIGURE 4.4: CTC loss function: sequence of labellings component

We have used CTC networks as a loss function, and the model is optimized using Adamax optimizer with its default learning rate and momentum. We trained for 100 epochs and use the batch size of 10. CTC typically aligns each label prediction with the corresponding part of the sequence. CTC implicitly models the correct inter-label dependencies and hierarchical CTC is where the labellings at one level (e.g. letters) become inputs for the labellings at the next (e.g. words).

4.2 Standard CNN Layers Integration

We have used different CNN architectures to see their effect in obtaining discriminating feature for the input word images. From the standard CNN architectural models we have demonstrated the VGG, ResNet, DenseNet and EfficientNet as a backbone of our feature extraction unit. We compared the result of our custom CNN with the other standard CNN architectures. We have modified the standard architectural model by removing the top layers based on the network pattern and reshaped it to $H \times 32 \times C$ feature maps because we want the input feature length to be 32. We generate sequential features by concatenating the column wise features $(H \times C)$ of all the feature maps which gives a sequential feature of size $32 \times (H \times C)$.

We assumed the standard CNNs each of which joined with the custom model (above $MaxPool4$) as *Model A* through *E* as shown in table 4.2. Thus, we can have a fair comparison by keeping the setup of the input shape, the two-layers BGRU and the CTC loss function architecture the same.

TABLE 4.2: The standard CNNs chopped off beyond the given layers for (32,128,1) input shape.

Id	Standard CNNs	Truncated beyond and used layers up to
A	VGG19	$10^{th}\ layers$ or $block3\_conv4$
B	ResNet152V2	$29^{th}\ layers$ or $conv2\_block3\_preact\_relu\ (Activaton)$
C	DenceNet121	$50^{th}\ layers$ or $pool2\_conv\ (Conv2D)$
D	DenseNet201	$50^{th}\ layers$ or $pool2\_conv\ (Conv2D)$
E	EfficientNet-B7	$157^{th}\ layers$ or $swish\_30$

The standard CNNs need to be chopped off beyond where the layers vary depending on the input shape and to be joined from $MaxPool4$ layer of the custom model. Again, table 4.3 shows the standard CNNs truncation and assumption of *Model A* through *Model E*.

TABLE 4.3: The standard CNNs chopped off beyond the given layers for (64,256,1) input shape.

Id	Standard CNNs	Truncated beyond and used layers up to
A	VGG19	$15^{th}\ layers$ or $block4\_conv4$
B	ResNet152V2	$119^{th}\ layers$ or $conv3\_block8\_preact\_relu\ (Activaton)$
C	DenceNet121	$139^{th}\ layers$ or $pool3\_conv\ (Conv2D)$
D	DenseNet201	$139^{th}\ layers$ or $pool3\_conv\ (Conv2D)$
E	EfficientNet-B7	$261^{th}\ layers$ or $swish\_51$

4.3 Gated CNN Layers Integration

Gated CNN (GCNN) means that it controls the path via which information flows in the network. The GCNN only requires a small number of epochs to reach the same accuracy. Thus, integrating GCNNs to the model yield better computational efficiency.

TABLE 4.4: Integration of GCNN to AWR model
Abbreviations: GatedConv (gated CNN layer), Conv (CNN layer), MaxPool (max pooling layer) and BN (batch normalization layer)

Type	**Description**	**Output size**
Input	Gray-level word-image	$64 \times 256 \times 1$
Conv + ReLU + BN	kernel 3×3	$64 \times 256 \times 64$
GatedConv + ReLU + MaxPool	kernel 3×3, pool 2×2	$32 \times 128 \times 64$
Conv + ReLU + BN	kernel 3×3	$32 \times 128 \times 128$
GatedConv + ReLU + MaxPool	kernel 3×3, pool 2×2	$16 \times 64 \times 128$
Conv + ReLU + BN	kernel 3×3	$16 \times 64 \times 256$
GatedConv + ReLU + MaxPool	kernel 3×3, pool 2×2	$8 \times 32 \times 256$
GatedConv + ReLU + BN	kernel 3×3	$8 \times 32 \times 256$
GatedConv + ReLU + MaxPool	kernel 3×3, pool 2×1	$4 \times 32 \times 256$
Conv + ReLU + BN	kernel 3×3	$4 \times 32 \times 512$
GatedConv + ReLU + BN + MaxPool	kernel 3×3, pool 2×1	$2 \times 32 \times 512$
GatedConv + ReLU + BN	kernel 3×3	$2 \times 32 \times 512$
Conv + ReLU	kernel 2×2	$1 \times 31 \times 512$
Map to sequence function	remove dimension	31×512
BGRU (or BLSTM) + dropout	each with 512 hidden cells	31×1024
BGRU (or BLSTM) + dropout	each with 512 hidden cells	31×1024
Dense layer + softmax (301 classes)	project onto 301 characters	31×301
CTC	decode or loss calculation	≤ 31

Chapter 5. Discussion of the Experimental Results

This chapter presented the results by using the proposed handwritten Amharic word recognition (AWR) system. Multiple experiments are conducted to analyze the effect of different parts of the system or aspects of the dataset. We used to execute all of the experiments on Google Co-laboratory (COLAB) with GPU (Tesla T4, 8GB) and 12GB of RAM.

5.1 Optimization Results

The test set, which is 10% randomly selected of the total dataset, is defined to be able to compare different models under the same conditions. The COLAB provided free access to the GPU as long as 24 hours per an email so that if the execution surpassed the time allowed within a day, it used to continue with out it. Executing the training without GPU can take up very much longer time. Therefore, we ran the optimizer algorithm only one after the other for each hyperparameter. Also, we have omitted some experiments that gave worse results from the report. Table 5.1 shows the result obtained using the best performing custom-model on our dataset, HAW-DB. This is consisting of twelve CNN layers from top to bottom: each two 64 and 128, then each four 256 and 512 filters with 3×3 kernel size except the last layer which is 2×2. ReLU activation is applied following every CNN. Adamax optimizer with learning rate of $1e - 3$ (i.e. 0.001). Finally, the two-layer BGRU with 512 units and two dropouts each with 0.5 rates.

TABLE 5.1: The best performing model results on HAW-DB.

10% test set	CER (%)	WER (%)	Jaro (%)	Ratio (%)
1,200	2.674	8.833	98.28	97.46

We discovered the best performance of our custom-model by using two techniques based on the nature of an hyperparameter that we are going to setup as optimal as possible. One is experimentally and two is using Bayesian optimization algorithm, which we mentioned in the earlier chapter.

Experimentally, we could optimize input images size and RNN input length. Table 5.2 shows the matrix of comparison on both accuracy and loss recorded by varying each over the other hyperparameter.

TABLE 5.2: The effect of image size and RNN input length on the accuracy/loss over the same network

RNN input length	Image Size			
	16×64	32×128	48×192	64×256
15	76.917 / 1.220	84.167 / 0.834	81.417 / 1.002	83.000 / 0.810
31	81.917 / 0.903	84.250 / 0.697	88.750 / 0.582	88.583 / 0.557
47	82.167 / 0.899	84.167 / 0.754	85.917 / 0.598	86.167 / 0.746
63	80.000 / 0.921	85.833 / 0.680	73.333 / 1.120	85.333 / 0.676

At this experimental analysis, Bayesian hyperparameter-tuning algorithm by itself finally resulted in the model, which possibly is of the least validation loss. Thus, we investigated the optimal value to a given hyper-parameter by using the optimization algorithm for finding out the convenient values to the hyper-parameters in order to configure our custom model architecture. Via which, we could at the end organize as the best performing setting for our specific problem domain, input data, and the model itself.

5.2 Data Augmentation Results

The augmentation of data has enhanced the performance of the models by increasing the data variation in training data. Table 5.3 shows the results of addition of 2-times augmented word-images on the two different input sizes. Here, it shows much better performance because the increased number of training samples, enabling the models to learn more robust features by preventing over-fitting and higher variance within data and symbols.

TABLE 5.3: Result of an addition of 2-times augmented HAW-DB

Input shape	Test accuracy (%)	Loss	CER (%)	WER (%)
(32, 128, 1)	91.667	0.516	2.171	7.250
(64, 256, 1)	92.167	0.495	1.819	6.833

5.3 10 – Fold Cross – Validation

This method has 10 independent evaluations used to be carried out, and with different partitions for training and validation. Figure 5.1 illustrates that in each evaluation, the system is trained with 90% of dataset samples and validated with the remaining 10%. Therefore, it made training and validation datasets definitely separated from each other. After the dataset has been taken for 10 fold cross-validation, we got the average test accuracy of $87.14\% (\pm 0.86\%)$ and loss of $0.6279 (\pm 0.0515)$ by averaging each accuracy and loss over the 10-fold cross validation.

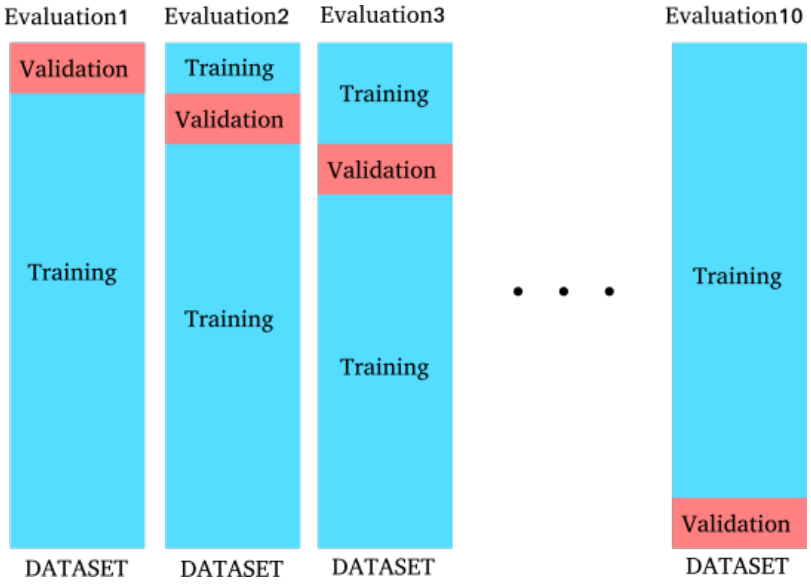


FIGURE 5.1: Representation of the dataset in 10 different evaluations.

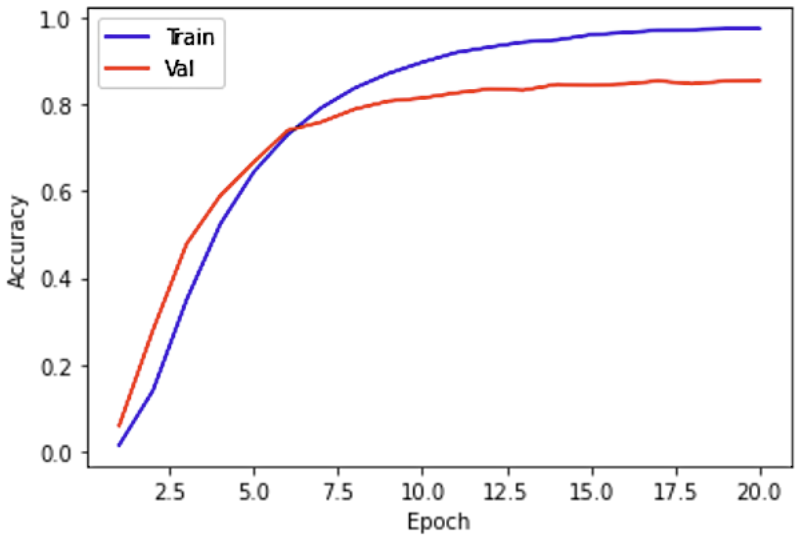


FIGURE 5.2: Average of 10-fold cross-validation of model accuracy

The graph depicted in figure 5.2 is given by averaging each accuracy in fold and by tracking their accuracy while the model training and validation, over only few epoch.

5.4 Standard CNN Layers Integration Results

Depending on the experimental results in table 5.4, *Model D* which is of DenseNet201 architecture showed the great recognition performance on 64×256 input size when compared to the rest models. This is due to its network architecture design which has to be suitable for the input data. The selection of an optimal size of the input word image has a profound effect on the performance of the handwritten text recognition task. When we use optimal input word image sizes to shallow network architectures extract more relevant feature representation of words.

TABLE 5.4: Result of the standard CNNs integration

Input shape	Model	Test accuracy (%)	Loss	CER (%)	WER (%)
(32, 128, 1)	A	86.667	0.669	3.547	11.083
	B	85.750	0.712	3.910	12.333
	C	86.833	0.648	3.519	11.583
	D	84.583	0.723	3.564	12.416
	E	83.667	0.790	4.080	13.917
(64, 256, 1)	A	82.167	0.839	4.610	14.750
	B	87.500	0.625	3.293	11.000
	C	87.083	0.645	2.996	10.333
	D	89.417	0.564	2.816	9.167
	E	84.333	0.826	4.151	13.083

Chapter 6. Conclusion and Recommendation

This chapter summarizes the research within the first section, Conclusion. Also, the section assesses the outcome of the thesis in terms of the initial set of objectives. And, it outlines suggestions to be done for the progress and possible future works within the second section, Recommendation.

6.1 Conclusion

This research presented the development of an handwritten Amharic word database after collecting a total of $12,064$ words from 60 different people. It also presented a CNN-BGRU-CTC based HTR system implemented with Tensoflow, Deep learning library on Google Co-laboratory (COLAB), with training using GPU, that currently has the highest popularity and percentage of use. During the design of the system, different models have been implemented and tested, to make fine adjustments of the hyper-parameters with the help of Bayesian optimization algorithm. This hyper-parameter tuning technique cross-validated the number of CNN layers, filters, kernel size, dropout rate, batch size and RNN units. We have used six quantitative measures to evaluate performance and these are: loss, accuracy, CER, WER, Jaro and Ratio.

Out of a total of four major experiments that we have conducted in this study, one is that we used 10-fold cross validation for the entire data. On this, the model scored an average accuracy of $87.14\% \, (\pm 0.86\%)$ and average loss of $0.6279 \, (\pm 0.0515)$. In addition, while using G-CNN in the place of CNN reduces the execution time with comparative performance.

Next, we compared the original dataset and with addition of its augmented version. To do this experiment, first we have generated additional $24,128$ word images "by dirtying" the original (i.e. by applying randomly from: adding some kind of noise to the images, by shifting or by blurring, etc.) and second, we used the same custom model algorithm trained with the augmented dataset. Thus, we found 69.24% and 77.35% improvement of the CER and WER respectively on the same test set to the experiment done on the original dataset.

As described in the earlier chapter, we cannot compare on different dataset, data-size or of different languages. Instead, we did integration of pre-designed standard architectural models as part of our custom model held in as another experiment. At last, we found out DenseNet201 is outperforming with 2.816 WER and 9.167 CER.

6.2 Recommendation

Even though, this research contributed a lot for the attempt to design and develop handwritten Amharic word recognition system, we believe improvements can be done by applying transformer into it. Thus, by placing attention layer immediately after the CNNs (feature extractor component) and by applying language models: BERT, ELMo, etc. And, we recommend to realize transfer learning of the model, training that's already trained on other datasets shows promising result.

This research also made an effort to alter the challenge especially in the experiments conducted on word-image augmentation; nevertheless, the system's input (another dataset) in the future can be prepared by collecting a number of digital images of aged historical handwritten documents (also called "Brana").

Moreover, sometimes handwritten text won't be easily recognizable due to human mistakes, broken characters or addition of noise. Also, historical documents usually lose some features of characters due to aging and potentially contain part of text degradation. Therefore, integrating the state-of-the-art neural networks to correct spellings of words or some feature recovery techniques should be studied in future.

References

[1] Y. Assabie and J. Bigün. "Offline handwritten Amharic word recognition". In: *Pattern Recognit. Lett.* 32 (2011), pp. 1089–1099.

[2] J. A. Sánchez and U. Pal. "Handwritten Text Recognition for Bengali". In: *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 2016, pp. 542–547. DOI: 10.1109/ICFHR.2016.0105.

[3] Reeve Ingle, Yasuhisa Fujii, Thomas Deselaers, et al. "A Scalable Handwritten Text Recognition System". In: *ICDAR*. 2019. URL: https://arxiv.org/abs/1904.09150.

[4] J. Pradeep, E. Srinivasan, and S. Himavathi. "Neural network based handwritten character recognition system without feature extraction". In: *2011 International Conference on Computer, Communication and Electrical Technology (ICCCET)*. 2011, pp. 40–44.

[5] Maria Bulakh and Denis Nosnitsin. "An Old Amharic poem from northern Ethiopia: one more text on condemning glory". In: *Bulletin of the School of Oriental and African Studies* 82.2 (2019), pp. 315350. DOI: 10.1017/S0041977X1900034X.

[6] Tom M. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN: 978-0-07-042807-2.

[7] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 282289. ISBN: 1558607781.

[8] Leonard E. Baum and Ted Petrie. "Statistical Inference for Probabilistic Functions of Finite State Markov Chains". In: *Ann. Math. Statist.* 37.6 (Dec. 1966), pp. 1554–1563. DOI: 10.1214/aoms/1177699147. URL: https://doi.org/10.1214/aoms/1177699147.

[9] Jingtao Fan, Lu Fang, Jiamin Wu, et al. "From Brain Science to Artificial Intelligence". In: *Engineering* 6.3 (2020), pp. 248 –252. ISSN: 2095-8099. DOI: https://doi.org/10.1016/j.eng.2019.11.012. URL: http://www.sciencedirect.com/science/article/pii/S2095809920300035.

[10] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2018. arXiv: 1603.07285 [stat.ML].

[11] J. Elman. "Finding Structure in Time". In: *Cognitive Science* 14.2 (1990), pp. 179–211.

[12] Alex Graves and Jürgen Schmidhuber. "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by D. Koller, D. Schuurmans, Y. Bengio, et al. Vol. 21. Curran Associates, Inc., 2009, pp. 545–552.

[13] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85 –117. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2014.09.003.

[14] Mike Schuster and Kuldip Paliwal. "Bidirectional recurrent neural networks". In: *Signal Processing, IEEE Transactions on* 45 (Dec. 1997), pp. 2673 –2681. DOI: 10.1109/78.650093.

[15] Asifullah Khan, Anabia Sohail, Umme Zahoora, et al. "A Survey of the Recent Architectures of Deep Convolutional Neural Networks". In: *Artificial Intelligence Review* 53 (Dec. 2020). DOI: 10.1007/s10462-020-09825-6.

[16] K. Cho, B. Van Merriënboer, D. Bahdanau, et al. "On the properties of neural machine translation: Encoder-decoder approaches". In: *arXiv* (2014). DOI: arXiv:1409.1259.

[17] Alex Graves, Santiago Fernández, Faustino Gomez, et al. "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks". In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2006, pp. 369376. ISBN: 1595933832. DOI: 10.1145/1143844.1143891. URL: https://doi.org/10.1145/1143844.1143891.

[18] Yoshua Bengio, Yann LeCun, Craig Nohl, et al. "LeRec: A NN/HMM Hybrid for On-Line Handwriting Recognition". In: *Neural Computation* 7.6 (1995), pp. 1289–1303. DOI: 10.1162/neco.1995.7.6.1289. eprint: https://doi.org/10.1162/neco.1995.7.6.1289. URL: https://doi.org/10.1162/neco.1995.7.6.1289.

[19] Alex Graves and Jürgen Schmidhuber. "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks". In: *Advances in Neural Information Processing Systems 21*. Ed. by D. Koller, D. Schuurmans, Y. Bengio, et al. Curran Associates, Inc., 2009, pp. 545–552. URL: http://papers.nips.cc/paper/3449-offline-handwriting-recognition-with-multidimensional-recurrent-neural-networks.pdf.

[20] Norhidayu Abdul Hamid and Nilam Nur Amir Sjarif. "Handwritten Recognition Using SVM, KNN and Neural Network". In: *CoRR* abs/1702.00723 (2017). arXiv: 1702.00723. URL: http://arxiv.org/abs/1702.00723.

[21] A. El-Yacoubi, R. Sabourin, C. Y. Suen, et al. "An HMM-Based Approach for Off-Line Unconstrained Handwritten Word Modeling and Recognition". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 21.8 (Aug. 1999), pp. 752760. ISSN: 0162-8828. DOI: 10.1109/34.784288. URL: https://doi.org/10.1109/34.784288.

[22] Alessandro Koerich, Robert Sabourin, and Ching Suen. "Lexicon-Driven HMM Decoding for Large Vocabulary Handwriting Recognition with Multiple Character Models". In: *Document Analysis and Recognition* 6 (Oct. 2003), pp. 126–144.

[23] Théodore Bluche. "Joint Line Segmentation and Transcription for End-to-End Handwritten Paragraph Recognition". In: *CoRR* abs/1604.08352 (2016). arXiv: 1604.08352. URL: http://arxiv.org/abs/1604.08352.

[24] Arindam Chowdhury and Lovekesh Vig. *An Efficient End-to-End Neural Model for Handwritten Text Recognition*. 2018. arXiv: 1807.07965 [cs.CL].

[25] Johannes Michael, Roger Labahn, Tobias Grüning, et al. "Evaluating Sequence-to-Sequence Models for Handwritten Text Recognition". In: *CoRR* abs/1903.07377 (2019). arXiv: 1903.07377. URL: http://arxiv.org/abs/1903.07377.

[26] T. Bluche and R. Messina. "Gated Convolutional Recurrent Neural Networks for Multilingual Handwriting Recognition". In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 01. 2017, pp. 646–651.

[27] Denis Coquenet, Yann Soullard, Clement Chatelain, et al. "Have Convolutions Already Made Recurrence Obsolete for Unconstrained Handwritten Text Recognition?" In: Sept. 2019, pp. 65–70. DOI: 10.1109/ICDARW.2019.40083.

[28] R. R. Ingle, Y. Fujii, T. Deselaers, et al. "A Scalable Handwritten Text Recognition System". In: *2019 International Conference on Document Analysis and Recognition (ICDAR)*. 2019, pp. 17–24.

[29] Mohamed Yousef, Khaled F. Hussain, and Usama S. Mohammed. "Accurate, Data-Efficient, Unconstrained Text Recognition with Convolutional Neural Networks". In: *CoRR* abs/1812.11894 (2018). arXiv: 1812.11894. URL: http://arxiv.org/abs/1812.11894.

[30] Mesay Samuel Gondere, Lars Schmidt-Thieme, Abiot Sinamo Boltena, et al. *Handwritten Amharic Character Recognition Using a Convolutional Neural Network*. 2019. arXiv: 1909.12943 [cs.CV].

[31] F. Abdurahman. "Handwritten Amharic Character Recognition System Using Convolutional Neural Networks". In: *Engineering Sciences* 14 (2019), pp. 71–87.

[32] M. Meshesha and C. Jawahar. "Optical Character Recognition of Amharic Documents". In: *African J. of Inf. and Commun. Technology* 3 (Aug. 2007). DOI: 10.5130/ajict.v3i2.543.

[33] J. Cowell and F. Hussain. "Amharic character recognition using a fast signature based algorithm". In: *Proceedings on Seventh International Conference on Information Visualization, 2003. IV 2003.* 2003, pp. 384–389. DOI: 10.1109/IV.2003.1218014.

[34] B. Belay, Tewodros Habtegebrial, M. Meshesha, et al. "Amharic OCR : An End-to-End Learning". In: *Applied Sciences* 10 (2020), p. 1117.

[35] B. Belay, Tewodros Habtegebrial, M. Liwicki, et al. "Amharic Text Image Recognition: Database, Algorithm, and Analysis". In: *2019 International Conference on Document Analysis and Recognition (ICDAR)* (2019), pp. 1268–1273.

[36] Rajib Ghosh, Chirumavila Vamshi, and Prabhat Kumar. "RNN based online handwritten word recognition in Devanagari and Bengali scripts using horizontal zoning". In: *Pattern Recognition* 92 (2019), pp. 203 –218. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2019.03.030. URL: http://www.sciencedirect.com/science/article/pii/S0031320319301384.

[37] U. Marti and H. Bunke. "The IAM-database: an English sentence database for offline handwriting recognition". In: *International Journal on Document Analysis and Recognition* 5 (2002), pp. 39–46.

[38] Hong-Phuong Tran, Andrew Smith, and Eric Dimla. "Offline Handwritten Text Recognition using Convolutional Recurrent Neural Network". In: Nov. 2019, pp. 51–56. DOI: 10.1109/ACOMP.2019.00015.

[39] Xinfeng Zhang and Kunpeng Yan. "An Algorithm of Bidirectional RNN for Offline Handwritten Chinese Text Recognition". In: *Intelligent Computing Methodologies*. Ed. by De-Shuang Huang, Zhi-Kai Huang, and Abir Hussain. Cham: Springer International Publishing, 2019, pp. 423–431.

[40] José Carlos Aradillas, Juan José Murillo-Fuentes, and Pablo M. Olmos. "Boosting Handwriting Text Recognition in Small Databases with Transfer Learning". In: *CoRR* abs/1804.01527 (2018). arXiv: 1804.01527. URL: http://arxiv.org/abs/1804.01527.

[41] Adeline Granet, Emmanuel Morin, Harold Mouchère, et al. "Transfer Learning for Handwriting Recognition on Historical Documents". In: *7th International Conference on Pattern Recognition Applications and Methods (ICPRAM)*. Madeira, Portugal, 2018. URL: https://hal.archives-ouvertes.fr/hal-01681126.

[42] K. Dutta, P. Krishnan, M. Mathew, et al. "Offline Handwriting Recognition on Devanagari Using a New Benchmark Dataset". In: *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*. 2018, pp. 25–30.

[43] R. Chamchong, W. Gao, and M. D. McDonnell. "Thai Handwritten Recognition on Text Block-Based from Thai Archive Manuscripts". In: *2019 International Conference on Document Analysis and Recognition (ICDAR)*. 2019, pp. 1346–1351.

[44] H. El Abed and V. Margner. "The IFN/ENIT-database - a tool to develop Arabic handwriting recognition systems". In: *2007 9th International Symposium on Signal Processing and Its Applications*. 2007, pp. 1–4.

[45] R. Maalej and M. Kherallah. "Convolutional Neural Network and BLSTM for Offline Arabic Handwriting Recognition". In: *2018 International Arab Conference on Information Technology (ACIT)*. 2018, pp. 1–6.

[46] Gupta Mehul, Patel Ankita, Dave Namrata, et al. "Text-based Image Segmentation Methodology". In: *Procedia Technology* 14 (2014). 2nd International Conference on Innovations in Automation and Mechatronics Engineering, ICIAME 2014, pp. 465–472. ISSN: 2212-0173. DOI: https://doi.org/10.1016/j.protcy.2014.08.059.

[47] O. Surinta, M. Holtkamp, F. Karabaa, et al. "A Path Planning for Line Segmentation of Handwritten Documents". In: *2014 14th International Conference on Frontiers in Handwriting Recognition*. 2014, pp. 175–180. DOI: 10.1109/ICFHR.2014.37.

[48] Alessandro Vinciarelli and Juergen Luettin. "A new normalization technique for cursive handwritten words". In: *Pattern Recognition Letters* 22.9 (2001), pp. 1043 –1050. ISSN: 0167-8655. DOI: https://doi.org/10.1016/S0167-8655(01)00042-3. URL: http://www.sciencedirect.com/science/article/pii/S0167865501000423.

[49] R. Manmatha and J. L. Rothfeder. “A scale space approach for automatically segmenting words from historical handwritten documents”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.8 (2005), pp. 1212–1225. DOI: 10.1109/TPAMI.2005.150.

[50] Peter T. Daniels and William Bright. *The World’s Writing Systems*. New York : Oxford University Press, 1996. Chap. Ethiopic Writing, p. 573. ISBN: 978-0-19-507993-7.

Appendix A. Amharic Writing System

A.1 Alphasyllabary

TABLE A.1: Chart of Amharic fidels [50]

	ä/e [ə]	u	i	a	ē	ə [ɨ], ∅	o	ʷä/ue [ʷə]	ʷi/ui	ʷa/ua	ʷē/uē	ʷə [ʷɨ/ū]
h	ሀ	ሁ	ሂ	ሃ	ሄ	ህ	ሆ					
l	ለ	ሉ	ሊ	ላ	ሌ	ል	ሎ			ሏ		
ḥ	ሐ	ሑ	ሒ	ሓ	ሔ	ሕ	ሖ			ሗ		
m	መ	ሙ	ሚ	ማ	ሜ	ም	ሞ			ሟ		
ś	ሠ	ሡ	ሢ	ሣ	ሤ	ሥ	ሦ			ሧ		
r	ረ	ሩ	ሪ	ራ	ሬ	ር	ሮ			ሯ		
s	ሰ	ሱ	ሲ	ሳ	ሴ	ስ	ሶ			ሷ		
ʃ	ሸ	ሹ	ሺ	ሻ	ሼ	ሽ	ሾ			ሿ		
q	ቀ	ቁ	ቂ	ቃ	ቄ	ቅ	ቆ	ቈ	ቍ	ቋ	ቌ	ቊ
b	በ	ቡ	ቢ	ባ	ቤ	ብ	ቦ			ቧ		
β	ቨ	ቩ	ቪ	ቫ	ቬ	ቭ	ቮ			ቯ		
t	ተ	ቱ	ቲ	ታ	ቴ	ት	ቶ			ቷ		
tʃ	ቸ	ቹ	ቺ	ቻ	ቼ	ች	ቾ			ቿ		

	ä/e [ə]	u	i	a	ē	ə [ɨ], ∅	o	ʷä/ue [ʷə]	ʷi/ui	ʷa/ua	ʷē/uē	ʷə [ʷɨ/ū]
x	ሃ	ሁ	ሂ	ሃ	ሄ	ህ	ሆ	ሏ	ሗ	ኋ	ኄ	ሟ
n	ነ	ኑ	ኒ	ና	ኔ	ን	ኖ			ኗ		
ɲ	ኘ	ኙ	ኚ	ኛ	ኜ	ኝ	ኞ			ኟ		
ʔ	አ	ኡ	ኢ	ኣ	ኤ	እ	ኦ			ኧ		
k	ከ	ኩ	ኪ	ካ	ኬ	ክ	ኮ	ኰ	ኵ	ኳ	ኴ	ኸ
x	ኸ	ኹ	ኺ	ኻ	ኼ	ኽ	ኾ	ዀ	ዅ	ዃ	ዄ	ዅ
w	ወ	ዉ	ዊ	ዋ	ዌ	ው	ዎ					
ʕ	ዐ	ዑ	ዒ	ዓ	ዔ	ዕ	ዖ					
z	ዘ	ዙ	ዚ	ዛ	ዜ	ዝ	ዞ			ዟ		
ʒ	ዠ	ዡ	ዢ	ዣ	ዤ	ዥ	ዦ			ዧ		
j	የ	ዩ	ዪ	ያ	ዬ	ይ	ዮ					
d	ደ	ዱ	ዲ	ዳ	ዴ	ድ	ዶ			ዷ		
dʒ	ጀ	ጁ	ጂ	ጃ	ጄ	ጅ	ጆ			ጇ		
g	ገ	ጉ	ጊ	ጋ	ጌ	ግ	ጎ	ጐ	ጕ	ጓ	ጔ	ጘ
ṭ	ጠ	ጡ	ጢ	ጣ	ጤ	ጥ	ጦ			ጧ		
tʃʼ	ጨ	ጩ	ጪ	ጫ	ጬ	ጭ	ጮ			ጯ		
pʼ	ጰ	ጱ	ጲ	ጳ	ጴ	ጵ	ጶ			ጷ		
ṣ	ጸ	ጹ	ጺ	ጻ	ጼ	ጽ	ጾ			ጿ		
ṣ́	ፀ	ፁ	ፂ	ፃ	ፄ	ፅ	ፆ					
f	ፈ	ፉ	ፊ	ፋ	ፌ	ፍ	ፎ			ፏ		
p	ፐ	ፑ	ፒ	ፓ	ፔ	ፕ	ፖ			ፗ		

A.2 Punctuation and Numerals

Punctuation includes the following:

TABLE A.2: Amharic punctuation

※	section mark
፡	word separator
።	full stop (period)
፣	comma
፤	semicolon
፥	colon
፦	preface colon (introduces speech from a descriptive prefix)
፧	question mark
፨	paragraph separator

The Ethiopic numerals include:

TABLE A.3: Ethiopic numerals

፩	1	፳	20
፪	2	፴	30
፫	3	፵	40
፬	4	፶	50
፭	5	፷	60
፮	6	፸	70
፯	7	፹	80
፰	8	፺	90
፱	9	፻	100
፲	10	፼	10000

N.B. The Ethiopic script used for Amharic is also used for other languages, including Ge'ez, Argobba, Gurage, and Tigre. Ge'ez, which is chiefly a liturgical language, uses only 26 basic letter forms from this table.

Appendix B. CTC Labels Representation

TABLE B.1: Amharic characters encoding in CTC–labels

0	ሀ	38	ረ	76	ቦ	114	ን	152	ዊ	190	ዲ	228	ጫ	266	ፖ
1	ሁ	39	ሩ	77	ቧ	115	ኖ	153	ዋ	191	ዳ	229	ጰ	267	ፕ
2	ሂ	40	ሪ	78	ቨ	116	ኗ	154	ዌ	192	ዴ	230	ጱ	268	!
3	ሃ	41	ራ	79	ቩ	117	ኘ	155	ው	193	ድ	231	ጲ	269	፡-
4	ሄ	42	ሬ	80	ቪ	118	ኙ	156	ዎ	194	ዶ	232	ጳ	270	‹
5	ህ	43	ር	81	ቫ	119	ኚ	157	ዏ	195	ዷ	233	ጴ	271	(
6	ሆ	44	ሮ	82	ቬ	120	ኛ	158	ዐ	196	ጀ	234	ጵ	272	«
7	ለ	45	ሯ	83	ቭ	121	ኜ	159	ዑ	197	ጁ	235	ጶ	273	፥
8	ሉ	46	ሰ	84	ቮ	122	ኝ	160	ዒ	198	ጂ	236	ጷ	274	%
9	ሊ	47	ሱ	85	ቯ	123	ኞ	161	ዓ	199	ጃ	237	ጸ	275	»
10	ላ	48	ሲ	86	ተ	124	ኟ	162	ዔ	200	ጄ	238	ጹ	276	)
11	ሌ	49	ሳ	87	ቱ	125	አ	163	ዕ	201	ጅ	239	ጺ	277	›
12	ል	50	ሴ	88	ቲ	126	ኡ	164	ዖ	202	ጆ	240	ጻ	278	.
13	ሎ	51	ስ	89	ታ	127	ኢ	165	ዘ	203	ጇ	241	ጼ	279	+
14	ሏ	52	ሶ	90	ቴ	128	ኣ	166	ዙ	204	ገ	242	ጽ	280	፣
15	ሐ	53	ሷ	91	ት	129	ኤ	167	ዚ	205	ጉ	243	ጾ	281	-
16	ሑ	54	ሸ	92	ቶ	130	እ	168	ዛ	206	ጊ	244	ጿ	282	¤
17	ሒ	55	ሹ	93	ቷ	131	ኦ	169	ዜ	207	ጋ	245	ፀ	283	/
18	ሓ	56	ሺ	94	ቸ	132	ኧ	170	ዝ	208	ጌ	246	ፁ	284	0
19	ሔ	57	ሻ	95	ቹ	133	ከ	171	ዞ	209	ግ	247	ፂ	285	1
20	ሕ	58	ሼ	96	ቺ	134	ኩ	172	ዟ	210	ጎ	248	ፃ	286	2
21	ሖ	59	ሽ	97	ቻ	135	ኪ	173	ዠ	211	ጐ	249	ፄ	287	3
22	መ	60	ሾ	98	ቼ	136	ካ	174	ዡ	212	ጓ	250	ፅ	288	4
23	ሙ	61	ቿ	99	ች	137	ኬ	175	ዢ	213	ጠ	251	ፆ	289	5
24	ሚ	62	ቀ	100	ቾ	138	ክ	176	ዣ	214	ጡ	252	ፈ	290	6
25	ማ	63	ቁ	101	ቿ	139	ኮ	177	ገ	215	ጢ	253	ፉ	291	7
26	ሜ	64	ቂ	102	ጎ	140	ኳ	178	ዥ	216	ጣ	254	ፊ	292	8
27	ም	65	ቃ	103	ጐ	141	ኸ	179	ዦ	217	ጤ	255	ፋ	293	9
28	ሞ	66	ቄ	104	ጒ	142	ኹ	180	ጊ	218	ጥ	256	ፌ	294	፡
29	ሟ	67	ቅ	105	ጋ	143	ኺ	181	የ	219	ጦ	257	ፍ	295	፤
30	ሠ	68	ቆ	106	ጌ	144	ኻ	182	ዩ	220	ጧ	258	ፎ	296	…
31	ሡ	69	ቋ	107	ጎ	145	ኼ	183	ዪ	221	ጨ	259	ፏ	297	*
32	ሢ	70	በ	108	ጓ	146	ኽ	184	ያ	222	ጩ	260	ፐ	298	#
33	ሣ	71	ቡ	109	ነ	147	ኾ	185	ዬ	223	ጪ	261	ፑ	299	?
34	ሤ	72	ቢ	110	ኑ	148	ዀ	186	ይ	224	ጫ	262	ፒ	300	null
35	ሥ	73	ባ	111	ኒ	149	ዂ	187	ዮ	225	ጬ	263	ፓ		
36	ሦ	74	ቤ	112	ና	150	ወ	188	ደ	226	ጭ	264	ፔ		
37	ሧ	75	ብ	113	ኔ	151	ዉ	189	ዱ	227	ጮ	265	ፕ		

FIGURE B.1: Sample results during model evaluation with unseen test set

(A) Incorrect predictions

(B) Correct predictions

Appendix C. Bayesian Hyper-optimization Source Code

```
def create_models_bayesian_opt(num_random_points, num_iterations,
                               results_dir, ml_algo_name):

  input_shape = X_train.shape[1:] # (64, 256, 1) - channels last
  number_of_classes = len(char_list)+1

  # parameters for cnn that do NOT have to be saved
  maximum_epochs = 100
  early_stop_epochs = 10
  learning_rate_epochs = 5

  # parameters that change for each iteration that must be saved
  list_early_stop_epochs = []
  list_validation_loss = []
  list_saved_model_name = []

  start_time_total = time.time()

  def create_model(num_cnn_blocks, num_filters, kernel_size, num_units,
                                                             batch_size,
                                                             drop_out):

    model_name = ml_algo_name + '_' + str(np.random.uniform(0,1,))[2:9]

    # variable parameters
    dict_params = {'num_cnn_blocks':int(3*num_cnn_blocks),
                   'num_filters':int(32*num_filters),
                   'kernel_size':int(kernel_size),
                   'num_units':int(128*num_units),
                   'batch_size':int(8*batch_size),
                   'drop_out':drop_out}
```

```
# start of cnn coding
input_tensor = Input(shape=input_shape)

# 1st cnn block
x = Conv2D(filters=dict_params['num_filters'],
          kernel_size=dict_params['kernel_size'],
          strides=(1,1), padding='same')(input_tensor)
#x = Dropout(dict_params['drop_out'])(x)

# additional cnn blocks
for iblock in range(dict_params['num_cnn_blocks'] - 1):
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(filters=dict_params['num_filters'],
              kernel_size=dict_params['kernel_size'],
              strides=(1,1), padding='same')(x)
    x = MaxPooling2D()(x)
    #x = Dropout(dict_params['drop_out'])(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 1))(x)

x = Conv2D(512, (2,2), activation = 'relu')(x)

squeezed = Lambda(lambda x: K.squeeze(x, 1))(x)

x = Bidirectional(GRU(dict_params['num_units'],
         return_sequences=True, name='gru1'))(squeezed)

x = Dropout(dict_params['drop_out'])(x)

x = Bidirectional(GRU(dict_params['num_units'],
         return_sequences=True, name='gru2'))(x)
x = Dropout(dict_params['drop_out'])(x)
output_tensor = Dense(number_of_classes, activation='softmax')(squeezed)

# instantiate and compile model
model = Model(inputs=input_tensor, outputs=output_tensor)
inputs = model.input
outputs = model.output
opt = Adamax(lr=0.001)  # default = 0.001
labels = Input(name='the_labels', shape=[max_label_len], dtype='float32')
input_length = Input(name='input_length', shape=[1], dtype='int64')
label_length = Input(name='label_length', shape=[1], dtype='int64')
```

```
def ctc_lambda_func(args):
    y_pred, labels, input_length, label_length = args
    return K.ctc_batch_cost(labels, y_pred, input_length, label_length)

loss_out = Lambda(ctc_lambda_func, output_shape=(1,))([outputs, labels,
                                                       input_length,
                                                       label_length])

#model to be used at training time
model = Model(inputs=[inputs, labels, input_length, label_length],
              outputs=loss_out)
model.compile(loss={'ctc': lambda y_true, y_pred: y_pred}, optimizer=opt,
              metrics=['accuracy'])

# callbacks for early stopping and for learning rate reducer
callbacks_list = [EarlyStopping(monitor='val_loss', patience=early_stop_epochs),
          ReduceLROnPlateau(monitor='val_loss', factor=0.1,
                            patience=learning_rate_epochs,
                            verbose=0, mode='auto', min_lr=1.0e-6),
          ModelCheckpoint(filepath=results_dir + model_name + '.h5',
                          monitor='val_loss', save_best_only=True)]

# fit the model
h = model.fit(x=[X_train, y_train, train_input_length, train_label_length],
        y=np.zeros(len(X_train)),
        batch_size=dict_params['batch_size'],
        epochs=maximum_epochs,
        validation_split=0.25,
        shuffle=True, verbose=0,
        callbacks=callbacks_list)

# record actual best epochs and valid loss here,
# added to bayes opt parameter df below
list_early_stop_epochs.append(len(h.history['val_loss']) - early_stop_epochs)

validation_loss = np.min(h.history['val_loss'])  # h.history['val_loss']
list_validation_loss.append(validation_loss)
list_saved_model_name.append(model_name)

# bayes opt is a maximization algorithm,
# to minimize validation_loss, return 1-this
bayes_opt_score = 1.0 - validation_loss

return bayes_opt_score
# end of create_model()
```

```
optimizer = BayesianOptimization(f=create_model,
          pbounds={'num_cnn_blocks':(2, 4.001),
                   'num_filters':(1, 4.001),  # *32
                   'kernel_size':(2, 4.001),
                   'num_units':(1, 4.001),  # *128
                   'batch_size':(1, 4.001),  # *8
                   'drop_out': (0.05, 0.5)},
                   verbose=2)

optimizer.maximize(init_points=num_random_points, n_iter=num_iterations)

print('nbest result:', optimizer.max)

elapsed_time_total = (time.time()-start_time_total)/60
print('\n\ntotal elapsed time =',elapsed_time_total,' minutes')

# optimizer.res is a list of dicts
list_dfs = []
counter = 0
for result in optimizer.res:
    df_temp = pd.DataFrame.from_dict(data=result['params'], orient='index',
                                     columns=['trial' + str(counter)]).T
    df_temp['bayes opt error'] = result['target']

    df_temp['epochs'] = list_early_stop_epochs[counter]
    df_temp['validation_loss'] = list_validation_loss[counter]
    df_temp['model_name'] = list_saved_model_name[counter]

    list_dfs.append(df_temp)

    counter = counter + 1

df_results = pd.concat(list_dfs, axis=0)
df_results.to_pickle(results_dir + 'df_bayes_opt_results_parameters.pkl')
df_results.to_csv(results_dir + 'df_bayes_opt_results_parameters.csv')
```