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

November 8, 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:*
ASST. PROF. KRIS C. CALPOTURA	_____	__ / __ / ____
Internal Examiner:	*Signed:*	*Date:*
DR. KINDE ANLAY	_____	__ / __ / ____
External Examiner:	*Signed:*	*Date:*
DR. MICHAEL MELESE	_____	__ / __ / ____

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	_____	25/06/2021

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

Abstract

Amharic (Amarñña: **አማርኛ**) is the official language of the Federal Government of Ethiopia, with more than 27 million speakers. But it is a low-resourced language, and only a few attempts have been made so far for handwritten Amharic text recognition. This is challenging due to the very high similarity between many of the alphabets, and handwriting calligraphy is personal. This paper presents offline handwritten Amharic word recognition using deep learning architecture, which comprises convolutional neural networks (CNNs) for feature extraction from input word images, bidirectional recurrent neural networks (BRNNs) for sequence encoding, moreover connectionist temporal classification (CTC) as a loss function.

To the authors knowledge, there was no any publicly available handwritten Amharic text dataset. Therefore, we commenced from preparing the dataset from scratch. 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 $12\,064$ word-images for our dataset. Data augmentation was employed by applying random transformations to the word-images of the training set to enhance performance of the proposed models.

In the main process, we have developed a custom model with CNN-BRNN-CTC framework, and Bayesian algorithm was used to set values for hyper-parameters. We have then compared four different state-of-the-art CNN models: EfficientNet, DenseNet, ResNet and VGG for robust feature extraction. We did this experiment by modifying their architectures to fit our problem domain. We have measured the word error rate (WER) and character error rate (CER) using our test set, which contains 1200 word images (they are randomly selected 10% of the total dataset). Hence, the outperforming model with DenseNet201 feature extractor network has achieved WER of 9.00 % and CER of 2.51 % on the non-augmented 64×256 word-image dataset and become advanced in WER of 6.83 % and CER of 1.82 % on augmented one. Whereas, the custom model has also achieved competitive performance on the offline handwritten Amharic word recognition with the state-of-the-art models.

Keywords: Deep learning, CNN-BRNN-CTC, offline handwritten Amharic word recognition.

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

Acronyms

AI	**A**rtificial **I**ntelligence
ANN	**A**rtificial **N**eural **N**etwork
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
CV	**C**ross **V**alidation
DC	**D**istrict of **C**olumbia
DIP	**D**igital **I**mage **P**rocessing
ELMo	**E**mbeddings from **L**anguage **Mo**dels
FLOPS	**FL**oating point **OP**erations per **S**econd
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
HAWR	**H**andwritten **A**mharic **W**ord **R**ecognition
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

As described in [1], text either in printed or handwritten form has been used widely for storing, accessing and transmitting information. An electronic document in this aspect has a great advantage to users of a language since it can be easily edited, shared, stored, and managed using text processing devices and tools. Whereas, handwritten and printed documents cannot be processed by computers which makes them difficult to easily edit, store, share and distribute them electronically. A large amount of printed and handwritten documents are available in the form of historical archives, invoices, tax forms, various documents in offices, etc. In order to convert such printed and handwritten documents in to computer readable format either it should be keyed manually, which is very time consuming and labor intensive, or systems of automated text recognition should be used [2]. Automated text recognition is a document digitization process using the techniques of pattern recognition, computer vision and natural language processing. Text recognition can be done either for machine printed or handwritten documents. The recognition of machine printed documents is much easier than the handwritten documents due to its specific structure of the document which makes the pre-processing task such as segmentation of characters, words and lines easier [3], [4]. However, the handwritten text recognition is more challenging due to the cursive nature of the document, myriad writing style of the individual, moreover, it is difficult to collect and prepare handwritten text dataset than machine printed document to develop automated text recognition models.

Although the field of text recognition has been studied for the past few decades and various techniques and systems have been implemented, it is still an active research area. Nowadays, optical character recognition (OCR) system for machine printed documents are able to digitize with a reliable performance. However, despite the huge effort made on the development of handwritten text recognition systems by various scholars and companies their performance is still limited. In order to achieve a reliable performance for recognition of handwritten text researchers investigated different sub-problems by applying some constraints on the recognition approach of the HTR task. One such constraint is on the process of data acquisition, the HTR task can be online where the handwriting is captured online using some kind of digitizer

input device and the recognition system do have both the temporal pen trajectory as well as text image information as an input. In the offline scenario, only the scanned image of the handwritten document is available for the recognition system as input, which makes it the more difficult one [5]. Figure 1.1 shows the general overview of text recognition system pipeline.

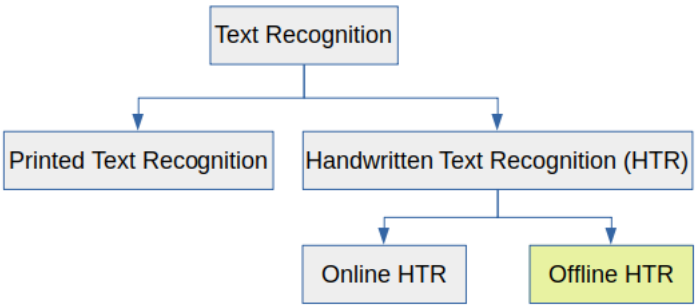


FIGURE 1.1: Types of text recognition based on data acquisition.

An handwritten text recognition (HTR) system often involves an interpretation of handwritten text image by a vector of feature values and normalization before the model, then convert into sequence of characters. And, the variation of handwritten documents is a major challenge for recognition of text. Therefore over the past years, researchers have proposed various text recognition techniques. By using classical machine learning-based HTR approaches such as hidden Markov's model (HMM [6]), which automatically learns from manually extracted features of handwritten text, have shown good results for the HTR task due to the sequential nature of a text. However, HMM based approaches become ineffective as the feature extracted from the handwritten text become too many and complex in variance. Recently, the use of artificial neural networks (ANNs) for learning feature representation from the scanned images of handwriting text benefit the HTR task. Many HTR researches have been done so far in the field of artificial intelligence (AI) for Arabic, Hebrew, Latin, Greek, Urdu, Hindi, Chinese, Japanese, etc. scripts [7]–[10]. Intuitively, models using the deep learning approach extract and learn the specific features of the data each with respect to a sequence of labels hierarchically using artificial neural network (ANN [11]) layers. State-of-the-art deep learning-based approach for solving HTR problems consists of the convolutional neural networks (CNNs), recurrent neural networks (RNNs), long-short term memory (LSTM) or gated recurrent units (GRU) with the addition of the connectionist temporal classifications (CTC) algorithm that perform the final labeling task from the RNN's output enhance the performance.

1.1.1 Amharic Language and Basics of its Writing System

The Ethiopian official language, Amharic (Amarñña: አማርኛ) uses the Ethiopic script as a syllabary [12]. It is second-most common language of Ethiopia (after Oromo) and second-most commonly spoken Semitic language in the world (after Arabic). 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]. Also, Amharic is considered as a holy language by the Rastafarian religion followers worldwide.

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.

Amharic alphabets, also called *"Fidel"* is a syllable writing system where consonants and vowels co-exist within each graphic symbol. In total, a *"Fidel"* has 238 core characters and 27 labialized forms (which have two sounds such as: ሏ, ቈ, ጧ, etc.) [5]. It has 34 base characters from which other six or more families formed by changing its shape (see Appendix A), which causes intra-class similarity. This similarity can be observed in the different orders of the writing system, for example: ሰ and ስ, ረ and ሬ, ም and ሞ, ሙ and ሙ, ፉ and ፋ, ፓ and ፖ, etc. Also, it contains 7 derived characters such as: ሸ from ሰ, ኸ from ከ, ጨ from ጠ, etc. Besides, the variability between classes is insignificant. This results in the inter-class similarity of characters in different families. In the same order, for example, ለ and ሰ, ደ and ጸ, ዐ and ፀ, ጰ and ጸ, ፈ and ረ, ጎ and ነ are which differs with a single stroke or mark. The inter-class shape similarity problem can also be observed in the numerals and punctuation as shown in the Table 1.1.

TABLE 1.1: The inter-class shape similarity problem of the script.

1^{st} Order	2^{nd} Order	3^{rd} Order	4^{th} Order	5^{th} Order	6^{th} Order	7^{th} Order
ለ and ሰ	ሉ and ሱ	ሊ and ሲ	ላ and ሳ	ሌ and ሴ	ል and ስ	-
ደ and ጸ	ዱ and ጹ	ዲ and ጺ	ዳ and ጻ	ዴ and ጼ	ድ and ጽ	ዶ and ጾ
ዐ and ፀ	ዑ and ፁ	ዒ and ፂ	ዓ and ፃ	ዔ and ፄ	ዕ and ፅ	ዖ and ፆ
ጰ and ጸ	ጱ and ጹ	ጲ and ጺ	ጳ and ጻ	ጴ and ጼ	ጵ and ጽ	ጶ and ጾ
ጎ and ነ	ጉ and ኑ	ጊ and ኒ	ጋ and ኃ	ጌ and ኔ	ጎ and ን	ኖ and ኖ
ፈ and ረ	ፉ and ሩ	ፊ and ሪ	ፋ and ራ	ፌ and ሬ	ፍ and ር	ፎ and ሮ
Numerals						
፩, ፬ and ፱	፫ and ፭	፮ and ፯	፲, ፵ and ፺	፻ and ፼		
Punctuations						
፥ and ፡-	፤ and ፧	፨ and ፧	፡ and ፧	፡ and ፨		

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

There are also ten among the base alphabets don't have unique sound (such as: {ሀ,ሐ,ኀ,ኸ}, {ሠ,ሰ}, {አ,ዐ}, {ጸ,ፀ}) and the characters enclosed in a brace can be used interchangeably in written documents. For example: ኃይል can be written as ሃይል or ሀይል. Likewise, the same word can have multiple spelling variants (e.g. አንበሳ, አምበሳ).

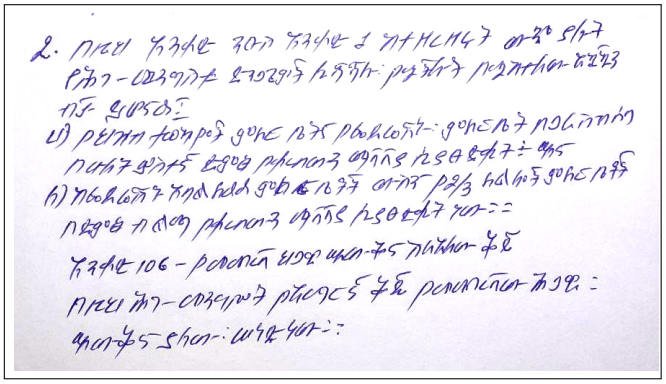


FIGURE 1.2: A sample handwritten Amharic text.

Handwritten Amharic word recognition (HAWR) is still active research area for the language, which transcribe handwritten word images into its respective digital forms. This thesis aims to develop an offline recognition algorithm that takes handwritten Amharic words as input and produces labelled sequence in digitized format. This is different from the one as character-level, line-level segmented, or as a page (text-level) kind of 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.

1.2 Motivation

The Amharic as one of the most commonly spoken language in Ethiopia, there are a bulk number of documents which are written in this language which are found in handwritten form in various locations including historical documents in churches and mosques, official and administrative documents, etc. Despite the availability of such huge number of documents in Amharic language which should be converted to digital form to make them easily stored and shared electronically, there are only a very few research attempts done for the handwritten Amharic text recognition task. Also, most of them focus on recognition of printed documents and recognition of handwritten documents at character level. In addition to that, most of the existing works use the

classical machine learning approaches which are claimed with low performance and unable to generalize in domain shift. In this thesis, we in detail have investigated the applicability of state-of-the-art deep leaning models for the very challenging handwritten Amharic word recognition. Also, the other motivation was absence of any public dataset for development of handwritten Amharic text recognition system. We have released a public dataset which can be used by other researchers for the future improvement of the recognition task.

1.3 Problem Statement

Handwritten documents will be used in various application areas including storage of historical documents, bank check processing and others. Due to age and various natural disaster, these documents may be damaged and can not be recovered again. Amharic language being used as one of the major languages in Ethiopia various handwritten documents are available which holds historical data of a given society or the country, medical and magical contents and other official document in governmental and non-governmental organizations. The conversion of these documents in to digital or editable form is crucial for storing, communicating and preserving these documents.

Even-though there are few research studies done for the Amharic language handwritten text recognition, most of them are done for isolated character recognition and printed document recognition. Few of the research works done for handwritten word recognition use classical machine learning approach with handcrafted features which are claimed by researchers they have low performance and unable to generalize in data domain shifts. The isolated character recognition based models for hand written text recognition struggle to segment characters compared with word and line level handwritten text recognition models since the cursive handwriting nature of the Amharic language. Handwritten text recognition is more challenging than printed text recognition due to the myriad writing styles, interconnection between adjacent characters, variation in size and shape of characters written by different writers or with in the same writer.

One of the major challenge in conversion of handwritten text in to digital form is the lack of consistency in writing style of different writers and the cursive nature of handwritten document. In addition, the writing system and vocabulary varies from language to language which makes it difficult to adopt developed OCR models of one language to the other. Due to the inherent complex morphology, large vocabulary size and the intra- and inter-class similarity between characters, the development of

HTR models for the Amharic language is very challenging. Also, 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 even the style of a particular person also changes in different instances of writing time. In contrast with the printed text, poor quality of the handwritten text pose harder hurdles in transcribing it to machine readable text.

Another challenge in development of HTR models is to extract relevant features from an handwritten text image which is used in the transcription to the target text in the image. In this thesis, we applied current the state-of-the-art deep learning feature extractors based on CNNs to extract features from an input word image. In addition, we approach the Amharic word recognition task as a sequence learning problem and applied the state-of-the-art deep learning based sequence learning algorithms based on RNN architecture. Even-though the existing handwritten text recognition models for other languages achieved state-of-the-art performance using deep learning techniques, they cannot be applied directly to the Amharic language due to its peculiar characteristics. In this thesis, we address these problems by using deep learning techniques, particularly such as: CNN, BGRU/BLSTM and CTC for transcribing handwritten Amharic words into the digital forms with robust feature extractions.

1.4 Research Questions

In this thesis, we want to answer the following research questions:

1. How effective are state-of-the-art deep learning based handwritten text recognition models for recognition of Amharic handwritten words?
2. Which model hyper-parameters should be optimized to benefit the recognition models to achieve high recognition rate for the Amharic handwritten word recognition problem?

1.5 Objectives

1.5.1 General Objective

The main objective of this study is to develop offline HAWR system using deep learning approaches.

1.5.2 Specific Objectives

The specific objectives of the study include:

- To review related works for understanding the concept of existing HTR models developed using the state-of-the-art deep learning approach.
- To collect, preprocess and standardize the dataset for handwritten Amharic words and for training the proposed HAWR model.
- To develop architecture of HAWR model using different combinations of deep learning approach.
- To optimize values for the hyper-parameters' setting of HAWR model.
- To analyze the application of pre-trained state-of-the-art models for HAWR task.
- To evaluate and perform comparative analysis of among the proposed HAWR models after training on the dataset with and without augmentation.
- To release publicly a new dataset for handwritten Amharic word.

1.6 Significance

Nowadays, usage of digital technologies arises in most sectors and many aspects of life to store and transfer information. People communicate through handwriting, but they still desire it's transcription into electronic text. Writers such as columnist, novelists, researchers, police officers, medical doctors, etc. can continuously have handwritten text and want it in computer text version. 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 proposed model, emphasizing from input data preparation to testing the developed model, pursuing high performance in the recognition task. Thus, the system is capable of taking an input word image and return a transcribed word, in the form of character sequence.

The potential HAWR applications include assists in preserving historical heritages and old historic documents (medicinal or magical books and manuscripts) all from being damaged and passing to the future generation. And also, it solves trouble in industries like bank, healthcare, insurance, etc. and transcribes students and lectures handwritten notes into digital format in education centers. The system can also be used in search engines, machine translations, other machine learning areas such as natural language processing and understanding (NLP/U) for the analysis of the text syntactic and semantics on a computer or an underlying machine. For transcription of offline handwritten Amharic word to the corresponding editable digital form, this thesis contributions are as follows:

1. There is no public dataset for handwritten Amharic text recognition. In this study, we produced a new and the first handwritten Amharic word dataset called HAW-DB to alleviate this issue. HAW-DB can be used for future studies and comparing related works.

2. We cross-validated different combinations of CNN and RNN architectures by using Bayesian hyper-parameter tuning, a model-based method for finding minimum of loss function. So, it has been used to tune the hyper-parameters' setting until a well-performing model found. As well as, we experimentally evaluated different input word image sizes with different RNN input lengths. And, we also analyzed the effect of data augmentation.

3. We investigated different state-of-the-art CNN models, such as EfficientNet, DenseNet, ResNet and VGG with modification to fit the problem domain, for robust feature extraction from handwritten Amharic words images. Finally, we have achieved state-of-the-art recognition accuracy for handwritten Amharic words using the proposed method.

1.7 Scope and Limitations

This thesis focuses on the recognition of handwritten Amharic word images, and on developing a deep learning model that results in the best performance for testing output sequence. Only an offline scenario is considered and recognizes handwritten Amharic words after trained with the dataset prepared by ourselves. Offline methods involve recognizing text once it's written-down often on a paper and scanned image of the document, which is with some background noises and possible degradation of part of the text, is given as an input to the recognition model. Thus, unlike online methods, there is no information to the strokes or the directions involved during writing. In general, this thesis is to come up with a model for an offline handwritten

Amharic word recognition purpose using deep learning approach and particularly CNN+BRNN+CTC frameworks. The main goal herewith deals not who is the writer, but instead what is written in an image for the computer to represent by its symbols, Unicode or ASCII.

The proposed model is writer-independent hence it is able to recognize handwritten Amharic word images whoever writes in a variety of writing styles. Also, these word images are inherently unconstrained and written in a natural handwriting. It knows a new word containing any character that's in the characters set which is given to train the recognition model. This indeed informs flexibility of the model besides increasing the language usability.

The proposed model did not consider the complete vocabulary of the language. Thus, the model do not handle any misspellings made in the source or by classifications in the model itself. In relation to that, some of Amharic alphabets have the same sound is still challenging to the recognition model; 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.

Unwanted stroke detection and removal is also an issue for the HTR models which has not been addressed perfectly yet and including the current work. Consequentially, this leads to incorrect spelling of the output. But in the recognition model, 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).

1.8 Thesis Structure

This thesis work have been done and organized as follows:

Chapter 1: Introduction to handwritten Amharic word recognition. In this chapter, we give the background of the study, problem statement, objectives, overview of the methods, significance, scope and limitations of the study.

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: Presents data collection and dataset preparation with the dataset statistics and word segmentation approaches, and data augmentation. This chapter also goes

into details on the fundamental knowledge serving as the neural network in deep learning used in custom network architecture, and applications of standard networks.

Chapter 4: 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 5: 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 proposed alternative ways to tackle further problems.

Chapter 2. Background Theory and Literature Review

In this chapter, we provide background information by briefly introducing the methods of HTR task by categorizing them into two parts: statistical (classical machine learning) and deep neural network based recognition approaches. In addition, we provide detail review of literature related to our work by summarizing the existing HTR techniques both in the classical machine learning based approach and current deep learning text recognition techniques.

2.1 Classical 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"* [13]. 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 [14] are among the common ones.

2.1.1 Hidden Markov Model (HMM)

The HMM is a statistical model that was first proposed by Baum L.E. [15] uses a Markov process that contains hidden (unknown) events. When we need to compute a probability for a sequence of observable events, Markov's chain is useful.

Generally, there are some drawbacks that exist when using HMM algorithms including adequate task-oriented knowledge is needed while designing HMM state models, and the assumptions need also be with explicit dependency to make their inference tractable or the assumption that observations in HMM should be independent. Also, HMM has discriminate sequence labeling although the training is generative.

2.2 Deep Learning Approaches

Deep learning, a powerful set of techniques use of artificial neural network (ANN), provide the best models for HTR tasks. There are multiple layers between the input and output layers for the sake of learning. The three often used deep learning models in HTR are convolutional neural network (CNN), recurrent neural network (RNN) and connectionist temporal classification (CTC).

2.2.1 Overview of Neural Network (NN)

The background of artificial neuron is the biological nervous system [16]. Thus, for example, Figure 2.1 shows an electrochemical process taking place to identify NaCl concentration.

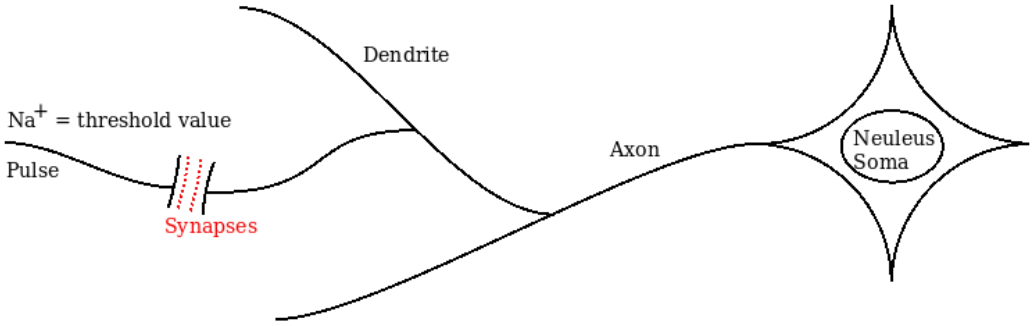


FIGURE 2.1: An illustration of biological neuron.

In ANN, a neuron is a node or a point at which certain mathematical operation is performed. There are two criteria need to be satisfied for a neuron to perform its task. First, the function it is going to perform must be known. Second, it has to have an input by which the function is executed. Generally, an artificial neuron structure is shown in Figure 2.2. Where, x is the input signals or vector and if f is already known neuron activation function, then $y = f(x)$.

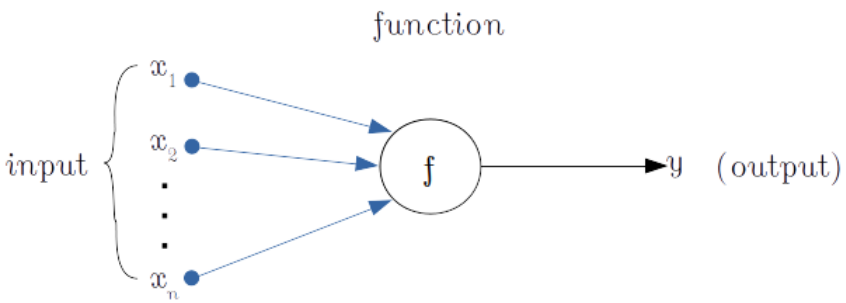


FIGURE 2.2: An illustration of artificial neuron model.

Depending on the value of executed function, the neuron is excited or is inhibited. Thus, a neuron is excited if the executed function is greater than certain threshold else it is inhibited.

2.2.2 Convolutional Neural Network (CNN)

CNN is inspired by the primary visual cortex of the brain. It is a specialized ANN 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 ANN 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.

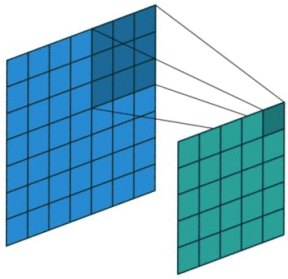


FIGURE 2.3: An illustration of convolution.

CNN involves convolution of the input tensors in two dimensions to give the output. 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 3×3 connectivity and stride equals 1 is applied to 7×7 input image so that the output would be 5×5. 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 [17]. CNN often followed by nonlinear activation functions like *ReLU* or *tanh*. This results in local networks, 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. RNN which is initially proposed by Elman in 1990 [18] and explored for use in HTR [19], [20] is best suited for decoding and extracting features from sequential data. Especially, RNN output is a function of not only the current input but also of the previous output, which is encoded into a hidden state h. That means, RNN has also a memory of the previous timestep by which it can encode the information present in the sequence itself. RNN can therefore 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.

2.2.3.1 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 [21]. 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 such as the input gate, forget gate and output gate, so allow information to have long range inter-dependencies. 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 [22]. 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 contained in the cell state.

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

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

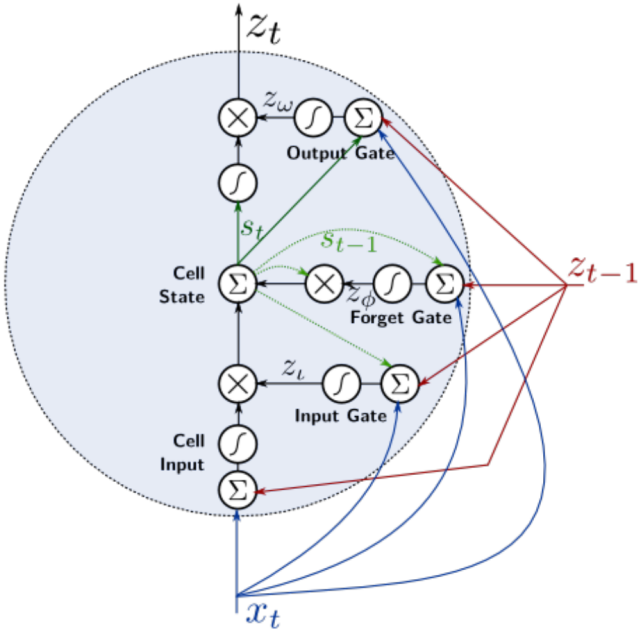


FIGURE 2.4: An illustration of long short-term memory unit [23].

Whereas, the outputs of these gates given by: $z_l^t = f(a_l^t)$ and $z_\phi^t = f(a_\phi^t)$ and the forget gate controls if the previous state is contained in the cell state, or if it is forgotten.

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

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.

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

For RNN component of HTR 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.

2.2.3.2 Bidirectional Gated – Recurrent Unit (BGRU)

GRUs with the assumption of $x = (x_1, ..., x_t, ..., x_T)$ for $x_t \in R^n$ be a sequence of T observations and $y \in C$ where C is ground truth class label [24]. 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}.

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

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 BGRU, $\left[h_{fw,T}^j, h_{bw,0}^j\right]$ implies to concatenate the forward and backward GRUs activation. Thus, the log-it value is modified as expressed by Equation 2.13.

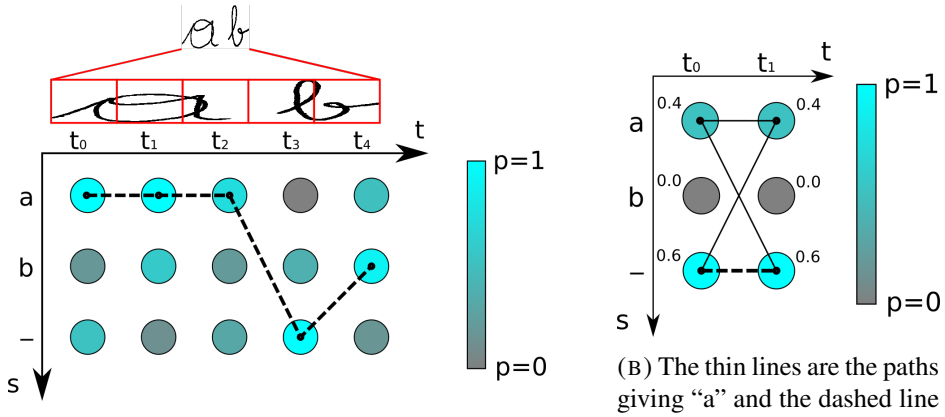
$$z^i = \sum_j w_s^{ij} \left[h_{fw,T}^j, h_{bw,0}^j\right] \tag{2.13}$$

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

$$softmax(z^i) = \frac{e^{z^i}}{\sum_j e^{z^j}} \tag{2.14}$$

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 example of output sequence CTC is fixing.

The CTC corresponds to the task of labeling unsegmented sequence data with RNNs [25]. It applies the output of the RNNs (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 its 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. Equation 2.15 is the probability of observing π, defining a distribution over the set of sequences L'^T, given the input sequences x.

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

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.

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

2.3 Commonly used Convolutional Neural Networks

In this subsection, we described in detail the most common CNN architectures which we have also used them in our proposed model during the early feature extraction step. These deep convolutional neural networks architectures have been modified to fit to our problem domain by seeing their performance experimentally.

2.3.1 Visual Geometry Group (VGG)

One of the deep convolutional neural net architecture, so called VGG was created by Visual Geometry Group at University of Oxford [26]. The VGG architecture is deeper than previously existing CNN architectures. In the VGG architecture blocks of CNN layers are grouped together with growing number of feature map sizes. There are various versions of the VGG architecture and we have used VGG19 in this study due to its high performance in feature extraction. The VGG19 is a latest version of the VGG architecture that is used to classify images. It has a total of 19 weight layers out of which 16 are the layers of CNNs and it can reduce computation by implementing layers of 3×3 filter size. Among its advantage, its implementations are easily available in keras and other deep learning frameworks.

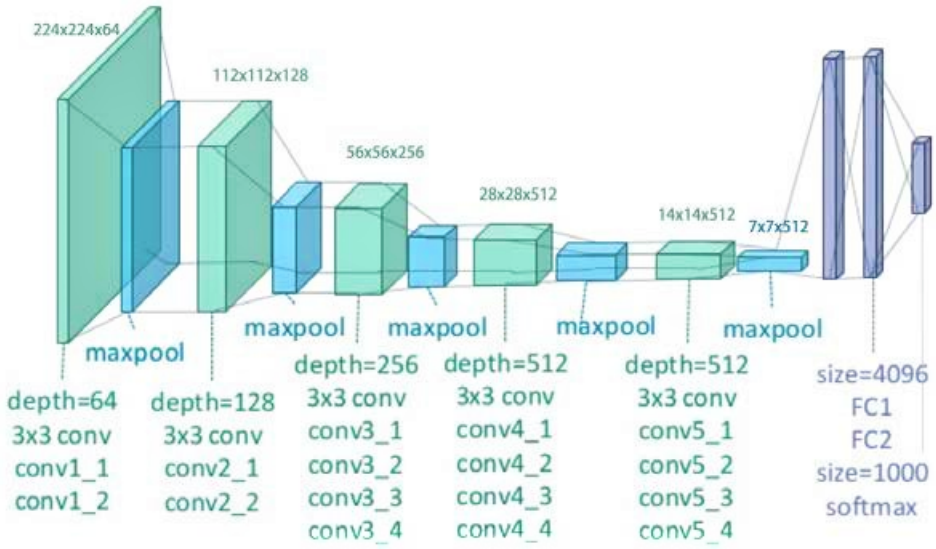


FIGURE 2.6: An illustration of the network architecture of VGG-19 model for ImageNet: conv (convolution), FC (fully connected) [27].

VGG19 first was used for competition on ImageNet dataset. It is composed of 3×3 convolutional layers stacked on top of each other in increasing depth. Diminishing size of volume is handled by max-pooling. Finally, each of two fully-connected layers with 4,096 nodes are then followed by a classifier which is softmax.

2.3.2 Residual Network (ResNet)

Microsoft Research proposed ResNet, which scaled up CNN layers from 18 to 200. It uses a technique called skip connections (or a shortcut connections) to overcome the problem of vanishing or exploding gradients despite the network is deep. If $\mathcal{H}(\mathrm{x})$ is representation of stacked layers and x is the inputs to the first of these layers, a residual function $\mathcal{F}$ can be approximated to $\mathcal{H}(\mathrm{x}) - \mathrm{x}$. The same dimensions of the input and output leads to identity shortcuts: $\mathrm{y} = \mathcal{F}(\mathrm{x}, \{\mathcal{W}_i\}) + \mathrm{x}$, where $\mathcal{F}(\mathrm{x}, \{\mathcal{W}_i\})$ is denoting residual mapping to be learned on multiple CNN layers, and y is output vectors. The $\mathcal{F} + \mathrm{x}$ is done by a shortcut connection and element-wise addition [28].

2.3.2.1 ResNet152 Architecture

When the network depth increased, and multiplied n of these large numbers become exploded, multiplied n of these small numbers become vanished. Despite the fact that deeper networks suffered a degradation problem, ResNet achieved more accuracy as the network depth increased without the accuracy got saturated. The deeper ResNet has smaller magnitudes of responses. Thus, 152-layer ResNet (11.3 billion FLOPs) has *lower complexity* than the shallower VGG19 model (19.6 billion FLOPs).

layer name	output size	152-layer
conv1	112×112	7×7, 64, stride 2
conv2_x	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax
FLOPs		11.3×10^9

TABLE 2.1: An architecture of ResNet152 model for ImageNet, Down-sampling is performed by $conv3\_1$, $conv4\_1$, and $conv5\_1$ with a stride of 2. [28]

2.3.3 Dense Convolutional Network (DenseNet)

DenseNet was developed by Tsinghua University and Facebook AI Research (FAIR) in Cornwell University. Unlike ResNet, the outputs of the preceding CNN layers within "dense blocks" of DenseNet are not summed but concatenated to the current layer output. This is shown in Figure 2.7[1]. Thus, each layer receives a "collective knowledge" from all of the preceding layers. Each layer adds k feature-maps of its own to this state. The growth rate regulates how much new information each layer contributes to the global state. The global state is not replicated layer by layer, instead it is written once and can be updated from everywhere within the network. There is also *bottleneck* layer to improve computational efficiency by reducing the number of input feature-maps [29]. DenseNet variants are: DenseNet-264, DenseNet-121, DenseNet-169 or DenseNet-201. DenseNet-264 ($k = 48$) outperformed.

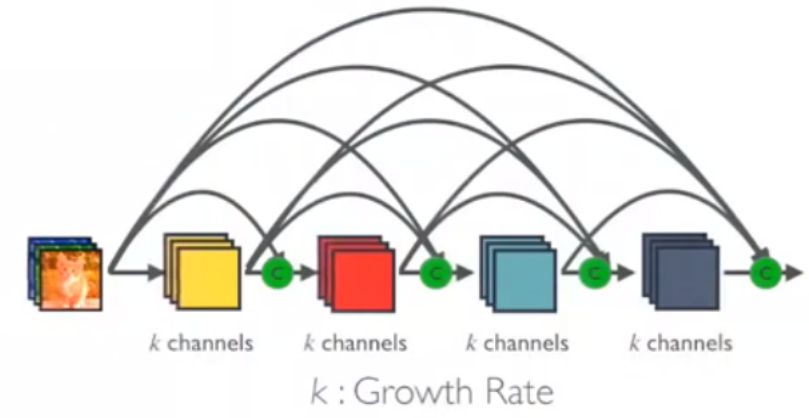


FIGURE 2.7: An illustration of the architecture of dense block.

2.3.4 Efficient Network for CNNs (EfficientNet)

As reported in [30], EfficientNets achieved greater efficiency than those models used previously on the ImageNet. Both accuracy and FLOPS can be effectively optimized. Intuitively, this was empirical evidence showing that the *compound scaling* method can be applied on the three dimensions of network width, depth, and resolution. This used a set of fixed scaling coefficients to uniformly scaled up all the three dimensions.

Hence, there are two steps followed to invent EfficientNet models. First, searching once on the smallest baseline network, which is EfficientNet-B0 (it has residual blocks of MobileNetV2 as a bottleneck and squeeze-and-excitation blocks). Next, scaling the baseline network using different compound coefficient, then EfficientNet-B1 to B7 are obtained. These EfficientNet models consistently reduce parameters and FLOPS.

[1] Review: Dense Convolutional Network (Image Classification) by Sik-Ho Tsang, Nov 25, 2018.

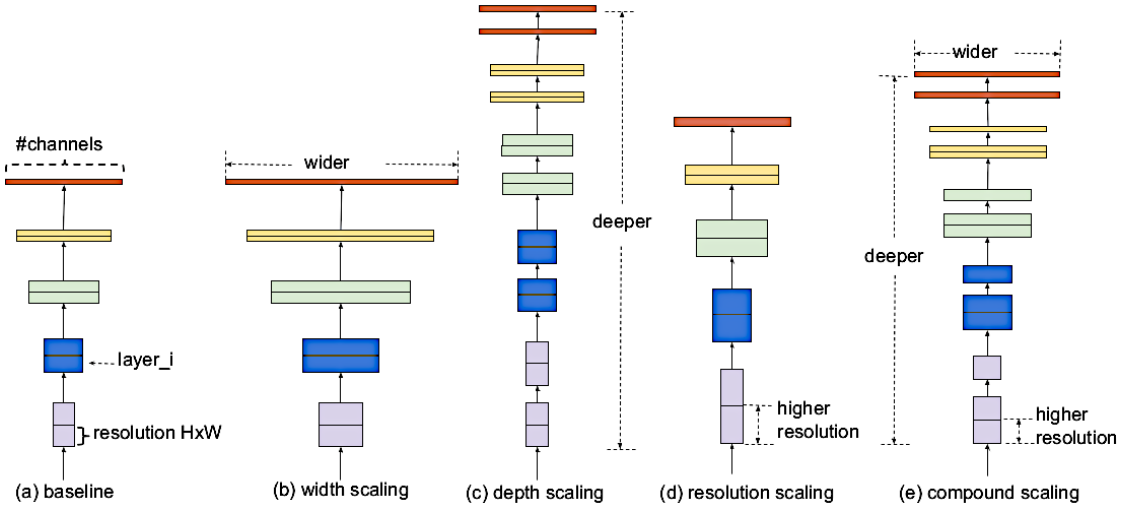


FIGURE 2.8: An illustration of model scaling. (a) is an example of baseline network; (b)-(d) are conventional scaling that only increases one dimension; (e) is compound scaling method that uniformly scales all three dimensions with a fixed ratio [30].

As it is depicted in Figure 2.8, the researchers note that there are three primary ways that is used to expand the network, these are in accordance with the comprehensive width, depth and resolution. Figure 2.8a is a baseline network, which is what we call the baseline. Figures 2.8b, 2.8c, and 2.8d expand the width, depth, and input resolution of the baseline network, and Figure 2.8e map for compound scaling. Thus mathematically, if the whole convolutional network is $\mathcal{N}$, and its i^{th} convolutional layer can be seen as a function map in Equation 2.24.

$$Y_i = \mathcal{F}_i(X_i) \tag{2.24}$$

Where, Y_i is the output tensor, X_i is the input tensor, assuming that the dimension of X_i is $\langle H_i, W_i, C_i \rangle$. By using the signal of the stage $1...s$, the i^{th} layer L^{th}, and $\mathcal{F}^{th}$ that has L^{th} in the i^{th} stage with the same structure convolutional layer, then the neural network $\mathcal{N}$ will have multiple identical convolutional layers, so it can be expressed as Equation 2.25.

$$\mathcal{N} = \bigodot_{i=1...s} \mathcal{F}_i^{L_i}(X_{\langle H_i, W_i, C_i \rangle}) \tag{2.25}$$

Now, having the three parameters including the depth L_i, the number of channels C_i, which is the width of the network, and the third one is resolution given by H_i and W_i, therefore the researchers main aim is that all convolutional layers of a convolutional network must be uniformly expanded by the same proportional constant.

2.4 Literature Reviews

The first handwriting recognition was in the 1995 which is character-level online-HTR based on Hidden Markov Models (HMMs) [31]. 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 [32].

An HTR system from traditional works that use of support vector machine (SVM), k-nearest neighbors (K-NN) algorithm, hidden Markov model (HMM) or any other classical machine learning approach has emerged to deep neural networks (DNNs), and hybrid of DNNs with machine learning algorithms [33] for its implementation.

For HAWR task, feature-level concatenation method outperformed over HMM-level concatenation across different document qualities and varying sizes of training and test data [5]. In this 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. However, HMMs are not enough to withstand many of the usual challenges especially that are associated with offline handwriting recognition: the input is a variable-sized two-dimensional image, the output is a variable-sized series of character, with no direct relation to the input size, and the prior segmentation into characters become difficult due to cursive nature of free-handwriting.

2.4.1 Related Works

Unfortunately, only few literature have been done for developing HAWR. Most of the existing handwritten recognition for Amharic language are based on the transcription of isolated handwritten characters [34]–[37]. An HMM for recognition of Amharic handwritten words used a traditional hand-engineered feature to represent character symbols [5]. 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.

Recently, there are only two researches done for printed and synthesized Amharic text-line images and published on the Amharic, printed and synthetic documents recognition (ADOCR[2]) [38], [39]. The studies attempted end-to-end learning to Amharic OCR and employed CNN for feature extraction and BLSTM (for sequence learning) and CTC for sequence labeling. They trained serially over a dataset curated

[2]http://www.dfki.uni-kl.de/belay/

using OCRopus. The dataset contains $337\,332$ Amharic text-line images, scaled them to 48×128 of spatial dimensions while data preprocessing, and trained with Adam optimizer. But, the dataset was limited to only two Amharic font types.

According to the baseline model used, there are also many other studies related and some of which are summarized in Table 2.2. 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 Amharic words consisting of $33\,672$ handwritten word-images[3].

[3]Handwritten Amharic word dataset (HAW-DB) used in this thesis is available online at this link: https://sites.google.com/view/hawdb-v1.

Year	Authors	Target scripts/Database	Baseline models	Results/Accuracy/CER	Citation
2020	B. Belay, T. Habtegebrial, Million Meshesha, G. B. Gebremeskel et al.	ADOCR: printed and synthetic benchmark Amharic Optical Character Recognition (OCR) database	CNN + BLSTM + CTC	CER of 6.98% and 1.05% for printed and synthetic fonts of Visual Geez and Power Geez, respectively.	[38]
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]
2020	S. Bansal, P. Krishnan and C.V. Jawahar	a collection of synthetic text in the Hindi language	CRNN (uses EmbedNet, a pre-trained word image embedding network)	Word recognition accuracy (WRA) of 85.364% at K textual transcriptions = 20.	[41]
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	[42]
2019	H.P. Tran, A. Smith, and E. Dimla	IAM Handwriting English Sentence Database [43]	CNN + BLSTM + CTC	88.18% on test data	[44]
2019	Xinfeng Zhang and Kunpeng Yan	offline Chinese handwritten text, a dataset collected from students	CNN + BRNN	83.0% on test set data	[45]
2019	R. Chamchong, W. Gao and M. D. McDonnell	Thai handwritten dataset collected from ancient archive documents	CNN + BLSTM + CTC	CER of 11.9% with the best architecture.	[46]

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.	[47]
2018	R. Maalej and M. Kherallah	Arabic handwritten text IFN/ENIT database [48]	CNN + BLSTM + CTC	The recognition rate of 92.21%	[49]

TABLE 2.2: Summary of some related works.

Chapter 3. Methodology

In this thesis, we have followed the experimental research approach in which we developed our custom CNN+BRNN+CTC architecture for the handwritten Amharic word recognition technique and we did extensive experimental study to evaluate and compare the performance of the handwritten Amharic word recognition (HAWR) models by optimizing various hyper-parameters of the architectures. The methods used in this study are described in this chapter including the procedures on how the dataset is prepared. As output from the pre-processing and input for HAWR model, we prepared a dataset which consists of a set of Amharic word images, with their corresponding transcripts, in the standard format. Then, a special emphasis has been placed on the deep learning components of the recognition system including: CNN, BRNN (such as BLSTM or BGRU) and CTC that make up the architecture.

3.1 Proposed End-to-End HAWR Model

In the proposed end-to-end HAWR model, we employed the state-of-the-art deep learning based HTR architecture, which comprises of three popular components namely CNN for feature extraction from input handwritten Amharic word images, RNN variants for sequence learning from the extracted features outputted from the CNN part and finally CTC as a loss function and sequence decoder algorithm. Figure 3.1 shows the general system overview of the CNN based feature extractor, BRNN based sequence modeling, and CTC based output transcription components.

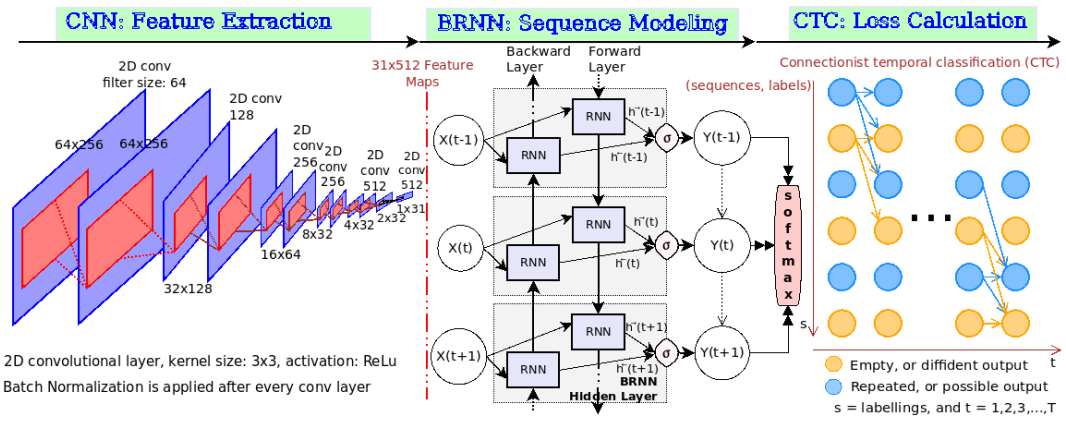


FIGURE 3.1: General overview of the proposed HAWR model.

Thus, the input data is prepared in the form of handwritten word images after data preprocessing, which is given as an input to stacked CNN layers. These layers are used to extract feature that represent the input word-image. The extracted feature from the CNN component is converted into one dimensional sequence of features before fed as an input to the RNN for learning temporal information of the characters in the input word image. Finally, the CTC is used to calculate the error made with the predicted, which are taken as output sequence, and actual word in the image.

3.2 Data Collection

There is no publicly available dataset for handwritten Amharic text or words. Due to this, we collected handwritten Amharic text afterwards conducted certain procedures for automatic word segmentation. We randomly split the original dataset into training set and test set then we did the data augmentation on the training set. Next, we applied data pre-processing on both dataset with and without augmentation, and the test set for standardizing their format.

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. Four documents from different aspects of views were selected, written with a pen on A4 paper and captured as image. These include the Aristotle monograph (writer: Hailegiorgis Mamo), 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. 4309 are unique and 2733 are single, which are written only one time out of the words in the dataset.

TABLE 3.1: Description of dataset.

Data Sources		Pages	Lines	Words
Aristotle monograph	[50]	7	178	1,389
Astronomy newsletter	[51]	7	188	1,476
Ethiopian constitution	[52]	50	1,074	7,817
Thus Spake Zarathustra	[53]	6	163	1,382
Total		**70**	**1,574**	**12,064**

3.3 Data Pre–processing

Data pre-processing was early stage that mainly categorized into five processes: an handwritten text image binarization, segmentation into word images, the word image labeling, data augmentation and data standardization.

3.3.1 Binarization

Binarization, is a digital image processing (DIP) technique, was applied to the text image and reduced it to binary form (often represented 1's on a background of 0's) using a thresholding transformation function followed by a thinning.

Thinning process 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.

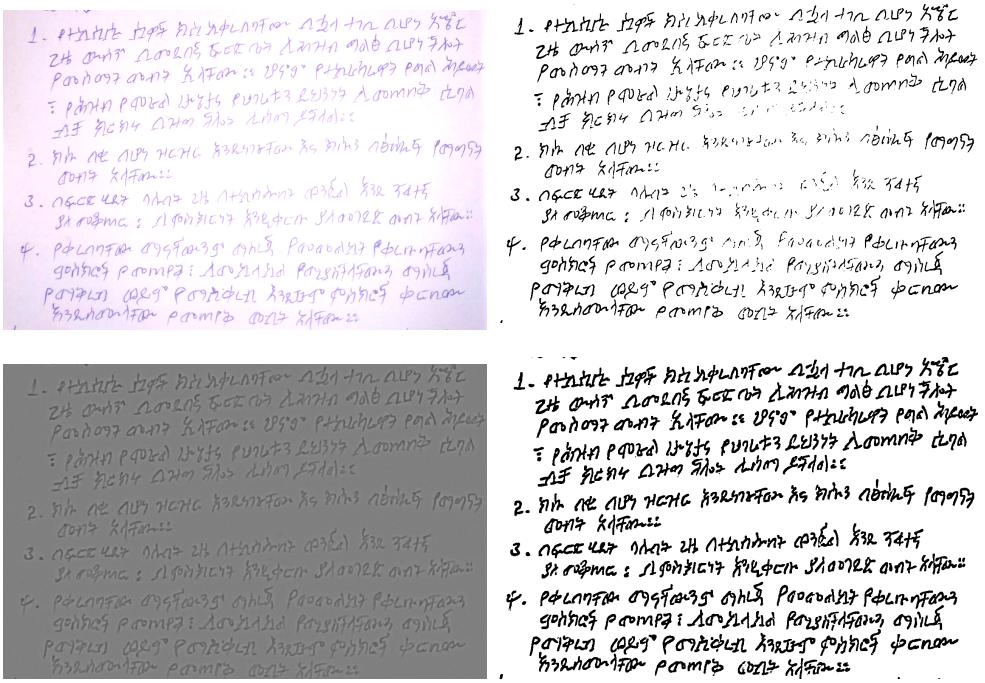


FIGURE 3.2: Challenges such as noise in binarization.

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 3.2, 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.

3.3.2 Image Cropping

The image was scanned and cropping frame was limited to handwriting text in the image. This was done by detecting the predominant contour in the image and by using four separate points of perspective transformation. Then, it was cropped.

3.3.3 Segmentation

Segmentation was employed to segment handwriting text image in levels of lines and words. A* path–planning algorithm [54] was performed to determine the path that separates each pair of lines; then, each line was segmented. Afterwards, *deslanting technique* applied for the removal of cursive and sloped handwriting styles from the line images [55]. Among the latest works in word segmentation, we used the scale space word partitioning algorithm [56] which provided 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 was 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 (see Figure 3.3).

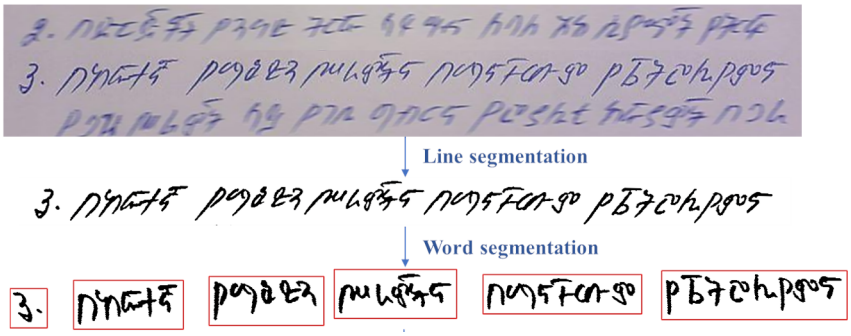


FIGURE 3.3: The word level segmentation from text image.

3.3.4 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 3.4 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.

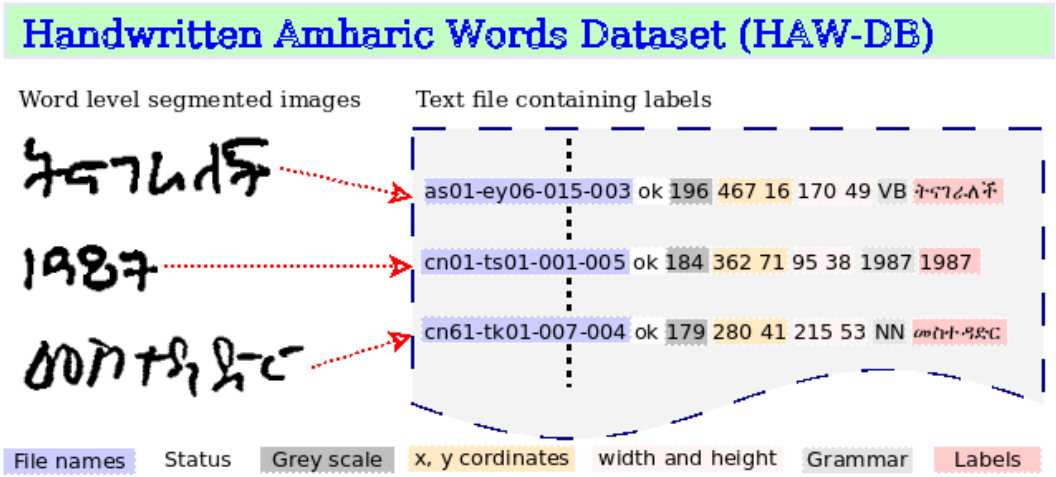


FIGURE 3.4: An illustration of word images and labels.

3.3.5 Data Augmentation

The size of dataset is an issue while training a deep learning model. Only $12\,024$ original handwritten Amharic word images contained in HAW-DB. And, this was relatively small dataset that very likely to cause over-fitting. The training set initially was $10\,824$ and could be extended to a total of $32\,472$ word-images with the addition of $21\,648$ augmented data, which were regenerated by the application of multiple random combination of transformation functions (we took care of that word-images should not have a function like flipping, instead we applied up to $10°$ x-and-y axis rotation, not more than 4 pixel erode, dilate and shifting, up to 10% resizing (zoom in, or zoom out), adding Gaussian noise and light intensity, opening or closing, etc. see samples in Figure 3.5). But the test set was remained the same for both the original and new dataset containing augmented ones, therefore to make fair comparison experiment. This new dataset when the test set added became $33\,672$ word images.

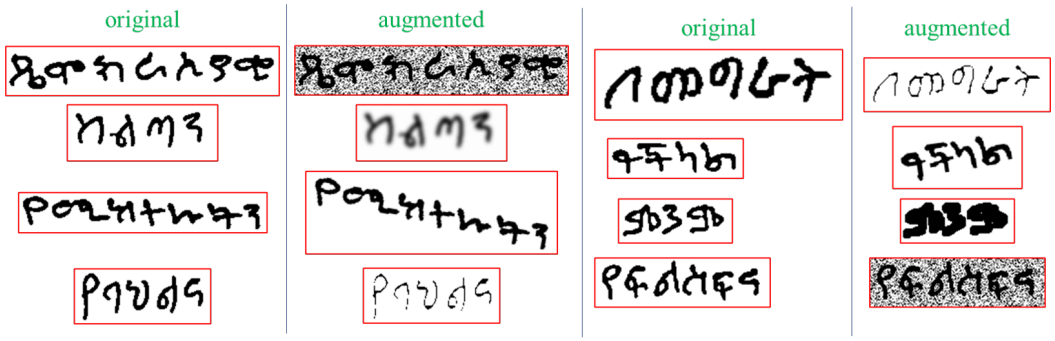


FIGURE 3.5: Samples showing application of data augmentation.

3.3.6 Data Standardization

These steps implemented the task of standardizing the format of the input data for training the model in Keras with TensorFlow backend. Thus, 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.

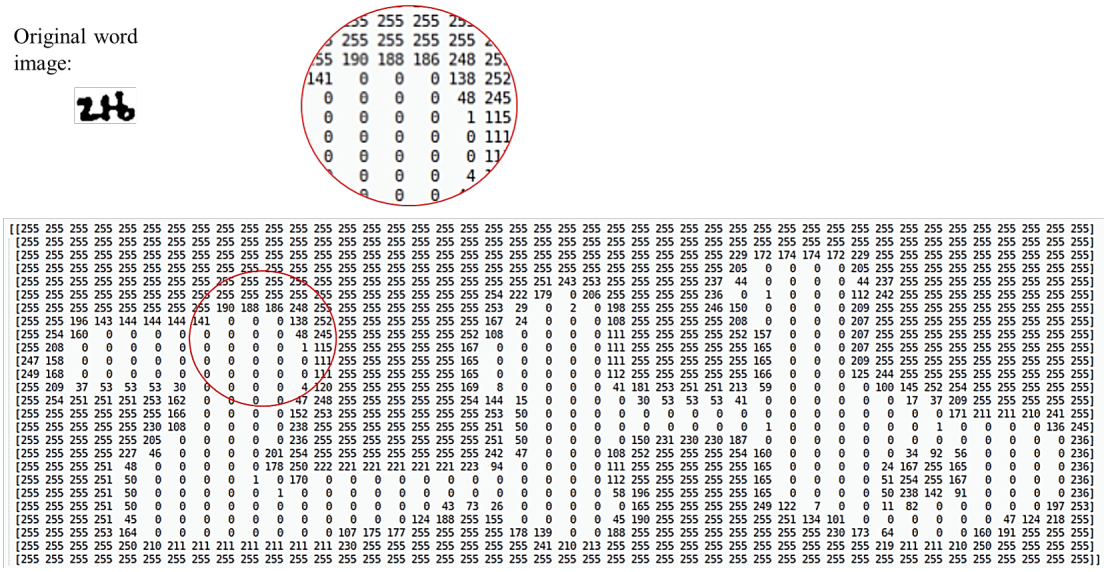


FIGURE 3.6: An example of gray-level handwritten Amharic word-image with its associated tensor for a word "ጊዜ".

This was 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 3.6 results as shown in Figure 3.7.

```
[[[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 word-image to shape (4, 8, 1) and normalization.

In Figure 3.7, 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 3.8.

```
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 3.8: 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.

3.4 Experimental Setup

In this section, we detailed the experimental analysis and investigation of optimal combination for the three major components (CNN, BRNN and CTC) by using different input handwritten Amharic word image. For the optimization experiment, we chose Bayesian algorithm and optimized the hyper-parameter settings of the CNN and BRNN architectures. In addition to the neural architecture search, we analyzed different input word image sizes and feature lengths that the BRNN received as input, and investigated their effect in a recognition performance of the model.

During all experiments, we randomly split the dataset with approximately 80% for training, 10% for validation and the remaining 10% for testing set. The more detail experimental settings for each component in our HAWR model is discussed in the following sub-sections.

3.4.1 CNN Based Feature Extraction

CNNs are well known techniques for extraction of spatial information from an input image. Hence, we did extensive experiments by using various architectures of CNN as feature extractors for robust representation of the input handwritten Amharic word images. We designed a custom CNN architecture by carefully optimizing values for the hyper-parameters of the network using our training and validation sets. Also, we applied the commonly used CNN architectures including VGG, ResNet, DenseNet and EffecientNet by modifying their architectures so as to fit our problem domain.

We modified the commonly used CNN architectures by removing the top n-layers until $(H \times W \times C)$ feature map could be obtained at the last convolution layer, where the spatial dimensions H is height, W is width and C is channel. When we increased the depth of CNN architecture, the down-sampled feature maps would have higher semantic information but the projected receptive field would cover larger area in the original image. Decreasing width of the feature map too much due to down-sampling would cause overlapping of features for two or more adjacent characters, which degrades performance of the whole model. In order to prevent such feature overlapping problem, we experimentally selected optimal values for the height and width of the feature map at the last convolution layers of our CNN architectures (both for the custom and commonly used CNN architectures). We did this by modifying strides of the max pooling layers at different layers of the proposed CNN architectures. We have achieved the best performance for our proposed CNN models with feature map dimensions, height (H) of 1 and width (W) of 31 generated at the last CNN layer.

3.4.1.1 Custom CNN Architecture

A custom CNN architecture that we designed for the comparable feature extraction task with the existing commonly used CNN architectures at less computational cost. We experimentally tuned each hyper-parameter including number of CNN layers, kernel size, pooling layer size, application of batch normalization and activation functions. Figure 3.9 illustrates the custom CNN layers used in our architecture. Each of these layers, consist the types of operations such as: convolutional, batch normalization and pooling layers.

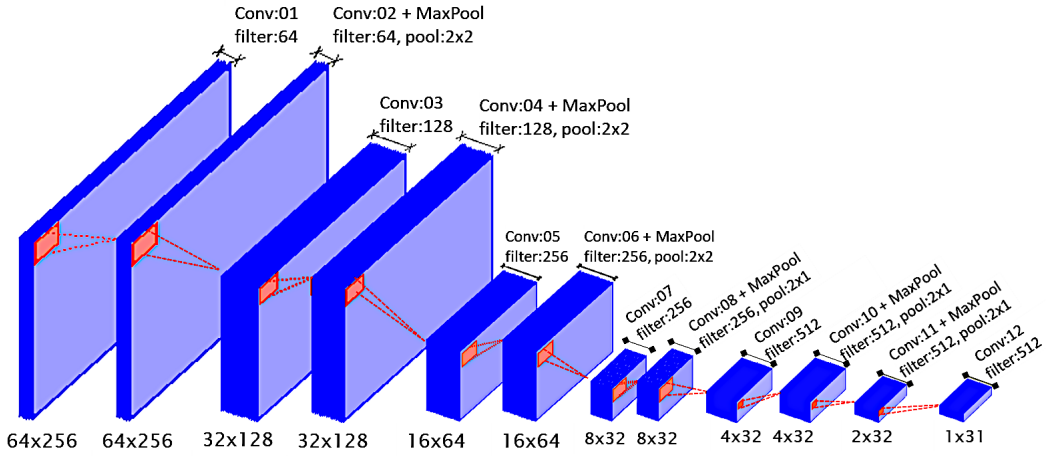


FIGURE 3.9: Feature extraction: Convolutional neural nets (CNN).

Our custom model consists of twelve CNN layers from bottom to top: 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. Batch Normalization [57] to resolve the internal covariate shift and rectified linear unit (ReLU) [58] non-linear activation function is applied at every convolution unit as depicted in table 3.2. Max-pooling layers are added after the second, fourth, sixth, eighth and tenth convolution layers. In the last three max-pooling layers, we set the pooling size to $(2, 1)$ in order to keep the width of the feature map at the last convolution layer at 31.

In the HAWR model, the custom CNN architecture is used to extract spatial features (a feature map) from the input word images. The maximum width of 31 and height of 1 is experimentally selected to be outputted at the last convolution layer of our custom CNN architecture, which is also used as the input sequence length of the RNN unit. The extracted feature map then was squeezed using map-to-sequence function to make it a one dimensional feature before feeding it to the RNN sequence learning component of our HAWR model. The CNN component provides the mapping onto a matrix (often it is one-dimensional input sequence), which is input for the two-layered BGRU (or BLSTM). Table 3.2 shows all the detailed layers of our custom

CNN+BRNN+CTC architecture. In addition to our custom CNN architecture we investigate in detail the applicability of other commonly used CNN architectures such as VGG, ResNet, denseNet and others as part of our HAWR model feature extractors. Their detail is provided in the next sub-section.

TABLE 3.2: Custom HAWR 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

3.4.1.2 Commonly Used CNN Architectures

In this thesis, as a feature extraction unit, we also applied four state-of-the-art CNN architectures including VGG, ResNet, DenseNet and EfficientNet. Experimentally, we then compared their performance to see their effect in obtaining discriminating feature for the input handwritten Amharic word images. For a given input tensor with a shape of $(H \times W \times C)$, where H and W are spatial dimension and C is the channel dimension and for RNN input length ℓ of feature sequence desired at the last layer of the convolutional network, we modified these feature extractors by removing n top layers until it outputs $(1 \times \ell \times C)$ feature map and by keeping the network patterns. But, the size of C at the last CNN layer varies from architecture to architecture.

Hence, we truncated the layers of the commonly used CNNs at different locations and add a new layer extension that is taken from our custom CNN architecture starting at $MaxPool4$, which are the eighth and above layers of the custom CNN architecture.

The truncation layer of a given CNN architecture vary based on the input shape of the word image and the maximum width of the feature map outputted at the last convolutional layer. Experimentally, we selected three input word image sizes of 32×128, 48×192 and 64×256 based on their performance in terms of accuracy and loss.

TABLE 3.3: Configuration of commonly used CNNs using input shapes of (32,128,1) with 31 and (48,192,1) with 47 as a maximum width of feature map at the last convolutional layer.

Id	CNN model	Truncation layer
A	VGG19	$10^{th}\ layer\ [block3\_conv4]$
B	ResNet152V2	$29^{th}\ layer\ [conv2\_block3\_preact\_relu\ (Activaton)]$
C	DenseNet121	$50^{th}\ layer\ [pool2\_conv\ (Conv2D)]$
D	DenseNet201	$50^{th}\ layer\ [pool2\_conv\ (Conv2D)]$
E	EfficientNet-B7	$157^{th}\ layer\ [swish\_30]$

TABLE 3.4: Configuration of commonly used CNNs using input shape of (64,256,1) with maximum width of 31 at the last convolutional layer.

Id	CNN model	Truncation layer
A	VGG19	$15^{th}\ layer\ [block4\_conv4]$
B	ResNet152V2	$119^{th}\ layer\ [conv3\_block8\_preact\_relu\ (Activaton)]$
C	DenseNet121	$139^{th}\ layer\ [pool3\_conv\ (Conv2D)]$
D	DenseNet201	$139^{th}\ layer\ [pool3\_conv\ (Conv2D)]$
E	EfficientNet-B7	$261^{th}\ layer\ [swish\_51]$

In the experimental setup, we named the models from the commonly used CNNs into five different models: *Model A* through *E*. Each of the models differ from one another by their CNN feature extraction unit. Table 3.3 shows the different configurations of the commonly used CNN architectures for the models using input word images sizes of 32×128 and 48×192 and table 3.4 shows for input word image size of 64×256.

3.4.2 BRNN Based Sequence Modeling

RNNs are a powerful dynamic network for sequence learning tasks. In the networks of bidirectional RNN (BRNN), each pair of 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 RNNs and uses past features

(through forward states) and the newer (by the backward) for a particular time frame. Figure 3.10 shows the architecture of BRNN with BGRU.

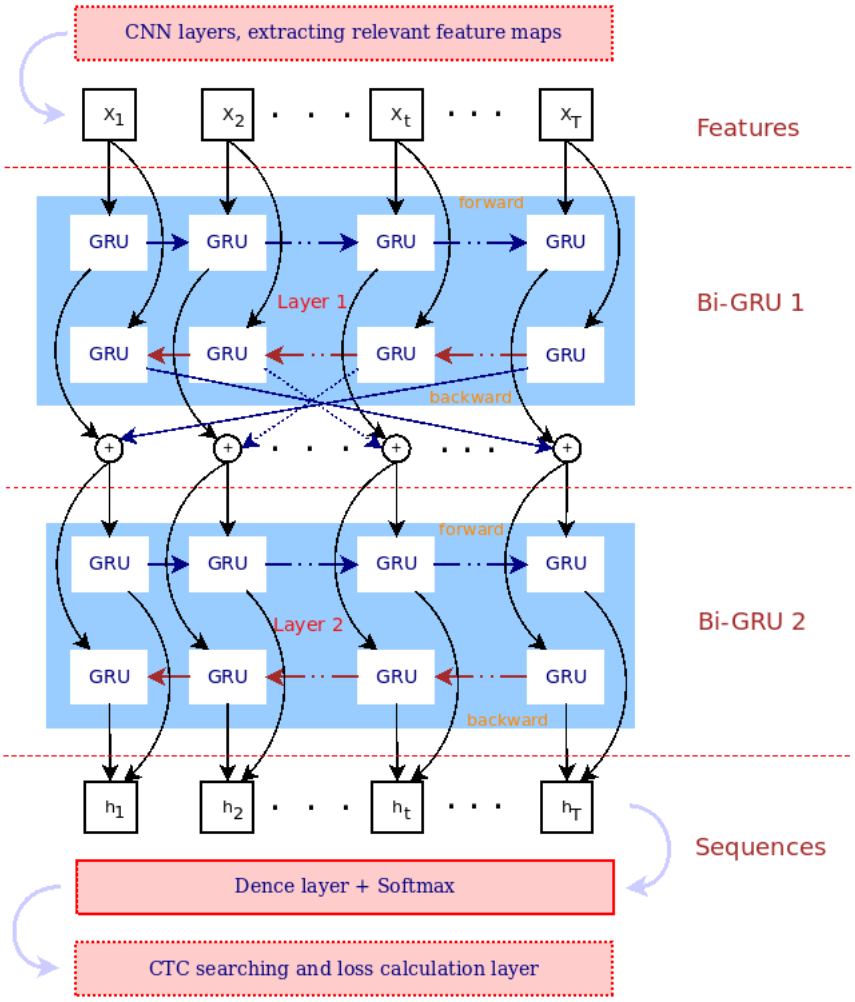


FIGURE 3.10: Architecture of bidirectional gated-recurrent units (BGRU) for sequence modeling.

In this work, we employed BRNNs to encode sequential context information of input word images extracted by using our CNN models described in the previous section. We have used the two abundantly used state-of-the-art BRNN frameworks, namely: the BGRU and BLSTM. We have compared different combinations of these two BRNN architectures by combining the two or using them separately, varying the number of layers and their hidden unit size. We have achieved optimal performance across all experiments when we use them separately rather than combining the two architectures. In addition, BGRU was much faster learning algorithm than BLSTM and better in terms of accuracy, since our dataset is not very large. BGRU is also often chosen over BLSTM because hidden states are totally exposed and so that they are easier to interpret. BGRU cells can reset all long-temporal information using internal gates and parameters are able to be optimized.

Two-layer architectures of both BLSTM and BGRU with a hidden unit size of 512 and dropout of 0.5 at each layer show better recognition performance in all setups our experiments. In our final HAWR model, we chose BGRU which performs better with such a small size of our dataset. For the output activation, we adopted softmax [59] with slightly better results than other candidates.

3.4.3 CTC Based Transcription

We employed CTC networks for two tasks, which include loss calculation and final decoding of the softmax layer output sequence into the correct labels with greedy search techniques. The softmax layer output sequence were represented by a matrix, which were used for two purposes including first for calculating the loss value while training the model and later for transcription (or decoding) the matrix to get the text contained in the handwritten Amharic word image. Figure 3.11 shows the ℓ labelling with respect to each time steps.

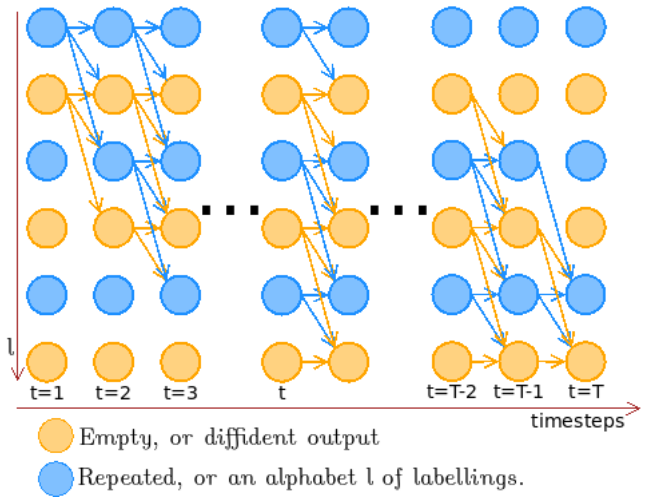


FIGURE 3.11: Sequence labellings with CTC loss function.

A CTC loss function controls the model training. We only fed the output sequence from the BRNN component of our HAWR model and the corresponding ground-truth (GT) text of the input word image to the CTC loss function. Empirically, it tries all possible alignments of the GT text in the image and takes the sum of all scores (the score of a GT text is high if the sum over the alignment-scores has a high value). The outputted probability distributions from the softmax layer at the end of the BRNN sequence modeling unit are used by the CTC to calculate the conditional probability of a possible alignment path given an input sequence.

In this thesis, CTC was given the probability distributions of a total of 301 symbols (300 character sequences and one extra character for representing the CTC blank

label) as input. Using these, the CTC finds the maximum probable label sequence. Thereupon, we could obtain character-scores for each sequence-element from the CTC. Therefore, the CTC decoder finally outputs a sequence (a word) which could have 31 characters at most. In addition, CTC implicitly used to model the correct inter-label dependencies and hierarchical CTC, where the labellings at one level (e.g. word) become inputs for the labellings at the next (e.g. character).

3.5 Bayesian Optimization Algorithm

Neural architecture search is one of the techniques to select the best optimal hyper-parameters of a given deep neural network. Researchers propose various techniques for neural architecture search. Bayesian optimization is one of the recent and best suited optimization technique to search optimal network hyper-parameters. This optimization approach uses Bayes theorem. This theorem works based on Bayes rule, which calculates a conditional probability of an event. Hence, the rule states that the probability p of hypothesis h is true under condition data D is observed, $p(h|D)$ is given by Equation 3.1.

$$p(h|D) = \frac{p(D|h) \times p(h)}{p(D)} \tag{3.1}$$

This algorithm provides efficient and effective search of the minimum or maximum cost of a global optimization problem and works better than the 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 optimizes explorations while it is predicting the result with a new hyper-parameter settings from knowing the different results so far obtained.

The Bayesian hyper-parameter tuning algorithm was used in this thesis to set values for hyper-parameters in our neural network (NN) model. In other words, this helps us to find a vector of hyper-parameters that works well on our problem domain. Hyper-parameters including number of CNN layers, number of filters, number hidden units, kernel size, batch size, learning rate, and dropout rates were optimized. Bayesian optimization was invoked as shown in Algorithm 2. Initially, we were required 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. The algorithm used to create models based on points for each iteration then find the more optimized one.

Algorithm 2: Bayesian hyper-parameters optimization algorithm.

$inputShape \leftarrow trainImages$.**shape[**$1:$**]** /* e.g. (64,254,1) */

$numClasses \leftarrow$ **len(**$charList$**)** $+ 1$ /* characters plus 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**$(function \leftarrow model,$
$parameterBounds \leftarrow \{numCNNs(6:24), numFilters(64:512),$
$numUnits(128:2048), kernelSize(1:4), batchSize(8:32),$
$learningRate(0.0001:0.01), dropout(0.2:0.6)\})$

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 */

$dataFrameForEpoches \leftarrow listEarlyStopEpoches\,[id]$

$dataFrameForLosses \leftarrow listValidationLosses\,[id]$

$dataFrameForModels \leftarrow listSavedModelNames\,[id]$

$dataFrameForBayesErrors \leftarrow res\,['target']$

3.6 Evaluation Metrics

During our experimental analysis, we have used different evaluation metrics to test and interpret a performance of the recognition model. For measurement of the HAWR model performance, we determined and selected four quantitative standard evaluation metrics and these are: loss, accuracy, character error rate (CER) and word error rate (WER).

As we mentioned in subsection 3.4.3, the loss and accuracy [60] were calculated by the CTC loss function. Loss is a distance between the ground truth (GT), which refers to the actual word contained in the word image, and the predicted word by the recognition model while accuracy is simply a measure of how many correctly predicted words are made by the model.

Next, we made the HAWR model performance evaluation by measuring similarity or dissimilarity (distance) between two text strings (i.e. the ground truth (GT) and the predicted text). The CER and WER are therefore the standard numerical error measures. The CER is the number of edit operations to match the recognized text with the GT, which essentially is the Levenshtein edit-distance, divided by the GT text length (see Equation 3.2).

$$CER = \frac{insertions + deletions + substitutions}{length(GT)} \tag{3.2}$$

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.

3.7 HAWR Model Training

During model training we split and grouped the handwritten Amharic word recognition dataset into three sets, these are: training, validation and testing. The set of handwritten Amharic word images that the model trains and learns was contained in the training set, and the dataset that we held back from training used to provide an unbiased assessment of the model skill on the training set was contained in the validation set. These two sets were used while training the model but the test set was used to validate the model skill with data that has not been before after training is done. In addition, we have used 10-fold cross validation [61] to make unbiased evaluation and increase the generalization capability of the recognition models. This method was carried out for 10 independent evaluations, and with different definitely separated partitions of training and validation sets from each other. Figure 3.12 illustrates that in each evaluation, the system is trained with 90% of handwritten Amharic word images in the dataset and validated with the remaining 10%.

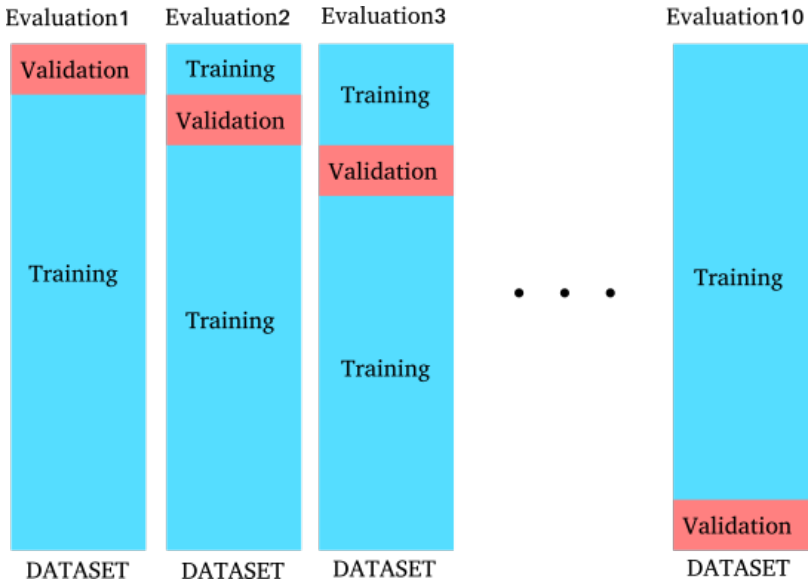


FIGURE 3.12: An illustration of the dataset in 10-fold CV.

The training was carried out on Google Co-laboratory (COLAB) with GPU (8 GiB Tesla T4, NVIDIA) and 12 GiB of RAM, which currently has the highest percentage of use and popularity. In all the experiments, the proposed HAWR models are trained using stochastic gradient decent (SGD) approach. And, we experimentally selected an optimizer Adamax [62] with initial learning rate of 0.001 and an epoch of 100 with mini-batch size of 10 to train all the models. Furthermore, we experimentally optimized four input handwritten word image sizes, these are: 16 (height) $\times$ 64, 32 (height) $\times$ 128, 48 (height) $\times$ 192 and 64 (height) $\times$ 256, and four RNN input lengths of 15, 31, 47 and 63 by exploring their effect on performance of the different

recognition models. During this experiment, we faced a challenge of incompatibility between the input size (word image size and RNN sequence length) and the custom model architecture. To solve such problem, the CNN architecture should be slightly modified in accordance with the given input size. Hence, we have modified max pooling layer size of $MaxPool2$ from $(2, 2)$ to $(3, 3)$ and omitted $MaxPool3$ for the combinations of the input word image size of 48 (height) $\times 192$ with RNN input sequence length of 31. Similarly, for RNN input sequence length of 63, we have modified size of the pooling layer $MaxPool1$ from $(2, 2)$ to $(2, 1)$, $MaxPool2$ from $(2, 2)$ to $(3, 3)$ and omitted $MaxPool3$. For the input word image size of 32 (height) $\times 128$, we omitted $MaxPool3$ when the RNN input sequence length is 31 and modified size of $MaxPool2$ from $(2, 2)$ to $(2, 1)$ when the sequence length is 63. Likewise, we omitted $MaxPool2$ and $MaxPool3$ for 16 (height) $\times 64$ input size and 31 sequence length, and modified $MaxPool1$ from $(2, 2)$ to $(2, 1)$.

We also did experimental analysis by training all the recognition models using the augmented data to improve models' recognition performance and to avoid overfitting problem.

We have designed different models for HAWR task, which we call them *Model A*, *Model B*, *Model C*, *Model D*, *Model E* and *Custom model*. All these models have similar sequence modeling unit with two layers of BGRU with 512 hidden units each and a dropout unit of 0.5 added at each GRU layers. They also have similar CTC loss and decoding function. These models differ from one another on the CNN feature extraction component where we use VGG-19, ResNet-152V2, DenseNet-121, DenseNet-201, EfficientNet-B7 and our custom CNN architecture as a feature extractor in *Model A*, *Model B*, *Model C*, *Model D*, *Model E* and *Custom model* respectively. For all the recognition models, we have done an experimental analysis using non-augmented data and input handwritten word image sizes of 16 (height) $\times 64$, 32 (height) $\times 128$, 48 (height) $\times 192$ and 64 (height) $\times 256$. We have also did an experimental analysis using the augmented data and input word image sizes which show better performance on the non-augmented data, these are: 32 (height) $\times 128$ and 64 (height) $\times 256$.

Chapter 4. Experimental Results and Discussion

This chapter presented the results of the different recognition models proposed for the handwritten Amharic word recognition (HAWR) task. Detailed experiments were conducted to analyze the effects of different model parameters and data preprocessing techniques including data augmentation. We have used the test dataset with randomly selected 1200 word images, which is 10% of the total dataset to test performance of the models in all the experiments.

4.1 Optimization Results of the Custom Model

We have done detailed experimental evaluations to obtain an optimized custom model for the HAWR task by analysing different hyper-parameters of our custom model. We have done the optimization techniques for the input word image size and input sequence length through trial and error approach, where as we use the Bayesian algorithm for the rest of hyper-parameters including the number of CNN layers, filter sizes, kernel sizes, dropout rates, batch sizes and RNN units, which hyper-tuned the model for achieving improved recognition performance. Based on the Bayesian optimization result, we have achieved high performance for our custom model with CNN layers, filter sizes, kernel sizes, dropout rates, batch sizes and RNN units as shown in the red colored results in Table C.1 of Appendix C.

In addition, the selection of an optimal size of input for the neural network has a profound effect on the performance HAWR models. When the optimal input word image size is used, CNNs can easily extract more relevant feature and more correctly represent the word in the image. Similarly, the maximum input sequence length to the BRNN affects the performance of the HAWR model. Hence, we have optimized input images size along with the RNN input length. Table 4.1 shows the experimental results in terms of accuracy and loss by varying the input word image size and the RNN input sequence length. The results in Table 4.1 are based on the custom model when trained and evaluated using non-augmented data.

From the experiment results shown in Table 4.1 using input word image size of 48 (height) $\times 192$ and RNN input sequence length of 47 we achieved the least loss from

TABLE 4.1: The effect of input image size and RNN input length on the accuracy/loss in the custom HAWR model. Abbreviation: seq. (sequence)

RNN input seq. length	**Input image size**			
	16 × 64	32 × 128	48 × 192	64 × 256
15	76.92 / 1.220	84.17 / 0.834	81.42 / 1.002	83.00 / 0.810
31	81.92 / 0.903	84.25 / 0.697	80.83 / 0.857	**86.17** / 0.682
47	82.17 / 0.899	84.17 / 0.754	85.92 / **0.598**	85.00 / 0.665
63	80.00 / 0.921	85.83 / 0.680	73.33 / 1.120	85.33 / 0.676

the rest of test results, but less accuracy than the word image size of 64 (height) $\times 256$ with RNN input sequence length of 31. And, input word image size of 64 (height) $\times 256$ with RNN input length of 31 resulted in the greatest accuracy than the rest due to its high input word image resolutions, but it has higher loss. Finally, we have chosen an input word image resolution of 64 (height) $\times 256$ with RNN input length of 31, since we could minimize the loss by adjusting other hyper-parameters settings of the custom model.

4.2 Experimental Results of the Commonly used CNN Architectures with the Custom Model

We have done comprehensive experimental analysis and evaluation of commonly used CNN models which use the CNN architectures described in section 3.4.1.2 for the handwritten Amharic text recognition task using out test set data. Table 4.2 shows the experimental results of the five models (*Model A* through *Model E*) using an input word image size of 32 (height) $\times$ 128, 48 (height) $\times$ 192 and 64 (height) $\times$ 256, which are selected as best performing values from the custom model optimization. The results in Table 4.2 are shown for the commonly used CNN models trained and evaluated using a non-augmented dataset. Additionally we have shown the results of the custom model in Table 4.2 for comparative analysis with the commonly used CNN based models. As we can see from the table, *Model D*, which is based on the architecture of DenseNet201 has shown higher recognition performance with a WER of 9.000% and CER of 2.506% using input word image size of 64×256 when compared to the rest of HAWR models. This is due to DenseNet201's of its network architecture design which enables for extraction of robust feature representation from the input word image and high resolution input word image size compared with the other models.

TABLE 4.2: Result of the commonly used CNN based model on the non-augmented dataset. $[Patience\,(early\ stopping) : 7]$

Input shape	**Model**	**Test accuracy**(%)	**Loss**	**CER**(%)	**WER**(%)
(32, 128, 1)	A	86.67	0.669	3.547	11.08
	B	85.75	0.712	3.910	12.33
	C	86.83	0.648	3.519	11.58
	D	84.58	0.723	3.564	12.42
	E	83.67	0.790	4.080	13.92
	CUSTOM	88.17	0.669	3.245	10.58
(48, 192, 1)	A	88.00	0.565	2.926	10.00
	B	84.00	0.773	3.704	12.92
	C	85.50	0.639	3.543	11.58
	D	86.17	0.602	2.900	10.42
	E	85.08	0.805	3.942	12.50
	CUSTOM	88.75	0.582	2.816	9.167
(64, 256, 1)	A	82.17	0.839	4.610	14.75
	B	87.50	0.625	3.293	11.00
	C	87.08	0.645	2.996	10.33
	D	**89.42**	**0.564**	**2.506**	**9.000**
	E	84.33	0.826	4.151	13.08
	CUSTOM	88.58	0.557	2.813	9.167

4.3 Experimental Results using Augmented Data

We have also conducted an experimental analysis to investigate the effect of data augmentation on the proposed HAWR models. The augmentation of data has enhanced the performance of the models by increasing the data variation in training data. Table 4.3 shows the results of addition of 2-times augmented input word-images with size of 64 (height) $\times\, 256$ selected for the model training and evaluation. As it is shown from our experimental result, by training the HAWR models using data augmentation much better performance is achieved on the test set data due to the increased number of training samples, which is enabling the models to learn more robust features from the data variations and this technique contributes for preventing over-fitting problem and higher variance within data and symbols. From the experimental result the best performing model in all our proposed models is *Model D*, which is using DenseNet201, and has achieved a WER of 6.833 and a CER of 1.819.

TABLE 4.3: Result of the recognition models on the augmented dataset $[Patience\,(early\ stopping) : 7]$.

Input shape	Model	Test accuracy(%)	Loss	CER(%)	WER(%)
(32, 128, 1)	D	90.50	0.538	2.298	8.167
	CUSTOM	90.42	0.569	2.374	8.417
(64, 256, 1)	**D**	**92.17**	**0.445**	**1.819**	**6.833**
	CUSTOM	90.17	0.498	2.319	8.250

4.4 10 – Fold Cross – Validation Result

Furthermore, we have employed 10-fold cross validation technique as a part of the experimental analysis in addition to the one that we split the dataset into training, validation and testing sets, which is discussed in the previous sections. The reason behind we have chosen this cross validation was that our dataset is small, and this approach has shown promising result by training and evaluating the model in varied set of samples during the learning process. During these experiments, we have used both the non-augmented and augmented dataset with the two input word image sizes of 32 (height) $\times$ 128 and 64 (height) $\times$ 256, and compared the performance of two HAWR models (particularly, the *Model D* and custom model).

Herewith, we reported the result graphically using non-augmented dataset with input word image sizes of 32 (height) $\times$ 128. Figure 4.1 shows a graph that is given by tracking each accuracy in the 10 folds of the *Model D* and custom model at the end of each fold, and the abscissa is the fold's number.

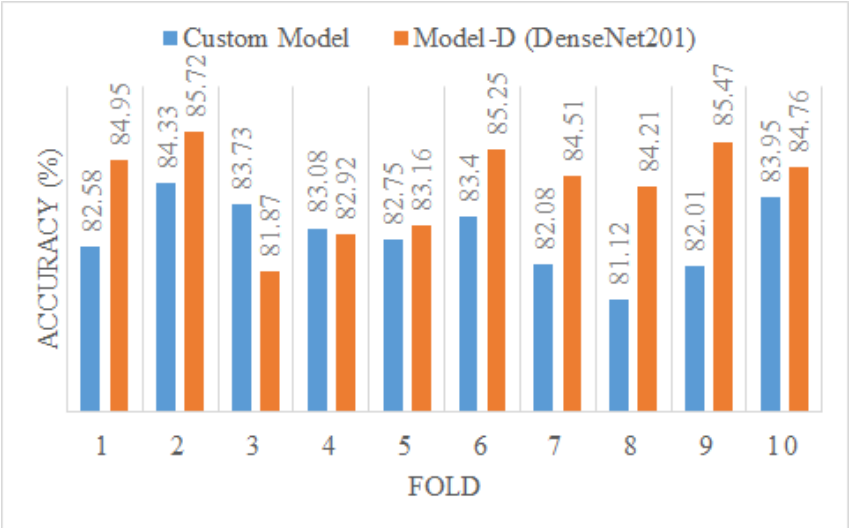


FIGURE 4.1: Graph of average accuracy while training 10-fold CV.

As the graph in Figure 4.2 reveals that the same is done to compare the models' loss.

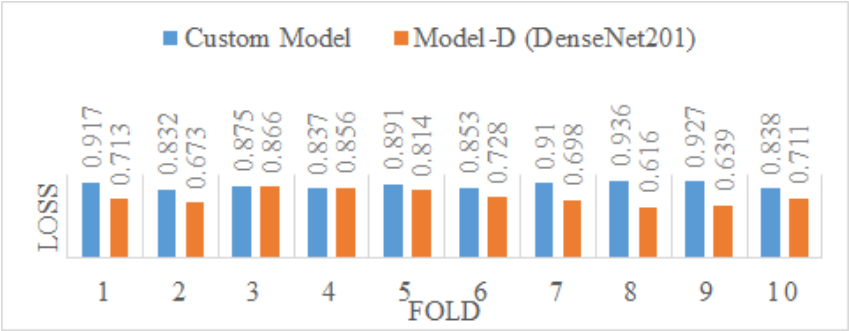


FIGURE 4.2: Graph of average loss while training 10-fold CV.

Thus, the dataset has been trained and evaluated using the 10-fold CV concept, and we have achieved the average validation accuracy of $(82.90 \pm 0.993)\%$ and loss of (0.6279 ± 0.0515) with the custom model, and the average validation accuracy of $(84.14 \pm 0.86)\%$ and loss of (0.6279 ± 0.0515) with the *Model D*. In similar fashion, the graphs depicted in Figure 4.3 and 4.4 indicate each CER and WER of the models.

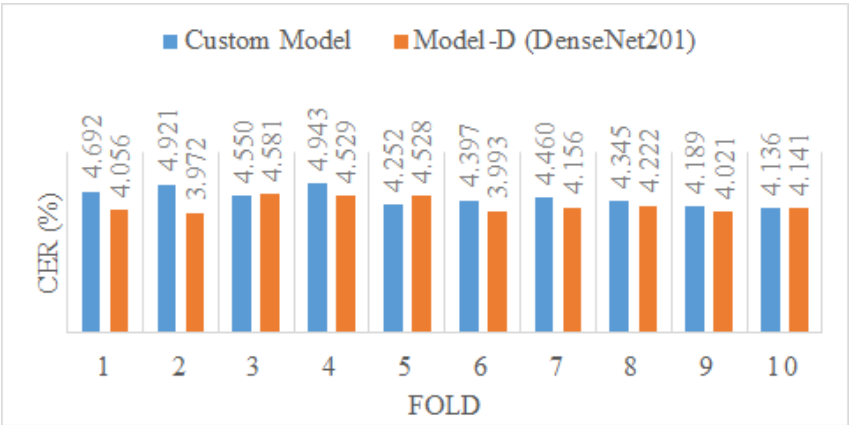


FIGURE 4.3: Graph of average CER while training 10-fold CV.

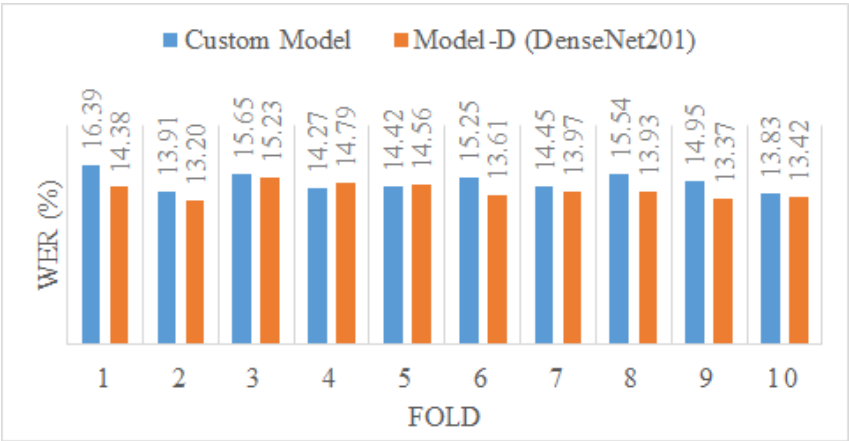


FIGURE 4.4: Graph of average loss while training 10-fold CV.

Generally, *Model D* employing DenseNet201 for feature extraction has considerable performance so far as our experimental results. Table 4.4 also reveals that it is outperforming when compared to the custom model.

TABLE 4.4: 10 fold CV result of the recognition models on the two dataset $[Patience\,(early\,stopping) : 5]$.

Data augmentation	**Input shape**	**Model**	**Test accuracy**(%)	**Loss**	**CER**(%)	**WER**(%)
without	**(32, 128, 1)**	D	84.08	0.731	4.220	14.05
		CUSTOM	82.90	0.882	4.489	14.87
	(64, 256, 1)	**D**	**86.25**	**0.649**	**3.653**	**12.65**
		CUSTOM	84.75	0.664	4.289	13.20
with	**(32, 128, 1)**	D	88.83	0.680	0.765	2.586
		CUSTOM	87.58	0.793	0.807	2.832
	(64, 256, 1)	**D**	**89.83**	**0.621**	**0.725**	**2.584**
		CUSTOM	88.17	0.631	0.829	3.095

4.5 Sample prediction Results

In this section, we have shown some examples of recognition results on sample handwritten Amharic word images taken from the test dataset for visual analysis in Figure 4.5. From Figure 4.5a, it can be seen that for randomly selected word images those our recognition models have predicted wrongly by deleting, substituting or inserting some of the characters in the transcribed text while Figure 4.5b shows some of the correct predictions. Amharic alphabets, the so-called *"Fidel"*, there are characters that have almost similar shapes with the only difference in a single stroke from the base character or from inter-character families. This similarity in shape becomes one of the challenges for Amharic handwritten recognition, unlike Latin script, which has completely different character shapes from one character family to other family. This makes the Amharic handwritten text recognition task very challenging and it needs thousands if not millions of training samples to achieved reliable and robust performance in recognition of Amharic language vocabulary symbols. One of the limitation for this thesis was lack of enough training data, which is containing all the vocabulary symbols in various shapes and types. As it is shown in the Figure 4.5a, the incorrectly predicted character "ውዲ" in the second row is due to its almost similar shape with the character from another family "ዉ". In some of the wrongly predicted symbols, the data augmentation technique creates its own effect when an input word image is dilated it creates a problem for the recognition model to understand the character shape well. For example, in the last row the characters are too much dilated and our recognition model wrongly predict the character "ራ" into two characters "ራ" and "ቦ". In General, most of the failure cases in our recognition model occur due to disconnected character parts during handwriting, cursive nature of handwritten text, and character shape variation due to pre-processing which in cumulatively contribute a character shape difference compared with the ground truth (GT).

(A) Incorrect predictions (B) Correct predictions

FIGURE 4.5: Sample prediction results during model evaluation with the test set.

Chapter 5. Conclusion and Recommendation

This chapter summarizes the thesis 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.

5.1 Conclusion

This thesis presented the development of models for handwritten Amharic word recognition (HAWR) task using deep learning approach. Firstly, we have designed the model with a CNN-BGRU-CTC framework and implemented with deep learning libraries on Google Co-laboratory (COLAB) with GPU. While designing phase, different models have been implemented and tested, to make fine adjustments of the hyper-parameters being assisted with the Bayesian optimization algorithm. Hence, we hyper-tuned and cross-validated the number of CNN layers, filters, kernel size, dropout rate, batch size and RNN units to reach our best performing custom model.

Next, we have compared the non-augmented dataset with its augmented version. To do this experiment, first we regenerated a total of $33\,672$ 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 have used the same model algorithm trained on the augmented dataset like the one used on the non-augmented. Another experiment that we have conducted is the 10-fold cross validation for the entire data. Hence, *Model D* (using DenseNet201) scored an average accuracy of $89.83\%\,(\pm 0.86\%)$ and loss of $0.649\,(\pm 0.051)$ on the augmented data.

At the end, we have concluded that *Model D* is outperforming with WER of 9.000 and CER of 2.506 on the non-augmented dataset as well as WER of 6.833 and CER of 1.819 on the augmented version. Thus, we have achieved up to $4.95\times$ and $1.94\times$ smaller of the CER and WER respectively using the augmented data on the same test set. The model with the designed custom CNN architecture has shown a comparable with the performance obtained when existing commonly-used CNN architectures incorporated into the recognition model as a feature extractor.

5.2 Recommendation

Even-though, this thesis contributed a lot for the attempt to design and develop handwritten Amharic word recognition system, we believe that 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 thesis also made efforts to solve the challenge especially in the experiments conducted on word-image augmentation; because, as the dataset size is increased that the model to be trained with, it is obvious that higher performance can be achieved. In the future use, this dataset size can be increased, or another dataset 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, incorporating some post-processing methods that can fix the model predictions to their correct form, or modifying the pre-processing by adding the state-of-the-art neural networks to correct spellings of words or some feature recovery techniques should be studied in future.

References

[1] Afiqah Amirah Hamzah, Saiful Farik Mat Yatin, Nurul Athirah Ismail, et al. "Data Capturing: Methods, Issues and Concern". In: *International Journal of Academic Research in Business and Social Sciences* 8.9 (20 October 2018), pp. 617629. DOI: 10.6007/IJARBSS/v8-i9/4642. URL: http://dx.doi.org/10.6007/IJARBSS/v8-i9/4642.

[2] Xiaoxue Chen, Lianwen Jin, Yuanzhi Zhu, et al. "Text recognition in the wild: A survey". In: *ACM Computing Surveys (CSUR)* 54.2 (2021), pp. 1–35.

[3] Fitehalew Demilew and Boran Sekeroglu. "Ancient Geez script recognition using deep learning". In: *SN Applied Sciences* 1 (Nov. 2019). DOI: 10.1007/s42452-019-1340-4.

[4] Deepak Sinwar, Vijaypal Dhaka, Nitesh Pradhan, et al. "Offline script recognition from handwritten and printed multilingual documents: a survey". In: *International Journal on Document Analysis and Recognition (IJDAR)* 24 (June 2021). DOI: 10.1007/s10032-021-00365-5.

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

[6] 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.

[7] Rio Anugrah and Ketut Bintoro. "Latin Letters Recognition Using Optical Character Recognition to Convert Printed Media Into Digital Format". In: *Jurnal Elektronika dan Telekomunikasi* 17.2 (2017), pp. 56–62. ISSN: 2527-9955. DOI: 10.14203/jet.v17.56-62. URL: https://www.jurnalet.com/jet/article/view/163.

[8] Najwa Altwaijry and Isra Al-Turaiki. "Arabic handwriting recognition system using convolutional neural network". In: *Neural Computing and Applications* 33 (December 2021). DOI: 10.1007/s00521-020-05070-8.

[9] Shahbaz Hassan, Ayesha Irfan, Ali Mirza, et al. "Cursive Handwritten Text Recognition using Bi-Directional LSTMs: A Case Study on Urdu Handwriting". In: *2019 International Conference on Deep Learning and Machine Learning in Emerging Applications (Deep-ML)*. 2019, pp. 67–72. DOI: 10.1109/Deep-ML.2019.00021.

[10] Wenchao Wang, Jun Du, and Zi-Rui Wang. "Parsimonious HMMs for Offline Handwritten Chinese Text Recognition". In: (March 2018).

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

[12] Edward Ullendorff. "Studies in the Ethiopic Syllabary". In: *Africa: Journal of the International African Institute* 21.3 (1951), pp. 207–217. ISSN: 00019720, 17500184. URL: http://www.jstor.org/stable/1156593.

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

[14] 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.

[15] 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.

[16] 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.

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

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] Théodore Bluche. "Deep Neural Networks for Large Vocabulary Handwritten Text Recognition". In: 2015.

[24] 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.

[25] 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.

[26] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[27] Yufeng Zheng, Clifford Yang, and Aleksey Merkulov. "Breast cancer screening using convolutional neural network and follow-up digital mammography". In: May 2018, p. 4. DOI: 10.1117/12.2304564.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[29] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks". In: *CoRR* abs/1608.06993 (2016). arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993.

[30] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *CoRR* abs/1905.11946 (2019). arXiv: 1905.11946. URL: http://arxiv.org/abs/1905.11946.

[31] 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.

[32] 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.

[33] 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.

[34] 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].

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

[36] 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.

[37] 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.

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

[39] 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.

[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] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] M. Hailegiorgis. *Aristotle Monogragh*. wisdom from Pilate. Addis Ababa, ET, 2010. URL: http://www.aau.edu.et/chls/hailegiorgis-mamo/.

[51] Y. Zemene. *Zoskales Astronomy newsletter*. 2007. URL: https://dirzon.com/Zon/DldAsync/Ftarget/Dtelegram/Azamanayohanese(zosekaalase).pdf.

[52] *Constitution of the Federal Democratic Republic of Ethiopia Proclamation No. 1/1995*. Federal Negarit Gazeta. Addis Ababa, ET, 21 Aug. 1995. URL: https://ethiopianembassy.be/wp-content/uploads/Constitution-of-the-FDRE.pdf.

[53] A. Esubalew. *Thus Spoke Zarathusra*. URL: https://drive.google.com/file/d/1nHYtt2jkYFQMo_mNHTJHEoX_kNRu7aFk/view.

[54] 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.

[55] 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.

[56] 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.

[57] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[58] Jason Brownlee. *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. Machine Learning Mastery. 8January 2019, Retrieved 27 July 2020.

[59] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. “6.2.2.3 Softmax Units for Multinoulli Output Distributions”. In: *Deep Learning, MIT Press* abs/1502.03167 (2016), pp. 180184.

[60] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[61] Yoonsuh Jung. “Multiple predicting K -fold cross-validation for model selection”. In: *Journal of Nonparametric Statistics* 30 (July 2017), pp. 1–19. DOI: 10.1080/10485252.2017.1404598.

[62] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[63] 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 [63].

	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	ሄ	ሄ	ህ	ሆ	ሏ	ሗ	ኋ	ኋ	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	ኳ	ኰ
x	ኸ	ኹ	ኺ	ኻ	ኼ	ኽ	ኾ	ዃ	ዅ	ዃ	ዃ	ዄ
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	ጋ	ጐ
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

Appendix B. CTC–Labels Representation

B.1 CTC–Labels for Amharic Characters

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	ፗ
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	ዖ	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	ሿ	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	ዤ	215	ጢ	253	ፉ	291	7
26	ሜ	64	ቂ	102	ኀ	140	ኳ	178	ዥ	216	ጣ	254	ፊ	292	8
27	ም	65	ቃ	103	ኁ	141	ኸ	179	ዦ	217	ጤ	255	ፋ	293	9
28	ሞ	66	ቄ	104	ኂ	142	ኹ	180	ዧ	218	ጥ	256	ፌ	294	፡
29	ሟ	67	ቅ	105	ኃ	143	ኺ	181	የ	219	ጦ	257	ፍ	295	፤
30	ሠ	68	ቆ	106	ኄ	144	ኻ	182	ዩ	220	ጧ	258	ፎ	296	…
31	ሡ	69	ቋ	107	ኅ	145	ኼ	183	ዪ	221	ጨ	259	ፏ	297	*
32	ሢ	70	በ	108	ኆ	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		

Appendix C. Bayesian Hyper-optimization

C.1 Bayesian Optimization Source Code

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

  input_shape = X_train.shape[1:] # (H, W, C) - input shape
  number_of_classes = len(char_list)+1
  #print(input_shape)
  # parameters for NN that do NOT have to be saved
  maximum_epochs = 100
  early_stop_epochs = 3
  learning_rate_epochs = 3

  # 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,
                   learning_rate,
                   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(num_cnn_blocks),
                   'learning_rate':int(learning_rate),
                   '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}
```

```
    # RNN input length
    size = 32
    train_inp_len=[]
    for i in range(trn_set):
      train_inp_len.append(size)
    train_inp_len=np.asarray(train_inp_len)
    test_inp_len=[]
    for i in range(val_set):
      test_input_len.append(size)
    test_input_len=np.asarray(test_inp_len)


    input_tensor = Input(shape=input_shape)
    x = BatchNormalization()(input_tensor)

    # start of cnn coding
    cnns = dict_params['num_cnn_blocks']

    # 1st iteration for CNN blocks
    for i in range(3*cnns):
      x = Conv2D(filters=((dict_params['num_filters'])*(i+1)),
                 kernel_size=dict_params['kernel_size'],
                 strides=(1, 1), use_bias=True,  padding='same',
                 kernel_regularizer=regularizers.l2(REG),
                 name=f'block1_conv{i+1}')(x)
      x = Activation('relu')(x)
      x = Dropout(dict_params['drop_out'])(x)
      x = BatchNormalization()(x)

      if i % cnns == 0:
        x = MaxPool2D(pool_size=(2,2), name=f'block1_pool{i+1}')(x)

    # 2nd iteration for CNN blocks
    for i in range(3*cnns):
      x = Conv2D(filters=((dict_params['num_filters'])*2*(i+3)),
                 kernel_size=dict_params['kernel_size'],
                 strides=(1, 1), padding='same',
                 kernel_regularizer=regularizers.l2(REG),
                 name=f'block2_conv{i+1}')(x)
      x = Activation('relu')(x)
      x = Dropout(dict_params['drop_out'])(x)
      x = BatchNormalization()(x)

      if i % cnns == 0:
        x = MaxPool2D(pool_size=(2,1), name=f'block2_pool{i+1}')(x)

```

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

    # instantiate and compile model
    model = Model(inputs=input_tensor, outputs=output_tensor)
    #model.summary()
    inputs= model.input
    outputs= model.output
    opt= Adamax(dict_params['learning_rate'])  # default = 0.001
    labels=Input(name='labels',shape=[maxLabelLen],dtype='float32')
    inp_len= Input(name='input_length', shape=[1], dtype='int64')
    label_len= Input(name='label_length', shape=[1], dtype='int64')

    def ctc_lambda_func(args):
      y_pred, labels, inp_len, label_len = args

      return K.ctc_batch_cost(labels,y_pred,inp_len,label_len)

    loss_out= Lambda(ctc_lambda_func,output_shape=(1,),name='ctc')
                          ([outputs, labels, inp_len, label_len])

    #model to be used at training time
    model = Model(inputs=[inputs, labels, inp_len, label_len],
                  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_inp_len,
                     train_label_length], y=np.zeros(len(X_train)),
                     batch_size=dict_params['batch_size'],
                     epochs=maximum_epochs, #validation_split=0.1,
                     validation_data=([X_test, y_test, test_inp_len,
                     test_label_length], [np.zeros(len(X_test))]),
                     #shuffle=True,
                     verbose=0,
                     callbacks=callbacks_list)

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

    # h.history['val_loss']
    validation_loss = np.min(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':(1, 4), # *6
                         'learning_rate':(0.0001, 0.01),
                         'num_filters':(1, 4),  # *32
                         'kernel_size':(1, 4),
                         'num_units':(1, 4),  # *128
                         'batch_size':(1, 4),  # *8
                         'drop_out': (0.2, 0.6)},
                         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+'bayes_results_parameters.pkl')
df_results.to_csv(results_dir + 'bayes_results_parameters.csv')
```

LISTING C.1: Python source code for Bayesian optimization.

C.2 Bayesian Optimization Sample Result for the Custom Model

TABLE C.1: A sample result during hyper-tuning for 50 trials (here, trial 3 is the best).

trial	batch size (×8)	cnn blocks (×6)	dropout rate	CNN filters (×32)	kernel size	learning rate	RNN units (×128)	bayes opt error	epochs	model names
0	1	1	0.38130	1	1	0.00345	2	-166.67	1	hawr_0112504
1	2	3	0.56179	2	1	0.00024	2	-169.20	1	hawr_0002319
2	3	2	0.50007	2	2	0.00868	3	-165.50	8	hawr_4888705
3	2	2	0.54272	2	1	0.00495	2	-144.65	12	hawr_8895593
4	2	3	0.45622	3	2	0.00158	1	-165.69	13	hawr_7888660
5	2	2	0.54107	2	1	0.00490	2	-165.49	3	hawr_4235324
6	3	1	0.33963	3	3	0.00787	3	-168.23	1	hawr_1413807
7	3	3	0.40692	3	1	0.00673	1	-168.78	1	hawr_5504421
8	1	3	0.54044	2	1	0.00466	1	-170.18	1	hawr_5874453
9	2	2	0.38639	3	2	0.00289	2	-167.05	18	hawr_6517570
10	3	1	0.57549	2	2	0.00050	1	-164.16	5	hawr_0074673
11	3	3	0.31867	2	1	0.00999	2	-149.24	8	hawr_7427297
12	2	1	0.50642	1	2	0.00856	2	-169.00	1	hawr_7065799
13	3	1	0.49877	1	1	0.00367	3	-165.98	7	hawr_9321885
14	1	1	0.49906	2	3	0.00410	2	-169.83	1	hawr_4718763
15	2	2	0.41710	1	3	0.00512	1	-168.20	1	hawr_7421182
16	3	2	0.31253	3	2	0.00730	2	-165.34	9	hawr_6470791
17	2	2	0.27829	3	1	0.00797	3	-168.78	4	hawr_3193661
18	2	3	0.41320	2	1	0.00056	2	-164.96	9	hawr_1804471
19	3	2	0.56304	2	3	0.00594	2	-166.94	1	hawr_9883286
20	1	2	0.53821	1	3	0.00302	1	-158.55	8	hawr_0822871
21	1	2	0.24980	3	3	0.00874	1	-162.91	3	hawr_6316071
22	1	1	0.58594	3	2	0.00164	3	-158.96	4	hawr_4027049
23	1	2	0.53265	3	3	0.00826	1	-169.12	1	hawr_2803434
24	1	1	0.46939	3	3	0.00826	2	-169.88	1	hawr_2379754
25	2	1	0.44754	1	2	0.00982	3	-165.75	14	hawr_0168926
26	2	3	0.29664	3	2	0.00752	3	-162.68	5	hawr_1539385
27	3	3	0.30110	1	2	0.00302	1	-164.67	16	hawr_6275462
28	2	3	0.49319	3	1	0.00591	1	-166.53	5	hawr_9702976
29	1	2	0.41063	3	1	0.00778	2	-166.95	6	hawr_1199178
30	1	1	0.43106	2	1	0.00888	1	-166.27	5	hawr_9756732
31	1	1	0.59074	3	3	0.00036	1	-170.81	1	hawr_1569277
32	1	2	0.41936	3	2	0.00106	2	-149.25	6	hawr_4514202
33	3	3	0.47076	3	2	0.00028	3	-165.56	8	hawr_6514668
34	2	2	0.55695	2	1	0.00568	2	-169.74	1	hawr_1179949
35	1	1	0.53929	2	1	0.00304	3	-162.94	13	hawr_0408958
36	2	2	0.37726	3	2	0.00019	3	-157.31	8	hawr_4886011
37	3	1	0.52930	2	1	0.00972	1	-160.04	5	hawr_8611309
38	1	2	0.59544	1	1	0.00149	1	-161.10	6	hawr_9177017
39	1	1	0.39657	2	1	0.00130	3	-156.55	5	hawr_5035048
40	2	3	0.37782	2	3	0.00073	1	-169.49	1	hawr_1432236
41	2	1	0.48703	3	1	0.00930	1	-169.00	1	hawr_2530413
42	1	2	0.52780	2	2	0.00506	1	-168.19	1	hawr_5785484
43	2	1	0.31183	1	1	0.00622	2	-170.38	1	hawr_4482785
44	3	1	0.31487	2	1	0.00679	1	-167.53	1	hawr_5052025
45	1	2	0.23887	2	1	0.00902	2	-164.27	2	hawr_5581458
46	2	3	0.58285	2	1	0.00518	3	-159.91	10	hawr_6649196
47	2	3	0.38500	3	2	0.00863	3	-169.09	1	hawr_4990366
48	2	1	0.44548	1	1	0.00558	3	-160.11	3	hawr_8886166
49	2	2	0.44279	3	1	0.00746	3	-162.41	15	hawr_6671779

Total elapsed time = 214.57529 minutes.