Handwritten Ge’ez Digit Recognition Using Deep Learning

Mukerem Ali Nur

A Thesis Submitted to Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master’s in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

June 2022

Adama, Ethiopia

Handwritten Ge’ez Digit Recognition Using Deep Learning

Mukerem Ali Nur

Advisor: Dr. Rajesh Sharma

A Thesis Submitted to Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master’s in Computer Science and Engineering

Office of Graduate Studies
Adama Science and Technology University

June 2022
Adama, Ethiopia

APPROVAL PAGE

I, the advisors of the thesis entitled "**Handwritten Ge'ez Digit Recognition Using Deep Learning**" and developed by Mukerem Ali Nur, hereby certify that the recommendation and suggestions made by the board of examiners are appropriately incorporated into the final version of the thesis.

Dr. Rajesh Sharma	_____	_____
Major Advisor/Supervisor	Signature	Date

We, the undersigned, members of the Board of Examiners of the thesis by Mukerem Ali Nur have read and evaluated the thesis entitled "**Handwritten Ge'ez Digit Recognition Using Deep Learning**" and examined the candidate during the open defense. This is, therefore, to certify that the thesis is accepted for partial fulfillment of the requirement of the degree of Master's of Science in Computer Science and Engineering.

_____	_____	_____
Chair Person	Signature	Date
_____	_____	_____
Internal Examiner	Signature	Date
Dr. Getachew Mamo	_____	_____
External Examiner	Signature	Date

Finally, approval and acceptance of the thesis is contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

_____	_____	_____
Department Head	Signature	Date
_____	_____	_____
School Dean	Signature	Date
_____	_____	_____
Office of Postgraduate Studies, Dean	Signature	Date

DECLARATION

I hereby declare that this Master's Thesis entitled "**Handwritten Ge'ez Digit Recognition Using Deep Learning**" is my original work. That is, it has not been submitted for the award of any academic degree, diploma, or certificate in any other university. All sources of materials that are used for this thesis have been duly acknowledged through citation.

Mukerem Ali Nur	_____	_____
Name of Student	Signature	Date

RECOMMENDATION OF ADVISORS/SUPERVISORS

I, the advisor of this thesis, hereby certify that I have read the revised version of the thesis entitled "**Handwritten Ge'ez Digit Recognition Using Deep Learning**" prepared under my guidance by Mukerem Ali Nur. submitted in partial fulfillment of the requirements for the degree of Master's of Science in Computer Science and Engineering. Therefore, I recommend the submission of a revised version of the thesis to the department following the applicable procedures.

Dr. Rajesh Sharma	_____	_____
Major Advisor/Supervisor	Signature	Date

ACKNOWLEDGMENT

In the name of Allah, the Most Gracious and the Most Merciful.

All praises to Allah and His blessing for the completion of this thesis. I thank God for all the opportunities, trials and strength that have been showered on me to finish writing the thesis. I experienced so much during this process, not only from the academic aspect but also from the aspect of personality.

First, I would like to sincerely thank my advisor Dr. Rajesh Sharma for his guidance, understanding, and patience and most importantly, he has provided positive encouragement to finish this thesis. It has been a great pleasure and honor to have him as my advisor. The deepest gratitude for Dr. Mesfin Abebe (Ph.D.) for his worth comments, guidance, and kind support.

I wish to thank all the members of CSEC ASTU who have helped me in submitting and partially gathering the data that were used for training. I would also want to extend my special thanks to Mebatsion Sahle, Yohannes Melese, Abdi Adem and Abdi Kasim for their support.

May God shower the above-cited personalities with success and honor in their life.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF SAMPLE CODES

LIST OF ACRONYMS AND ABBREVIATIONS

1D	One Dimension
2D	Two Dimension
3D	Three Dimension
ADAM	Adaptive Moment Estimation
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
ASTU	Adama Science and Technology University
CNN	Convolutional Neural Network
Conv2D	Convolutional Layer with 2D
ConvNets	Convolutional Networks
CPU	Central Processing Unit
CSEC	Computer Science and Engineering Club
DCNN	Deep Convolutional Neural Network
DL4J	Deep Learning for Java
DNN	Deep Neural Network
EMML	Ethiopian Manuscripts Microfilm Library
ENALA	Ethiopian National Archives and Library Agency
EOTC	Ethiopian Orthodox Tewahedo Church
GB	Giga Byte
GHz	Giga Hertz
GPU	Graphics Processing Unit
HD	Hard Drive
HDD	Hard Disk Drive
IDE	Integrated Development Environment
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
JPEG	Joint Photographic Experts Group
JPG	Joint Photographic Experts Group
KNN	K-Nearest Neighbor
LR	Learning Rate
LSTM	Long Short Term Memory

MD-LSTM	Multi-Dimensional Long Short Term Memory
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology
NADAM	Nesterov-accelerated Adaptive Moment Estimation
NIN	Network In Network
OCR	Optical Character Recognition
PC	Personal Computer
PNG	Portable Network Graphics
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RGB	Red Green Blue
RNN	Recurrent Neural Network
RTX	Ray Tracing Texel eXtreme
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
TPU	Tensor Processing Unit
UML	Unified Modeling Language
UNESCO	United Nations Educational, Scientific and Cultural Organization
USB	Universal Serial Bus
WD	Western Digital
YOLOv3	You Only Look Once, Version 3

LIST OF NOTATIONS AND SYMBOLS

Notation	Meaning
Y	Indicates the image domain
≈	Indicate approximate to
$\sum$	Indicates the summation
exp	Indicates the exponential
n	Total number of training inputs
a	Actual output
w	Weight of the neural network
b	Bias of the neural network
$\frac{\partial C}{\partial w}$	Partial derivative of Cost function respect to weight of the network
$\frac{\partial C}{\partial b}$	Partial derivative of Cost function respect to bias of the network
η	Learning rate
δ	Back-propagation of the network
m	Batch size
f'(z)	First derivate of of the output of the network

ABSTRACT

Amharic language is the second most spoken language in the Semitic family after Arabic. In Ethiopia and neighboring countries more than 100 million people speak the Amharic language. Many historical documents are written using the Ge'ez script. Digitizing historical handwritten documents and recognizing handwritten characters is essential to preserving valuable documents. Handwritten digit recognition is one of the tasks of digitizing handwritten documents from different sources. Currently, handwritten Ge'ez digit recognition researches are very few, and there is no available organized dataset for the public researchers. Convolutional Neural Network (CNN) is preferable for pattern recognition like in handwritten document recognition by extracting a feature from different styles of writing. In this thesis, the proposed model is to recognize Ge'ez digits using CNN. Deep neural networks, which have recently shown exceptional performance in numerous pattern recognition and machine learning applications, are used to recognize handwritten Ge'ez digits, but this has not been attempted for Ethiopic script. Our dataset, which contains 51,952 images of handwritten Ge'ez digits collected from 524 individuals, is used to train and evaluate the CNN model. The application of CNN improves the performance of several machine-learning classification methods significantly. Our proposed CNN model has an accuracy of 96.21% and loss of 0.2013. In comparison to earlier research works on Ge'ez handwritten digit recognition, we were able to attain higher recognition accuracy using the developed CNN model.

***Keywords**: CNN, Feature Extraction, Digit Recognition, Pattern Recognition, Ge'ez Handwritten Digit Recognition, Deep Learning, Handwritten Recognition*

CHAPTER ONE

1. INTRODUCTION

1.1 Background of study

Amharic language is the only African language with its alphabet and writing system while most the other African languages use Latin and Arabic alphabets for their writing system (Ashagrie & Boran, 2019). The Federal Democratic Republic of Ethiopia and other regional states use the Amharic language as their official working language. It is the mother language for over 50 million people and the second language for over 100 million people in Ethiopia (Ashagrie & Boran, 2019). Arabic is the only Semitic language spoken more than Amharic in the world. Some people in neighboring countries like Eritrea, Djibouti, and Somalia (Belay et al., 2020) also speak Amharic.

There are many historical documents written in Ge'ez scripts found in Ethiopia. There are around 80 different languages spoken in Ethiopia, with up to 200 dialects. The Ge'ez alphabet is used as the writing system in some languages. Amharic, Ge'ez, and Tigrinya are the most spoken language in Ethiopia that use the Ge'ez alphabet (Ashagrie & Boran, 2019).

Ge'ez script consists of 265 characters including 27 labialized characters (characters representing 2 sounds), 20 symbols for numerals, and 8 punctuation marks (Gondere et al., 2019). Our research focused on only the Ge'ez digits. Ge'ez, numerals have been used in Ethiopian calendars, Amharic bibles, and historical documents. Ge'ez, numbers consist of twenty different symbols to represent the numerical values. Unlike Latin numbers 0 is not represented by any symbol. Twenty numbers are represented by independent symbols such as 1-9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, and 10000 as shown in figure 1.1. Other numbers are represented by the combination of those twenty symbols. Each digit symbol has a dash (horizontal line) above and below the digit character.

Figure 1.1 Ge'ez number representations (Beyene, 2019)

In the last two decades, digitizing historical documents to electronic format has been done to preserve the documents. Digitizing documents is an efficient way to preserve valuable historical documents and information. This makes these documents to be easily accessible. Historical manuscripts are available for the research community and the public which gives efficient indexing, searching, and information retrieval (Kusetogullari et al., 2020).

Great research works are being done on machine learning to achieve and improve the efficiency of pattern recognition from the data. One of the uses of pattern recognition is handwritten character recognition. Handwritten document recognition is to understand and convert handwritten image documents to readable text formats which include characters, numerical values, words, sentences, and paragraphs (Kusetogullari et al., 2020).

Handwritten recognition extracts textual characters, words, sentences, and numerical digits from handwritten document images (Kusetogullari et al., 2020). Recognition of handwritten documents is challenging and complex because of some factors. Such as great variation of handwriting styles, touching strings, various character scales, different orientations angle of the characters, ink flying in time, character overlap, bleed-through, faint characters, and other factors (Kusetogullari et al., 2020; Nair et al., 2021).

Digit detection and recognition is one of the main problems in handwritten character recognition. A machine can detect and recognize different human handwritten digits from different sources like papers, images, documents, etc. (Nair et al., 2021). Handwritten digit recognition has many applications in various sectors of our daily life. Postal mail sorting, bank cheque processing, document indexing based on dates, and number plate recognition are the main application of handwritten digit recognition.

Random Forest, SVM, KNN, and other machine learning techniques have been developed to recognize handwritten digits. Deep learning methods like CNN have the highest accuracy when compared to the most commonly used machine learning algorithms for handwritten digit recognition (Dutt & AashiDutt, 2017). Pattern recognition and large-scale image classification are both done with CNN. Handwriting character recognition is a research field in computer vision, artificial intelligence, and pattern recognition (Ashagrie & Boran, 2019; Nair et al., 2021). It might be claimed that a computer application that conducts handwriting

recognition can acquire and recognize characters in photographs, paper documents, and other sources, and convert them to electronic or machine-encoded form.

Nowadays, deep learning is becoming a popular technique to learn to recognize patterns and deep patterns and extract them. It has a deep learning level to generate patterns from a given dataset. It is an amazing algorithm with diverse libraries to extract patterns, recognize from images, and classify them. Among the deep learning algorithms, CNN is efficient and has good image classification, image recognition, pattern recognition, feature extraction, and so on.

1.2 Motivation of the Study

Ge'ez manuscripts and scrolls contain a wealth of knowledge. For instance, Abushair or Bahre Hasab (**ባህረ ሐሳብ (የቀመርና የሥነፈለክ ምሥጢር)**) - This Ethiopian manuscript, is written in brana (**ብራና**) in the languages of Amharic and Ge'ez, is a book explaining the mathematical system for fixing the movable feasts and fasts of the EOTC. EOTC bible also uses Ge'ez numbers for its chapters. Another example is the Ethiopian calendar, which is prepared with Ge'ez numbers.

In almost all manuscripts of Ge'ez there are Ge'ez number symbols, digitizing them makes it easier for recognition and a better understanding of the scripts. To understand all the knowledge passed from our forefathers, simplifying the accessibility, providing search ability of Ge'ez numbers as a text, and learning them is vital. To keep our history, the manuscripts in text format are essential to preserving them and data extraction. On a small scale, it seems easier to extract the symbols but when it comes to large data of brana this is a good start to do further research.

There was a shortage of public digital datasets when doing this research. As a result, any researcher who is interested in this topic must collect data manually. This research will help people who require a digitized dataset of Ge'ez numbers to overcome this problem. This study is a first step in demonstrating how image classification for Ge'ez numerals can be done. To go over combined numbers and Ge'ez letters, more research is needed.

1.3 Statement of the Problem

In Ethiopia there are many manuscripts, basically they are handwritten. So to preserve the data digitally, some research has been done by many Ethiopians and foreign elites so far.

Even if some research has been done on handwritten character recognition in Ge'ez documents, they mainly focused on alphabets and ignore the digits, and there are only a few numbers of researches done on handwritten Ge'ez digit recognition. The two main challenges in Ge'ez handwritten digit recognition are the lack of organized data and the complexity of Ge'ez digits. While conducting this research, I have found only one research that focuses on Ge'ez digits, (Beyene, 2019). This research performance of detecting and classifying images is around 89.88%, the dataset he used is small and has 560 instances, which can lead to inconclusive results. The initiative of doing this project is to address the following two problems. In the world, there is not enough research done on Ge'ez digits, yet. The other problem is that there is a lack of organized datasets online for handwritten Ge'ez digits.

1.4 Research questions

In this study, an attempt is made to answer the following questions.

RQ1. What are the different techniques and methods used in handwriting digit recognition?

RQ2. How can the performance of handwriting recognition systems be improved using deep learning?

RQ3. What parameters and hyper-parameters improve the recognition model?

1.5 Objective of the Study

1.5.1 General objective

The general objective of the study is to develop a model, which can recognize handwritten Ge'ez digits using deep learning.

1.5.2 Specific Objectives

The following specific objectives are addressed to achieve the general objective.

- ➢ To collect handwritten Ge'ez digit data.
- ➢ To convert the handwritten collected data to an electronic form (digitization).
- ➢ To design a pre-processing technique to enhance the quality of the data.
- ➢ To train the proposed algorithm with the dataset.
- ➢ To evaluate the performance of the model with a test data set using different metrics.

1.6 Scope and Limitations of the Study

1.6.1 Scope of the Study

Within the time and resources available, the scope of the thesis study comprises the following:

- Recognize handwritten Ge'ez digits using deep learning approaches to extract features from the data.
- Augment the data to increase the amount of the data and increase the variety of the data for different handwriting styles.
- The research works only for the handwriting of one digit. It does not consider the multi-handwritten digits.
- Use a detection-free recognition approach. The study does not focus on the detection of the digits.

1.6.2 Limitation

Some of the limitations faced in the doings of this research work are listed as follows:

- There is no public available dataset for Ge'ez numbers. One of the limitation for this research is the availability of the organized data.
- The collected data amount is enough to train the model, but it is not a large dataset.
- Students dominate the respondent of the data gathering. Most of the respondent is student, so the model is performed well for the students and for other individual group the model does not perform well like the students.
- The dataset does not include the historical document and manuscript images. The collected data is only from individuals not including other sources.
- There are only a few researches done on this topic for Ge'ez handwritten digit recognition.

1.7 Contribution and Significance of the Study

1.7.1 Contribution of the Study

The contribution of this study is listed as follows

➢ Introduce a new public dataset for Ge'ez handwritten digits for future researchers. This research create a new dataset for which contains 52,400 instances of images of different individuals handwriting styles.

- ➢ The other contribution of the research is develop a new approach of deep learning model for recognition of Ge'ez digit. It is a new model for the handwritten digit.
- ➢ Performing analysis to identify the gaps in the previously proposed Ge'ez handwritten digit recognition research.
- ➢ Comparing the state-of-the-art deep learning approaches to identify the best approach for the Ge'ez handwritten digit recognition. For our dataset compare the performance of different deep learning algorithms related to the problem and select the best algorithm based on evaluation metrics.

1.7.2 Significance of the study

The National Library of Ethiopia has a rich collection of 859 manuscripts, mainly in Ge'ez but also in Arabic and Amharic. Up to the number 617, they are described in a manual catalog, in Amharic, available in the reading room in the Manuscripts section of the department of Ethiopian Studies.

Some of them have been registered in UNESCO's Memory of the World program because of their antiquity, their rareness, and/or their interest in the history of the Ethiopian State. Ancient volumes are: n° 132, a Lectionary of the Holy Week from the 15th century; n° 27, Epistles of Saint Paul from Hayq, 15th century; n° 28, the famous illustrated Gospel of Täsfana Krestos from Hayq, 14th century.

The collection of microfilms in ENALA includes all the volumes of the EMML collection which is 9 238 volumes. It also comprises a few copies of Ethiopian manuscripts from the British Library, as well as 111 manuscripts preserved in Israel, at the Jerusalem Ethiopic Manuscript Microfilm Library or originating from the Faitlovitch Library in Tel Aviv. The digitization campaign called Ethiopian Manuscripts Microfilm Library (EMML) has been done over about 20 years (1973-1991), as a result of a collaboration between the Patriarchate of the Ethiopian Orthodox Täwahdo church, the Ethiopian Ministry of Culture, and the American University of Saint-Louis, Collegeville.

This thesis is significant for digitizing valuable documents found in different institutions in Ethiopia. ENALA and other institutions are the end-user of this work. In digitizing Ge'ez documents one of the tasks is to recognize Ge'ez digits found in the documents. Converting historical or any document to electronic format is done by extracting the textual or features

of the character, word, or numerical digits from handwritten image documents. So handwritten Ge'ez digit recognition is used in digitizing Ge'ez documents.

Digital documents are searched using different techniques to extract specific information in a short period of time. One of the techniques of searching is indexing the document by using different index methods like page number, dates, keywords, etc. Indexing a document using a date like birth date, marriage date, death date, document date, or any event date found in a document is one of the techniques. Dates are a crucial part of indexing because it relates to events.

1.8 Organization of the Thesis

This research work is divided into seven chapters, which are briefly discussed below.

Chapter One: This chapter provides an outline of the research work as well as justifications for how and what will be accomplished.

Chapter Two: This chapter explains the background literature and related works regarding deep learning, CNN, convolution layer, and handwritten digit recognition.

Chapter Three: This chapter covers research methodologies such as data collection, data preprocessing, design tools, development frameworks and platforms, and evaluation methods.

Chapter Four: Using different block diagrams, pseudo-codes, and flow charts of the implementation with corresponding mathematical descriptions, this chapter aims to provide comprehensive information on how the proposed solution attempts to address the problem.

Chapter Five: This chapter explains how to implement the proposed solution to the problem into action. It also contains a code snippet for the proposed solutions as well as the environment setup.

Chapter Six: The outcomes of the proposed solution are discussed in this chapter, along with a comparison of the results utilizing figures, graphs, and tables.

Chapter Seven: The research is summarized in this chapter. The conclusion of the work is given, as well as suggested future works.

CHAPTER TWO

2. LITERATURE REVIEW AND RELATED WORKS

2.1 Introduction

Handwritten character and digit recognition works are done in different languages to improve the efficiency of the recognition when they digitize historical and handwritten documents (Granell et al., 2019). Digit recognition is a well-known problem that has been used to document indexing using dates such as document date, birth date, marriage date, and death date (Kusetogullari et al., 2020). Digit recognition and detection have been utilized in a variety of applications, including automated reading of the number of bank cheques, postal numbers and codes, tax forms, and document indexing based on dates (Elitez, 2015). There are two types of architectures for handwritten digit string recognition. The two strategies for recognizing the digit string are detection-free and segmentation-based recognition (Ribas, 2015). In segmentation based the system first detect the numerical string that may contain multiple digits. Splitting digits should be done before a recognition to isolate each digit (Shi, 2016; Saabni, 2016). But detection free recognition approach recognizes each digit without any splitting and detection pre-processes (Choi & Oh, 1999; Ciresan, 2008).

Deep learning algorithms are well known for complex pattern recognition like feature extraction in handwritten digit recognition. CNN is the most well-known deep learning technique being used for image classification problems such as face detection, pattern recognition, object detection, etc. (Siddique et al., 2019; Krizhevsky et al., 2012). Manufacturing companies also use CNN classification for fault detection in the semiconductor manufacturing process (Lee et al., 2017).

A pixel image is used as input for the handwritten digit classification model. The model is trained using gradient descent and back-propagation algorithms. CNN has similar architecture as ANN. In these algorithms, there is an input layer, an output layer, and multiple hidden levels between the input and output layers (Siddique et al., 2019). Each layer consists of several neurons. The input of a neuron in the following layer is the weighted total of all the neurons in the previous layer, with a biased value added (Siddique et al., 2019). The cost

function is minimized by updating the backpropagation using shared weights and biases, which improves model performance (LeCun, 2015; Russell & Norvig, 2016).

Many researchers focus this area and they are striving to develop different models and algorithms to increase the accuracy of the recognition system (Siddique et al., 2019). They also create a dataset for the public and research communities.

2.2 Deep Learning

Deep learning is a popular field of machine learning that uses hierarchical structures to learn high-level abstractions from data. According to (LeCun et al., 2015 and Krizhevsky et al., 2012), The availability of technology CPUs, GPUs, and Hard drives, among other things), machine learning algorithms, and large data, such as MNIST handwritten digit data sets and ImageNet data, are all factors in deep learning's success.

Handwritten digit recognition, facial recognition, computer vision, audio and visual signal analysis, voice recognition, disaster recognition, and automated language processing are all areas where deep learning is applied (Hinton et al., 2006). The accuracy of 98.67% for 10 classes of crop pest images with a difficult field background is valuable in agriculture for reducing damage caused by agricultural pests and improving agricultural productivity and crop storage (Luo et al., 2017). Grinbat and his colleagues (Grinblat et al., 2016) use DCNN to classify three different legume species in the same region: white beans, red beans, and soybeans. Lu and his colleagues (Lu et al., 2017) used DCNN techniques using image recognition to identify rice illnesses, with an accuracy of 95.48% using 500 natural photos of infected and healthy rice leaves and stems. Nogueira and his colleagues (Nogueira et al., 2017) investigate three different methodologies for remote sensing scene classification: complete training, fine-tuning, and employing ConvNets as feature extractors.

Deep learning is a type of Neural Network Algorithm that takes data as an input and processes it via a series of non-linear transformation layers to get an output classification. Previously, in traditional pattern recognition, as shown in figure 2.1, the feature extraction procedure was carried out by a subject matter expert and/or a programmer, and the resulting list was then presented to a traditional network neuron for the classification of input data. The handcrafted feature extractor process is time-consuming, needs input data understanding, and takes a long time. This painful step is avoided with deep learning.

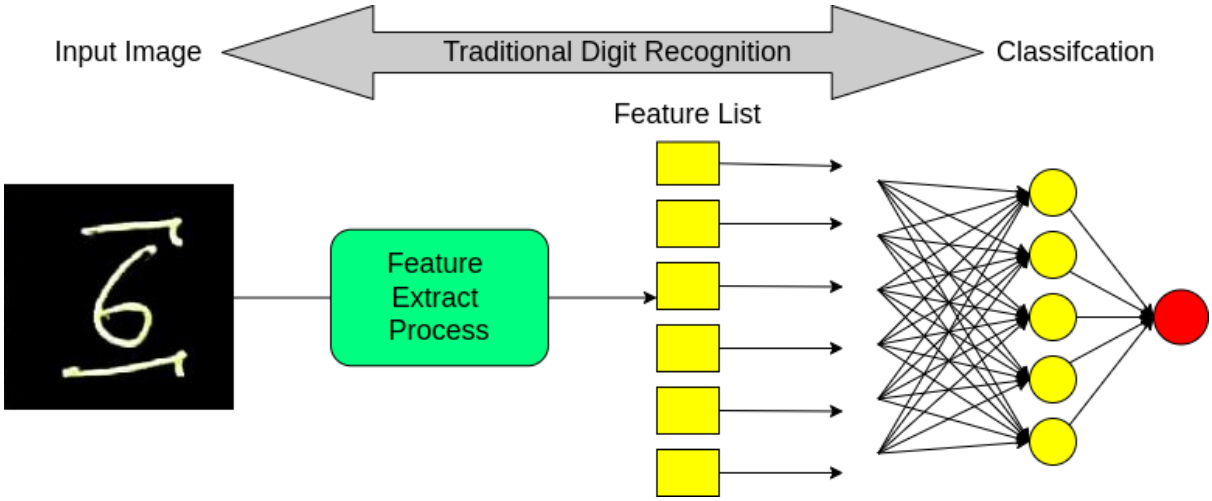


Figure 2.1 Traditional digit Recognition: Fixed/Handcrafted Feature Extractor

The automated extraction of features is a key characteristic of deep learning systems as shown in figure 2.2. This signifies that the algorithm grasps the important features necessary for the problem's solution automatically. This eliminates the need for the expert to explicitly choose features, as is the case with standard pattern recognition systems (Figure 2.1). This can be used for problems that are supervised, unsupervised, or semi-supervised. If it has more than one stage of non-linear feature transformation, it is considered deep (LeCun et al., 2015). The deep learning neural network comprises at least three hidden layers, generally multiple layers. Each hidden layer, which is made up of a group of neurons, is in charge of learning a unique collection of characteristics depending on the output. Each hidden layer, which is made up of a group of neurons, is in charge of training a unique collection of characteristics depending on the preceding layer's output (Figure 2.2). As the number of hidden layers grows, so does the data's complexity and abstraction.

2.3 Convolutional Neural Network (CNN)

CNN's, Restricted Boltzmann Machines, Autoencoders, and Sparse Coding are four categories that some academics use to categorize deep learning methods (Guo et al., 2016). CNN (Figure 2.2) is extensively adopted for image classification with high performance by making convolutional networks fast to train (Krizhevsky et al., 2012; Barbedo, 2013).

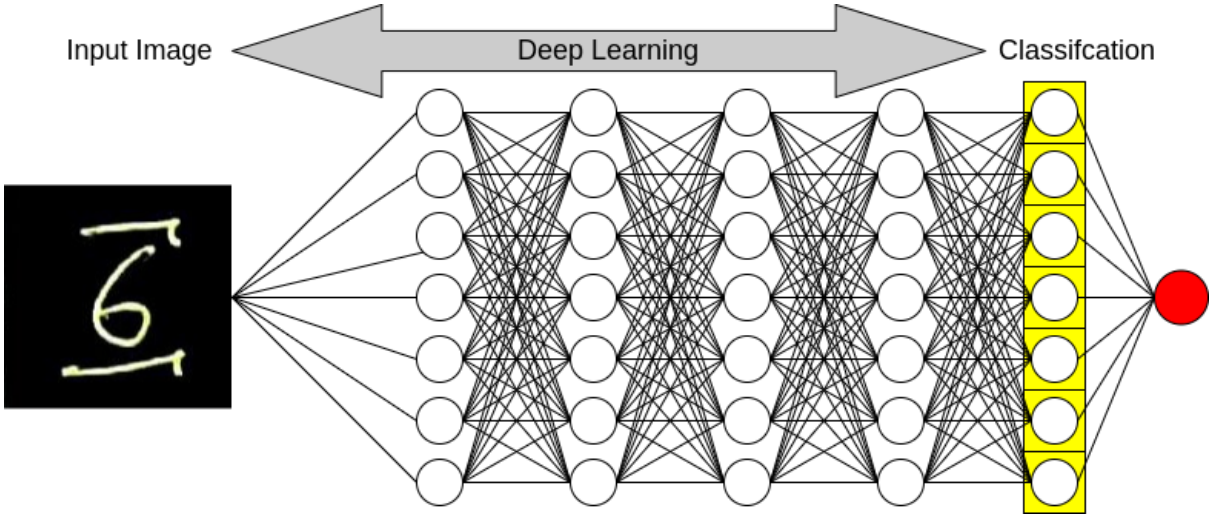


Figure 2.2 Deep Learning Neural Network

With architecture that have multiple hidden layers that are learned, CNN techniques are widely used. A forward stage and a backward stage are used to train the network. The forward stage's main purpose is to represent the input image using the current layer parameters (weights and bias). The first layers find for low-level features such as edges, lines, and corners, while the second and third layers find for mid-level and high-level features such as structures, objects, and forms (LeCun et al., 2015).

The loss cost to the ground truth labels is then calculated using the prediction result. Second, the backward stage uses chain rules to compute the gradient of each parameter based on the loss cost. The gradients are used to update all of the parameters before moving on to the next forward computation. The network learning can be terminated after several iterations of the forward and backward stages.

Because the term "convolution" is used to describe mathematical concepts and ideas about feature transformation strategy and process., it would be helpful to clarify what it means. In mathematics, specifically algebraic topology, convolution is a mathematical transformation of two functions (g and h) that gives a third function that is commonly considered to be a transformed version of one of the original functions.

Convolution, often known as g*h, is a function defined by (Hirschman and Widder, 2017) in the case of two real or complex functions, g and h:

$$(g * h)(x) = \int_{-\infty}^{\infty} g(x - t)\, h(t)\, dt \qquad (2.1)$$

To ensure that the qualities of their corresponding geometric images are retained, the convolution fulfills several algebraic properties (e.g., commutativity, associativity, distributivity, and multiplicative identity).

Commutativity $g * h = h * g$ (2.2)

Associativity $g * (h * u) = (g * h) * u$ (2.3)

Associativity with scalar multiplication

$a (g * h) = (ag) * h$ (2.4)

Distributivity $g * (h + u) = (g * h) + (g * u)$ (2.5)

As shown in figure 2.4, CNN's use multiple cascaded convolution kernels with deep learning applications in artificial intelligence. A discrete convolution is a formal name for the filtering process performed by a feature map. Discrete convolution may be thought of as matrix multiplication. The matrix, has numerous elements that must be equal to other ones.

The convolution of two finite sequences is calculated for functions defined on the set of integers by extending the sequences to finitely applicable functions on the set of integers (e.g., a finite summation can be used for finite support). To define discrete convolution, use the following formula (Hirschman and Widder, 2012):

$$(g * h)(n) = \int_{m=-\infty}^{\infty} g(n - m)\, h(m)\, dt \quad (2.6)$$

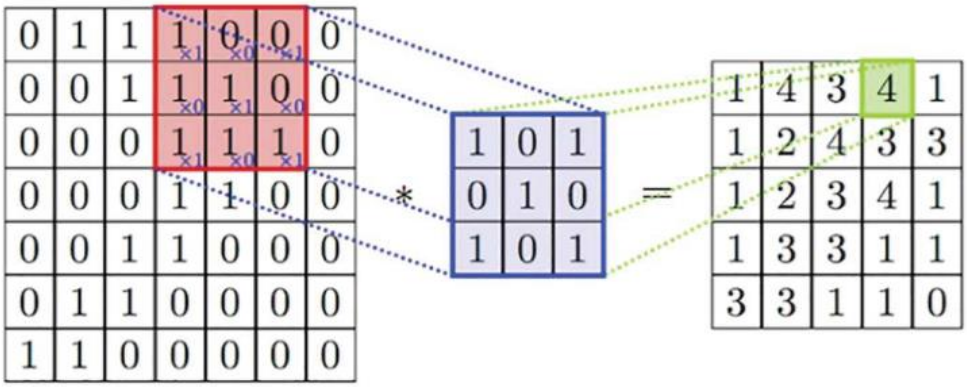


Figure 2.3 Convolution process (Traoré et al., 2018)

Convolution is done in image processing by performing a mathematical operation between matrices representing a kernel and an image, as shown in figure 2.3. Each image element is convoluted by adding it to its closest neighbors, which are weighted by the kernel (Traoré et al., 2018).

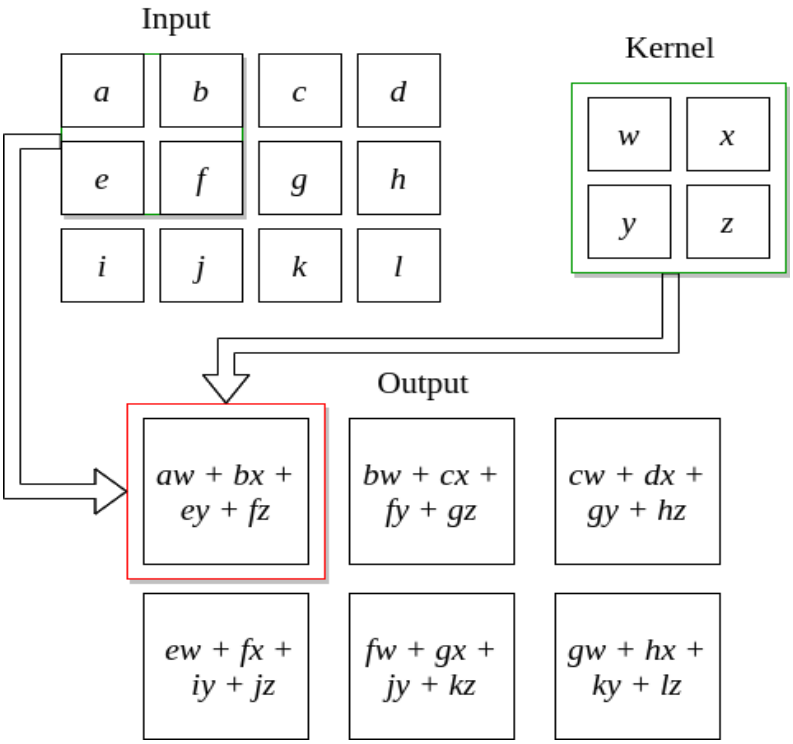


Figure 2.4 A convolution with a kernel size of 2 x 2

To determine the values of a specific pixel in the output image, each kernel value is multiplied by the corresponding input image pixel values. Then add the multiplication result of each kernel value. The pseudo-code shown below can be used to express this algorithmically:

Algorithm 2.1 Convolution operation

```
for each image row in input image:
    for each pixel in images row:
        set accumulator to zero
        for each kernel row in the kernel:
            for each element in the kernel row:
                if element position corresponding to pixel position then
                    multiply element value corresponding to the pixel value
                    add the result to the accumulator
                endif
            endfor
        endfor
    endfor
endfor
set output image pixel to accumulator
```

The building blocks of basic CNN architecture are the convolution layer, ReLU layer, pooling layer, and fully connected layer, as described in Table 2.1.

2.3.1 Convolution Layer

Convolutional layers provide the true potential of deep learning, particularly for image and pattern recognition. It is the most crucial and primary layer. This layer uses a CNN to convolve both the whole image and the intermediate feature maps, resulting in multiple feature maps. A feature map is a map that connects input layers to hidden layers.

The depth, stride, and zero-padding are the three hyper-parameters that determine the size of the convolutional layer's output volume.

Depth: The depth of the output volume determines the number of neurons in the layer that connect to the same region of the input volume. All of these neurons will learn to respond to different input attributes by activating.

Table 2.1 The parameters of different layers of a CNN architecture.

Layers	**Parameters descriptions**
Input	Image files (png, jpg, jpeg, etc.) are defined in grayscale format, which can be represented as a 2D matrix.
Conv(i, o, w, h, sw, sh, zw, zh)	i = with the given image, the number of predicted input channels o = after convolution, the number of output channels (Conv) phase w = The convolution's kernel width h = The convolution's kernel height sw = The width size of the convolution's step (stride) sh = The height size of the convolution's step (stride) zw = Additional zeros on both sides of the width axis added to the input image data zh = Additional zeros on both sides of the height axis added to the input image data
ReLU(r)	If the input is less than 0, the output of a ReLU is 0; otherwise, the output remain the same.

MaxPooling(w,h,u,v)	A max-pooling operation that examines w x h windows and determines the maximum stride length by u x v. w = The filter (kernel) width of pooling h = The filter (kernel) height of pooling u = The size of the pooling step's width (stride) v = The size of the pooling step's height (stride)
FullyConnected(x,y)	x = Input image size e.g. 1x32x32 (1 color channels, 32 pixels of height and 32 pixels of width) y = Number of output classes less than input image size
Loss (x, y)	The loss function calculates a number that measures the model's performance by combining the predicted and true labels. x = the predicted label y = true label
Output	Classification and pattern analysis (e.g. image recognition, video analysis)

Stride: How depth columns are assigned around the spatial dimensions is determined by stride (width and height). The filters are moved one pixel at a time when the stride value is one. This results in a lot of overlapping receptive fields between the columns, as well as a lot of output volume. When we slide the filters around with a stride of 2, they leap 2 pixels at a time. The receptive fields will overlap less, resulting in a smaller output volume with reduced spatial dimensions.

Zero-padding: Zero-padding is the third hyper-parameter, and it's used to pad the input with zeros on the input volume's boundary. The regulation of the output volume spatial size is dealt with via zero padding. It is occasionally necessary to precisely preserve the spatial size of the input volume. For example, the input volume is 28x28x3. By padding two borders of zeros around the volume, we get a 32x32x3 volume. When we apply the convolution layer with our 5x5x3 filters and a stride of 1, we'll get a 28x28x3 output volume.

The convolution operation has three key advantages (Zeiler and Fergus, 2014)

- The weight sharing procedure reduces the number of parameters in the same feature map.
- Local connection learns correlations between pixels in close surroundings.
- Object location invariance.

The Network In Network (NIN) technique (Lin et al., 2013) proposes replacing linear filters with nonlinear neural networks by replacing the traditional layer with a tiny multilayer perceptron consisting of many fully linked layers with nonlinear activation functions. In terms of image classification, this technique performs well.

2.3.2 ReLU Layer

The Rectified Linear Units Layer is what it's called. This layer of neurons applies the non-saturating non-linearity function, often known as the loss function:

$$f(x) = max(0, x) \quad (2.7)$$

It provides the nonlinear features of the decision function and the overall network without affecting the receptive fields of the convolution layer. We have saturated nonlinear functions that are significantly slower (Krizhevsky et al., 2012). The tanh function is also available:

$$f(x) = tanh(x) \quad (2.8)$$

Alternatively, the logistic sigmoid function:

$$f(x) = \frac{1}{1+e^{-x}} \quad (2.9)$$

The neural network is trained many times faster as a result of ReLU, with no notable difference in generalization accuracy.

2.3.3 Pooling Layer

Its task is to reduce or simplify the spatial dimensions of information obtained from feature maps. Average pooling, L2-norm pooling, and max pooling (Scherer et al., 2010) are the three types of pooling. Max pooling is the most common because of its speed and superior convergence. As seen in figure 2.5, this essentially requires a filter (typically of size 2x2) and a stride of the same length. It then applies the filter to the input volume and, with the filter moving around, outputs the highest number in each sub-region.

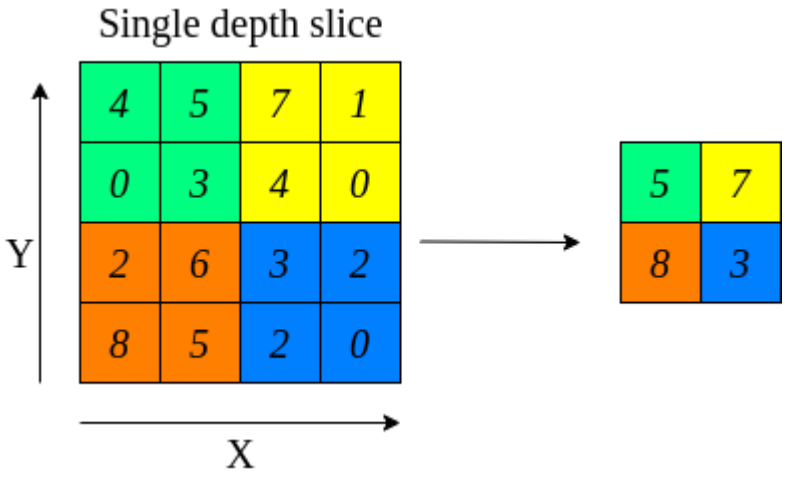


Figure 2.5 Max pooling process

2.3.4 Fully Connected Layer

This layer receives a vector of numbers as input. The word "fully connected" refers to the fact that each of the inputs is connected to each of the outputs. It is the last layer in the CNN process, generally after the last pooling layer. Fully connected layers are similar to classic neural networks in that they contain roughly 90% of the CNN parameters. This layer takes the previous pooling layer's output as input and outputs an N-dimensional vector, where N is the number of classes the program has to choose from. It enables us to feed forward the neural network into a vector of a predetermined length. For image classification, we feed the vector forward into a set of categories (Krizhevsky et al., 2012). The outcome is also a vector of numbers.

2.3.5 Loss Layer

Using functions that take in the model's output and the target, the loss layer calculates a number that measures the model's performance.

It has two primary purposes:

- Forward (input, target): computes the loss value depending on the input and target values.
- Backward (input, target): returns the result after calculating the gradient of the loss function linked with the criteria.

Backpropagation is a concept employed in CNNs to determine the gradient of the loss function (which estimates the cost associated with a given state). Backpropagation weight changes are commonly done using the stochastic gradient descent process.

2.3.6 Algorithm of stochastic gradient descent

In practice, most practitioners use a technique called Stochastic Gradient Descent (SGD). This method shows the input vector for a few examples, computes the outputs and errors, calculates the average gradient for those samples, and adjusts the weights as needed. The technique is repeated for several small numbers of instances from the training set until the average of the objective functions does not decline anymore (LeCun et al., 2015). It's called stochastic because each small number of samples provides a noisy estimate of the average gradient over all cases. SGD algorithm using an iterative method is provided below.

Algorithm 2.2 Stochastic gradient descent

begin
choose an initial vector of parameters and learning rate α
repeat until an approximate minimum is obtained
    randomly shuffle examples in the training datasets
    for *i = 1,2, …, n:*
        $w = w - \eta \nabla Q_i(w)$
        $f = f - \alpha \nabla Q_i(f)$
    endfor
endrepeat
end

This is a stochastic approximation of the gradient descent optimization method for minimizing an objective function given as the sum of several functions.

The training set is used by this method, and each training example is updated as described above. Until the algorithm converges, many runs over the training set can be done. Data can be shuffled for each run to avoid cycles if this is done. To ensure that the method converges, unique implementations may use an adjustable learning rate α.

The gradient is computed against many training examples called a mini-batch at each step as a compromise between computing the genuine gradient and the gradient at a single sample. Because the method may use vectorization libraries instead of computing each step independently, it can produce considerably better results than real stochastic gradient descent.Because the gradient produced at each step includes more training instances, it may also result in smoother convergence.

2.4 Related Works

Kusetogullari et al. (2020) introduced a deep learning architecture known as DIGITNET to detect and recognize English handwritten digits that are found in historical documents in Sweden. The authors also created a large-scale handwritten digit dataset for the public known as DIDA. The data were collected from the Swedish handwritten historical documents written by different priests in the nineteenth century. The dataset consists of 100,000 handwritten digit images. The DIGITNET consists of two different architectures to detect a digit and recognize the digit. The first architecture is DIGITNET-dect, which detects the digit strings from handwritten documents, and the second architecture is DIGITNET-rec which recognizes the handwritten digit. The authors used a deep learning approach to train both models and used regression-based deep CNN methods to detect the digit. YOLOv3 was designed by the authors to detect and classify a digit from an image. In the recognition phase, the authors proposed three different CNN architectures. Convolutional, batch normalization, max-pooling, fully-connected layers, and softmax layers are all included in each proposed model. But still, it has a limitation of some of the image data having high resolution, so it increases the computational cost in the training of the model and some digits are not labeled due to their bad appearance. Low digit detection accuracy because of negative sampling is also a limitation of the research work.

Chen et al. (2018) compared five machine learning classification models to recognize handwritten digits offline. The authors compared the performance of the KNN, Neural Network, Random Forest, Decision Tree, and Bagging with gradient boost. 70,000 digit images are used to develop the classifier models. KNN and Neural Network show better accuracy than other classifiers and KNN achieves 10 times faster speed than the Neural Network model. The preprocessing stage is the crucial part of the recognition system in handwritten recognition. The authors used some pre-processing techniques to enhance the data. They used normalization to give equal weight to each attribute. Then they used a median filter for the noise reduction step. Image sharpening and image attribute reduction are the other steps in the preprocessing phase, but still, it has some limitations from those, the bewilder tool is not effective to preprocess handwritten image data and they didn’t find a threshold value for the binarization preprocess technique, then, they ignore binarization technique. Image is blurred after median filter and sharpening in preprocessing techniques.

Siddique et al. (2019) proposed a seven-layer CNN that contains 1 layer for the input, 1 layer for the output, and the remaining 5 layers for the middle hidden layer. The input layer consists of 28 by 28-pixel grayscale images. The output layer consists of 10 classes that represent the output of the image from 0 to 9. Each hidden layer extracts a feature and reduces the dimension of the image. Convolution layer 1 is the first hidden layer that extracts the first features from the input data. Pooling layer 1 is the next hidden layer, and by reducing the output information from the previous layer, it reduces the number of parameters and complexity of the model. A max-pooling layer is used in this layer. The model's third layer is convolution layer 2, which operates and functions similarly to convolution layer 1 except for the variation of kernel size and feature map. Pooling layer 2 is the fourth hidden layer. It also works in the same way as pooling layer 1, with the exception of the kernel size and feature map variations. The last hidden layer is a flattened layer, which is used to convert a 2D, featured map matrix to 1D. The authors used 70,000 scanned images of handwritten digits from the Modified National Institute of Standards and Technology (MNIST) dataset to develop the recognizing system. They compare the accuracy of the handwritten digit by varying the number of hidden layers for the 15 epochs. But, still, this research lack in preprocessing technique for the dataset.

Beyene (2019) proposed a multi-layered feed-forward propagation ANN for offline handwritten and machine-printed Ge'ez number recognition. The author collected only 560 instances for the model. He used 460 for the training and 100 for the test data. The author collected the data manually because there is no public data for Ge'ez handwritten digits. The overall classification accuracy is 89.88% which is poor because he used a very small amount of the data to develop his model. (Beyene, 2019) research is the first research in this specific area of handwritten Ge'ez digits. Some other researchers have done for all Ge'ez character recognition but (Beyene, 2019) did his research specifically on Ge'ez digits. But still, it has some limitations from those, a small number of data is used to train the algorithm, the work does not give any information about the preprocessing technique, and the accuracy of the proposed model is low to recognize the digit.

Hossain & Ali (2019) proposed a handwritten digit recognition using a CNN on MNIST handwritten dataset. The authors used MatConvoNet to increase the speed of the operation of building the proposed model. MatConvoNet is a MATLAB function that supports an

efficient computation on CPU and GPU allowing the training of complex models on large datasets such as Image Net ILSVRC. But it has some limitations such as the research does not give any information about the preprocessing technique and the number of hidden convolution layers is small in the proposed model.

Ashagrie & Boran (2019) proposed an ancient Ge'ez script recognition model by using deep learning. The authors developed a deep CNN model to recognize Ethiopian ancient Ge'ez characters found in historical documents. They proposed an architecture that only recognizes Ge'ez characters and not words or full sentences. The dataset is a total of 22,913 images collected from libraries, private books, and the Ethiopian Orthodox Tewahedo Church. They also developed a recognition system to recognize twenty-six base characters only. In Ge'ez scripts, there are around 265 characters and 34 base characters, but they classified each character to its base character class, not to its specific character. There are 7 characters found in each base class including the base class. One of the challenges in recognizing handwritten Ge'ez script is the similarity between the characters which are found in the same base class. The authors classified all of the seven characters found in the same class into one base class and ignored the difficult task in their model, but still, it has the problem of low image quality, the number of instances is not balanced for each character. And also the research work does not mention the methods that are used for character detection. The proposed model classified all of the seven characters found in the same class into one base class, this is the other limitation.

Gondere et al. (2019) designed a handwritten Amharic character recognition system using a CNN. The authors used multi-task learning to enhance the model from the relationships of the characters. They ran the experiment by some hyper-parameters of a CNN. The parameters are 100 batches in size, 0.3 keeping probability for dropout, 0.0001 learning rate, and 0.01 L2 regularization. They organized a dataset from different previous research works. But still, it has some problems in the research work. The first one is they used a unique handwritten dataset that affected the performance of the models and the work does not mention the preprocessing technique.

Ali et al. (2019) proposed a model to recognize a handwritten digit. The authors used a CNN algorithm to develop the model. They used deeplearning4j with a CNN for the recognition system. The CNN is composed of two main tasks. The first task is to extract a feature from

each layer. Each layer takes input from the output of the previous layer and forwards the current output to the next layer. The second task of the CNN architecture is feature classification. This unit generates or classifies the predicted output. The authors used the MNIST dataset for their work. 60,000 handwritten digit images were used for training and testing the model. But it has some limitations from those, the proposed model used a large kernel size in the convolution layer, and because of that, it consumes a longer training time. and the work does not give any detailed information about the preprocessing technique.

Upadhyay et al. (2020) proposed a neural network architecture to recognize the handwritten character. The authors used multi-dimensional long short-term memory (MD-LSTM) and recurrent neural networks. Their model is based on a convolutional encoder of the input handwritten image and a bidirectional LSTM decoder predicting the character sequences. But it has some gaps in the work. The first one is the work uses small instances in the dataset to train the model, and the second limitation is the work does not mention the techniques and methods used for preprocessing the image data.

Most of the researchers did digit recognition on English numbers. They achieved high performance using different methods to recognize handwritten digits. For English handwritten digits, there are many resources and datasets ready to be used by the research community. It encourages the researchers to focus on that area. But for Ge'ez handwritten digits, there are no organized data in public for researchers to work on recognition of handwritten digits. Some researchers did Ge'ez character recognition for machine-printed and handwritten characters but they did not focus on digits, especially for handwritten. (Beyene, 2019) is the first researcher to work on recognizing handwritten Ge'ez digits, but the dataset he used was very small and low performance made.

2.5 Summary of Related Works

The following table illustrates the summary of related works that have been described in section 2.4

Table 2.2 Summary of related works

Related Papers	**Architecture**	**Dataset**	**Limitation**
(Kusetogullari et al., 2020) DIGITNET : A deep handwritten digit detection and recognition method using a new historical handwritten digit dataset	YOLO and CNN	DIDA dataset	➢ Some of the data have high resolution, so it increases the computational cost. ➢ Some digits are not labeled due to their bad appearance. ➢ Low digit detection accuracy because of negative sampling.
(Chen et al., 2018) Offline handwritten digits recognition using machine learning	KNN and Neural Network	MNIST dataset	➢ Bewilder tool is not effective to preprocess handwritten image data. ➢ They don’t find a threshold value for the binarization preprocess technique. because of that, the authors ignore this technique. ➢ The image is blurred after median filter and sharpening in preprocessing techniques.

(Siddique et al., 2019) Recognition of handwritten digit using CNN in Python with TensorFlow and comparison of performance for various hidden layers	CNN	MNIST dataset	➢ Does not give any information about the preprocessing technique.
(Beyene, 2019) Handwritten and machine printed OCR for Ge'ez numbers using artificial neural network	ANN	560 numbers of characters	➢ A small number of data used to train the algorithm. ➢ Does not give any information about the preprocessing technique. ➢ Low digit recognition accuracy.
(Hossain & Ali, 2019) Recognition of handwritten digit using CNN	CNN	MNIST dataset	➢ Does not give any information about the preprocessing technique. ➢ The number of hidden convolution layers is small in the proposed model.
(Gondere et al., 2019) Handwritten Amharic character recognition using a CNN	CNN	Used a dataset of (Assabie U. Bigun, 2011)	➢ The performance of the models was influenced by a unique handwritten dataset. ➢ Does not give any information about the preprocessing technique.

(Ashagrie & Boran, 2019) Ancient Ge'ez script recognition using deep learning	CNN	22,913 images collected from EOTCs, libraries, and private books	➢ The number of instances is not balanced for each character. ➢ Low image quality ➢ Does not mention the methods used for character detection. ➢ The authors classified all of the seven characters found in the same class into one base class.
Ali et al., (2019) An efficient and improved scheme for handwritten digit recognition based on CNN	CNN	MNIST dataset	➢ Large kernel size. Consumed longer training time for larger filter sizes. ➢ Does not give any information about the preprocessing technique.
(Upadhyay et al., 2020) A review paper on handwritten character recognition using machine learning	MD-LSTM Recurrent Neural Network (RNN)	IAM and Rimes databases	➢ Small instance in the dataset. ➢ Does not give any explanation about the techniques used for preprocessing.

CHAPTER THREE

3. METHODOLOGY

3.1. Chapter Overview

This section expounds on the methodology that is used to recognize handwritten Ge'ez digits. This section describes the techniques used for data collection from the initial phase of data gathering, to data preprocessing, and the evaluation metrics used for the models were explained extensively.

Figure 3.1 illustrates the steps required from gathering the image to its implementation. The following block diagram is composed of the different series of steps involved in gathering the data, preprocessing the data, and using a dataset for model training and testing, as well as evaluating the model.

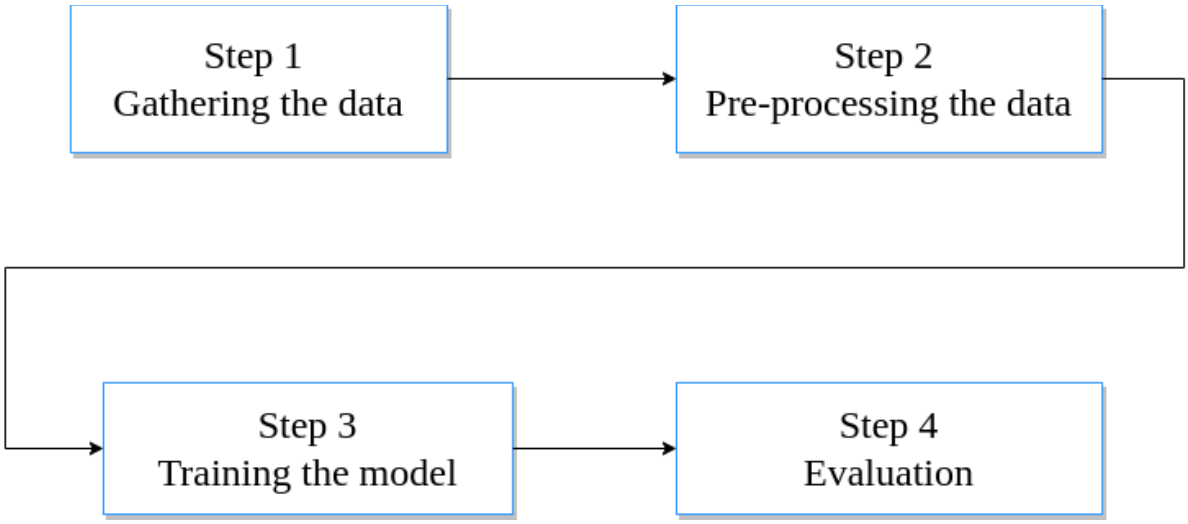


Figure 3.1 Research Methodology

3.2 Data collection

Models for deep learning greatly depend on data, without data it is impossible for the model to learn. In this kind of research, which is handwritten digit recognition in documents, using deep learning, the main task is collecting the required data and then preparing them for additional processing. The benchmark dataset for Ge'ez characters and digits is still not readily available for public use, especially for the general community of researchers. The

research deals with pattern recognition, specifically digit recognition, so the data collected are classified into two. The first is the data that is used to train the deep learning model while the second is for testing the performance of the model constructed.

For this study, handwritten data was collected from a variety of people with various writing styles. Instead of manual feature extraction, which is difficult for humans to do, deep learning models are utilized, which are life-simplifying and efficient techniques to extract with high accuracy, and performance.

A data-gathering paper was created for this purpose. The data gathering paper is prepared in a way to make the preprocessing easier. The paper is A4 size which consists of the symbol of all 20 Ge'ez numbers, in 2 rows and 10 columns in a box, and other same-sized empty boxes prepared and repeated 5 times. This means an individual has to handwrite 100 instances or digits.

The data was collected from 524 different individuals and each person gave 100 instances of digits. According to calculations, since the collected data is from 524 different individuals, 52,400 instances are obtained.

People from many demographic groups participated in the data collection. The data was gathered from elementary pupils, high school students, high school staff members, university students, and university academic staff (lecturers). The majority of information was acquired from university students, which totaled roughly 250 at Adama Science and Technology University.

Respondent-driven sampling techniques were used to collect the data from different individuals. It is a chain-referral sampling method where the participants recommend other people they know and gather the data from their surrounding people. The collection process started with a small number of respondents and expanded the network by collecting the data from their friends, families, and colleague.

The data collection in the university was successfully conducted with the help of Computer science and Engineering Club ASTU (CSEC-ASTU) members. The club had 100 members at the time of data collection, thus the data was gathered from them and through their

connections on the campus. As mentioned earlier, data were obtained from 250 university students, 150 of whom are male and 100 of whom are female.

The data collection from elementary and high school was a school in Addis Ababa. The data is collected from roughly 50 students in elementary school and around 100 students in high school. The data collection process took 2 months.

3.3 Image Acquisition and Digitization

Obtaining an input image with a handwritten digit is the first step in the image acquisition process. In this scenario, the image should be saved in a certain format, such as JPG. A digital camera or a scanner is used to capture the image. The digitization process, on the other hand, entails converting the paper on which the digit was written into an electronic format. The conversion is performed by scanning the original document that is found on paper and converting it to a digital image that can be saved on a computer. The pre-processing phase requires the digital image.

After collecting the data, it must be converted from paper to digital format before it can be processed. The documents were scanned using a Tecno mobile with a 50 Mega Pixel camera and a software app called camscanner for this process. The advantage of using a camscanner is that it detects the paper and provides only the digital format (in image format) of the paper part after removing the background, reducing noise.

Following document scanning, each image has the same dimensions and margins. The distance between the paper's edge and the data was then measured and cropped. As a result, there are five partitions with equal gaps between them, each of which is a group of tables from a paper with two rows and five columns of data as shown in figure 3.2. One partition contains 20 Ge’ez digits. The partition contains 20 identically sized data.

Python's OpenCV library was used for data extraction during the preprocessing technique. This program's input is one partition, and its output is the extracted data. Once prepared for one partition, the same would go for others.

Figure 3.2 Sample handwritten Ge'ez digit data

3.4 Evaluation Methods

The measurement of the quality of the model is done through the evaluation metrics and they are also used to check whether the model is performing correctly and optimally or not. Different evaluation methods such as training accuracy, validation accuracy, training loss, and validation loss will help us give the qualitative analysis and quantitative metrics to evaluate the models made for the handwritten recognition.

3.4.1 Training Accuracy

The measurement of the ability of the model to classify the given images during the training phase using the training dataset is done using the training accuracy metric. Originally the model is trained on the training set, which is the subset of the total dataset.

3.4.2 Validation Accuracy

The measurement of the ability of the model to classify the given data during the training phase using the validation dataset is done using the validation accuracy metric. Here the validation dataset is a part of data put aside from model training that is used to measure model performance while the model is being trained. Validation accuracy checks the model's performance on validation datasets.

3.4.3 Training Loss

The measurement of how well the deep learning model matches the training data is the training loss metric. So the model's errors on the training data are evaluated using this metric. The training loss is determined computationally by adding the sum of errors for each example in the training set. Here the main thing to keep in mind is that the training loss is calculated after every batch. The training loss is commonly illustrated by plotting a curve.

3.4.4 Validation Loss

The performance of the deep learning model on the given validation set is evaluated using the validation loss metric. The validation set is a partitioned portion of the dataset set aside to test the model's performance. Similar to the training loss, the validation loss is calculated by adding up all of the errors for each sample in the validation set. After each epoch, the validation loss is also calculated. This tells us whether the model needs more tuning or tweaks. We normally achieve this by showing a learning curve for the validation loss.

3.5 Development Tools

Numerous types of development tools do the implementation and design of the proposed thesis work. The tools consist of tools and platforms for developing prototypes, modeling tools like UML, and other tools that are relevant to doing the research. The sections that follow will explain these development tools briefly. In this research we use UML to draw a figures in different sections and to design a proposed model. The UML tool drew the figures found in this research document.

3.5.1 Design Tools

Design concepts are created, presented, and interpreted using design tools. Enterprise Architect is the main design tool that is used to design the system. It is used to create flowcharts, network diagrams, and other design concepts which are professional-looking and it is lightweight, and powerful.

3.5.2 Hardware tools

The following are the hardware tools used in this study implementation:

Table 3.1 Hardware tools

No.	Tools	Specification	Used for
1.	GPU	GeForce RTX 2070 Super 6 GB RAM	Computation boost and accelerate the training.
2.	RAM	8 GB RAM	For the computer's speed and performance
3.	Hard Drive (HDD)	WD 1TB USB 3.2 Gen 1	Used as storage for large data sets.

3.5.3 Software tools

Different IDEs (Integrated Development Environments), code editors, and other coding tools are used for the implementation part of this research, and they are briefly explained as follows:

Table 3.2 Software tools

No	Software tools	Version	Description
1	Python	3.8.5	Python is a programming language which is high-level and general-purpose that is abundant in libraries and frameworks that facilitate coding and save development time.

2	Jupyter Notebook	6.1.4	The most popular and convenient IDE amongst deep learning and AI researchers is Jupyter Notebook which is built upon python.
3	Scikit-learn	0.23.2	It is an open-source software library in python that contains machine learning algorithms.
4	Anaconda	2020.11	It is a program that is used to install the most recent versions of Python, as well as its various modules and IDEs, as well as to implement the recommended solution.
5	TensorFlow	1.15	TensorFlow is an open-source software library designed for numerical modeling and processing with high performance. Its modular architecture makes it simple to use.
6	Keras	2.4.0	Keras is a Python-based, user-friendly deep learning API that runs on top of TensorFlow.

CHAPTER FOUR

4. PROPOSED SOLUTION

4.1. Chapter Overview

The proposed solution architecture for the handwritten Ge'ez digit recognition problem is presented in this chapter, together with mathematical equations and the necessary diagrammatic representation. This chapter is divided into three sections: the first section covers the dataset, the second describes several preprocessing techniques, and the third portion introduces the proposed model architecture for recognizing handwritten Ge'ez digits.

4.2 Dataset

Handwritten Ge'ez digit dataset was collected from different individuals for the proposed methodology. The dataset consists of 51,952 images. In table 4.1 the Digits column represents the Latin numeric values for each symbol of the Ge'ez digits. Table 4.1 shows the number of instances for each digit in the train, validation, and test dataset.

Table 4.1 Number of instances for each class

Digits	**Number of Training data**	**Number of Validating data**	**Number of Testing data**	**Subtotal**
1	2079	260	260	2599
2	2082	260	260	2602
3	2078	259	260	2597
4	2088	261	261	2610
5	2084	260	260	2604
6	2085	261	261	2607
7	2080	260	260	2600
8	2085	260	260	2605
9	2085	260	260	2605
10	2077	259	260	2596

20	2063	258	258	2579
30	2062	257	258	2577
40	2076	259	260	2595
50	2081	260	260	2601
60	2062	257	258	2577
70	2072	259	259	2590
80	2076	259	260	2595
90	2079	260	260	2599
100	2086	260	261	2607
10000	2086	260	261	2607
Total	41,566	5,189	5,197	51,952

Out of 51,952 scanned images of Ge'ez handwritten digits, 41,566 scanned images of digits are used to train the model, representing for 80% of the dataset, 10% of the scanned images of digits are used to validate the model, and the remaining 10% are used to test the model's performance. All of the images used to train, validate, and test the network are grayscale images with a dimension of 32 by 32 pixels.

4.3 Preprocessing

The second phase of the proposed model is the preprocessing phase that occurs after the digital image has been made. The digitized image is first checked for skewing before being pre-processed to reduce noise. Pre-processing is necessary for creating data that is simple to recognize using handwritten digit recognition systems, and the goal is to reduce background noise, enhance the image's region of interest, and produce a clear distinction between foreground and background. We use Python openCV library for the pre-processing technique.

In this research, we use some pre-processing techniques. Such as resizing the original image to the desired dimension for the proposed model, converting the image to grayscale, color inversion, normalizing the image pixel value, and data augmentation are the techniques we apply to the dataset.

4.3.1 Resize Image

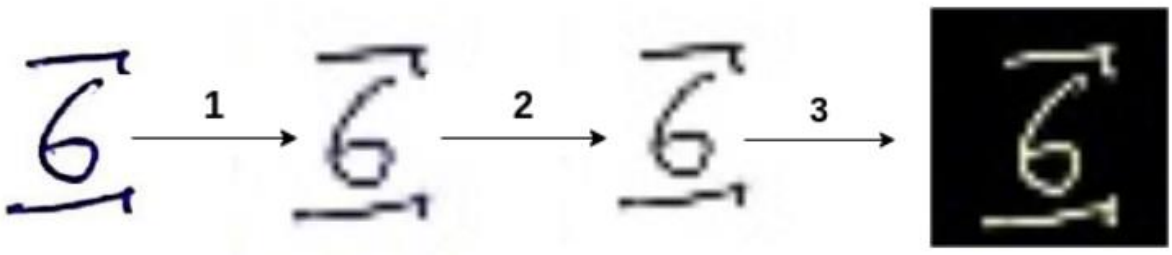
Because the data is available in a range of sizes, it must be resized to fit the network's input size. All images are resized to 32x32 pixels in this work. This scaling is important for reducing computational complexity and for concentrating on the region of interest by cropping it.

4.3.2 RGB to Grayscale Conversion

The simplest color model is grayscale, which specifies colors using only one component: lightness. A value ranging from 0 (black) to 255 (white) is used to define the amount of brightness (white). All the original images in the dataset are in RGB color format. To reduce the color channel we convert to grayscale and it reduces the computational complexity compared with RGB color images. In our proposed model the input images are grayscale so, the original images should be converted to the grayscale color format.

4.3.3 Color Inversion

The dominant color of the original image is white, which has a value of 255. For grayscale image dataset models changing the dominant color to black is preferable to reduce the complexity of the mathematical operations. Because black color has 0 values, a convolution operation with the dominant part with a 0 value is reducing the computational complexity of the model. Figure 4.1 shows the pre-processing techniques used in our dataset. As shown in figure 4.1 (d) the dominant part of the image is the background of the image. In the color inversion technique, we convert the background from white to black color.



(a) (b) (c) (d)

Figure 4.1 Ge'ez handwritten digit image after some pre-processing techniques.

(a) original image (b) resize image (c) grayscale image (d) black background image

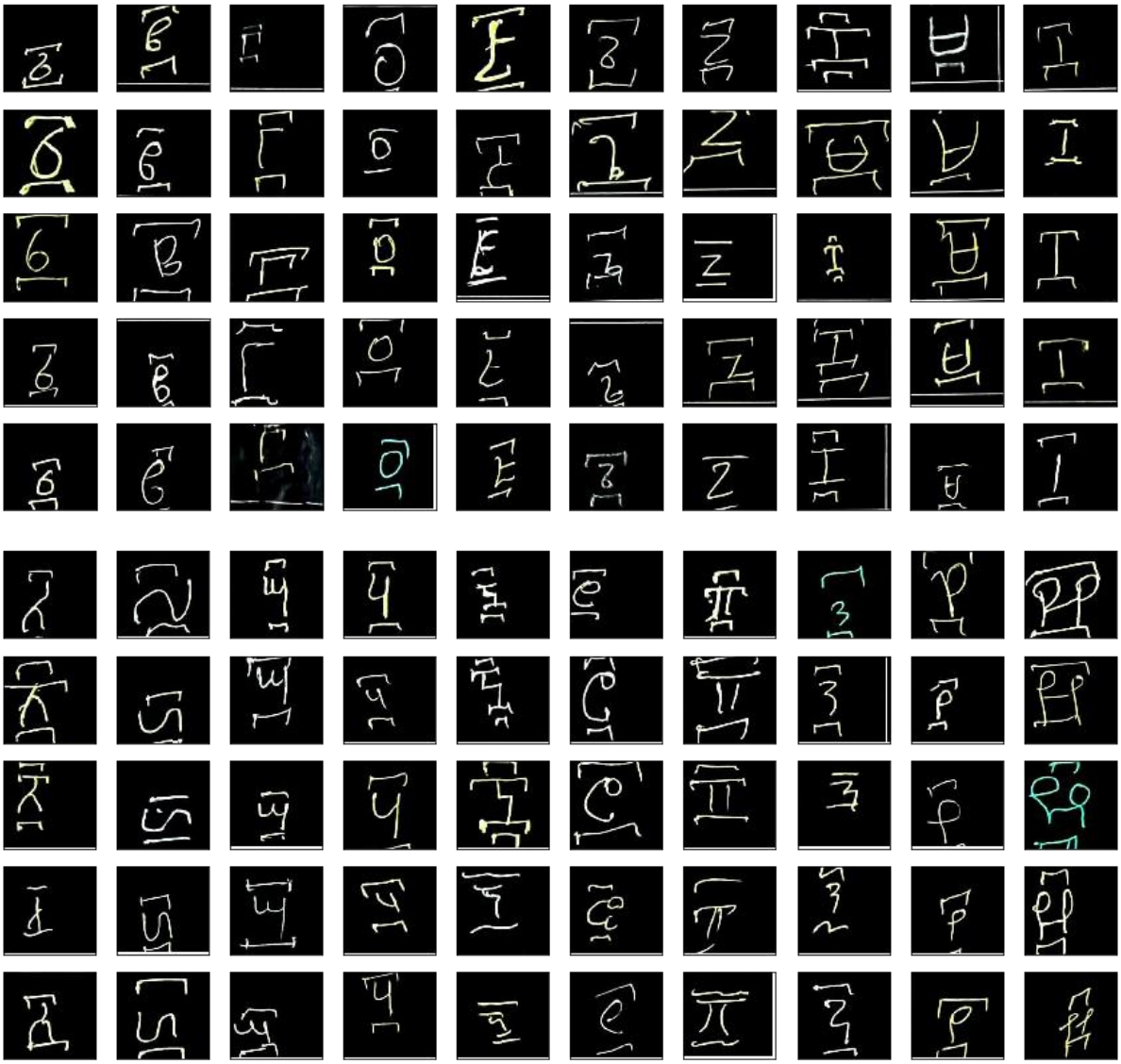


Figure 4.2 Sample images of Ge'ez handwritten digit dataset

Figure 4.2 shows the sample image of the dataset for the Ge'ez handwritten digits. In the figure each digit is represented by 5 images in a column. The first 5 rows of the image represent the same digits for each column, and the last 5 rows of the image represent another digit for each column.

4.3.4 Normalization:

The range of pixel intensity values is changed in this image processing process. Its main purpose is to convert an input image into a set of pixel values. The term "normalization"

refers to the process of transforming images to a standard size. It's a technique used during the data pre-processing stages. When the data set's characteristics have different ranges, the values of the numerical values are changed to use the common scale of [0,1]. And this is calculated from the input image by the following equation.

$$\text{Output} = \left(\frac{inputimage}{255}\right) \quad (4.1)$$

The output values will be in the range of 0 to 1. The normalization will minimize the computations, fasten the forward propagation process and provide results that are more accurate.

4.3.5 Data Augmentation

To give a good performance, the deep learning process typically requires a large amount of data. Data augmentation is a significant way to improve efficiency when there is limited data available. Augmentation is the process of generating training images artificially using various pre-processing techniques such as random rotation, horizontal or vertical flipping, shifting, and so on. It's a great approach to increasing the data's size for better outcomes.

Image Data Generator: This built-in Keras class provides developers to easily access enhanced images by using techniques such as a 30-degree rotation, a horizontal flip, and a sheer range of 0.15.

This Image Data Generator uses the following algorithm:

Algorithm 4.1 Data augmentation algorithm

```
for all images do:
        Load the original image from the disk.
        Transform the images randomly rotate
            from -30 degrees to 30 degrees.
        Write the transformed image to the disk.
endfor
```

4.4. Proposed Model Architecture

Convolutional Neural Network (CNN) is the proposed model to address the Ge'ez handwritten digit recognition. To recognize the digits, a CNN-based digit classifier is used. Six different CNN-based handwritten digit classifiers consist of several layers such as a convolutional layer, max-pooling layer, dropout layer, flatten layer, fully-connected layers, and softmax layer to achieve high recognition accuracy. Furthermore, the training was performed by applying the back-propagation approach of Stochastic Gradient Descent.

Finally, based on the evaluation metric, choose the best model for recognizing digit strings. Each classifier is constructed with a different number of convolutional layers, kernel sizes, and filters. The parameters applied in all six classifiers are summarized in Table 4.2. Model 6, for example, is shown in Figure 4.3 has 8 convolutional layers, 4 max-pooling layers, 3 dropout layers, 2 fully-connected layers, and 20 output layers. The kernel size, stride, and a number of filters in the first convolutional layer are 3x3, 1, and 32 (3x3@1@32), respectively. The second and third convolution layers are similar to the first. After three convolution layers, the max-pooling layer (2x2@2@32) is applied. The convolutional layer (3x3@1@64) is used in the fifth layer, and it consists of 64 filters with a kernel size of 3 3 and a stride of 1. The following two layers are convolution layers, with the same hyper-parameter as the fifth layer. The max-pooling layer (2x2@2@64) is applied in the eighth layer. After the max-pooling layer, dropout is applied. The convolutional layer (3x3@1@64) is applied next, which consists of 64 filters with a kernel size of 3 3 and a stride of 1. The max-pooling layer (2x2@2@64) is the next hidden layer. After the max-pooling layer, dropout is applied. The convolutional layer (3x3@1@128) is applied next, which consists of 128 filters with a kernel size of 3x3 and a stride of 1. Max-pooling layer along with dropout layer is used before the fully connected layer. Fully-connected layers are used, which consist of 128 nodes. In the convolutional and fully-connected layers, ReLU is used as an activation function. Softmax is used as a last layer to compute the probabilities of output classes in the last layer. The class with the highest probability produces the desired result. The epoch size is 30 and the total number of training instances in a single batch is 32. As indicated in Table 4.2, the other five classifiers have varying numbers of convolutional and fully connected layers, as well as different layer organizations. The first fully connected layer contains 128 neurons and the second contains 20 neurons in all cases.

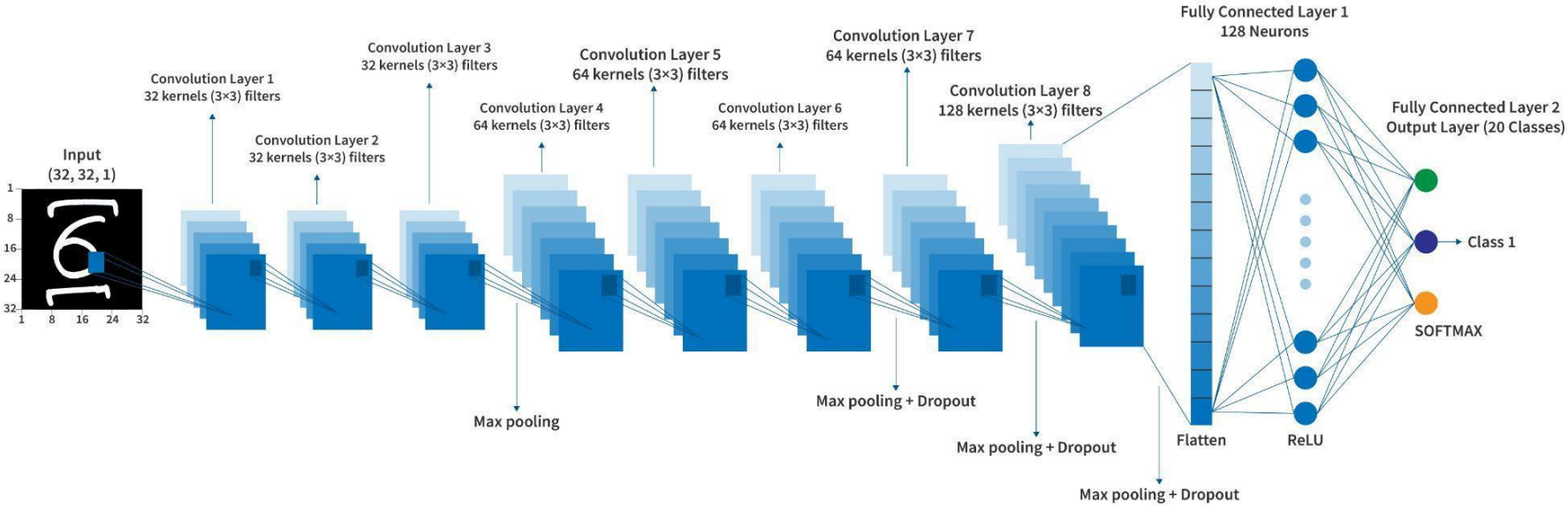


Figure 4.3 Convolutional neural network architecture of the proposed system

Table 4.2 The CNN models for Ge'ez handwritten digit recognition

Model	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Layer 6	Layer 7	Layer 8	Layer 9	Layer 10	Layer 11	Layer 12
1	Conv 3x3@1@32 and ReLU	Conv 3x3@1@32 and ReLU	Max-pooling 2x2@2@32	Conv 3x3@1@64 and ReLU	Conv 3x3@1@64 and ReLU	Max-pooling 2x2@2@64 and Dropout	Conv 3x3@1@64 and ReLU	Conv 3x3@1@128 and ReLU	Max-pooling 2x2@2@128 and Dropout			
2	Conv 3x3@1@32 and ReLU	Conv 3x3@1@32 and ReLU	Max-pooling 2x2@2@32	Conv 3x3@1@32 and ReLU	Conv 3x3@1@32 and ReLU	Max-pooling 2x2@2@32	Conv 3x3@1@64 and ReLU	Conv 3x3@1@64 and ReLU	Max-pooling 2x2@2@64 and Dropout	Conv 3x3@1@64 and ReLU	Conv 3x3@1@64 and ReLU	3 additional layers such as pooling, Conv, pooling
3	Conv 3x3@1@32 and ReLU	Conv 3x3@1@32 and ReLU	Max-pooling 2x2@2@32	Conv 3x3@1@32 and ReLU	Conv 3x3@1@32 and ReLU	Max-pooling 2x2@2@32 and Dropout	Conv 3x3@1@64 and ReLU	Conv 3x3@1@64 and ReLU	Conv 3x3@1@64 and ReLU	Max-pooling 2x2@2@64	Conv 3x3@1@64 and ReLU	Convolution layer with pooling and dropout layer
4	Conv 3x3@1@32 and ReLU	Conv 3x3@1@32 and ReLU	Conv 3x3@1@32 and ReLU	Max-pooling 2x2@2@32	Conv 3x3@1@64 and ReLU	Conv 3x3@1@64 and ReLU	Conv 3x3@1@64 and ReLU	Max-pooling 2x2@2@64 and Dropout	Conv 3x3@1@128 and ReLU	Max-pooling 2x2@2@128 and Dropout		
5	Conv 3x3@1@32 and ReLU	Conv 3x3@1@32 and ReLU	Conv 3x3@1@64 and ReLU	Max-pooling 2x2@2@64 and Dropout	Conv 3x3@1@64 and ReLU	Conv 3x3@1@64 and ReLU	Conv 3x3@1@128 and ReLU	Max-pooling 2x2@2@128 and Dropout				
6	Conv 3x3@1@32 and ReLU	Conv 3x3@1@32 and ReLU	Conv 3x3@1@32 and ReLU	Max-pooling 2x2@2@32	Conv 3x3@1@64 and ReLU	Conv 3x3@1@64 and ReLU	Conv 3x3@1@64 and ReLU	Max-pooling 2x2@2@64 and Dropout	Conv 3x3@1@64 and ReLU	Max-pooling 2x2@2@64 and Dropout	Conv 3x3@1@128 and ReLU	Max-pooling 2x2@2@128 and Dropout

4.4.1 Feature Extraction

The main phase of image classification is feature extraction. The CNN algorithm extracts the important features that are needed to classify the image. The height of the digit character, the number of horizontal lines, the widths of the digit character, the number of circles, pixels, the position of different characteristics, and the number of vertically oriented arcs, to list a few, are all extracted and defined in this phase.

4.4.2 Stochastic Gradient Descent (SDG)

The character x is used to represent a training input, where x is a 1024-dimensional vector with a 32-pixel input. y(x), where y is a 20-dimensional vector, is the corresponding desired output. The network's goal is to find the most feasible weights and biases so that the output approximates y(x) for all training inputs x, which is dependent on the weight and bias values. A cost function, as defined below, is used to calculate network performance:

$$C(w,b) = \frac{1}{2n}\sum_x [y(x) - a^2]^2 \quad (4.2)$$

Where w represents the overall weights of the network, b for all biases, n for the total number of training inputs, and a for the actual output. The actual output an is determined by the variables x, w, and b. Because all of the terms in the total are non-negative, C(w,b) is non-negative.

Furthermore, when the desired output y(x) is substantially identical to the actual output, a, C(w,b)=0 for all training inputs, n. To reduce the cost C(w,b) as a function of weight and biases, the training algorithm must select a set of weights and biases that cause the cost to become as minimal as possible as a function of weight and biases. Gradient descent is the algorithm used to do this. Gradient descent, in other terms, is an optimization technique that iteratively twists its parameters to minimize a cost function to its local minimum. To set the weights and biases, the gradient descent algorithm uses the following.

$$w^{new} = w^{old} - \eta \frac{\partial C}{\partial w^{old}} \quad (4.3)$$

$$b^{new} = b^{old} - \eta \frac{\partial C}{\partial b^{old}} \quad (4.4)$$

Also, as illustrated in figure 4.4, to achieve a global minimum of the cost C(w,b).

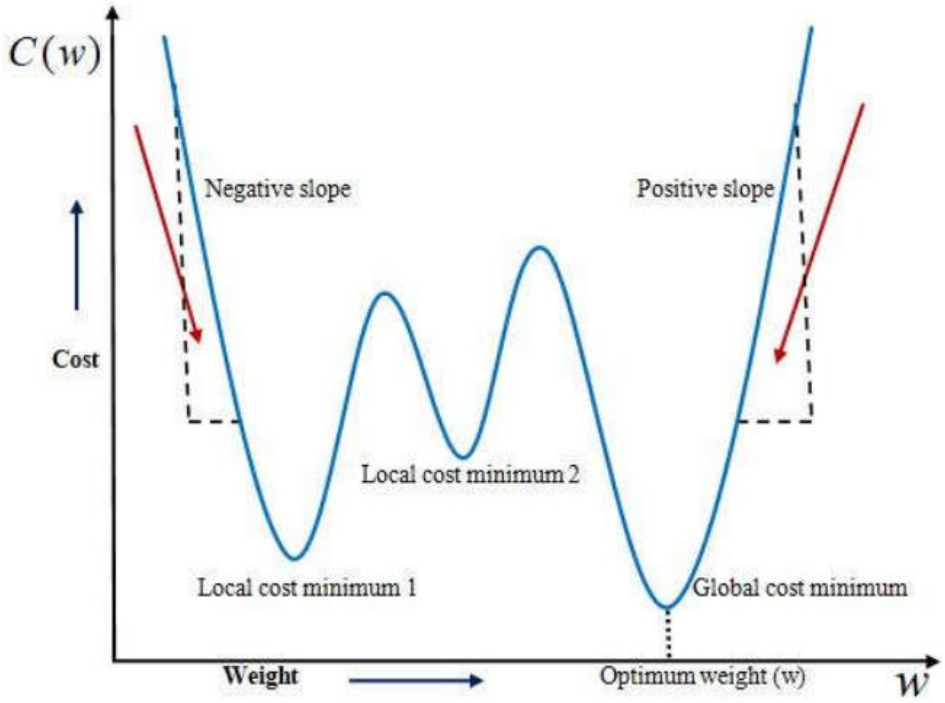


Figure 4.4 Graphical Representation of Cost vs. Weight (Siddique et al., 2019)

When the training data is very large, however, the gradient descent approach may be ineffective. As a result, a stochastic variant of the technique is used to improve the network's performance. In SDG, only a few iterations are required to find effective solutions to optimization problems. Furthermore, in SDG, a good solution can be found after only a few repetitions. The following equations are used in the Stochastic Gradient Descent algorithm:

$$w^{new} = w^{old} - \frac{\eta}{m}\frac{\partial C_{xj}}{\partial w^{old}} \quad (4.5)$$

$$b^{new} = b^{old} - \frac{\eta}{m}\frac{\partial C_{xj}}{\partial b^{old}} \quad (4.6)$$

The output of the network can be expressed by:

$$a = f(z) = f(wa + b) \quad (4.7)$$

4.4.3 Backpropagation

The back-propagation method is used to determine the amount of weight that contributes to the total error of the network. The following equations show how the network's back-propagation works.

$$\delta^{L} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} = \frac{1}{n}\left(a^{(L)} - 1\right) f'\left(z^{(L)}\right) \quad (4.8)$$

$$\delta^{l} = \frac{\partial C}{\partial Z^{(l)}} = \frac{\partial C}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial z^{(l)}} = \frac{\partial z^{(l+1)}}{\partial z^{(l)}} \delta^{(l+1)} = w^{(l+1)} \delta^{(l+1)} f'(z^{l}) \quad (4.9)$$

$$\frac{\partial C}{\partial b^{(l)}} = \delta^{l} \quad (4.10)$$

$$\frac{\partial C}{\partial w^{(L)}} = a^{(l-1)} \delta^{l} \quad (4.11)$$

4.4.4 Activation function

In the CNN model, many alternative activation functions are used in the experiments: ReLU, Softmax and Sigmoid are the most common activation functions for CNN models. The sigmoid activation functions, in particular, in the model's output layer. ReLU is preferable for convolution and fully connected layers.

4.4.5 Epochs

During training, it is the number of iterations the entire training data is trained to the network. It is the number of iterations that the complete training dataset runs through the model or network forwarded and backward. 30 epochs were used to train this model. Find the optimal number of epochs for the model's best performance. This was done to obtain the best possible optimum epoch.

4.4.6 Batch size

It is the total number of training inputs that have been given to train the model. To update the parameters, the single set is divided into small subsets called mini-batches. The 32 batch size was used to train this model. Small batch size is best to achieve high performance but consumes high computational cost.

4.4.7 Learning rate

The CNN model was trained using back-propagation, and the learning rate is used during weight updates. It determines how much weight should be updated during back-propagation. Choosing the right learning rate during the experiment was the most difficult part. Learning rates with a lower value take longer to train than those with a larger value. When given a lesser value, however, the model performs better than one with a higher learning rate. The learning rates used in the experiment were 0.01, 0.001, and 0.0001. This was done to determine the best learning rate.

4.4.8 Hyper-parameters for CNN Model

All of the CNN models were used with different hyper-parameters. These factors open up some architectural possibilities. It took a long time to train them with such a large dataset. To make efficient computing with available resources, hyper-parameter optimization is used. For the proposed CNN model, the following hyper-parameters were chosen:

- Architecture for deep learning:
  - CNN
- Training mechanism:
  - Starting at the Beginning
- Distribution of training, validation, and testing sets:
  - Train: 80%, Test: 10%, Validation: 10%
- Data set type:
  - Grayscale image
- Batch size:
  - 32
- Number of Epoch:
  - 30

CHAPTER FIVE

5. IMPLEMENTATION OF THE PROPOSED SOLUTION

5.1. Chapter Overview

This chapter deals with the implementation process. From the initial setup of the working environment to the data processing and training procedures, everything is covered. In addition, the implementation's code snippets are shown and discussed in depth.

5.2. Working Environments

This section describes the environment setup of the hardware components that are used in the implementation of the modeling process with their specifications.

- ➢ Laptop:
  - Operating System: Linux Mint 20.2 Cinnamon
  - Processor: Intel© Core™ i7-7500U CPU @ 2.70GHz × 2
  - Installed memory (RAM): 8.00 GB
  - Graphics Card: Intel Corporation HD Graphics 620
  - System Type: 64-bit operating system, x64-based processor

5.3. Environmental Setup

Specific programming tools, online and locally installed IDEs were used in this study.

JupyterLab: is the well-known and helpful web-based interactive development environment for AI, data science, scientific computing, machine learning, and deep learning examiner to work with python.

Colab: is a Google cloud-based notebook that allows you to create Python code on any browser with sufficient computing power (2 vCPU, 12GB RAM) and resources to train with the TPU and GPU.

Keras: the main goal is to make it as simple as possible to train deep learning models for research and development with models that are suited for deep learning and allow for quick calculations and prototyping. Because the software library uses a GPU in addition to the computer's CPU. The module is currently found in the Tensor flow.

TensorFlow: by setting up, training, and exploiting massive data sets of the neural networks, it is appropriate for deep learning. Keras library, which is used to train and model deep neural network (DNN) classifiers and to use them to predict for testing sets, uses TensorFlow as a back end.

Visual studio code (VS code): is utilized as an improved and simplified IDE, with python version of 3.8, for preprocessing the data.

5.4. Training Procedure

In the training procedure the following process has taken place in order to achieve the goal of this research.

- Data collection
- Data preprocessing
- Configuring the GPU and the dataset
- Implementing the proposed solution.

The technique is outlined in detail in each phase to provide further information on what has been done.

5.5. Implementation Techniques for Data Preprocessing

5.5.1. Image Data Augmentation

Data augmentation is a technique that is used to artificially increase the amount of data by generating new data variations from the existing data. Adding small changes to the original data or training a model to create those data can do this either. Nowadays this Image augmentation becoming popular in many deep learning models since deep learning requires a large data and deeper models to increase the generalizability and performance of models. Additionally, It can increase the model performance by applying to some of the original images without adding new images to the dataset. This will allow the model to learn more variation in the image, which will simulate real-world data. From the augmentation techniques shifting, rotation and zooming are used as shown in figure 5.1.

So all we have to do now is make minimal changes to our existing dataset to gain extra data. Zoom, translations, and rotations are all minor adjustments. In any case, our neural network would distinguish these images.

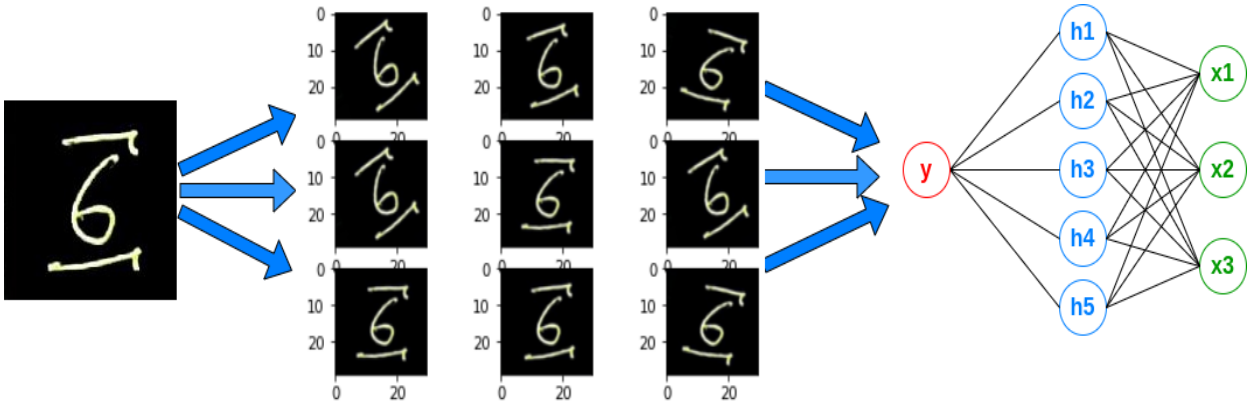


Figure 5.1 Sample augmented data

```
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
  rescale=1.0 / 255,
  rotation_range=30,
  width_shift_range=0.2,
  height_shift_range=0.2,
)
```

Snippet code 5.1 Data augmentation code

ImageDataGenerator has all the methods for the data augmentation process but this research used the parameters:

- **rescale** specifies the scaling factor in which the image range value is decreased to 1-255
- **rotation_range** specifies the range of angle in which the images are rotated,
- **width_shift_range** specifies the limit by which the image is shifted either to the right or left,
- **height_shift_range** specifies the limit by which the image is shifted either to upward or downward direction,

From the code snippet given above the image data generator object randomly applies the techniques, but not adds new images to the dataset. The rotation_range parameter is 30^0 which means that the rotation of the image is limited from 30^0 to the left to 30^0 to the right. The width and height shift range parameters specify by how much percent of the original size should the image be shifted.

5.6. Implementation Techniques for CNN

5.6.1. Load Image

To load images from a google drive first upload the data from local storage to Google Drive, then connect Google Drive to Colab and mount. To load the dataset to the working directory or in our case Google colab the **image_dataset_from_directory()** is used. This method converts the directory to a valid dataset to be used by a deep learning model.

The **image_size** and **batch_size** parameters are passed to the above method to indicate the image size in which the data is loaded and the number of images that are to be loaded per single step. If the subset parameter is “training” then the loaded dataset is used for training the model. The other option for the subset parameter s “validation”.

```
batch_size = 32
img_height = 32
img_width = 32
data_dir = "/content/gdrive/MyDrive/PreProcessV1/"
train_ds = tf.keras.utils.image_dataset_from_directory(
       data_dir,
       validation_split=0.111,
       subset="training",
       seed=123,
       image_size=(img_height, img_width),
       color_mode="grayscale",
       batch_size=batch_size
)
```

Snippet code 5.2 Dataset Load code

The batch size is chosen to be 32 images per one iteration. the more number batch chosen the accuracy tends to decrease and also the lesser the size the lower the model performance. So 32 is the optimal one. The images passed the first stages of processing techniques and are stored in the specified folder. As mentioned above the validation split used was 10% as specified in the above snippet. the image size is decreased to 32x32 in grayscale.

From the dataset to get the number of classes for digit recognition.

```
class_names = train_ds.class_names
print(class_names)
```

Snippet code 5.3 Return classes of the dataset

The output for the above code snippet is

```
['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '20', '30', '40',
'50', '60', '70', '80', '90', '100', '10000']
```

The output indicates the Ge'ez digits represented by one symbol.

5.6.2. Rescale

A preprocessing layer that rescales input image values to a new range. This layer rescales every value of an image pixel by multiplying by the scale and adding an offset. The original image pixel value is in the range of [0, 255]. To rescale an input value of [0, 255] to in the range of [0, 1] the operation is to multiply the value by 1/255. After rescaling all values of the image is between 0 and 1. 0 represents black and 1 represent white.

```
normalization_layer = tf.keras.layers.Rescaling(1./255)
normalized_ds = train_ds.map(lambda x, y:(normalization_layer(x), y))
```

Snippet code 5.4 Rescale image value code

5.6.3. The hyper-parameter Tuning

Hyper-parameter are the parameters that control other parameters. Tuning them to get the optimal value is called hyper-prarameter tuning. This procedure is essential because choosing the right set of parameters for the model will robust the performance of the model. The AUTOTUNE API makes it easier to create input pipelines that are both flexible and efficient. Other hyperparameter tunings were also performed in other parameters which will be described later.

```
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Snippet code 5.5 Hyper-parameter tuning code

5.6.4. Optimizer

```
adam = keras.optimizers.Adam(lr = 1e-3)
```

Snippet code 5.6 Adam optimizer code

For the optimizer section Adam optimizer with initial learning rate of 0.001 is used.

5.6.5 Learning Rate Adjustment

The learning rate must not be fixed but rather reduced to some amount after some epoch if the loss no longer improves, reduce the learning rate. This could a better performance to the model by enabling it to find the minimum value. If the learning rate is fixed the model may fluctuate around the global minimum.

```
reduce_lr = ReduceLROnPlateau(monitor = 'val_loss', patience = 2,
                verbose = 1, factor = 0.1, min_lr = 1e-7
)
```

Snippet code 5.7 Learning rate modify code

Keras provides the ReduceLROnPlateau function which will allow to adjust the learning rate. In the above snippet code, it will decrease the learning rate by a factor of 0.1, if the validation loss is not improving in two successive epochs until it reaches 10^{-7}.

The learning rate is a hyper-parameter that controls how much our network's weights alter in response to the loss gradient. A number that is too little may result in a delayed and stalled training process, while a value that is too big may result in learning a sub-optimal set of weights too rapidly or an unstable training process. The stochastic gradient descent approach uses instances from the training dataset to estimate the error gradient for the present state of the model, then updates the model's weights using the back-propagation of errors process, sometimes known as simply back-propagation. The "learning rate," or step size, is the amount by which the weights are altered during training. The learning rate is an adjustable hyper-parameter used in neural network training that has a moderate positive value, generally between 0.0 and 1.0. The learning rates used in this experiment were 0.01, 0.001, and 0.0001. The best-performed models are found by a learning rate of 0.001.

5.6.6. Define Model

Define a baseline convolutional neural network model for the problem. The model's two main components are the feature extraction front end, which comprises convolutional and pooling layers, and the classifier backend, which generates a prediction. A CNN model has two wheels: a convolutional layer and a pooling layer. The success of CNN for image classification challenges is due to its ability to work with grid structured data. For model compilation, we employ the Adam optimizer.

Each Conv2D and MaxPooling2D layer generates a three-dimensional tensor of the form (height, width, channels). The width and height proportions start to decrease as you move deeper into the network. The first option determines how many output channels each Conv2D layer has (e.g., 32 or 64). As the width and height of each Conv2D layer shrink, you can usually afford to add more output channels (computationally).

```
def define_model():
      model = keras.Sequential(
      [
          keras.Input(shape = (32, 32, 1)),
          layers.Conv2D(32, 3, padding = 'same', activation = 'relu',
                                                  name="Conv1"),
          layers.Conv2D(32, 3, padding = 'same', activation = 'relu',
                                                  name="Conv2"),
          layers.Conv2D(32, 3, padding = 'same', activation = 'relu',
                                                  name="Conv3"),
          layers.MaxPooling2D(name="Maxpool1"),

          layers.Conv2D(64, 3, padding = 'same', activation = 'relu',
                                                  name="Conv4"),
          layers.Conv2D(64, 3, padding = 'same', activation = 'relu',
                                                  name="Conv5"),
          layers.Conv2D(64, 3, padding = 'same', activation = 'relu',
                                                  name="Conv6"),
          layers.MaxPooling2D(name="Maxpool2"),
          layers.Dropout(0.2, input_shape=(32,)),

          layers.Conv2D(64, 3, padding = 'same', activation = 'relu',
                                                  name="Conv7"),
          layers.MaxPooling2D(name="Maxpool3"),
          layers.Dropout(0.2, input_shape=(32,)),

          layers.Conv2D(128, 3, padding = 'same', activation= 'relu',
                                                   name="Conv8"),
          layers.MaxPooling2D(name="Maxpool4"),
          layers.Dropout(0.15, input_shape=(32,)),

          layers.Flatten(name="Flatten1"),
          layers.Dense(128, activation = 'relu', name="Dense1"),
          layers.Dense(20, activation = 'softmax', name="Dense2"),
      ]
)
```

Snippet code 5.8 Define the model code

We use three successive convolutional layers with a small filter size 3 by 3 and a 32 filter channel for the convolutional front-end, followed by a max-pooling layer. Then three 64-channel Conv2D with a 3x3 size filter were applied followed by a max-pooling layer. The dropout layer is applied to remove 20% of the data to prevent the overfitting of the model on the training data. Another 64 channel layer was applied with a max-pooling and dropout layer. Then a 128 channel convolutional layer is used with a filter size of 3 by 3. A max-pooling and dropout was applied before the flattened process. The filter maps can then be flattened to provide features to the classifier. All the convolution layers uses the ReLU as an activation function.

To finish the model, we input the convolutional base's final output tensor into two dense layers for classification. Dense layers accept vectors as input (which are 1D), with a 3D tensor as the current output. We start by flattening (or unrolling) the 3D output to 1D, then build two dense layers on top of it.

For the stochastic gradient descent optimizer, we use a conservative set up with a learning rate of 0.001. The categorical cross-entropy loss function is optimized for multi-class classification, and the classification accuracy measure is monitored, which is reasonable provided that each of the 20 classes has the same amount of samples. The define_model() function below defines and return this model.

The implementation of the model structure is shown in the above snippet. successive Conv2D layers followed by MaxPooling layers having an activation function of relu with preserving the size and then flattening the image to 1D and then adding two Dense layers the first one having an activation function of relu and the last one having an activation function of softmax which will determine the probability of each class. Some Dropout layers are added to the layers this will increase the model performance by making it not dependable on specific attributes of pixel values.

The model is trained for 30 epochs since our learning rate is very small.

```
classifier = model.fit(train_ds, batch_size = 32, epochs = 30)
```

Snippet code 5.9 Train model code

5.7. Learning hyper-parameters

Hyper-parameters should be properly chosen for deep learning models like CNN to learn the relationship between features and outputs. Hyper-parameters are variables that affect other performance of the model parameters as well as network training methods. So, in order to gain a better result in this study, the following hyper-parameters were used, and Adam and Nadam were used to represent some of them for model optimization. Adam is a form of stochastic gradient descent optimization technique that uses training data to update network weight iterations. It's popular because it's easy to use, comes with Keras, uses less memory than other optimization techniques, is computationally efficient, and is good for situations with a lot of variables (data).

The learning rate (LR), which controls the speed at which the model learns or the rate at which the change in weight is determined, is the other hyper-parameter. A higher value of learning rate might make the model fluctuate around the global minimum value of the loss function and also if the learning rate is small the model may not reach the global minimum at all. In section 5.6.5, It’s shown that the learning rate won’t be fixed after some epochs if there is no iteration. But the initial learning rate value must be selected carefully.

Table 5.1 Learning hyper-parameters

No.	**Hyper-parameters**	**Value**
1	Learning rate	0.001
2	Optimizer	Adam
3	Epoch	30
4	Batch size	32

Table 5.2 Model summary table

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Conv1 (Conv2D)              (None, 32, 32, 32)        320

 Conv2 (Conv2D)              (None, 32, 32, 32)        9248

 Conv3 (Conv2D)              (None, 32, 32, 32)        9248

 Maxpool1 (MaxPooling2D)     (None, 16, 16, 32)        0

 Conv4 (Conv2D)              (None, 16, 16, 64)        18496

 Conv5 (Conv2D)              (None, 16, 16, 64)        36928

 Conv6 (Conv2D)              (None, 16, 16, 64)        36928

 Maxpool2 (MaxPooling2D)     (None, 8, 8, 64)          0

 dropout (Dropout)           (None, 8, 8, 64)          0

 Conv7 (Conv2D)              (None, 8, 8, 64)          36928

 Maxpool3 (MaxPooling2D)     (None, 4, 4, 64)          0

 dropout_1 (Dropout)         (None, 4, 4, 64)          0

 Conv8 (Conv2D)              (None, 4, 4, 128)         73856

 Maxpool4 (MaxPooling2D)     (None, 2, 2, 128)         0

 dropout_2 (Dropout)         (None, 2, 2, 128)         0

 Flatten1 (Flatten)          (None, 512)               0

 Dense1 (Dense)              (None, 128)               65664

 Dense2 (Dense)              (None, 20)                2580

=================================================================
Total params: 290,196
Trainable params: 290,196
Non-trainable params: 0
```

CHAPTER SIX

6. RESULTS AND DISCUSSIONS

6.1 Chapter Overview

This chapter covers the results obtained from the proposed solution and provides a comparative description of the results with graphs and tables, having previously discussed the concepts for implementation and the details of this work.

6.2 Result Discussion

The CNN is used to observe and see the differences of the accuracies among different results from the handwritten Ge'ez digit models. The accuracies are calculated using two of the most popular libraries in Python for machine learning and deep learning, namely TensorFlow and Keras. Training and validation accuracy were measured for 30 different epochs by changing out hidden layers for various combinations of convolution layers and using batch size 32 in all cases. Figures 6.1, 6.3, 6.5, 6.7, 6.9, and 6.11 illustrate the accuracy of the CNN, and figures 6.2, 6.4, 6.6, 6.8, 6.10, and 6.12 show the loss of the CNN with various convolution and hidden layer combinations. Table 6.1 shows the maximum and minimum training and validation accuracies of the CNN determined after experiments for six different

Table 6.1 Performance of CNN for the six different cases for various hidden layers

Case	Number of Hidden Layers	Batch Size	Minimum Training Accuracy		Minimum Validation Accuracy		Maximum Training Accuracy		Maximum Validation Accuracy		Overall Performance Test Accuracy (%)
			Epoch	Accuracy (%)	Epoch	Accuracy (%)	Epoch	Accuracy (%)	Epoch	Accuracy (%)	
1	9	32	1	88.15	1	91.75	28	99.23	18	95.82	95.65
2	14	32	1	85.01	1	89.00	28	98.74	20	94.99	94.71
3	13	32	1	85.96	1	89.63	26	98.63	20	95.28	94.98
4	10	32	1	88.88	1	91.89	30	99.36	27	95.64	95.42
5	8	32	1	87.41	1	89.90	29	99.77	27	94.84	94.42
6	12	32	1	88.77	1	91.94	30	98.44	12	96.15	96.21

Table 6.2 Loss of CNN for the six different cases for various hidden layers

Case	Number of Hidden Layers	Batch Size	Minimum Training Loss		Minimum Validation Loss		Maximum Training Loss		Maximum Validation Loss		Overall Test Loss
			Epoch	Loss	Epoch	Loss	Epoch	Loss	Epoch	Loss	
1	9	32	30	0.0232	8	0.2412	1	0.4515	1	0.3306	0.2946
2	14	32	28	0.0456	10	0.2571	1	0.5546	1	0.4149	0.2928
3	13	32	30	0.0423	14	0.2363	1	0.5214	1	0.4035	0.2908
4	10	32	27	0.0187	11	0.2371	1	0.4271	1	0.3266	0.3032
5	8	32	30	0.0068	8	0.3267	1	0.4705	28	0.4841	0.5504
6	12	32	24	0.0574	7	0.2077	1	0.4399	1	0.3213	0.2013

cases with different hidden layers, and Table 6.2 shows validation loss of the CNN in various cases for the recognition of Ge'ez handwritten digits.

The first hidden layer in the first case presented in figure 6.1 and 6.2 is the convolutional layer 1, which is used for feature extraction. It has 32 filters with a kernel size of 3x3 pixels, and it uses ReLU as an activation function. The next hidden layer is convolutional layer 2, which consists of 32 filters with a kernel size of 3x3 and ReLU. To minimize the spatial size of the output of a convolution layer, a pooling layer 1 is defined, with max-pooling and a pool size of 2x2 pixels. The next layers are two convolutional layers of a 64 filter with a kernel size of 3x3 and the ReLU activation function is applied to the model. A max-pooling layer 2 is applied after the convolution layer4. Next to the pooling layer 2, a regularization dropout layer is used to reduce the overfitting of the model by randomly eliminating 25% of the neurons in the layer. Convolution layer 5 with the channel size of 64 and filter size of 3x3 is applied after dropout. Convolutional layer 6 is the next hidden layer, which is made up of 128 filters with a kernel size of 3x3 pixels and ReLU. The next layer is max-pooling layer 3 with a dropout. A flatten layer is utilized to turn the 2D filter matrix into a 1D feature vector before entering the fully connected layers. After the flatten layer, the fully connected layer 1 is used, which comprises 128 neurons and ReLU. Finally, the fully connected layer 2 output layer, which determines the digits, has 20 neurons for 20 classes.

To output digits, the output layer has a softmax activation function. With a batch size of 32, the CNN is trained over 30 epochs. The performance has an overall test accuracy of 95.65%. The minimum validation accuracy is 91.75% at epoch 1, while the minimal training accuracy is 88.15% at epoch 1. At epoch 28, the highest training accuracy is 99.23%, whereas the highest validation accuracy is 95.82% at epoch 18. The overall model loss, in this case, is estimated to be around 0.2946. The training loss decreases exponentially when the iteration goes. The validation loss decreases from the pick value to the optimum value and then increases up to the 17th epoch. After the 19th epoch, the validation loss remains constant.

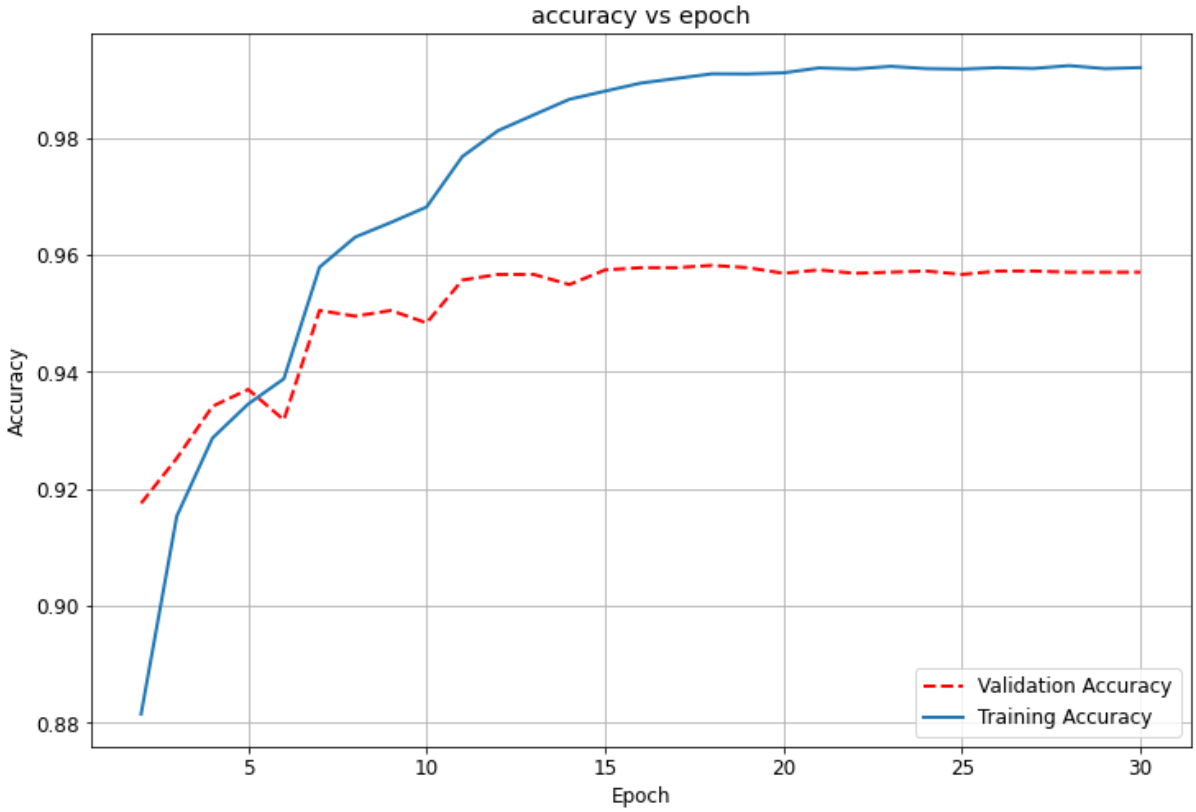


Figure 6.1 Observed accuracy for case 1

Figure 6.3 and figure 6.4 are defined for case 2, where convolution 1, convolution 2, pooling 1, convolution 3, convolution 4, and pooling 2 are used one after another. The first four convolution layers consist of 32 filters with a 3x3 kernel size. The next two hidden layers are convolution layers which are made up of 64 filters with a kernel size of 3x3 pixels. Max pooling and dropout layers are applied after the convolution layers. The next two layers are convolution layers with a channel size of 64 followed by a max-pooling layer.

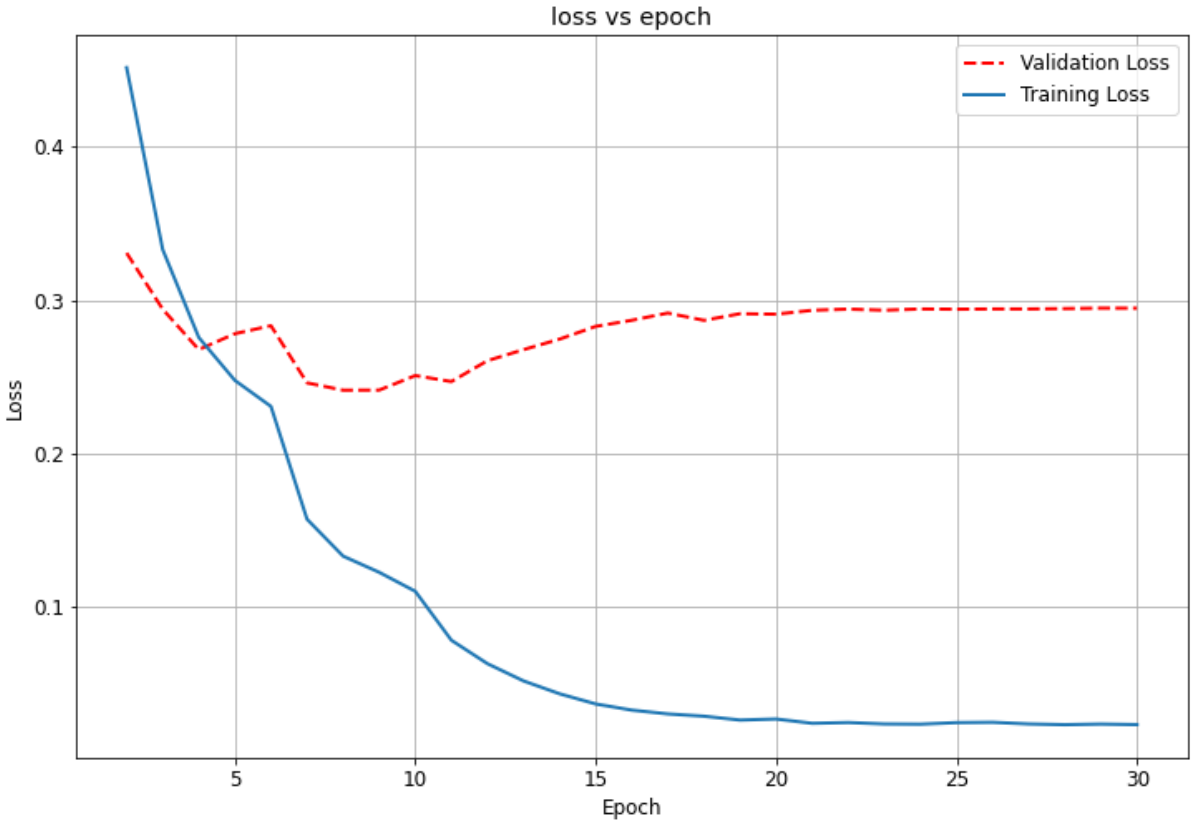


Figure 6.2 Observed loss for case 1

The next hidden layer is convolution layer 9 with a 3x3-kernel size of 128 filters. A max-pooling layer with a dropout is applied after the convolution layer. ReLU are used as an activation function in all convolution layers. The dimensions and hyper-parameters used in this and the next cases are the same as those used in case 1. The overall performance test accuracy is found to be 94.71%. The minimal training and validation accuracy is determined at epoch 1. The training accuracy is 85.01%, and the validation accuracy is 89.00%. Epoch 28 has the highest training accuracy, while epoch 20 has the highest validation accuracy. The maximum accuracy for training and validation is 98.74% and 94.99%, respectively. The total model loss is estimated to be approximately 0.2928.

Two convolutions are taken one after the other in case 3, as shown in figure 6.5 and 6.6, followed by a pooling layer. Two other convolution layers are next, followed by a max-pooling layer and dropout layer. The next layers are three consecutive convolution layers followed by a max-pooling layer. Before the flatten layer, two convolutional layers, the max-pooling layer, and the dropout layer was applied. A flatten layer is followed by the two fully connected layers.

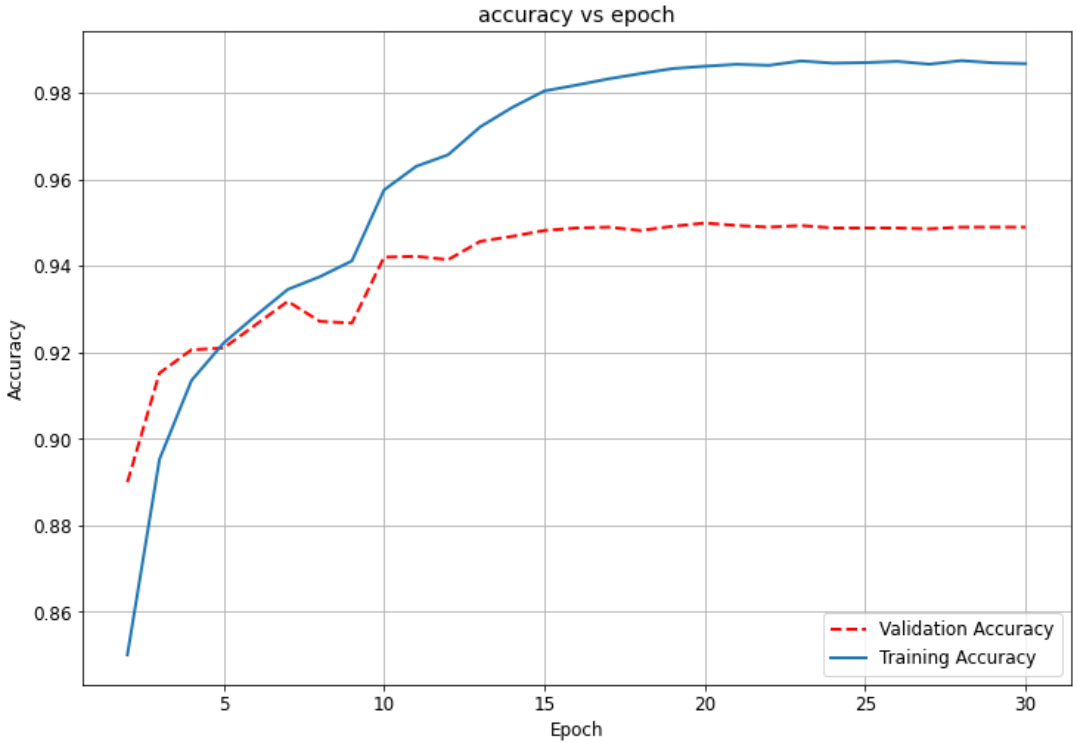


Figure 6.3 Observed accuracy for case 2

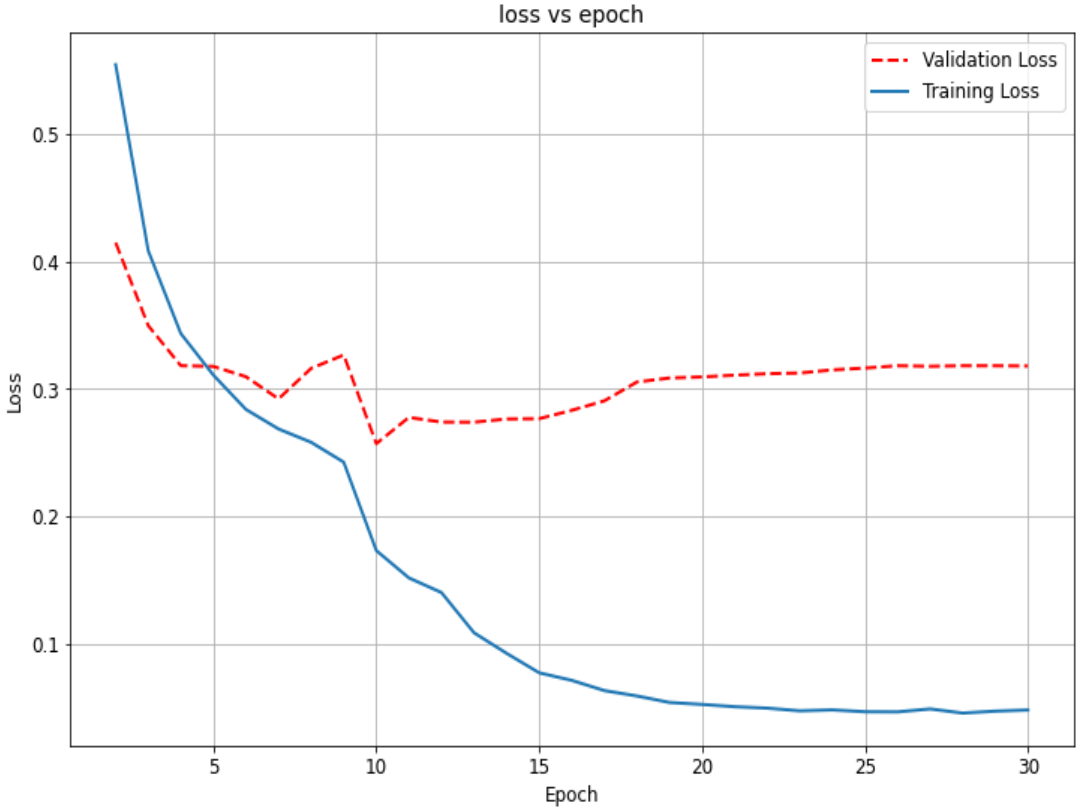


Figure 6.4 Observed loss for case 2

The overall performance test accuracy is found to be 94.98%. At epoch 1, the minimum training accuracy is 85.96%, whereas the minimum validation accuracy is 89.63%. The maximum training and validation accuracies are 98.63% and 95.28% found at epochs 26 and 20 respectively. The total model loss is found at approximately 0.2908.

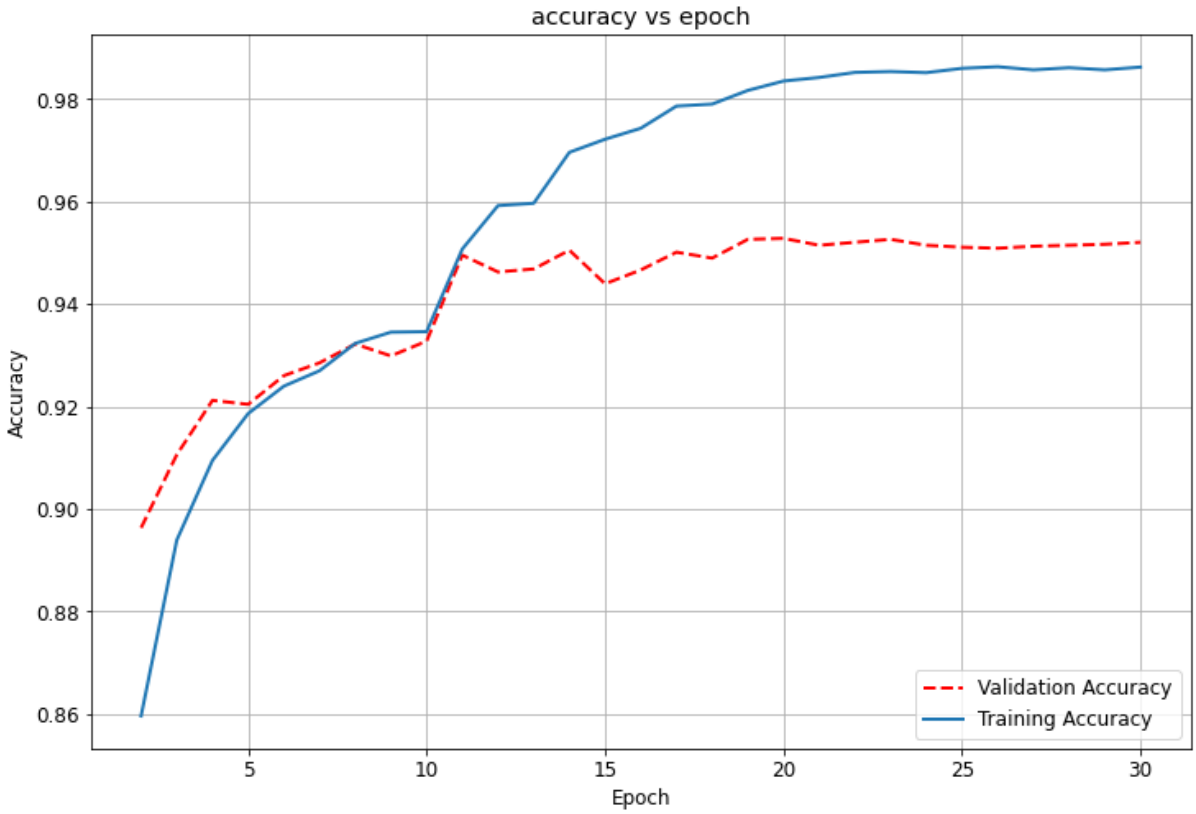


Figure 6.5 Observed accuracy for case 3

For case 4, shown in figure 6.7 and figure 6.8, the max-pooling layer was applied after the three convolutional layers. The max-pooling layer is followed by three convolution layers which are followed by a max-pooling layer with a dropout. The next layer is convolution layer 7 with a max-pooling layer and dropout. After a flatten layer, there are two fully connected layers with no dropout. The overall test accuracy in the performance is found 95.42%. At epoch 1, the minimum training and validation accuracies were found to be 88.88% and 91.89%, respectively. The maximum training accuracy is 99.36% is found at epoch 30, and the maximum validation accuracy is 95.64% is found at epoch 27. The total model loss is 0.3032.

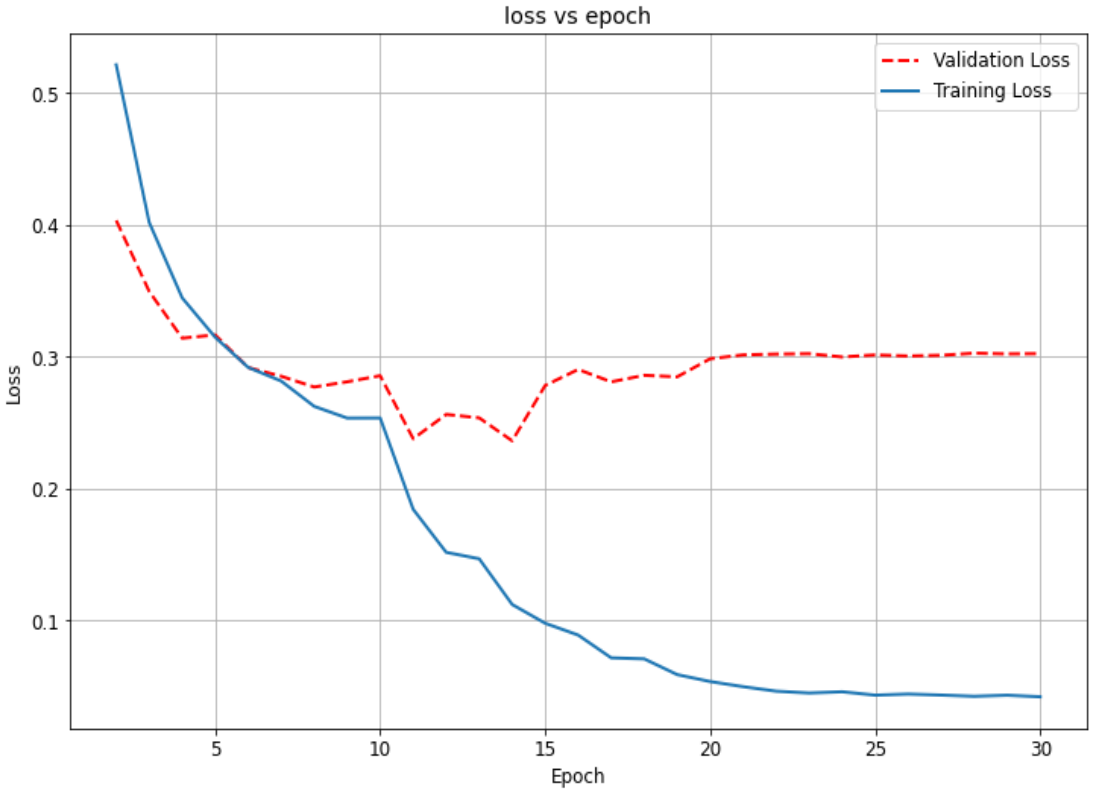


Figure 6.6 Observed loss for case 3

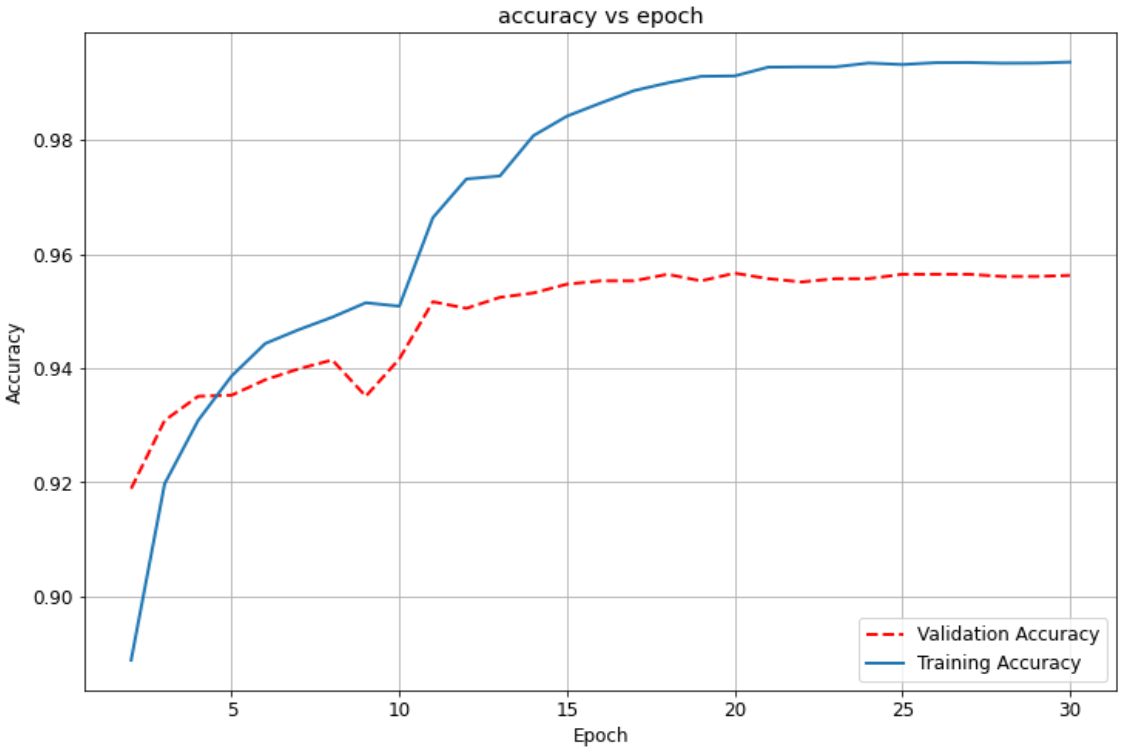


Figure 6.7 Observed accuracy for case 4

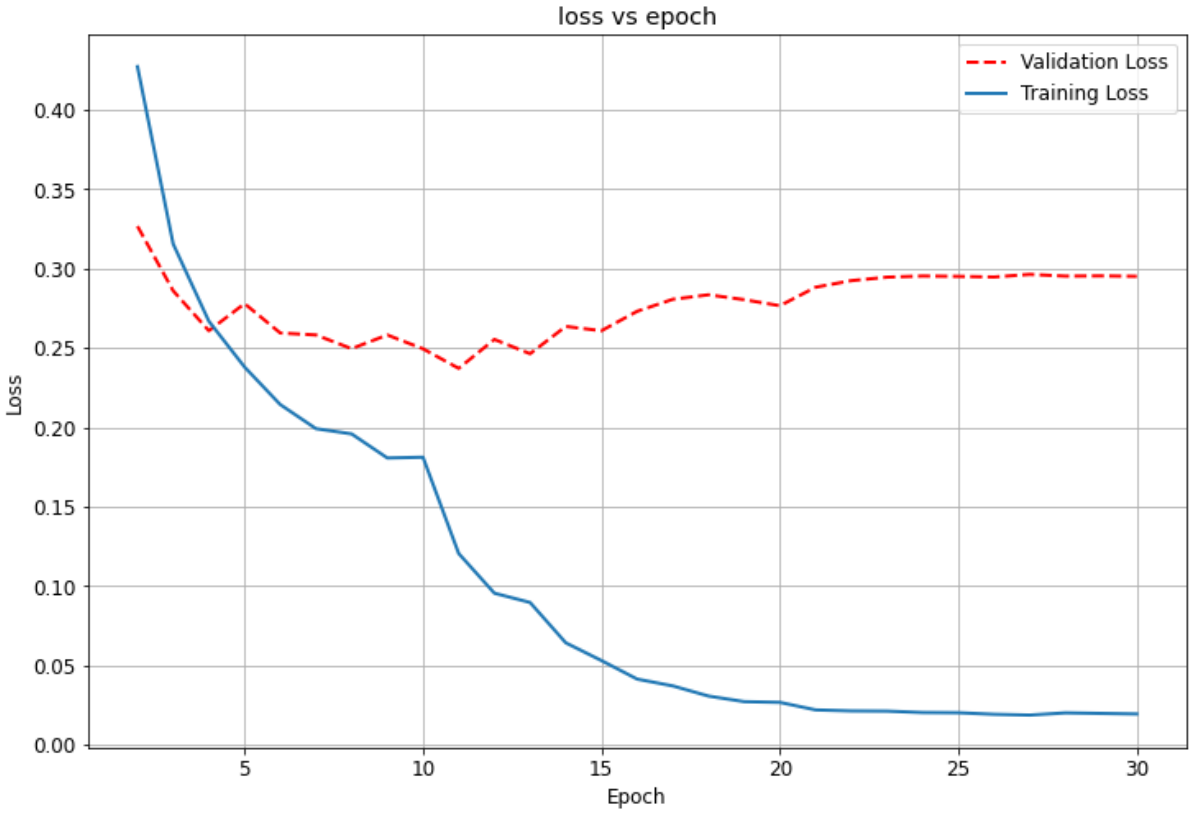


Figure 6.8 Observed loss for case 4

Case 5 is shown in figure 6.9 and figure 6.10, and for this case, three convolutions are taken and these are followed by a max-pooling layer. Next to the pooling layer, a regularization layer dropout is used to reduce overfitting by randomly eliminating 20% of the neurons in the layer. The next layers are three convolution layers followed by a max-pooling layer and a dropout layer. The two fully connected layers are followed by a flatten layer. The overall performance test accuracy was found to be 94.42%. At epoch 1, the minimum training accuracy is 87.41%, while the minimum validation accuracy is 89.90%. Epoch 29 has the highest training accuracy, while epoch 27 has the highest validation accuracy. The maximum accuracy for training and validation is 99.77% and 94.84%, respectively. The total test loss of the model, in this case, is 0.5504. The validation loss of the model increase when the iteration goes. It shows the model became overfit to the training data. The maximum model loss is occurred in this case from all the six cases. Also, the minimum model accuracy among all cases occurred in case 5. It shows that overfitted models give a high model loss and low accuracy for a new test dataset.

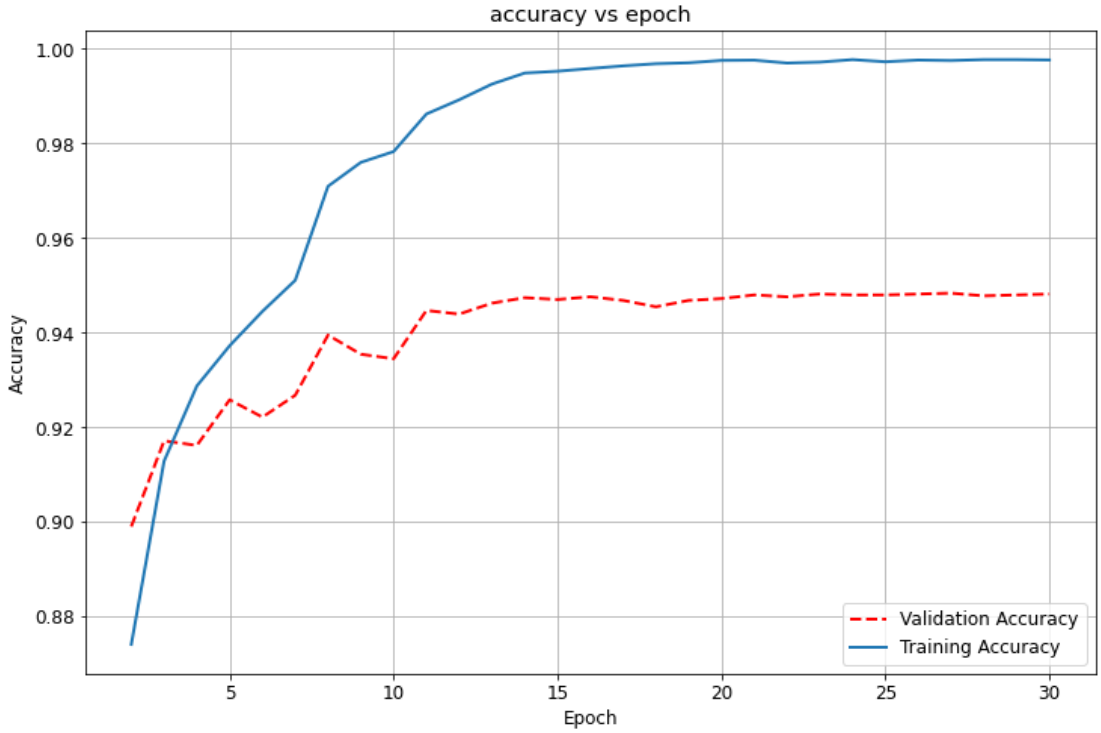


Figure 6.9 Observed accuracy for case 5

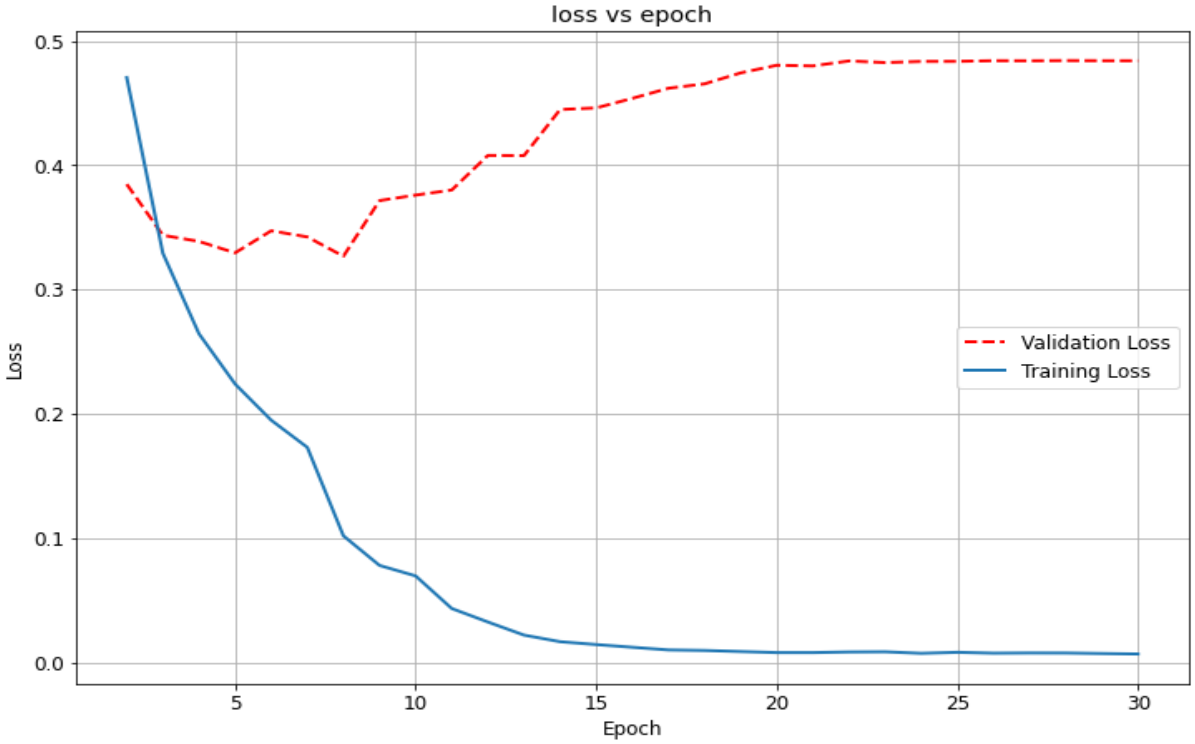


Figure 6.10 Observed loss for case 5

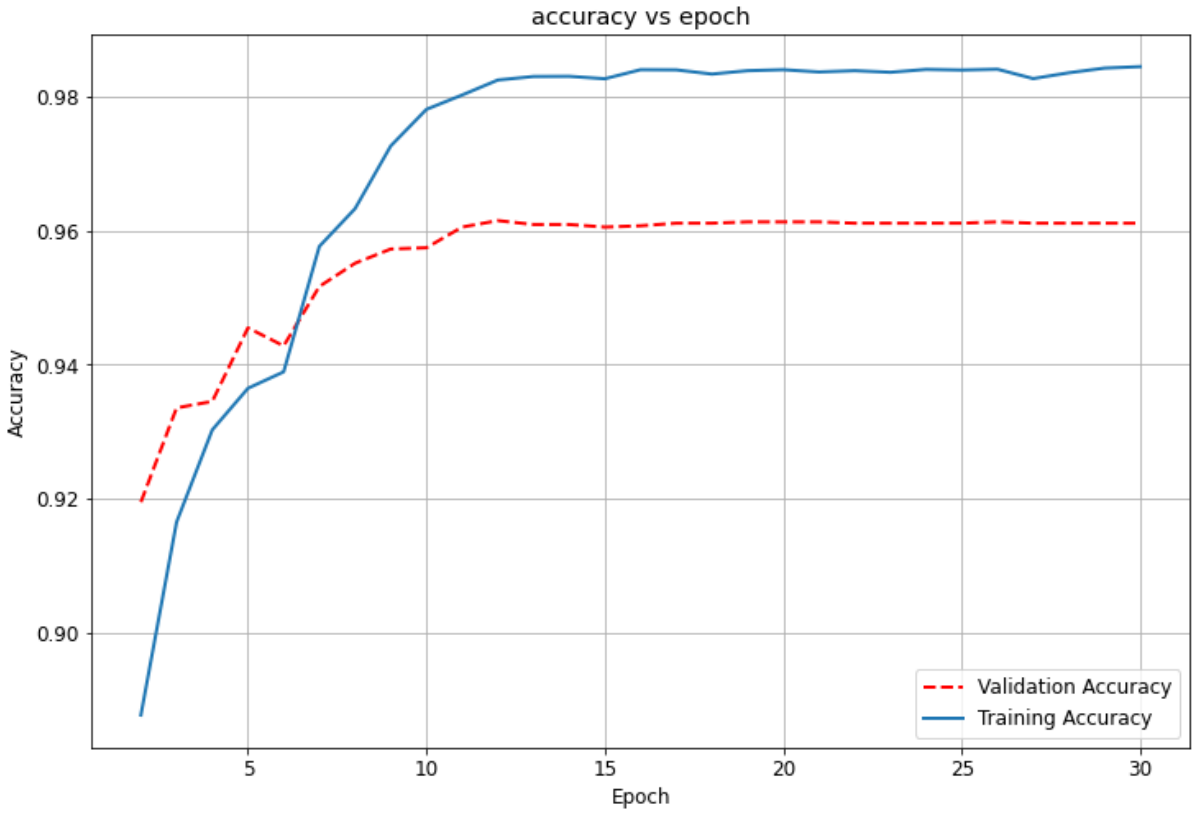


Figure 6.11 Observed accuracy for case 6

Finally, in Case 6 (figure 6.11 and 6.12), three convolutions are taken one after the other, followed by a pooling layer. Three convolution layers are next, followed by a max-pooling layer. Next to the pooling layer 2, a regularization layer dropout is applied to reduce overfitting by randomly eliminating 20% of the neurons in the layer. Convolutional layer 7 is the next hidden layer, followed by a max-pooling layer and a dropout layer. Convolution layer 8 is used after the dropout layer. Max-pooling layer 4 with a dropout was applied after the convolution layer 8. The flatten layer, followed by two fully connected layers, is applied. The overall performance test accuracy was found to be 96.21%. At epoch 1, the minimum training and validation accuracies were found to be 88.77% and 91.94%, respectively. Epoch 30 has the highest training accuracy, while epoch 12 has the highest validation accuracy. The maximal training and validation accuracy is 98.44% and 96.15%, respectively. The total model loss is found approximately 0.2013. The training loss decrease when the number of epoch goes, but the validation loss fluctuate for 10 epochs and then remain constant for the remaining number of epochs.

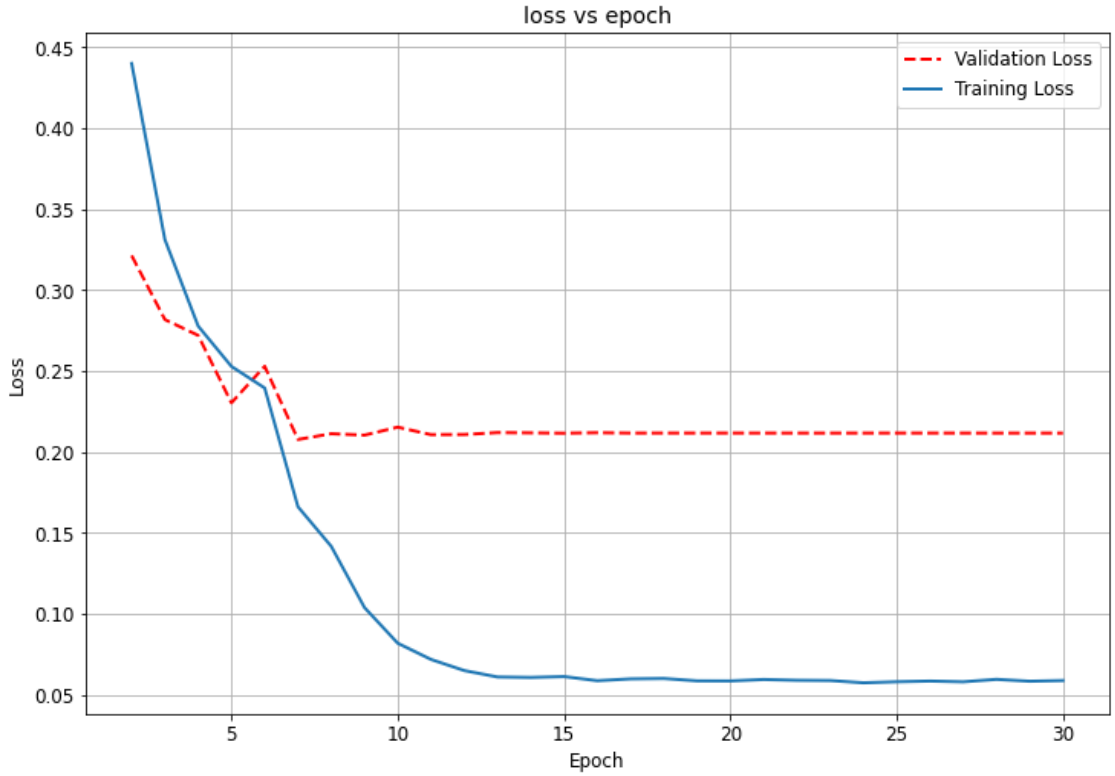


Figure 6.12 Observed loss for case 6

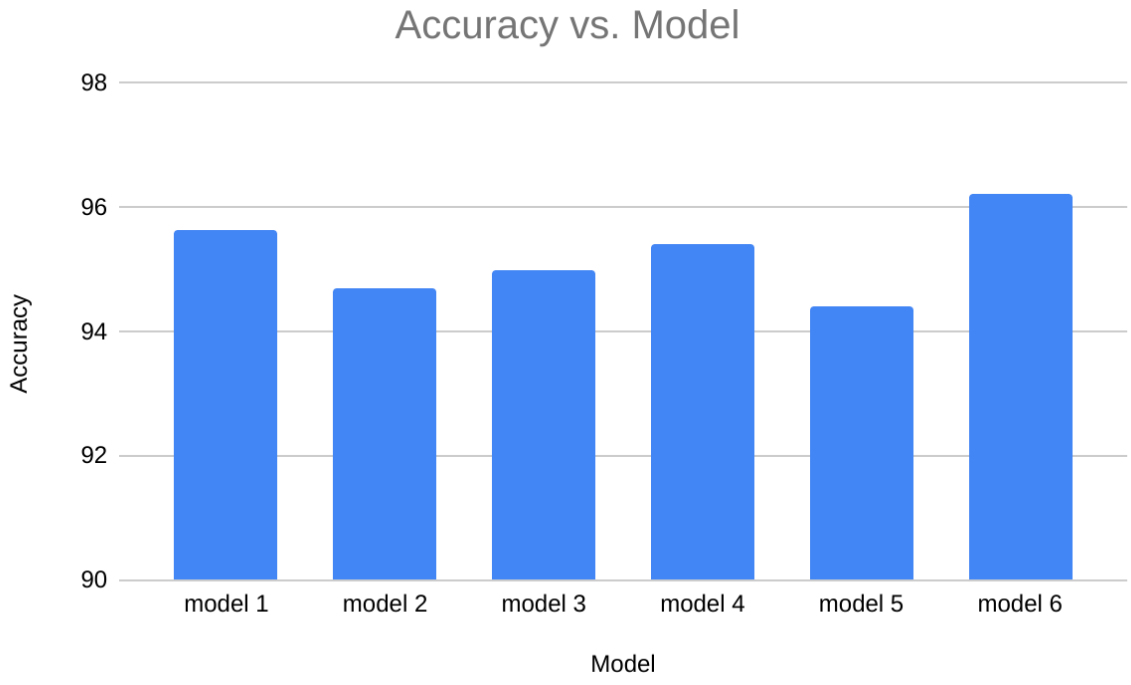


Figure 6.13 Accuracy of the different models

By varying the hidden layers, the change of the performance for handwritten digits was observed over 30 epochs in the experiment. Accuracy curves for the six cases for each parameter were generated using a handwritten Ge'ez digit dataset. The six cases behave differently due to the different combinations of hidden layers. The maximum and minimum accuracies for several hidden layer variations were recorded using a batch size of 32. As shown in the figure 6.13, the highest test accuracy in performance was found to be 96.21% for 30 epochs in case 6 among all the observations (Conv1, Conv2, Conv3, pool1, Conv4, Conv5, Conv6, pool2 with dropout, Conv7, pool3 with dropout, Conv8, pool4 with dropout, flatten layer, 2 fully connected layers). This type of greater accuracy will work in Ge'ez handwritten digit recognition to help the machine execute more efficiently. In case 5, however, the lowest accuracy among all observations in the performance was discovered to be 94.42% (Conv1, Conv2, Conv3, pool1, Conv4, Conv5, Conv6, pool2, flatten layer, 2 fully connected layers). Furthermore, the total highest model loss in case 5 is 0.5504, while the total lowest model loss in case 6 with dropout is around 0.2013 (Figure 6.14). With this minimal loss, CNN will be able to achieve greater image quality and noise processing. From the observed result, we choose the best model from six cases that have the highest model test accuracy and lowest test loss. Therefore, case 6 model with the highest accuracy of 96.21% and the lowest loss of 0.2013 is our proposed model for this research work.

Figure 6.14 Loss of the different models

Table 6.3 Model comparison of CNN vs. ANN

No.	Model	Number of Hidden Layers	Accuracy	Loss
1	Artificial Neural Network (ANN)	12	56.71%	3.9041
2	Convolutional Neural Network (CNN)	12	96.21%	0.2013

We have conducted experiments using other deep learning algorithms such as Artificial Neural Network (ANN) to compare them with the CNN model that we proposed. We used them on our dataset and measured their performance. As shown in table 6.3 ANN has an accuracy of 56.71% and model loss of 3.9041. The model is learning the training data well but fails to generalize. The validation loss continues to increase as we train the model for more epochs, indicating an unstable and unusable model. The model accuracy plateaus around 56.71%, while the model overfits on the training data. 56.71% accuracy for a 20-class dataset with 41,566 training images is almost as bad as it can get. The reason is simple dense layers are not designed to capture the complexity of 2-dimensional image data. The model needs a convolutional layer to do the job right. Compared to CNN, ANN has a poor performance when using it with the image dataset. This concludes that the CNN algorithm that we proposed is the best.

The previous work on Ge'ez handwritten digit recognition is done by (Beyene, 2019) achieved 89.88% accuracy using an ANN model. This study evaluates CNN models with different layers with different hyper-parameters. Compared with the previous work, we improve the accuracy of the recognition from 89.88% to 96.21% by using CNN, increasing the dataset size, and enhancing the quality of the image by using pre-processing techniques on the dataset. Comparing the proposed solution with (Beyene, 2019) work in terms of the state of the art, his work uses ANN to develop the model, but the proposed model was develop by using CNN. CNN shows a better performance compared with ANN because for image dataset convolution layers extract much information compared with a dense layer, which is the main part of ANN. Therefore, the main difference of the two models is the presence and absence of the convolution layer which is the crucial part of the CNN.

6.3 Research Question Discussion

This study answers to each of the research questions raised in the first chapter. This research provides the following answers to each question:

Answer for Q1: For handwritten digit recognition, we use some methods and techniques. The first technique is to collect data for the extraction of a pattern by using machine-learning algorithms. If there is a public dataset then we use it, but for some languages like Amharic, there is no available dataset for the recognition study. So collecting from individuals or documents is the first step of the work. The next step is digitizing the collected data into electronic image format. Then pre-process the image data to increase the quality of the images. Resize the image dimension, normalization, color conversions, and other techniques are used for the pre-processing method. Before feeding, a data into an algorithm to train the model the data should be of good image quality. The quality of the image has a high effect on the performance of the model.

Answer for Q2: Deep learning algorithms are preferable for complex problems like image classification. Deep learning algorithms extract detailed information deeply by using multi-hidden layers. The first layers extract high-level features like edge, corner, etc. The next layers extract the low-level features from the raw data. Deep learning methods are used to learn feature hierarchies, which are made up of features from higher levels of the hierarchy that are composed of lower-level features. Without depending only on human-crafted features, a system can learn complicated functions mapping the input to the output directly from data by automatically learning features at multiple levels of abstraction. For small data deep learning have the same accuracy as other machine learning algorithms, but for large dataset deep learning have the highest accuracy compared with other algorithms. So deep learning algorithms improve the performance of the model for complex problems like handwritten recognition.

Answer for Q3: In deep learning models, we use parameters and hyper-parameters to improve the model performance by tuning them. The parameters of the neural network models are weight and bias. Initially start with random values and update those values by using error cost functions like SGD and learning rate. The deep learning models are trained for several iterations (epochs). For each epoch, the weight and bias are update based on the

value of the learning rate, and cost functions values. The hyper-parameters used to improve the recognition models are learning rate, epoch, number of hidden layers, and batch size. All of them are used for improving the performance of the model to find the optimal values of the parameters.

CHAPTER SEVEN

7. CONCLUSION AND FUTURE WORK

7.1. Conclusion

In this research work, we used convolutional neural networks to recognize Ge'ez handwritten digits with 20-digit classes. CNN's are the current state-of-the-art algorithm for classifying image data and are widely used. On a prepared form for data collection, we collected a large number of Ge'ez handwritten digits from individual handwriting. The handwritten documents are scanned and pre-processed to get 32 x 32-pixel digit images, which are input into the CNN for classification. We offered a new public dataset for the Ge'ez handwritten digit dataset, which is open to all researchers (publicly available from: https://drive.google.com/file/d/1abJWvSYSyw8mLQ5Blg_lYAJng1K3LtGS/view?usp=sharing). We used CNN architecture from the deep learning approaches to develop Ge'ez handwritten digit recognition system. Many trial and error neural network configuration tuning mechanisms were used to get the best fit model of CNN-based architecture. In comparison to earlier research works on Ge'ez handwritten digit recognition, we were able to achieve higher recognition accuracy using the developed CNN model. We have achieved an accuracy of 96.21% and a model loss of 0.2013. Even though that much work has been done in the English language to recognize handwritten digits, only a small amount of work has been done in the Amharic language. Due to a lack of research work on the area, there is a big challenge to get a dataset for the Amharic language. In this research, we develop a dataset that can be used by other researchers in the future.

7.2. Future Work

This thesis covers many grounds, but still, there are some concepts to be refined and some to be added. Many works have to be done in the future, and the ones I will be stating here are the main ones. As we can see from this thesis, the current work only supports a single handwritten Ge'ez digit, but in the future, add the support for multi-digit. Multi-digit support as most of the numbers we write is multi-digit, so as a future work the model should be able to support multi-digit.

We can also see that the dataset is relatively small in size (only 52, 400 gathered from different individuals), but this will impact the accuracy, so in the future, increase the dataset, and broaden the diversity of the individuals to better the accuracy. In this work we have only used data gathered from individuals, but also supporting or using historical documents as our data we can increase the size of the dataset tremendously. So in the future, the dataset will also have historical data as the dataset for the model.

The data used in this thesis came from a diverse group of individuals but mainly consists of individuals from the educational sector especially students from different levels; in the future I will include more diverse groups by demography, age, background, and lifestyle. This will enable the data to cover a broader range of handwriting, which in turn will increase the accuracy of the model. The other main thing to be added in the future is that the current work focuses on the recognition of the handwritten digit, but in the future, handwritten digit detection will also be added alongside the recognition part. This will in turn enhance the work because before even attempting to do the recognition the model should know whether the document or image contains the content that is needed or not.

In general, this thesis covers a wide range of concepts, but still, there is a lot of work to be done and I have mentioned some of the above.

SPECIAL ACKNOWLEDGMENTS

This research project is funded by Adama Science and Technology University under the grant number:

ASTU/SM-R/559/22

Adama, Ethiopia

REFERENCES

Ali, S., Shaukat, Z., Azeem, M., Sakhawat, Z., & Mahmood, T. (2019). An efficient and improved scheme for handwritten digit recognition based on convolutional neural network. 1(9), 1–9. https://doi.org/10.1007/s42452-019-1161-5

Arnal Barbedo, J. G. (2013). Digital image processing techniques for detecting, quantifying and classifying plant diseases. SpringerPlus, 2(1), 1-12.

Ashagrie, F., & Boran, D. (2019). Ancient Ge'ez script recognition using deep learning. 1(11), 1–7. https://doi.org/10.1007/s42452-019-1340-4

Assabie, Y., Bigun J. (2011). Offline handwritten Amharic word recognition. In: Pattern Recognition Letters, Juni, 10Nr. 8, S. 1089-1099. http://dx.doi.org/10.1016/j.patrec.2011.02.007

Belay, B., Habtegebrial, T., Meshesha, M., & Liwicki, M. (2020). Applied sciences Amharic OCR : An End-to-End Learning. 1–13. https://doi.org/10.3390/app10031117

Beyene, E. G. (2019). Handwritten and machine printed OCR for Ge'ez numbers using artificial neural network. NeurIPS 2019 Workshop on Machine Learning for the Developing World, Nips.

Chen, S., Almamlook, R., Gu, Y., & Wells, L. (2018). Offline handwritten digits recognition using machine learning. Proceedings of the International Conference on Industrial Engineering and Operations Management Washington DC, USA, September 27-29, 2018, 133(June 2000), 274–286.

D. Ciresan (2008). Avoiding segmentation in multi-digit numeral string recognition by combining single and two-digit classifiers trained without negative examples, in: International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, IEEE, pp. 225–230.

Dutt, A., & AashiDutt. (2017). Handwritten digit recognition using deep learning. International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), 6(7), 990–997.

E. Granell, E. Chammas, L.L. Sulem, C.D.M. Hinarejos, C. Mokbel, B.I. Cirstea (2018). Transcription of Spanish historical handwritten documents with deep neural networks. Imaging 4 (15) 1–22.

Elitez, O. (2015). Handwritten digit string segmentation and recognition using deep learning (Master's thesis, Middle East Technical University).

F.C. Ribas, L.S. Oliveira, A.S. Britto Jr, R. Sabourin (2015). Handwritten digit segmentation: a comparative study, Int. J. Doc. Anal. Recognition. 16 127–137.

Gondere, M. S., Schmidt-thieme, L., Boltena, A. S., & Jomaa, H. S. (2019). Handwritten Amharic character recognition using a convolutional neural network. 1–12. https://doi.org/10.5445/KSP/1000098011/09

Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., & Lew, M. S. (2016). Deep learning for visual understanding: A review. Neurocomputing, 187, 27-48.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural computation, 18(7), 1527-1554.

Hirschman, I.I., Widder, D.V. (2012). The Convolution Transform. Courier Corporation.

Hossain, A., & Ali, M. (2019). Recognition of handwritten digit using convolutional neural network (CNN). Global Journal of Computer Science and Technology: Neural & Artificial Intelligence, 19(2).

Lee, K. B., Cheon, S., & Kim, C. O. (2017). A convolutional neural network for fault classification and diagnosis in semiconductor manufacturing processes. IEEE Transactions on Semiconductor Manufacturing, 30(2), 135-142.

Krizhevsky, I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, pp. 1097-1105.

Kusetogullari, H., Yavariabdi, A., Hall, J., & Lavesson, N. (2020). DIGITNET: A deep handwritten digit detection and recognition method using a new historical handwritten digit dataset. 23, 100182. https://doi.org/10.1016/j.bdr.2020.100182

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature (2015). May, 28(521), 7553.

LeCun, Y. (2015). LeNet-5, convolutional neural networks. URL: http://yann. lecun. com/exdb/lenet, 20(5), 14.

Lin, M., Chen, Q., & Yan, S. (2013). Network in network. arXiv preprint arXiv:1312.4400.

Lu, Y., Yi, S., Zeng, N., Liu, Y., & Zhang, Y. (2017). Identification of rice diseases using deep convolutional neural networks. Neurocomputing, 267, 378-384.

Luo, X., Shen, R., Hu, J., Deng, J., Hu, L., & Guan, Q. (2017). A deep convolution neural network model for vehicle recognition and face recognition. Procedia Computer Science, 107, 715-720.

Nair, A. M., Aldo, C., Bose, B., Joseph, A., Praseetha, V. M., & J, S. E. M. (2021). Handwritten character recognition using deep learning in android phones. International Research Journal of Engineering and Technology (IRJET), July, 61–66.

Nogueira, K., Penatti, O. A., & Dos Santos, J. A. (2017). Towards better exploiting convolutional neural networks for remote sensing scene classification. Pattern Recognition, 61, 539-556.

R. Saabni (2016). Recognizing handwritten single digits and digit strings using deep architecture of neural networks. International Conference on Artificial Intelligence and Pattern Recognition, IEEE, pp. 1–6.

S. Choi, I. Oh (1999). A segmentation-free recognition of two touching numerals using neural networks. International Conference on Document Analysis and Recognition, IEEE, pp. 253–256.

S. J. Russell and P. Norvig (2016). Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited.

Scherer, D., Müller, A., & Behnke, S. (2010, September). Evaluation of pooling operations in convolutional architectures for object recognition. In International conference on artificial neural networks (pp. 92-101). Springer, Berlin, Heidelberg.

Siddique, F., Sakib, S., & Siddique, A. B. (2019). Recognition of handwritten higit using convolutional neural network in Python with Tensorflow and comparison of performance for various hidden layers. 5th International Conference on Advances in Electrical Engineering (ICAEE), 2–7. https://doi.org/10.1109/ICAEE48663.2019.8975496

Traoré, Boukaye Boubacar, Kamsu-Foguem, Bernard & Tangara (2018). Fana deep convolution neural network for image recognition. Ecological Informatics, 48. 257-268. ISSN 1574-9541

Upadhyay, B. D., Chaudhary, S., & Gulhane, M. (2020). A review paper on handwritten character recognition using machine learning. International Journal of Future Generation Communication and Networking, 13(2), 1256–1266.

Z. Shi, DateFinder (2016). Detecting date regions on handwritten document images based on positional expectancy, Master's thesis, University of Groningen, The Netherlands.

Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham.

APPENDIXES

Appendix A: Sample Code

Data preprocessing

```
import os
from PIL import Image
baseDIR = "/home/andalus/Documents/MSc/Thesis/Dataset"
_dirs = "/home/andalus/Documents/MSc/Thesis/Dataset/CamScanner2-
rename/"
def load_image(_dir):
  img = Image.open(_dir)
  return img

def save_crop_images(img, number):
  dir = f"{baseDIR}/Crop/crop/"
  if not os.path.isdir(dir):
      os.mkdir(dir)
  path = f"{dir}{number}.jpg"
  img.save(path)

def crop(img, pleft=5, ptop=20, pright=4.5, pbottom=13,
   number=None):
  width, height = img.size
  left = pleft * width // 100
  top = ptop * height // 100
  right = width - pright * width // 100
  bottom = height - pbottom * height // 100

  img = img.crop((left, top, right, bottom))
  save_crop_images(img, number)
  return img

def save_partition_images(img, number, level):
  dir = f"{baseDIR}/Crop/partition/"
  if not os.path.isdir(dir):
      os.mkdir(dir)
  path = f"{dir}{number}_{level}.jpg"
  img.save(path)
```

```
def partion_image(img, ratio=1/44.0, n=5, number=None):
  images = []
  srt = 0
  width, height = img.size
  space = ratio*height
  size = (height - space*(n-1)) // n
  threshold = 0.2 * space
  for i in range(n):
      _img = img.crop((0, srt, width, srt+size - threshold))
      if i == n - 1:
          _img = img.crop((0, srt, width, height))
      images.append(_img)
      srt += size+space
      save_partition_images(_img, number, i+1)
  return images

def extract_numbers(img):
  width, height = img.size
  sizex = width // 10
  sizey = height // 2
  numbers = []
  SHIFTX = 5
  SHIFTY = 5
  srty = 0

for y in range(2):
      srtx = max(0, sizex - 1600)
      for x in range(10):
       _img = img.crop((srtx + SHIFTX, srty + SHIFTY, srtx+sizex -
                        SHIFTX, srty+sizey - SHIFTY))
          numbers.append(_img)
          srtx += sizex
      srty += sizey
  return numbers

fnames = list(range(1, 11))+list(range(20, 101, 10))+ [1000]
START1 = 1
END1 = 283
START2 = 286
END2 = 287
START3 = 295
END3 = 465
```

```
START4 = 520
END4 = 524
IMAGE_LIST = list(range(START1, END1 + 1)) \
  + list(range(START2, END2 + 1)) \
  + list(range(START3, END3 +1)) \
  + list(range(START4, END4 +1))
image_list = {i: _dirs + str(i) +".jpg" for i in IMAGE_LIST}
for COUNTER, img_dir in image_list.items():
  img = crop(load_image(img_dir), number=COUNTER)
  images = partion_image(img, number=COUNTER)
  for c, im in enumerate(images):
      nums = extract_numbers(im)
      for i, im in enumerate(nums):
          curr = f"{baseDIR}/Digits/{fnames[i]}/"
          if not os.path.isdir(curr):
              os.mkdir(curr)
          im.save(f"{curr}{COUNTER}_{c+1}.jpg")
```

image resize and inverse

```
import cv2
import os
saveBaseDIR = "/home/andalus/Documents/MSc/Thesis/Hand
written/Original Dataset/PreProcessDataset"
def resize_crop_inverse(path, num, file):
  im = cv2.imread(path)
  crop_img = im[:-10, 10:]
  imS = cv2.resize(crop_img, (32, 32))
  imagem = cv2.bitwise_not(imS)
  dir = f"{saveBaseDIR}/{num}"
  if not os.path.isdir(dir):
      os.mkdir(dir)
  new_path = os.path.join(dir, file)
  cv2.imwrite(new_path, imagem)
baseDIR = "/home/andalus/Documents/MSc/Thesis/Original Dataset"
dir = f"{baseDIR}/Crop/"
path = f"{dir}/digits.csv"
root_path = '/home/andalus/Documents/MSc/Thesis/Original
Dataset/Digits/'
all_files = []
```

```
for root, dirs, files in os.walk(f"{baseDIR}/Digits/"):
  if root == root_path:
      continue
  num = os.path.basename(os.path.normpath(root))
  for file in files:
      if not file[:3] in [str(i) for i in range(520, 525)]:
          continue
      if file.endswith(".jpg"):
          path = os.path.join(root, file)
          resize_crop_inverse(path, num, file)
```

Telegram Bot

```
import logging
import time
from telegram import *
from telegram.ext import *
from PIL import Image
from train import model_construct
from image_processing import process_image

model = model_construct()

def start(update, context):
       update.message.reply_text('Wellcome to Ge'ez Digit
                   Recognition bot\n Enter your drawing')

def error_message(update, context):
      update.message.reply_text("Send your drawing for prediction")

def predict(update, context):
         photo = update.message.photo
         images = []
         for image in photo:
               img = context.bot.getFile(image.file_id)
               path = f'image/{int(time.time() *
                                100)}_{image.file_unique_id}.png'
               img.download(path)
               images.append(path)
         data = images[0] # 4 different size images are recieved,
                    index 0 is the minimum size among them
```

```
        image = Image.open(data)
        x = process_image(image)
        if x is None:
               update.message.reply_text(
                         "invalid image",
                         reply_markup=ReplyKeyboardRemove())
              update.message.reply_text(
                         "Send your drawing for prediction",
                         reply_markup=ReplyKeyboardRemove())
               return 0
        prediction = model.predict(x)[0]
         classes = [
                   '1', '10', '100', '10000', '2', '20','3', '30',
                   '4', '40', '5', '50', '6', '60', '7', '70',
                    '8', '80', '9', '90']
        result = [(prediction[i], classes[i]) for i in range(20)]
        result.sort(reverse=1)
        update.message.reply_text(
                   f"The predicted value is {result[0][1]} with a
                    probability {(100 * result[0][0]):.2f}%",
                   reply_markup=ReplyKeyboardRemove())
        update.message.reply_text(
                   "Send your drawing for prediction",
                   reply_markup=ReplyKeyboardRemove())
logging.basicConfig(format="%(asctime)s - %(name)s - %(levelname)s
               -  %(message)s", level=logging.INFO)
logger = logging.getLogger(__name__)

def main():
        persistence =PicklePersistence(filename='conversationbot')
        updater =Updater(token="APItoken",persistence=persistence)
        dispatcher = updater.dispatcher
        dispatcher.add_handler(CommandHandler("start", start))
        dispatcher.add_handler(MessageHandler(Filters.photo |
                     Filters.document.category("image"), predict))
        dispatcher.add_handler(MessageHandler(Filters.all,
                    error_message))
        updater.start_polling()
        updater.idle()
if __name__ == '__main__':
  main()
```

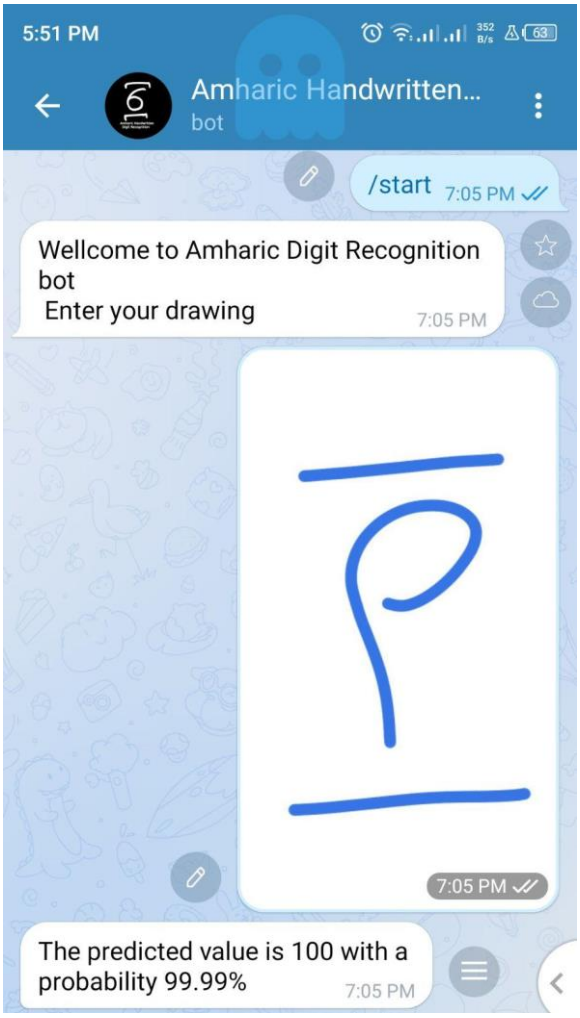
```
import numpy as np
from tensorflow import keras
from tensorflow.keras.preprocessing.image import img_to_array
from PIL import Image, ImageOps

def to_grayscale(image):
  return image.convert('L')

def invert_colors(image):
  return ImageOps.invert(image)

def resize_image(image):
  return image.resize((32, 32), Image.LINEAR)

def process_image(image):
  image = to_grayscale(image)
  image = invert_colors(image)
  image = resize_image(image)
  img = img_to_array(image)
  img = np.expand_dims(img, axis=0)
  return img

def process_batch_image(images):
  if not images: return []
  batch = []
  for image in images:
      image = to_grayscale(image)
      image = invert_colors(image)
      image = resize_image(image)
      img = img_to_array(image)
      img = np.expand_dims(img, axis=0)
      batch.append(img)
  con = batch[0]
  for img in batch[1:]:
      con = np.concatenate((con, img), axis=0)
  return con

def model_construct():
  model = keras.models.load_model(
              '30-epoch-val-acc-96-21-lr-1e-3-batch-32-adam.h5')
  return model
```

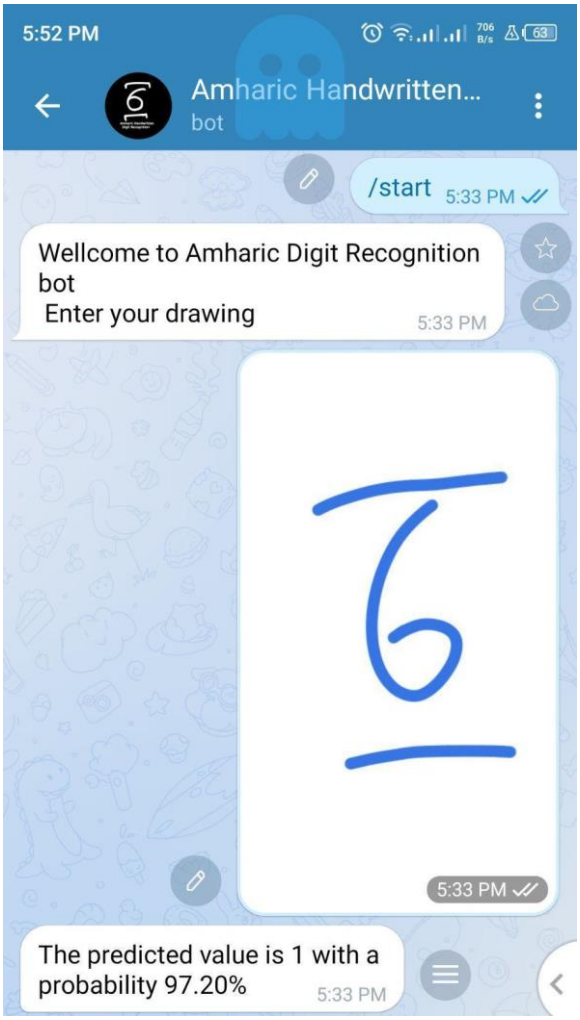
REST API

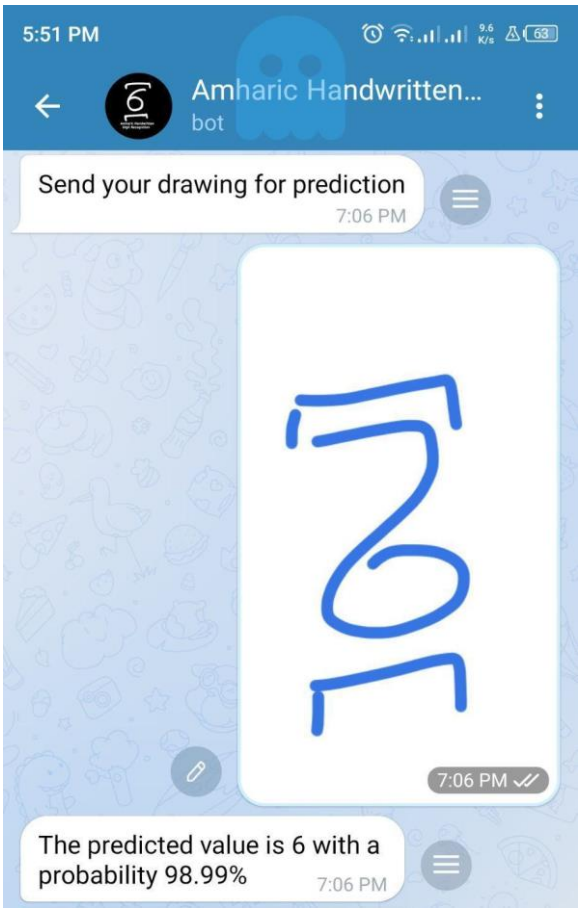
```
from rest_framework import permissions
from rest_framework.decorators import action
from rest_framework.parsers import FormParser, MultiPartParser
from rest_framework.response import Response
from rest_framework.viewsets import ViewSet
from PIL import Image
from .train import model_construct
from .image_processing import process_image, process_batch_image
class DigitRecognitionAPI(ViewSet):
  parser_classes = [MultiPartParser, FormParser]
  permission_classes = (permissions.AllowAny,)
  def create(self, request):
      form_data = request.data.dict()
      data = form_data.get('image', None)
      if data is None:
          return Response("image is required")
      image = Image.open(data)
      x = process_image(image)
      if x is None:
          return Response("invalid image")
      model = model_construct()
      prediction = model.predict(x)[0]
      classes = [
                '1', '10', '100', '1000', '2', '20', '3', '30',
                '4', '40', '5', '50', '6', '60', '7', '70', '8',
                '80', '9', '90']
      result = [(prediction[i], classes[i]) for i in range(20)]
      result.sort(reverse=1)
      return Response(result)
```

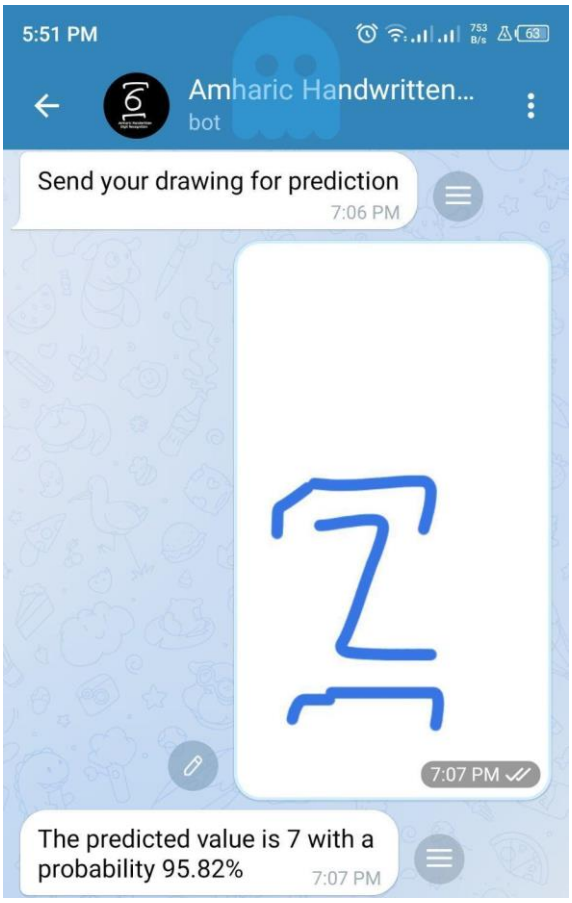
```
@action(methods=["post"], detail=False)
def batch(self, request):
    data = request.POST.getlist("image", [])
    if data is []:
        return Response("image is required")
    images = []
    for im in data:
        image = Image.open(im)
        images.append(image)
    x = process_batch_image(images)
    model = model_construct()
    predictions = model.predict(x)
    classes = [
              '1', '10', '100', '10000', '2', '20', '3', '30',
              '4', '40', '5', '50', '6', '60', '7', '70', '8',
              '80', '9', '90']
    response = {}
    for idx, prediction in enumerate(predictions):
        result = [(prediction[i],classes[i]) for i in range(20)]
        result.sort(reverse=1)
        response[data[idx]] = result[0][1]
    return Response(response)
```

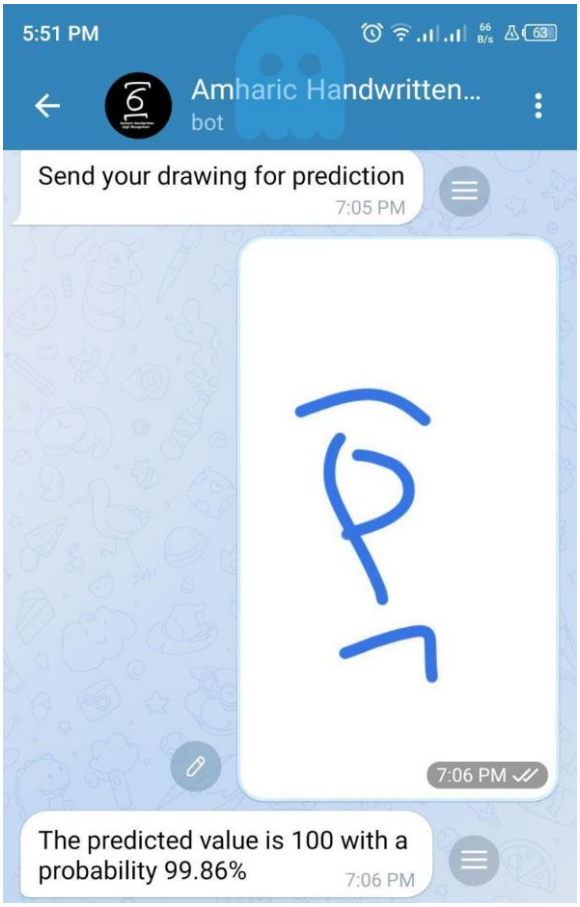
Appendix B: Telegram Amharic Handwritten Digit Recognition Bot Result

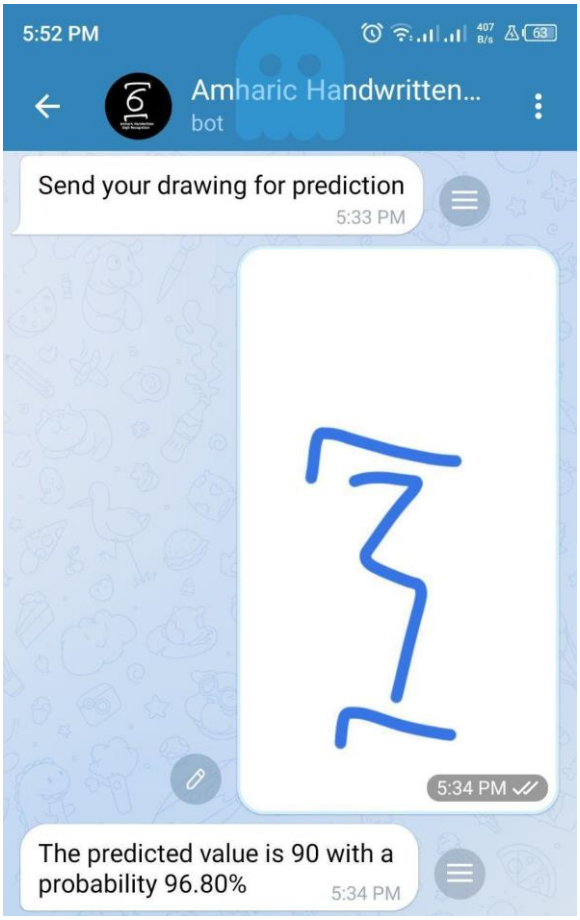


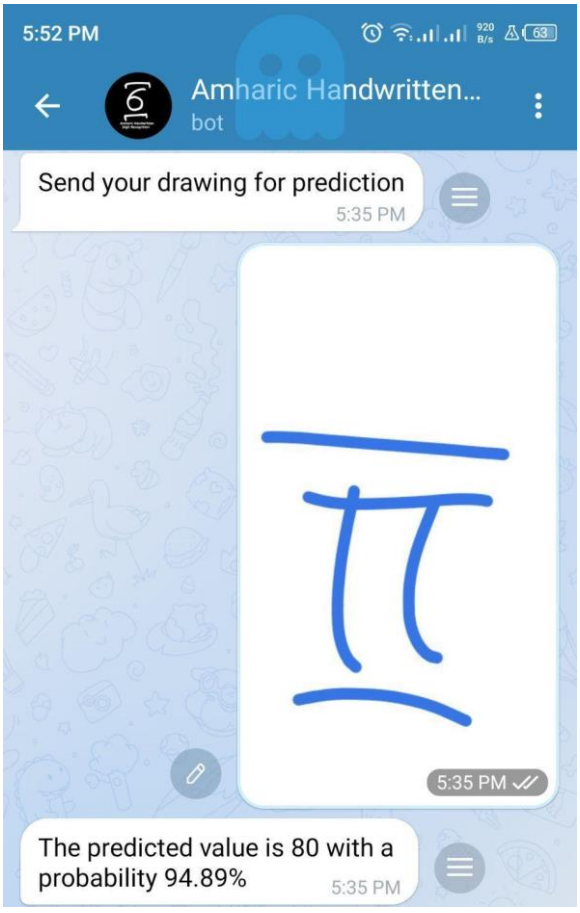


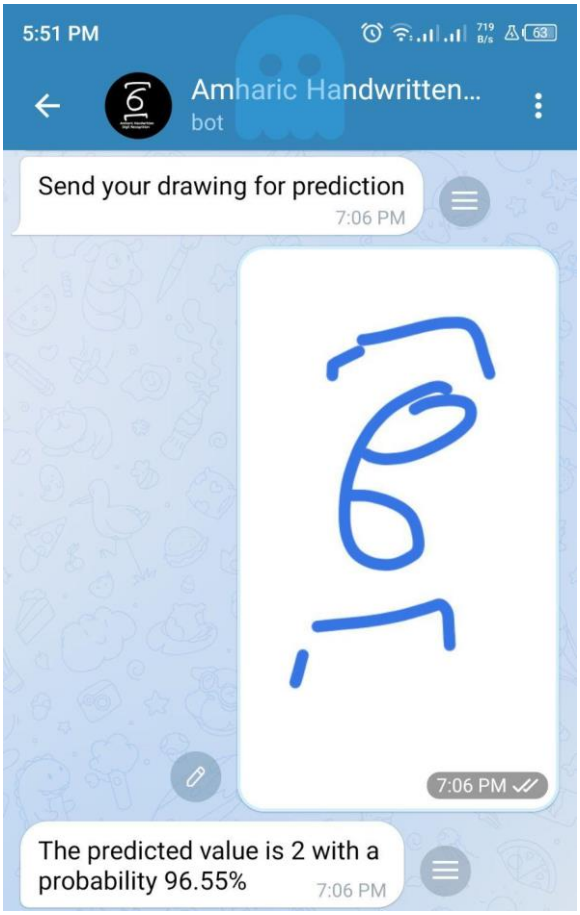

5:51 PM
Amharic Handwritten...
bot
Send your drawing for prediction
7:06 PM
7:06 PM
The predicted value is 6 with a probability 98.99%
7:06 PM


5:51 PM
Amharic Handwritten...
bot
Send your drawing for prediction
7:06 PM
7:07 PM
The predicted value is 7 with a probability 95.82%
7:07 PM


5:51 PM
Amharic Handwritten...
bot
Send your drawing for prediction
7:05 PM
7:06 PM
The predicted value is 100 with a probability 99.86%
7:06 PM


5:52 PM
Amharic Handwritten...
bot
Send your drawing for prediction
5:33 PM
5:34 PM
The predicted value is 90 with a probability 96.80%
5:34 PM


5:52 PM
Amharic Handwritten...
bot
Send your drawing for prediction
5:35 PM
5:35 PM
The predicted value is 80 with a probability 94.89%
5:35 PM


5:51 PM
Amharic Handwritten...
bot
Send your drawing for prediction
7:06 PM
7:06 PM
The predicted value is 2 with a probability 96.55%
7:06 PM