A TRANSFORMER BASED TRANSLITERATION TECHNIQUE FOR MANIPURI LANGUAGE

(Bengali to Meitei Mayek Script)

Thongram Sanajaoba Singh
Enrollment No.: 16103005

Ningthoujam Justwant Singh
Enrollment No.: 16103028

Laishram Somorjit Singh
Enrollment No.: 16103015

A TRANSFORMER BASED TRANSLITERATION TECHNIQUE FOR MANIPURI LANGUAGE

(Bengali to Meitei Mayek Script)

A report submitted to National Institute of Technology Manipur for the award of the degree

of

Bachelor of Technology in Computer Science & Engineering

submitted by

THONGRAM SANAJAOBA SINGH

(16103005)

NINGTHOUJAM JUSTWANT SINGH

(16103028)

LAISHRAM SOMORJIT SINGH

(16103015)

Under the supervision of

Mr. SANABAM BINESHWOR SINGH
Lecturer, NIT Manipur

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY MANIPUR
MAY 2020

DECLARATION

We declare that:

i. The work contained in this report is original and has been done by me under the guidance of my supervisor.

ii. The work has not been submitted to any other institute for any degree or diploma.

iii. I have followed the guidelines provided by the institute in preparing the report.

iv. I have conformed to the norms and guidelines given in the Ethical code of Conduct of the institute.

v. Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the report and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whoever necessary.

Signature of the students

Thongram Sanajaoba Singh
16103005

Ningthoujam Justwant Singh
16103028

Laishram Somorjit Singh
16103015

Department of Computer Science and Engineering
राष्ट्रीय प्रोद्योगिकी संस्थान, मणिपुर
NATIONAL INSTITUTE OF TECHNOLOGY MANIPUR

Langol, Imphal-795004, Phone: 0385-2445812, email: jina.yambem@gmail.com
(An Autonomous Institute under MHRD, Govt Of India)

Ref. No. NITMN.3/(89-Acad)/CSE/B.Tech-Project/2020/08 *Dated: 20/05/2020*

Certificate

*This is to certify that the dissertation Report entitled, "**A transformer based transliteration technique for Manipuri language (Bengali to Meitei Mayek script)**" submitted by "**Thongram Sanajaoba Singh (16103005), Ningthoujam Justwant Singh (16103028), Laishram Somorjit Singh (16103015)**" to National Institute of Technology Manipur, India, is a record of bonafide project work carried out by them under the supervision of **Mr. Sanabam Bineshwor Singh**, Lecturer of NIT Manipur under the department of **Computer Science & Engineering**, NIT Manipur and is worthy of consideration for the award of the degree of Bachelor of Technology in **Computer Science & Engineering Department** of the institute.*

(Dr. Yambem Jina Chanu)
HOD, CSE Department
NIT Manipur
Date:

(Mr. S. Bineshwor Singh)
(Project Supervisor)
Lecturer , CSE Department
NIT Manipur
Date:

ACKNOWLEDGEMENT

It is our proud privilege to have Mr. Sanabam Bineshwor Singh as our supervisor and mentor for this project, whose encouragement, guidance and support from the initial to the final stage enabled us to develop an outstanding of the topic.

We are thankful to Dr. Yambem Jina Chanu, Head of Department, Computer Science and Engineering Department for providing us all the facilities and resources required during the course of the project. We also want to thank our friends and family for their help and support throughout.

Lastly, we offer our regards and gratitude to all of those who supported us in any respect during the completion of this project.

Thongram Sanajaoaba Singh
16103005

Ningthoujam Justwant Singh
16103028

Laishram Somorjit Singh
16103015

ABSTRACT

Manipuri language [1] has its own script, which was used until the 18th century. Its earliest use is not known. Pamheiba, the ruler of the Manipur Kingdom who introduced Hinduism, banned the use of the Meitei script and adopted the Bengali script. Between 1709 and the middle of the 20th century, the Manipuri language was written using the Bengali script. So, most of the literature and historical records during that period were written in the Bengali script. Now in schools and colleges, the Bengali script is gradually being replaced by the Meitei Mayek. And a major transliteration is in need to reproduce those documents in Meitei Mayek. To help the process of transliteration we developed a machine transliteration model which was trained to transliterate Bengali script to Meitei Mayek. Firstly we scanned and convert the copy of the manuscript, textbook, or newspaper to editable text format with the help of Tesseract – OCR. Then the output from the OCR is used as an input for the transliteration model. We achieved an accuracy of 94.54% for the OCR model and 99.6% for the transliteration model.

LIST OF ABBREVIATION

OCR	Optical Character Recognition
Ben	Bengali
Mni	Manipuri
WER	Word Error Rate
CER	Character Error Rate
FFN	Feed-Forward Network
PE	Positional Encoding
GUI	Graphical User Interface
LSTM	Long Short Term Memory
UTF-8	8-bit Unicode Transformation Format
EOF	End-of-File
CSV	Comma-Seperated Values

LIST OF FIGURES

LIST OF TABLES

CONTENTS

CHAPTER 1

INTRODUCTION

For the last few years, OCR [2] for Manipuri language [1] has become very popular. Meitei Mayek is the official script for Manipuri language until the 18th century which was then replaced by Bengali Script. Almost every manuscripts, books written during 18th,19th, and early 20th century were in Bengali Script. In the late 20th century people start reclaiming the Meitei Mayek and started replacing the Bengali script.

So, a process of transliteration [3] is very necessary to translate the Bengali script to Meitei Mayek. Now in schools and colleges, the Bengali script is gradually
being replaced by the Meitei script.

In this project, we developed a technique for OCR for Manipuri Language i.e. Bengali script and transliteration for Manipuri Language i.e. Bengali script to Meitei Mayek along with a user-friendly GUI.

For OCR we used Tesseract-OCR [4] 4.xx, which is an open-source project developed and maintained by Google as a backend engine. The Tesseract 4.xx engine is based on LSTM neural networks. The Bengali script model has been adapted from the existing Bengali model trained with the dataset (words) of Assamese, Bengali by google. Fine-tuning is used to retrain the model to give higher accuracy of the subject language. For transliteration, we used the Transformer [5] to convert the Bengali script to Meitei Mayek.

1.1 OBJECTIVES

To transliterate the old manuscript or books written in Bengali script to Meitei Mayek.

1.2 MOTIVATION

Transliteration is necessary to reproduce our valuable works of literature and historical records which were written in Bengali in Meitei Mayek to make it readable or accessible to the younger generation. There are no proper tools for transliteration, most of the work is done manually. Manual transliteration takes a lot of time, effort, and is prone to error. So, developing a proper machine tool for transliteration will significantly help the transliteration process.

1.3 ORGANISATION OF THE REPORT

1. Chapter 1 provides a brief introduction on Manipuri Language, objectives and motivation of our work.

2. Chapter 2 provides some of the related works related to our work, their methodology and their findings.

3. Chapter 3 gives brief on Manipuri Language and its Scripts

4. Chapter 4 explains the proposed method

5. Chapter 5 provides a brief on OCR and Tesseract

6. Chapter 6 provides a brief on transliteration and Transformer

7. Chapter 7 provides the implementation details of our proposed method.

8. Chapter 8 provides the results and analysis on the outcome of the method.

9. Chapter 9 discuss the conclusion and the future work

CHAPTER 2

LITERATURE REVIEW

1. Md. Mahedi Azad. (02 October 2009) [Optical Character Recognizer for Bangla (Bangla-OCR)] Retrieved 17 May 2020 form [6] . In this project, the Bangla-OCR is implemented using Tesseract maintained by Google, which is considered to be one of the most accurate free open source OCR engines currently available.

2. Ashish Vaswani, Noam Shazeer, Niki Parmar akob, Uszkoreit Llion, Jones Aidan, N. Gomez, Łukasz Kaiser, Illia Polosukhin (2017) [Attention is all we need] Retrieved 17 May 2020 from [7]. Introduced and explained the Transformer.

3. Nongmeikapam K., Singh N.H., Thoudam S., Bandyopadhyay S. (2011) [Manipuri Transliteration from Bengali Script to Meitei Mayek: A Rule-Based Approach]. In: Singh C., Singh Lehal G., Sengupta J., Sharma D.V., Goyal V. (eds) Information Systems for Indian Languages. ICISIL 2011. Communications in Computer and Information Science, vol 139. Springer, Berlin, Heidelberg Retrieved 17 May 2020 from [8]. In this project the transliteration from Manipur language i.e. Bengali script to Meitei Mayek.

4. Neelakash Kshetrimayum. (2011) [A comparative study of Meetei Mayek] Retrieved 17 May 2020 from [9]. In this paper, the Meetei Mayek which is the official of Manipur language is thoroughly explained.

CHAPTER 3

MANIPURI LANGUAGE AND ITS SCRIPTS

3.1 MEITEI MAYEK

Some properties of Meitei Mayek script are given below:

1. Meitei Mayek script flows from left to right.

2. Characters are not classified as uppercase or lowercase.

3. Meitei Mayek letters do not connect with each other when written.

3.1.1 Meitei Mayek Alphabet

In Meitei Mayek, there is a total of 27 alphabets. Among this 27 alphabets 18 are original letterforms called Eeyek Eepee and 9 letterforms which were added later also called Lom Eeyek.

The original 18 letterforms are as follow:

ꯀ	ꯁ	ꯂ
ꯃ	ꯄ	ꯅ
ꯆ	ꯇ	ꯈ
ꯉ	ꯊ	ꯋ
ꯌ	ꯍ	ꯎ
ꯏ	ꯐ	ꯑ

Table3.1.1(a) shows the original 18 letterforms of Meitei Mayek script.

The 9 letterforms which were added later are as follow:

ꯒ	ꯓ	ꯔ
ꯕ	ꯖ	ꯗ
ꯘ	ꯙ	ꯚ

Table 3.1.1(b) shows the 9 letterforms which were added later.

3.1.2 Semi Consonant

There is a total of 8 semi consonants or half sound called Lonsum Eeyek. It is derived from some alphabets.

8 semi consonants along with their corresponding alphabets:

ꯀ	ꯛ	ꯂ	ꯜ
ꯃ	ꯝ	ꯅ	ꯟ
ꯇ	ꯠ	ꯉ	ꯡ
ꯏ	ꯢ	ꯄ	ꯞ

Table 3.1.2 shows the Lonsum Eeyek along with their corresponding alphabets.

3.1.3 Vowel signs

There are 8 vowel signs(Cheitap Eeyek) which are use to form word shapes.

Vowel signs:

◌ꯦ	◌ꯥ	◌ꯣ	◌ꯩ	◌ꯧ	◌ꯨ	◌ꯤ	ꯪ

Table3.1.3(a) shows the vowel signs(Cheitap Eeyek).

Letter along with the diacritic form of vowel signs as follows:

	◌ꯣ	◌ꯤ	◌ꯥ	◌ꯦ	◌ꯧ	◌ꯨ	◌ꯩ	ꯑ
ꯀ	ꯀꯣ	ꯀꯤ	ꯀꯥ	ꯀꯦ	ꯀꯧ	ꯀꯨ	ꯀꯩ	ꯀꯑ
ꯁ	ꯁꯣ	ꯁꯤ	ꯁꯥ	ꯁꯦ	ꯁꯧ	ꯁꯨ	ꯁꯩ	ꯁꯑ
ꯂ	ꯂꯣ	ꯂꯤ	ꯂꯥ	ꯂꯦ	ꯂꯧ	ꯂꯨ	ꯂꯩ	ꯂꯑ
ꯃ	ꯃꯣ	ꯃꯤ	ꯃꯥ	ꯃꯦ	ꯃꯧ	ꯃꯨ	ꯃꯩ	ꯃꯑ
ꯄ	ꯄꯣ	ꯄꯤ	ꯄꯥ	ꯄꯦ	ꯄꯧ	ꯄꯨ	ꯄꯩ	ꯄꯑ
ꯅ	ꯅꯣ	ꯅꯤ	ꯅꯥ	ꯅꯦ	ꯅꯧ	ꯅꯨ	ꯅꯩ	ꯅꯑ

[...]

ꯕ	ꯕꯣ	ꯕꯤ	ꯕꯥ	ꯕꯦ	ꯕꯧ	ꯕꯨ	ꯕꯩ	ꯕꯑ
ꯖ	ꯖꯣ	ꯖꯤ	ꯖꯥ	ꯖꯦ	ꯖꯧ	ꯖꯨ	ꯖꯩ	ꯖꯑ
ꯗ	ꯗꯣ	ꯗꯤ	ꯗꯥ	ꯗꯦ	ꯗꯧ	ꯗꯨ	ꯗꯩ	ꯗꯑ
ꯘ	ꯘꯣ	ꯘꯤ	ꯘꯥ	ꯘꯦ	ꯘꯧ	ꯘꯨ	ꯘꯩ	ꯘꯑ
ꯙ	ꯙꯣ	ꯙꯤ	ꯙꯥ	ꯙꯦ	ꯙꯧ	ꯙꯨ	ꯙꯩ	ꯙꯑ
ꯚ	ꯚꯣ	ꯚꯤ	ꯚꯥ	ꯚꯦ	ꯚꯧ	ꯚꯨ	ꯚꯩ	ꯚꯑ

Table3.1.3(b) shows the letters along with diacritic form of vowel signs.

3.1.4 Numerals

Numerals(Cheising Eeyek):

꯱	꯲	꯳	꯴	꯵
1	2	3	4	5
꯶	꯷	꯸	꯹	꯰
6	7	8	9	0

Table 3.1.4 shows the numerals of Meitei Mayek script.

3.1.5 Punctuation marks

Khudam Eeyek:

—	|	||	≍
“joining” underline	comma	fullstop/period	question mark

Table3.1.5 shows the punctuation marks of Meitei Mayek script.

3.2 BENGALI SCRIPT

Some common properties in Bengali language are given below :

1. Bengali script flows from left to right.

2. Characters are not classified as uppercase or lowercase.

3. A vowel with a consonant takes a modified shape known as vowel modifier.

4. A consonant with a consonant takes a modified shape known as consonant modifier.

5. The vowel (অ) always occurs at the beginning of the word.

6. There are about 250 compound characters.

7. Many characters have "matra" or headlines with them. Some characters have a signature extended above the head line (e.g. ই, ঈ, উ).

8. In a standard text piece, 95.63% percent of the characters are basic characters and the rest 4.27 percent are compound characters.

3.2.1 Vowels

In Bengali Alphabet, there is a total of 11 vowel symbols, whereas in the Bengali – Manipuri alphabet there are 14 vowel symbols in total.

Manipuri Vowels:

অ	আ	ই	ঈ
উ	ঊ	ঋ	ঌ
এ	ঐ	ও	ঔ
	অং	অঃ	

Table 3.2.1(a) shows Manipuri vowels.

Bengali Vowels:

অ	আ	ই	ঈ
উ	ঊ	ঋ	
এ	ঐ	ও	ঔ

Table 3.2.1(b) shows Bengali vowels.

3.2.2 Consonants

In Bengali Alphabet, there is a total of 35 consonant symbols, whereas in the Bengali Manipuri alphabet there are 41 consonant symbols in total.

Bengali consonants used in written Manipuri are as follow:

ক	খ	গ	ঘ	ঙ
চ	ছ	জ	ঝ	ঞ
ট	ঠ	ড	ঢ	ণ
ত	থ	দ	ধ	ন
প	ফ	ব	ভ	ম
য	র	ল	ৰ	শ
ঐ	স	হ	ক্ষ	ড়
ঢ়	য়	ং	ঃ	ঁ
		ৎ		

Table 3.2.2(a) shows consonant used in written Manipuri.

Bengali script Consonants are as follows:

ক	খ	গ	ঘ	ঙ
চ	ছ	জ	ঝ	ঞ
ট	ঠ	ড	ঢ	ণ
ত	থ	দ	ধ	ন
প	ফ	ব	ভ	ম
য	র	ল	শ	ষ
স	হ	ড়	ঢ়	য়

Table 3.2.2(b) shows Bengali script Consonants.

Some consonants along with the **diacritic** form of the vowels অ, আ, ই, ঈ, উ, ঊ, ঋ, এ, ঐ, ও and ঔ:

	ô	a	i	ī	u	ū	ṛ	e	oi	o	ou
kô	ক	কা	কি	কী	কু	কূ	কৃ	কে	কৈ	কো	কৌ
khô	খ	খা	খি	খী	খু	খূ	খৃ	খে	খৈ	খো	খৌ
gô	গ	গা	গি	গী	গু	গূ	গৃ	গে	গৈ	গো	গৌ
ghô	ঘ	ঘা	ঘি	ঘী	ঘু	ঘূ	ঘৃ	ঘে	ঘৈ	ঘো	ঘৌ
ṅô	ঙ	ঙা	ঙি	ঙী	ঙু	ঙূ	ঙৃ	ঙে	ঙৈ	ঙো	ঙৌ

Table3.2.2(c) shows the Bengali consonants along with the diacritic

3.2.3 Conjuncts

Bengali is full of conjunct consonants. Here are some of them:

ক্ক kkô	ক্ট kṭô	ক্ত ktô	ক্ব kbô	ক্ম kmô	ক্র krô	ক্ল klô	ক্ষ kṣô	ক্ষ্ম kṣmô
ক্স ksô	গ্ধ gdhô	গ্ন gnô	গ্ব gbô	গ্ম gmô	গ্ল glô	ঘ্ন ghnô	ঙ্ক ṅkô	ঙ্ক্ষ ṅkṣô
ঙ্থ ṅthô	ঙ্গ ṅgô	ঙ্ঘ ṅghô	ঙ্ম ṅmô	চ্ছ cchô	চ্ছ্ব cchbô	চ্ঞ cñô	জ্জ jjô	জ্জ্ব jjbô
জ্ঝ jjhô	জ্ঞ jñô	জ্ব jbô	ঞ্চ ñcô	ঞ্ছ ñchô	ঞ্ঝ ñjhô	ট্ট ṭṭô	ট্ব ṭbô	ণ্ট ṇṭô
ণ্ঠ ṇṭhô	ণ্ড ṇḍô	ণ্ণ ṇṇô	ণ্ম ṇmô	ত্ত ttô	ত্ত্ব ttbô	ত্থ tthô	ত্ন tnô	ত্ব tbô
ত্ম tmô	ত্র trô	দ্দ ddô	দ্ধ ddhô	দ্ব dbô	দ্ভ্র dbhrô	ন্ট nṭô	ন্ড nḍô	ন্ত ntô
ন্ত্ব ntbô	ন্ত্র ntrô	ন্দ ndô	ন্ধ ndhô	ন্ন nnô	ন্ব nbô	ন্স nsô	প্ট pṭô	প্ত ptô
প্ন pnô	প্প ppô	প্ল plô	প্স psô	ফ্ল phlô	ভ্র bhrô	ভ্ল bhlô	ম্ন mnô	ম্ফ mphô
ম্ব mbô	ম্ল mlô	ল্ট lṭô	ল্ড lḍô	ল্ব lbô	ল্ল llô	শ্চ shchô	ষ্ক ṣkô	ষ্ট ṣṭô
ষ্ণ ṣṇô	স্ক্র skrô	স্ত stô	স্ত্র strô	স্ব sbô	হ্ন hnô	হ্ম hmô	হ্ব hbô	হ্ল hlô

Table3.2.3(a) shows conjunct consonants of Bengali.

Some of the Manipuri Conjunts are as follows:

ক্য	ক্র	ক্ল	ক্ব	হ্ন	ক্ম	র্ক	ক্ক	ক্ট্র
খ্য	খ্র	প্ল	খ্‌ব	ষ্ণ	ঙ্ম	র্খ	ক্ত	ঙ্‌খ্র
জ্য	চ্র	ম্ল	গ্ব	ক্ন	ত্ম	র্গ	ক্ষ	ঙ্‌গ্র
ন্য	ত্র	ল্ল	দ্ব	ত্ন	ন্ম	র্জ	ক্স	ন্ত্র
প্য	থ্র	হ্ল	শ্ব	ন্ন	ম্ম	র্ত	ঙ্ক	ন্‌থ্র
ব্য	প্র	চ্চ	স্ব	প্ন	ল্ম	র্থ	ঙ্খ	প্‌ত্র
শ্য	ভ্র	ঞ্চ	ট্ট	ম্ন	স্ম	র্প	ঙ্গ	ম্‌ব্র
হ্য	হ্র	জ্ঞ	ণ্ঠ	হ্ন	শ্ম	র্ফ	ঙ্‌ঙ	ন্ধ্য
ঙ্য	ঙ্র	ঞ্জ	ণ্ড	স্ন	ষ্ম	র্ভ	ঙ্ক	ক্ষ্ম
ত্য	জ্র	চ্ছ	ন্দ	শ্ন	হ্ম	র্ম	ত্ত	ন্দ্র
ব্র	গ্র	ঞ্ঝ	প্ত	ল্ট	ন্ত	র্ত্য	ত্থ	স্প্র
স্ফ	দ্র	ষ্ঠ	প্প	শ্ম	ন্থ	র্দ্ধ্য	দ্দ	ন্‌
স্ব	ফ্র	স্থ	প্স	স্প				

3.2.4 Numerals

০	১	২	৩	৪
0	1	2	3	4
৫	৬	৭	৮	৯
5	6	7	8	9

Table 3.2.4 shows numerals of Bengali script.

CHAPTER 4

PROPOSED METHOD

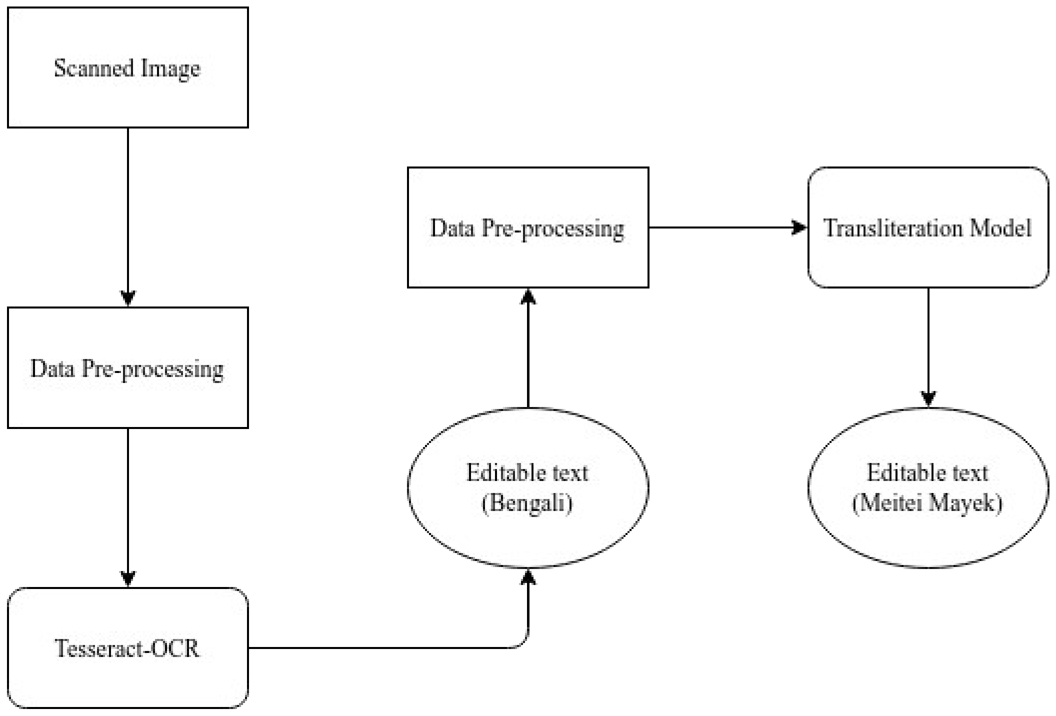


Fig. 4.1 Block diagram of the complete proposed model

4.1 WORK FLOW OF THE PROPOSED METHOD

Our proposed method can be grouped into two general phases as

1. OCR phase
2. Transliteration phase

4.1.1 OCR Phase

Steps involved in this phase are as follow:

1. **Scanning the inputs** – The pages to be transliterate are scanned and convert into images.

2. **Pre-Processing the scanned image** – Our OCR model gives higher accuracy if the image dpi is set at 300. So, we need to set the image dpi to 300. Then, combine all the scanned images (of same set) into a single tiff file.

3. **Performing OCR** – The text paragraphs on the image are extracted in this step with the help of the trained Tesseract-OCR. And we will get an editable ‘txt’ file containing all the text from the image.

4.1.2 Transliteration Phase

Steps involved in this phase are as follow:

1. **Data Pre-Processing** – The output from the OCR phase will act as an input for the Transliteration phase. The input should be in ‘csv’ format and with the shape of Nx1, where N is the number of total bengali words to be transliterated. So conversion should be perform accordingly.

2. **Performing Transliteration** – Now transliteration from Bengali to Meitei Mayek will be performed on the processed input.

After completion of all the steps mentioned in 4.1.1 and 4.1.2, we will get a machine transliterated editable text from Bengali to Meitei Mayek.

CHAPTER 5

OCR AND TESSERACT

5.1 OCR

An OCR [2] is scanning of printed or handwritten document which will be converted to machine encoded text i.e. first we will scaned the manuscript or printed textbook which where written in Bengali script and later we can translitrate the scaned words to Meitei Mayek.

5.2 TESSERACT

Tesseract [4] which was originally developed by Hewlett-Packard at Hewlett-Packard Laboratories Bristol and at Hewlett-Packard Co, Greeley Colorado between 1985 and 1994, with some more changes made in 1996 to port to Windows, and some C++izing in 1998. In 2005 Tesseract was open sourced by HP and the University of Nevada, Las Vegas. Since 2006 it is developed by Google. The latest (LSTM based) stable version is 4.0.0, released on October 29, 2018.

5.2.1 Why do we choose Tesseract Engine?

Tesseract is an OCR engine with support for unicode (UTF-8) and the ability to recognize more than 100 languages out of the box. It can be trained to recognize other languages.
Tesseract supports various output formats: plain text, hOCR (HTML), PDF, invisible-text-only PDF, TSV. The master branch also has experimental support for ALTO (XML) output.
And this OCR engine has a higher accuracy than its counterparts.

OCR	Accuracy for Ordinary Document(e.g Insurance)	Accuracy for Identity Document	Missing character detection	Similar/Special character replacement	Trainable	Free Tier
Microsoft OCR	93%	86%	85%	15%	No	5000 page/month
Google Vision	96%	84%	53%	47%	No	1000 page/month
ABBYY Finereader	91%	54%	62%	36%	No	Not available
Nuance Omnipage	92%	53%	53%	48%	No	Not available
Tesseract	84%	54%	72%	28%	Yes	Open Source

Table 5.2.1(a) shows the comparison of OCR(s)

Tesseract Version	Year	Develpoment
	1984	PhD project sponsered by HP
	1987	Second person assigned
	1988	Joint HPLaba/Scanner Division Project
	1990	Scanner product cancelled
	1994	HPLaba project cancelled
	1995	UNLV Evaluation proted to Windows
	2005	Open Sourced by HP
1.01-4	2006	Google takes it on
2.01-2	2007	Expanded to 6 European Langs
2.04 – 2.03	2008	Expanded to 6 European Langs
3.00	2010	Layout analysis + 60 langs
3.01	2011	Word blgrams, multi-lang, equation detection
3.02	2012	Word blgrams, multi-lang, equation detection
3.03	2014	Training Tools PDF
3.04	2015	100 Langs
4.00?	2016	LSTM

Table 5.2.1(b) shows the timeline of Tesseract

CHAPTER 6

TRANSLITERATION

Transliteration is the process of converting a word from the alphabet of one language to another language.

Some models or architecture used for transliteration:

1. Recurrent Neural Network.

2. Convulational Neural Network.

3. Sequence to Sequence.

4. Transformer.

In this project we are going to use Transformer model.

6.1 TRANSFORMER AND WHY?

Transformer is a type of neural network architecture which is use to transform one sequence into another sequence by using encoder and decoder. In this architecture all the recurrent layers present in sequence transduction model based on recurrent networks are replaced by the multi-head attention layers. Some of the advantages of Transformer over other architectures are as follows.

1. In Transformer model, all the inputs can be passed simultaneously.

2. When we train data, Transformer model takes lesser time than other models.

3. Positional Encoding **:** Vector embedding about the relative or absolute position of the token in the sequence.

Attention reduces sequential operations and maximum path length, which facilitates long range dependencies.

Layer Type	Complexity per layer	Sequential Operations	Maximum Path Lengths
Self-Attention	O$(n^2 - d)$	O(1)	O(1)
Recurrent	O$(n - d^2)$	O(n)	O(n)
Convolutional	O$(k\text{-}n\text{-}d^2)$	O(1)	O$(logk(n))$
Self-Attention (restricted)	O$(r\text{-}n\text{-}d)$	O(1)	O(n/r)

Table 6.1(a) comparison of different complexities between transduction architecture

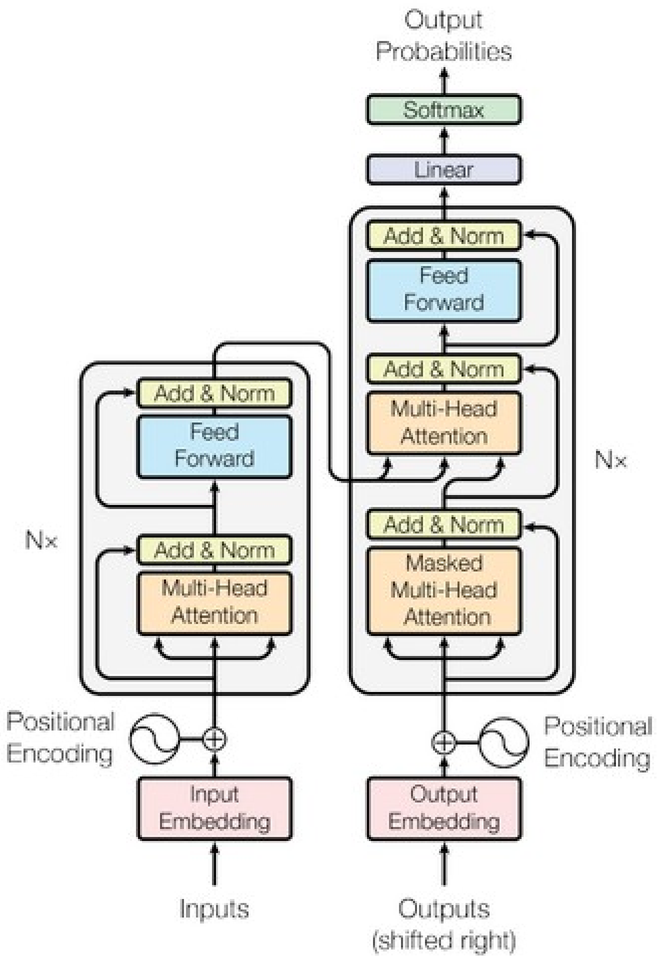


Fig. 6.1(a) shows the architecture of the Transformer

6.1.1 Encoder

The Encoder consists of a stack of N = 6 layers. The layers are all identical in structure and each layers consist of two sub layers. The first layer is multi-head self attention layer and the second layer is feed-forward layer.

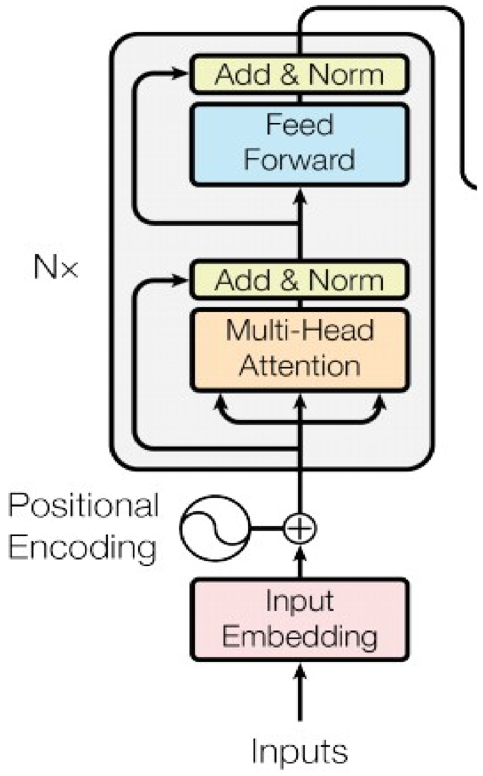
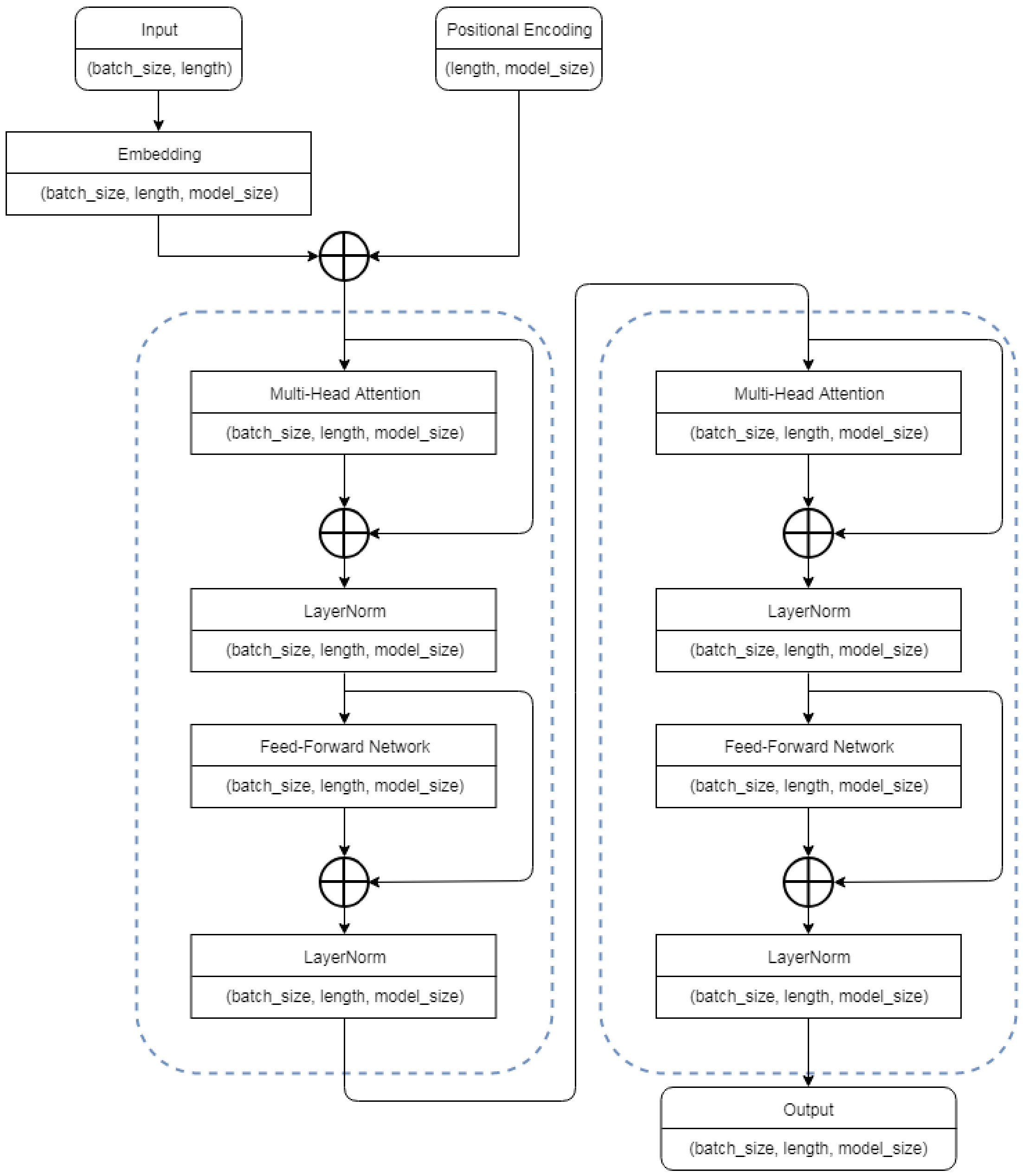


Fig. 6.1.1(a) shows the details of an encoder in Transformer


Input
(batch_size, length)
Positional Encoding
(length, model_size)
Embedding
(batch_size, length, model_size)
Multi-Head Attention
(batch_size, length, model_size)
LayerNorm
(batch_size, length, model_size)
Feed-Forward Network
(batch_size, length, model_size)
LayerNorm
(batch_size, length, model_size)
Multi-Head Attention
(batch_size, length, model_size)
LayerNorm
(batch_size, length, model_size)
Feed-Forward Network
(batch_size, length, model_size)
LayerNorm
(batch_size, length, model_size)
Output
(batch_size, length, model_size)
Fig. 6.1.1(b) Data shape inside the Encoder.

6.1.2 Decoder

The Decoder also consists of a stack of N = 6 layers and the layers are all identical in structure. The Decoder also have multi-head self attention layer and feed-forward network but it has another attention layer that performs multi-head attention with the output of the encoder.

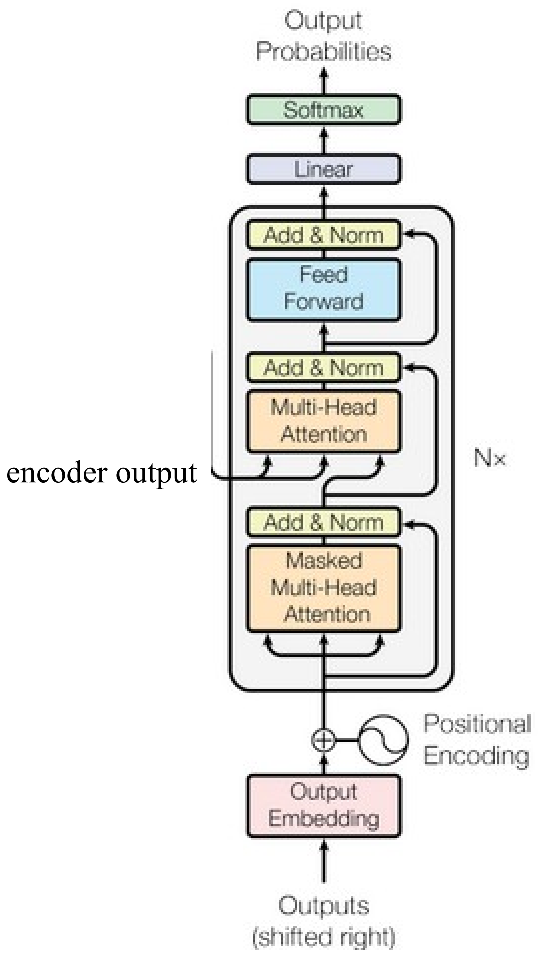


Fig. 6.1.2(a) shows the details of a decoder in Transformer.

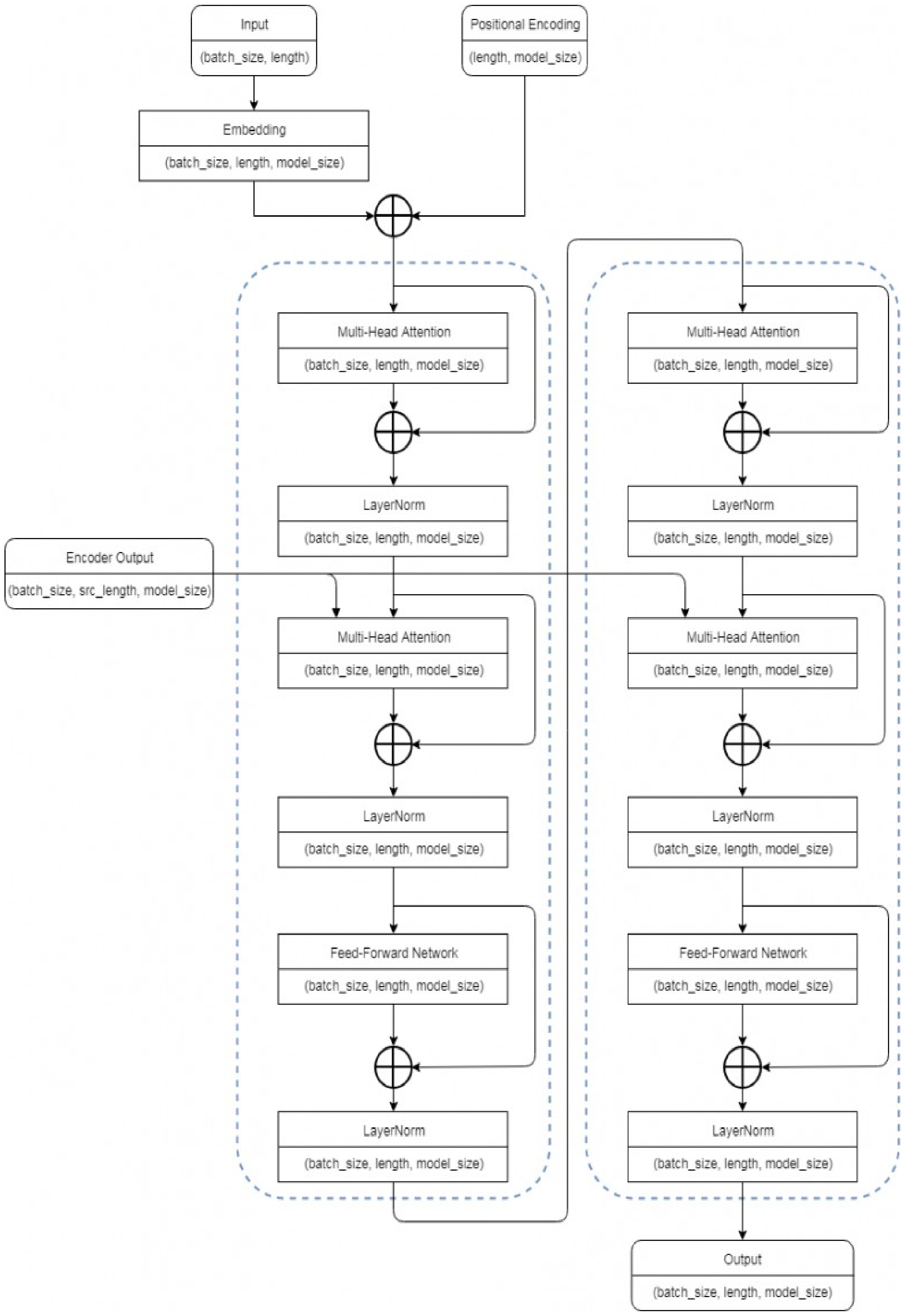

Input
(batch_size, length)
Positional Encoding
(length, model_size)
Embedding
(batch_size, length, model_size)
Multi-Head Attention
(batch_size, length, model_size)
LayerNorm
(batch_size, length, model_size)
Encoder Output
(batch_size, src_length, model_size)
Multi-Head Attention
(batch_size, length, model_size)
LayerNorm
(batch_size, length, model_size)
Feed-Forward Network
(batch_size, length, model_size)
LayerNorm
(batch_size, length, model_size)
Output
(batch_size, length, model_size)

Fig. 6.1.2(b) Data shape inside the Decoder.

6.1.3 Working of Encoder & Decoder

In Encoder section, first the inputs of the encoder fed to the input embedding and perform positional encoding. And this output is flow through the multi-head self attention layer and perform normalization. The outputs from this layer are again fed to a feed-forward network and perform normalization again.

In Decoder section, we feed the inputs in the decoder in the form of vector and perform positional encoding. Then we pass these vectors to the main decoder block that has three main components, two are similar to the encoder block. The other attention layer performs multi-head attention with the outputs from the encoder section. And the attention vectors pass to the feed-forward layer to make the output vector more digestible by next decoder block or linear layer. The linear layer is another feed-forward layer. The outputs from the linear layer is go through softmax layer to produce the probability distribution and we get the actual output.

Input / Output Embedding

In this layer the input tokens are mapped into a vector of dimension d_{model}.

Positional Encoding

It is used to get the information between the words and the sentences. Position Encoder is a vector that gives context based on the position of word in sentence.

We use sine and cosine function to generate this vector.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d}{}_{model})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d}{}_{model})$$

Attention block

It computes the attention vector of each outputs from the position encoding section.

Feed Forward Network

In each attention sub-layers of encoder and decoder contain a fully connected feed-forward network, which is applied to each position separately and identically. The Feed-Forward Network are used to transform the attention vectors into a form which is digestible by the next encoder block or decoder block.

$$FFN(x) = max(0,xW_1 + b_1)W_2 + b_2.$$

Normalization Layer

It normalizes the values in each layer to have 0 mean and 1 variance.

6.1.4 Attention

Attention is the mapping of a query and a set of key-value pairs to an output, where the query, key-value and output are all vectors. There are two types of attention:

1. Scaled Dot-Product Attention.

2. Multi-Head Attention.

Working of Attention

In Decoder section, the input

Scaled Dot-Product Attention

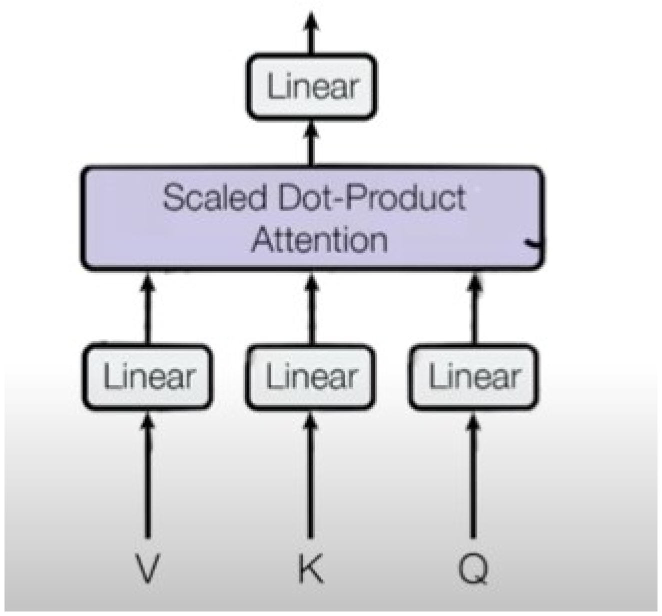


Fig. 6.1.4(a) Scaled Dot-Product Attention.

Scaled Dot-Product Attention consists of queries and keys of dimension d_k and values of dimension d_v. We compute the dot products of the query with all keys, divide each by$\sqrt{dk}$, and apply a softmax function to obtain the weights on the values.

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V . We compute the matrix of outputs as:

$$Attention(Q,K,V) = softmax\left(\frac{Q.K^T}{\sqrt{dk}}\right)V$$

Multi-Head Attention

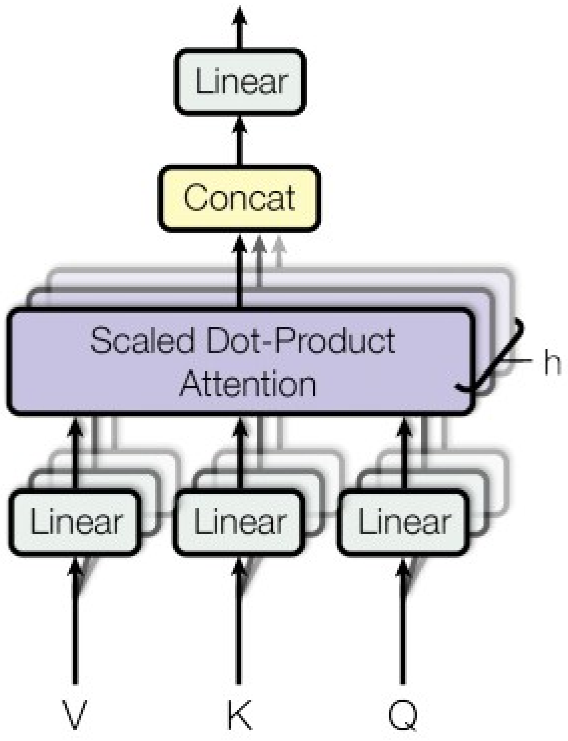


Fig. 6.1.4(b) Multi-Head Attention.

The Multi-Head Attention computes multiple attentions per query with different weights.

$$MultiHead(Q,K,V) = Concat(head_1,...,head_h)W^0$$

$$\text{where } head_i = Attention(QW^Q{}_i, KW^K{}_i, V W^V{}_i)$$

Masked Multi-Head Attention

Masked Multi-Head Attention is multi-head where some values are masked (i.e., probabilities of masked values are nullified to prevent from being selected). When decoding, an output value should depend only on previous outputs (not future outputs). Hence we mask future outputs.

$$\textit{Masked Multi-Head Attention}(Q,K,V) = \mathrm{softmax}\left(\frac{Q.K^T + M}{\sqrt{dk}}\right)V$$

Where M is a mask matrix of 0's and -∞'s

CHAPTER 7

IMPLEMENTATION

7.1 SYSTEM REQUIREMENTS

7.1.1 Operating system and Environment

1. Ubuntu 18.04.4 LTS
2. Python 3.7.3
3. System memory: 8 GB

7.1.2 Tools

1. Tesseract 4.1.0-rc4 [4]
2. GPL Ghostscript 9.26 (2018-11-20)
3. OCR-evaluation-Tools by UNLV [10]

7.1.3 Dependencies

1. A compiler for C and C++: GCC or Clan
2. GNU Autotools: autoconf, automake, libtool
3. pkg-config
4. Leptonica
5. libpng, libjpeg, libtif
6. Tensorflow 2.1.0
7. numpy 1.16.2
8. nltk 3.4
9. scikit-learn 0.20.3
10. tqdm 4.31.1
11. Tkinter

7.2 FINE TUNING THE OCR

There is already a trained model of Tesseract-OCR for Bangla language. The model can be found at [11] as ben.traineddata (following the ISO 639-2 code). We perform OCR on Manipuri text using the existing ben.traineddata. To our surprise we get a very low accuracy from the model even though the script are same.

Fine tuning [12] is a method provided by Tesseract to retrain an existing model with additional data. So, we consider to fine tune the model by adding Manipuri text written in Bengali script.

7.2.1 Data Pre-Processing and Feature Extraction

Since we are going to fine tune the tesseract-engine with lstm it will require a large number of words and trained model of Bengali (ben.traineddata) to recognise them correctly. So we collect 63404 typed Manipuri words in bengali script from a variety of sources including online newspaper [13], journal, and articles. The collection is named as **ben.training_text**.

The ben.training_text is then refined and checked manually to ensure correctness of spelling and to remove any typing mistakes. And we collect the ben.traineddata and eng.traineddata from tessdata_best available at tesseract main repository to used as primary and secondary language for the fine tuning respectively.

And from the langdata_lstm_master (data used to trained the ben.traineddata) we collect the ben.config, radical-stroke.txt files which will be used to create the training data.

Overview of feature extraction process

1. Prepare training text.

2. Render text to image + box file. (Or create hand-made box files for existing image data.)

3. Make unicharset file. (Can be partially specified, i.e., created manually).

4. Make a starter traineddata from the unicharset and optional dictionary data.

5. Run tesseract to process image + box file to make training data set.

The steps mentioned above can be done manually and individually which will take a lot of time and prone to error. Luckily tesseract 4.xx provides a tool, tesstrain.sh which execute all the steps automatically.

We used the font “Bangla Medium” and “Lohit Bengali” to create the training dataset and starter traineddata.

Uses of tesstrain.sh

```
~/tesseract-4.1.0-rc4/src/training/tesstrain.sh \
--fonts_dir /usr/share/fonts --lang ben \
--linedata_only --save_box_tiff –langdata_dir \
~/langdata_lstm_master --fontlist "Bangla Medium" "Lohit Bengali" \
--tessdata_dir ~/tesseract-4.1.0-rc4/tessdata --output_dir ~/tesstutorial/BTP/mnp2
```

We run tesstrain.sh on the ben.training_text, using the font, Bangla Medium and Lohit Bengali, then generate the following files:

1. ben/ben.traineddata

2. ben/ben.charset_size=127.txt

3. ben/ben.unicharset

4. ben.Bangla_Medium.exp0.box, ben.Lohit_Bengali.exp0.box

5. ben.Bangla_Medium.exp0.tif, ben.Lohit_Bengali.exp0.tif

6. ben.Bangla_Medium.exp0.lstmf, ben.Lohit_Bengali.exp0.lstmf

7. ben.training_files.txt

7.2.2 Tools and Files use to make training data

As with base Tesseract, the completed LSTM model and everything else it needs is collected in the traineddata file. Unlike base Tesseract, a starter traineddata (*ben/ben.traineddata*) file is given during training, and has to be setup in advance. It can contain:

1. Config file providing control parameters.

2. Unicharset defining the character set.

3. Unicharcompress, aka the recoder, which maps the unicharset further to the codes actually used by the neural network recognizer.

4. Punctuation pattern dawg, with patterns of punctuation allowed around words.

5. Word dawg. The system word-list language model.

6. Number dawg, with patterns of numbers that are allowed.

A new tool: *combine_lang_model* is provided to make a starter traineddata from a unicharset and optional wordlists.

The training data is provided via .lstmf files, which are serialized Document data. They contain an image and the corresponding UTF8 text transcription, and can be generated from tif/box file pairs using Tesseract in a similar manner to the way .tr files were created for the old engine.

Unicharset - Tesseract's unicharset file contains information on each symbol (unichar) the Tesseract OCR engine is trained to recognize. The first line of a unicharset file contains the number of unichars in the file. After this line, each subsequent line provides information for a single unichar. The first such line contains a placeholder reserved for the space character. Each unichar is referred to within Tesseract by its Unichar ID, which is the line number (minus 1) within the unicharset file. Therefore, space gets unichar 0. It can be generated using *unicharset_extractor* tool provided by Tesseract.

lang.font.exp0.tif – image file converted from ben.training_text. Can be generated using *text2image* tool provided by Tesseract.

lang.font.exp0.box - Each line in the box file matches a ‘character’ (glyph) in the tiff image.

<symbol> <left> <bottom> <right> <top> <page>

Where *<left> <bottom> <right> <top> <page>* could be bounding-box coordinates of a single glyph or of a whole textline.

To mark an end-of-textline, a special line must be inserted after a series of lines.

<tab> <left> <bottom> <right> <top> <page>

Box files can be generated using Tesseract 4.0 using the lstmbox config from image data using

```
tesseract <image name including extension> <box file name> lstmbox
```

7.2.3 Fine tuning on the training data

In tesseract-ocr 4.xx we can fine tune a existing model using:

```
[i]     training/combine_tessdata -e tessdata/lang.traineddata \
        ~/lang.lstm
```

```
[ii]    training/lstmtraining --model_output /path/to/output [--max_image_MB
        6000]\
        --continue_from /path/to/existing/model \
        --traineddata /path/to/traineddata/with/new/unicharset \
        --old_traineddata /path/to/existing/traineddata \
        [--perfect_sample_delay 0] [--debug_interval 0] \
        [--max_iterations 0] [--target_error_rate 0.01] \
        --train_listfile /path/to/list/of/filenames.txt
```

[i] and [ii] will give the output as model_name.checkpoint.

The flag with its default values:

• --*debug_interval* 0, the trainer outputs a progress report every 100 iterations.

• --*perfect_sample_delay* 0, discards perfect samples if there haven't been that many imperfect ones seen since the last perfect sample. The current default value of zero uses all samples. In practice the value doesn't seem to have a huge effect, and if training is allowed to run long enough, zero produces the best results.

• --*target_error_rate* 0.01, Stop training if the mean percent error rate gets below this value.

• --*max_iterations* 0, Stop training after this many iterations.

Note: All the flags are set to its default value for the whole training.

7.2.4 Combining the output

The lstmtraining program outputs two kinds of checkpoint files:

- *<model_base>_checkpoint* is the latest model file.

- *<model_base><char_error>_<iteration>.checkpoint* is periodically written as the model with the best training error.

Either of these files can be converted to a standard traineddata file as follows:

```
training/lstmtraining --stop_training \
--continue_from ~/tesstutorial/eng_from_chi/base_checkpoint \
--traineddata ~/tesstutorial/engtrain/eng/eng.traineddata \
--model_output ~/tesstutorial/eng_from_chi/eng.traineddata
```

This will extract the recognition model from the training dump, and insert it into the --traineddata argument, along with the unicharset, recoder, and any dawgs that were provided during training.

After converting the outputs into mni.traineddata (according to **ISO** 639-2) the details are as below:

Sl.no	Ben.training_text	Old traineddata	New traineddata	Error	No of Iteration
1	ben.training_text	tessdata_best/ ben.traineddata	mni.traineddata	0.01	125700

Table 7.2.4 shows the training error rate and other training details

7.3 TRAINING THE TRANSLITERATION MODEL

As mentioned earlier we use the Transformer for our transliteration model as it has multiple advantages over other transliteration models. We used the exact same architecture as described at [5]. The model has the encoder-decoder architectures using stacked self-attention and point-wise and fully connected layers for both the encoder and decoder. The details of the same can be found at Chapter 6.

7.3.1 Data Collection and Pre-Processing

To train the model we collect a total of 21306 unique Manipuri words written in Bengali script alongside its translated words written in Meitei Mayek. We store the data in a csv file (*ben-man.csv*) of shape Nx2, where N is the number of unique words in the collection. The collection has noise or unwanted rows containing scripts other than Bengali and Meitei Mayek. So we remove the noise and split the collection into three as follows:

1. *ben-man-train.csv* – The main training data containing unique 19805 Bengali words along with its label (in Meitei Mayek).

2. *ben-man-val.csv* – CSV file containing unique 498 Manipuri words written in Bengali script along with its label (in Meitei Mayek) which will be used for evaluation of the model.

3. *ben-man-test.csv* – CSV file containing unique 1000 Manipuri words written in Bengali script along with its label (in Meitei Mayek) which will be used to test the model.

Using the *ben-man.csv,* we perform tokenization at the character level for both of the scripts. And we collect all the unique characters including alphabets and numerals, store them in a file named ben-man.tokens.

We automate the task of noise removal, data splitting and tokenization with the help of programs written in python.

7.3.2 Training the Transliteration Model

Training was perform on a linux machine with 1 GPU – Nvidia 920MX and a quad-core Intel i3 CPUs with a system memory of 8 GB. Generally the transformer model can be trained faster than architectures based on recurrent or convolutional layers. Due to limited computational resources and taking in account the advantages of Transformer mentioned above we trained the model for 50 epochs with batch size of 100 taking a total time period of 12 hrs 24 mins 59.97 secs.

We used the Adam optimizer to update the learning rate of the model with the value of

beta_1 = 0.9, beta_2 = 0.98, epsilon=1e-9, warmup_steps = 4000.

Learning rate is updated according to the formula:

$$learning\_rate = d^{-0.5}.min(step\_num^{-0.5}, step\_num.warmup\_steps^{-1.5})$$

where d = 512.

After completion of the training process our model gives an all time minimum loss of *0.005365396849811077.* During the training process evaluation has been perform on *ben-man-val.csv*. The detail error reports and visualization can be found at Chapter 8, section 2.

Visualization and illustration of the training process can be found at [13]

7.4 SOFTWARE DESIGN SPECIFICATION

The software can be divided into two main modules and are as follows:

1. OCR

2. Transliteration

We developed separate UI design for both of the modules and integrated into one single shell script program to make it easier to use.

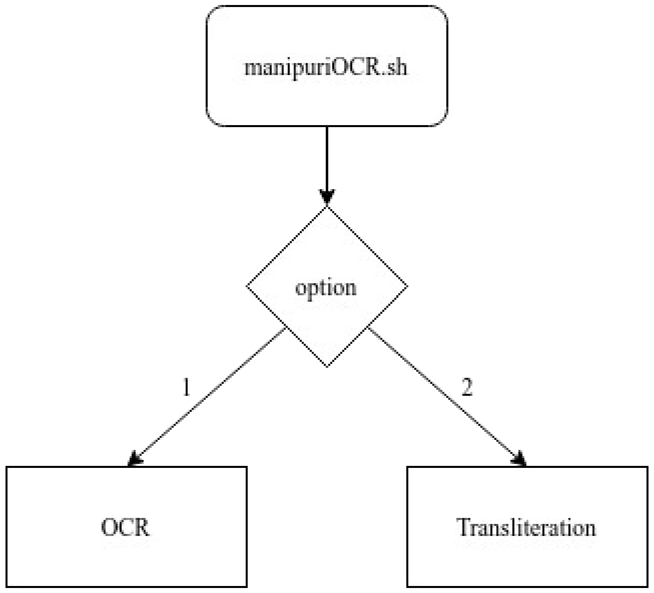
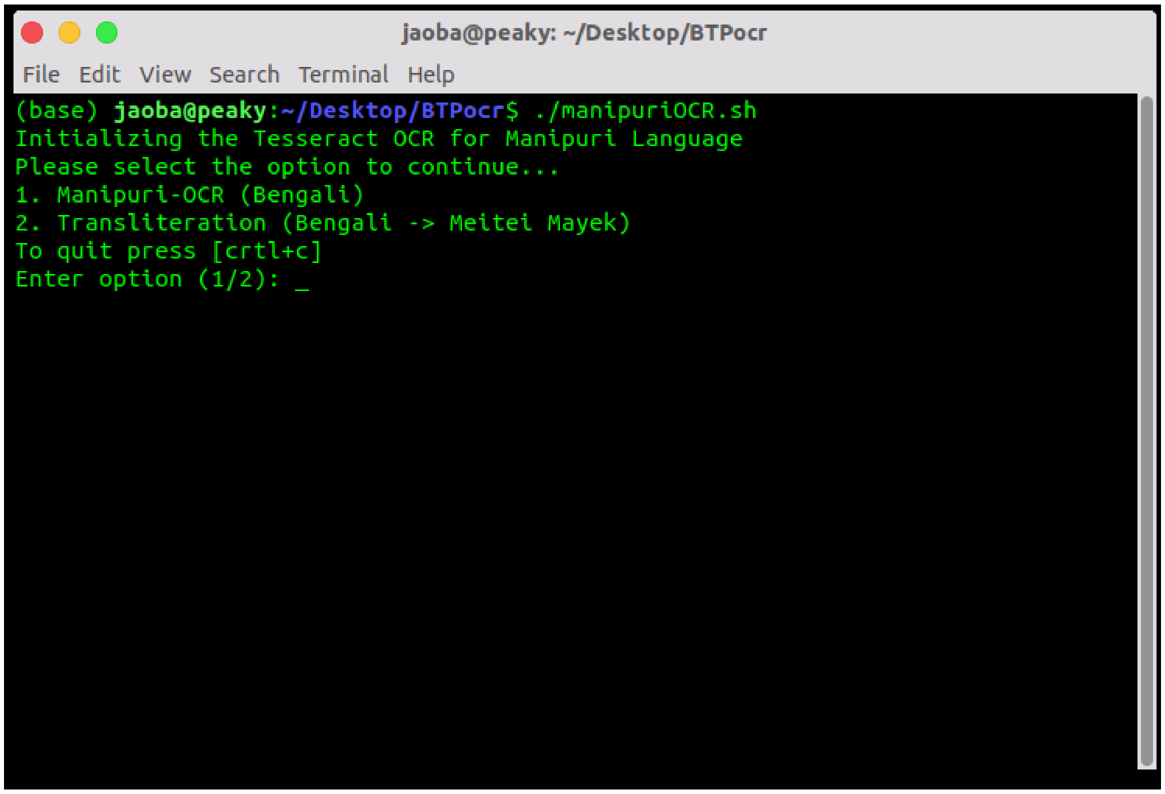


Fig. 7.4 (a) Software design overview

Following are some snapshots from the GUI


jaoba@peaky: ~/Desktop/BTPocr
File Edit View Search Terminal Help

```
(base) jaoba@peaky:~/Desktop/BTPocr$ ./manipuriOCR.sh
Initializing the Tesseract OCR for Manipuri Language
Please select the option to continue...
1. Manipuri-OCR (Bengali)
2. Transliteration (Bengali -> Meitei Mayek)
To quit press [crtl+c]
Enter option (1/2): _
```

Fig. 7.4 (b) Option selection page – manipuriOCR.sh

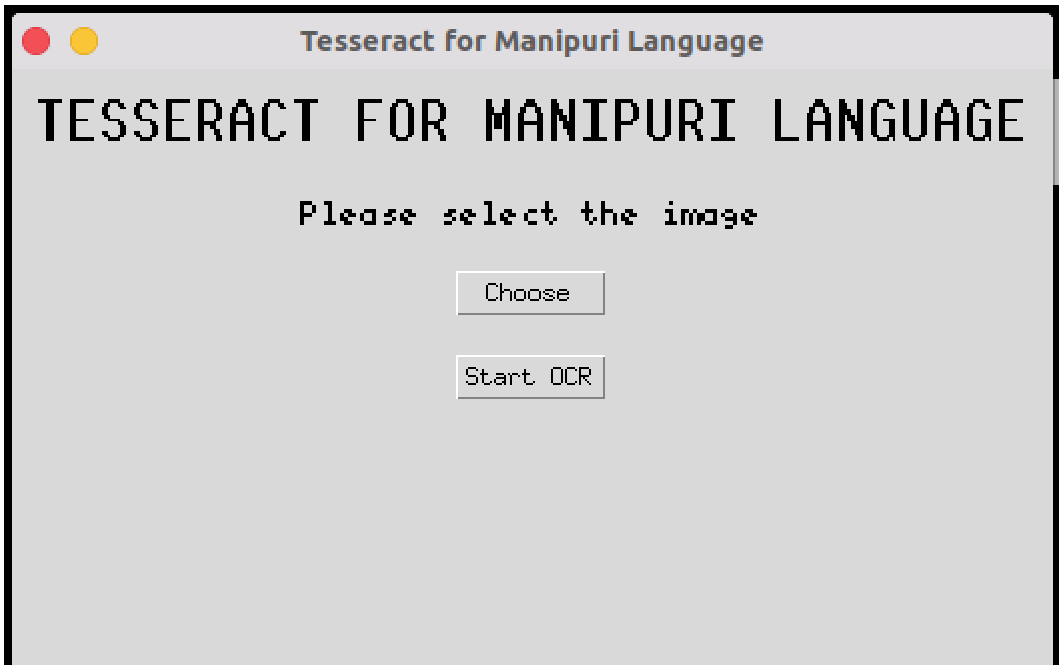
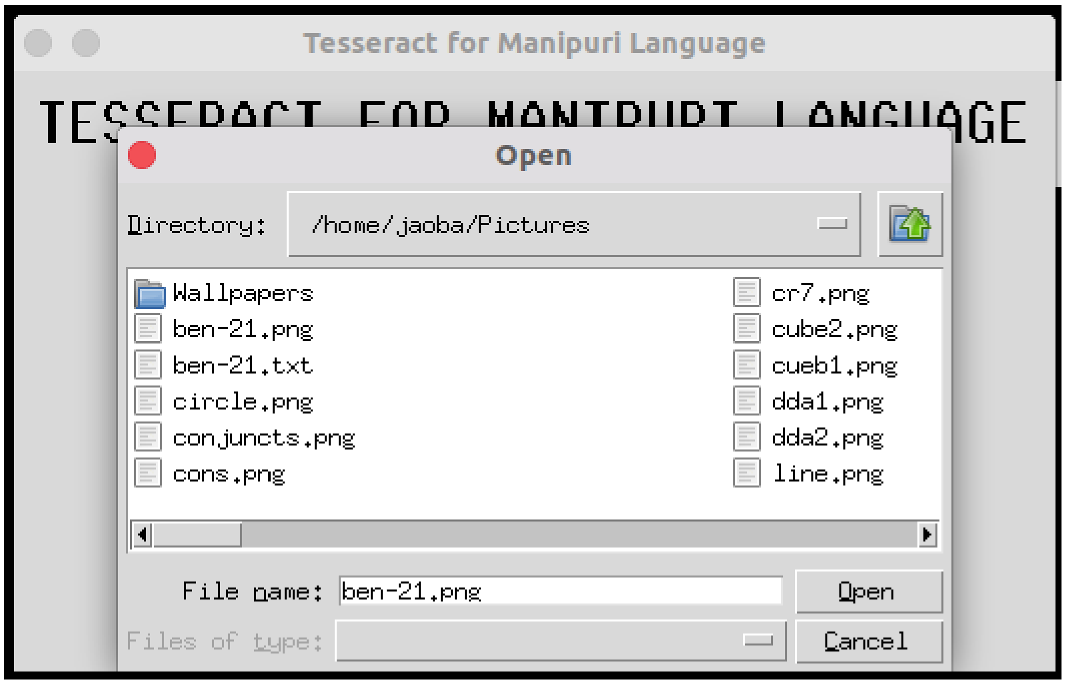

Tesseract for Manipuri Language
TESSERACT FOR MANIPURI LANGUAGE
Please select the image
Choose
Start OCR

Fig. 7.4 (c) Index page of the OCR module


Tesseract for Manipuri Language

TESSERACT FOR MANIPURI LANGUAGE

Open

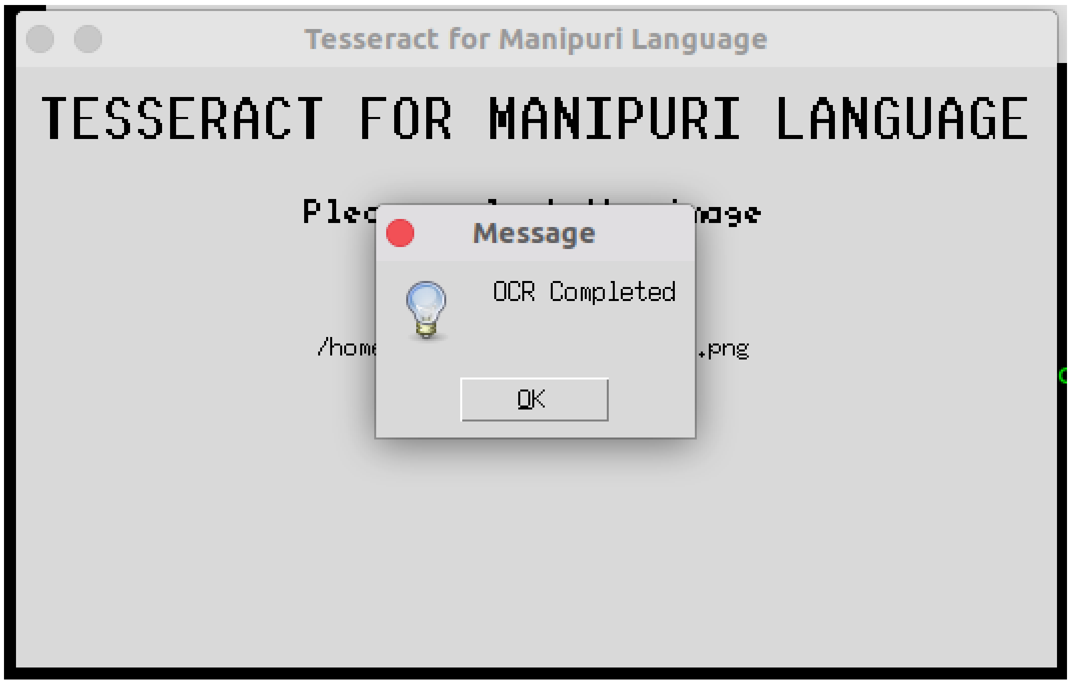
Directory: /home/jaoba/Pictures

Wallpapers
ben-21.png
ben-21.txt
circle.png
conjuncts.png
cons.png
cr7.png
cube2.png
cueb1.png
dda1.png
dda2.png
line.png

File name: ben-21.png

Files of type:

Fig. 7.4 (d) Image file selection interface

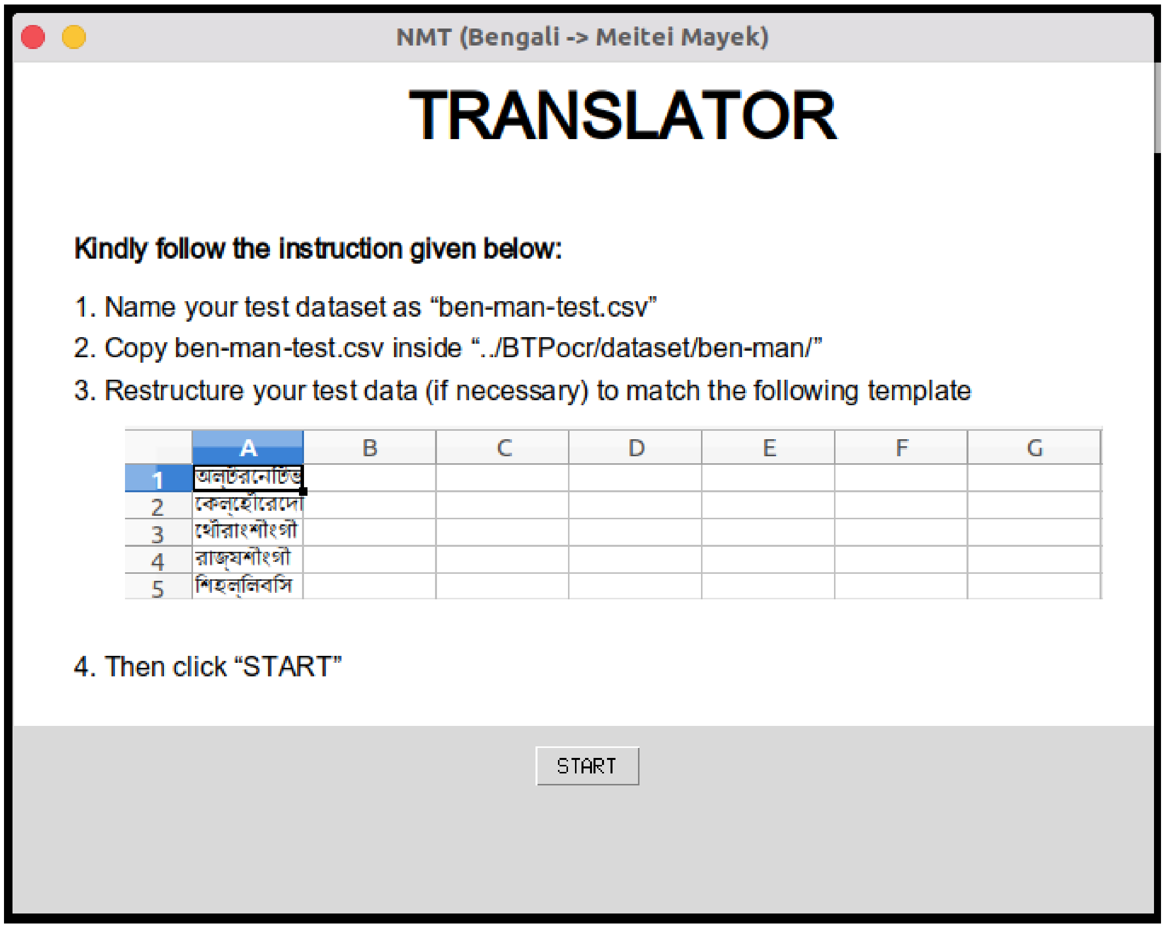

Tesseract for Manipuri Language

TESSERACT FOR MANIPURI LANGUAGE

Message

OCR Completed

Fig. 7.4 (e) Message box


NMT (Bengali -> Meitei Mayek)

TRANSLATOR

Kindly follow the instruction given below:

1. Name your test dataset as "ben-man-test.csv"
2. Copy ben-man-test.csv inside "../BTPocr/dataset/ben-man/"
3. Restructure your test data (if necessary) to match the following template

	A	B	C	D	E	F	G
1	অল্টরনেটিভ						
2	কেল্হৌরেদো						
3	থৌরাংশীংগী						
4	রাজ্যশীংগী						
5	শিহল্লিবসি						

4. Then click "START"

Fig. 7.4 (f) Index page of the Translation module

CHAPTER 8

RESULTS AND ANALYSIS

8.1 OCR MODEL

The OCR model **mni.traineddata** and the existing **ben** were tested with the same set of input data.

```
Tesseract input.tiff output[.txt] -l model_name [--psm 3] [--oem 0]
```

page segmentation mode:

--psm 3, fully automatic page segmentation, but no OSD. (Default)
--psm 6, Assume a single uniform block of text.

OCR Engine mode:

--oem 0, legacy mode.
--oem 1, lstm engine.

We prepared a ground truth (.txt) and a pdf for the same, using Ghostscript with size -r720x720 -g6120x7920 the pdf is converted into a .tiff file.

The result (out.txt) generate by tesseract 4.1.0-rc4 with each of the three models are evaluate using the **ocr-evaluation-tools** by UNLV. All the five models that we used are shown, along with the psm, oem, model used (-l), and the accuracy rate of the respective model in the following table.

Model (-l)	Font	psm	oem	Accuracy rate (%)
ben	Bangla Medium	3	1	90.81
ben	Bangla Medium	6	1	91.20
ben	Lohit Bengali	3	1	88.59
ben	Lohit Bengali	6	1	90.86
mni	Bangla Medium	3	1	94.25
mni	Bangla Medium	6	1	94.57
mni	Lohit Bengali	3	1	91.87
mni	Lohit Bengali	6	1	94.21

Table 8.1 shows the ben and mni model testing results

Here is a sample of the evaluation report generated by OCR-evaluation-tool[10].

```
UNLV-ISRI OCR Accuracy Report Version 5.1
-----------------------------------------
  53897   Characters
   2926   Errors
  94.57%  Accuracy

      0   Reject Characters
      0   Suspect Markers
      0   False Marks
   0.00%  Characters Marked
  94.57%  Accuracy After Correction

    Ins    Subst     Del   Errors
      0        0       0        0   Marked
    246     1703     977     2926   Unmarked
    246     1703     977     2926   Total

  Count   Missed   %Right
    107      107     0.00   Unassigned
   7717      477    93.82   ASCII Spacing Characters
    465        5    98.92   ASCII Special Symbols
      6        1    83.33   ASCII Digits
      1        1     0.00   ASCII Uppercase Letters
      4        4     0.00   ASCII Lowercase Letters
  45585     1348    97.04   Bengali
     12        6    50.00   General Punctuation
  53897     1949    96.38   Total

 Errors   Marked   Correct-Generated
   1604        0   {য়}-{য়}
    430        0   { }-{<\n>}
     96        0   {}-{}
     57        0   { য়}-{<\n>য়} //not EOF, full evaluation report can be found here
```

In the report above most of the error count for characters " য়" and " ড়" are due to the misclustering of the characters "য়" as (য ◌়), and "ড়" as (ড ◌়) and not because of misclassifications by the model. This type of errors are easily fixable by using a simple post-processing algorithm.

As we can see the newly generated model perform more accurately than the existing Bengali models on the Manipuri words with the model “mni” *[--psm 6 and –fontlist Bangla Medium], [input tiff -r300 (300 dpi)]* giving the highest accuracy of 94.54% and with post-processing the accuracy can be increase upto 97.35% (considering the report shown above).

8.2 TRANSLITERATION MODEL

The transliteration model achieved an all time lowest error rate of *0.0053653968498110.* Evaluation of the model has been performed during the training process and the outputs are stored locally. Below is the visualization of the same.

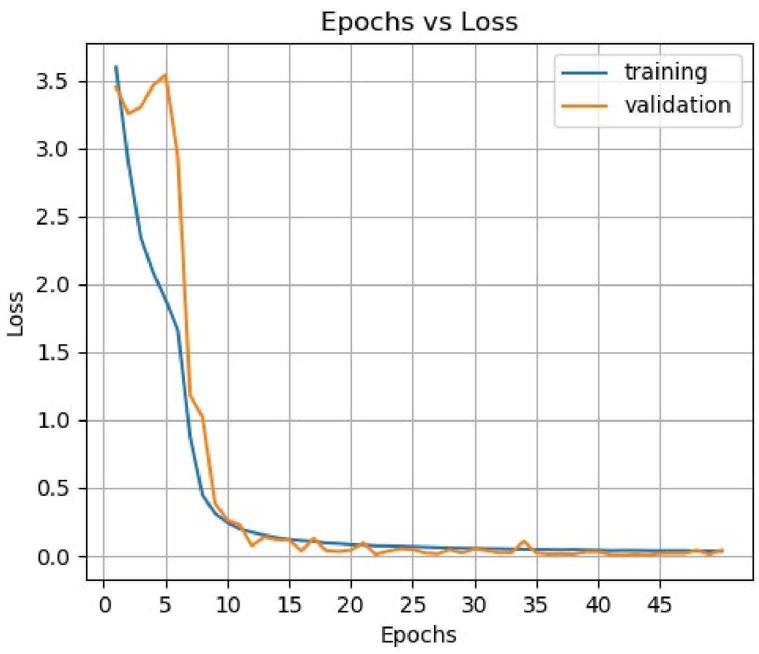


Fig. 8.1(a) Epochs vs Loss graph for training and validation

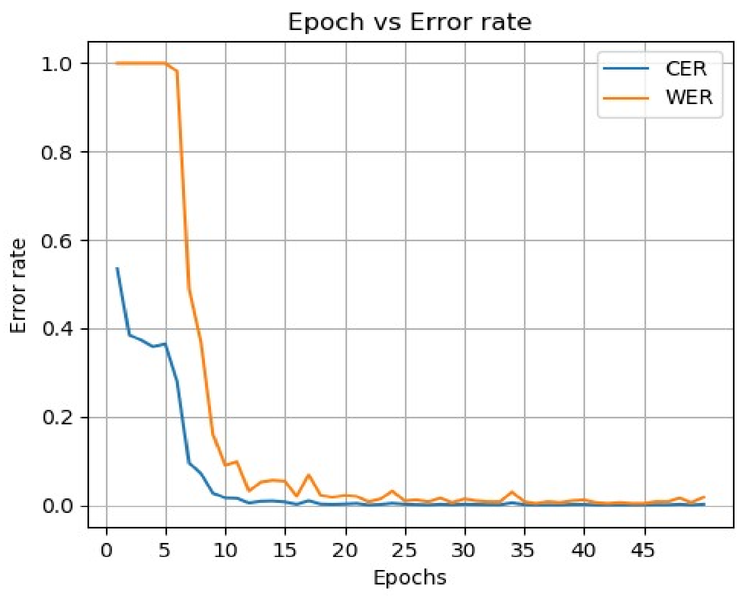


Fig. 8.1(b) Epoch vs Error rate – *CER & WER*

Fig. 8.1(a) & 8.1(b) shows the gradual decreasing of the loss or error rate of the model over epochs.

We have test the model using a seperate dataset (*ben-man-test.csv*). The dataset contains 1000 parallel instances. Out of 1000 instances only 4 instances were perdicted incorrectly. The report on the testing are given below.

test_dataset size	CER	WER	Accuracy
1000	0.0004	0.0040	99.6

Table 8.2 shows the detail testing report on *ben-man-test.csv* data

Accuracy is calculated using (number of words predicted correctly /1000)*100

CHAPTER 9

CONCLUSION AND FUTURE WORKS

9.1 CONCLUSION

In this work, we developed and presented the first Bengali to Meitei Mayek transliteration model which used Transformer, the sequence transduction model based entirely on attention. Since the model does not have any recurrent layers in the encoder-decoder architectures, the model can be trained significantly faster than other based on recurrent or convolutional layers. And the model achieved higher accuracy from other transduction model as claimed at [5].

And for the OCR, the Tesseract engine that is used in our work is actively maintained by google. We can easily re-train the existing model to recognise additional characters or scripts. The newly trained *mni* model gives higher accuracy than *ben* when we apply ocr on Manipuri words.

The transliteration model we have developed in this work is script independent. We can train the same model with different pair of source and target script and the accuracy will be acceptable.

9.2 FUTURE WORKS

The proposed method can be improved by adding segmentation of image and text parts from the scanned image before applying ocr. And for transliteration model, we can apply basic algorithms to rearrange the paragraph to give the proper format of the text and image.

So, our future work will be based on the same mentioned above.

REFERENCES

[1] https://en.wikipedia.org/wiki/Meitei_language Accessed: April, 2020.

[2] https://en.wikipedia.org/wiki/Optical_character_recognition Accessed: April, 2020.

[3] https://en.wikipedia.org/wiki/Transliteration Accessed: May, 2020.

[4] Tesseract-OCR available at https://github.com/tesseract-ocr Accessed: May, 2020.

[5] Ashish Vaswami, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser. "Attention Is All You Need" presented at the 31st Conference on Neural Information Processing System (NIPS 2017), Long Beach, CA, USA.

[6] Md. Abul Hasnat, Muttakinur Rahman Chowdhury, Mumit Khan. "An Open Source Tesseract Based Optical Character Recognizer for Bangla Script" presented at the 10th International Conference on Document Analysis and Recognition, 2009, Barcelona, Spain.

[7] https://arxiv.org/pdf/1706.03762.pdf Accessed: May, 2020.

[8] Nongmeikapam K., Singh N.H., Thoudam S., Bandyopadhyay S. (2011) Manipuri Transliteration from Bengali Script to Meitei Mayek: A Rule Based Approach. In: Singh C., Singh Lehal G., Sengupta J., Sharma D.V., Goyal V. (eds) Information Systems for Indian Languages. ICISIL 2011. Communications in Computer and Information Science, vol 139. Springer, Berlin, Heidelberg.

[9] Neelakash Kshetrimayum, "A comparative study of Meitei Mayek". 2011, [Online]. Available: http://www.typoday.in/2011/papers/Neelakash_Meetei- Mayek.pdf. Accessed: May, 2020.

[10] ocr-evaluation-tools from http://ancientgreekocr.org/ , 2018. [Online]. Available: https://github.com/ryanfb/ancientgreekocr-ocr-evaluation-tools Accessed: April, 2020.

[11] tesseract-ocr, "tessdata_best/ben.traineddata". 2017. [Online]. Available: https://github.com/tesseract-ocr/tessdata_best/blob/master/ben.traineddata Accessed: March, 2020.

[12] How to use the tools provided to train Tesseract 4.00. [Online]. Available: https://tesseract-ocr.github.io/tessdoc/TrainingTesseract-4.00.html Accessed: March, 2020.

[13] Naharol gi Thoudang, e-paper version. [Online]. Available: http://naharolgithoudang.in/web/ Accessed: March, 2020.

[14] Jay Alammar, "The Illustrated Transformer". [Online]. Available: https://jalammar.github.io/illustrated-transformer/ Accessed: May, 2020.

[15] tesseract-ocr, "langdata_lstm/ben". [Online]. Available: https://github.com/tesseract-ocr/langdata_lstm/tree/master/ben Accessed: March, 2020.

[16] Tensorflow, "Transformer model for language understanding". [Online]. Available: https://www.tensorflow.org/tutorials/text/transformer Accessed: May, 2020

[17] Tensorflow, "tensor2tensor". Available: https://github.com/tensorflow/tensor2tensor/tree/master/tensor2tensor Accessed: May, 2020.

APPENDIX A

A.1 CODE FOR THE MODEL SET UP – TRANSFORMER

```
class Transformer(tf.keras.Model):
    def __init__(self, num_layers, d_model, num_heads, dff,
input_vocab_size,
                 target_vocab_size, pe_input, pe_target, rate=0.1):
        super(Transformer, self).__init__()
        self.encoder = Encoder(num_layers, d_model, num_heads, dff,
                               input_vocab_size, pe_input, rate)
        self.decoder = Decoder(num_layers, d_model, num_heads, dff,
                               target_vocab_size, pe_target, rate)
        self.final_layer = tf.keras.layers.Dense(target_vocab_size)

    def call(self, inp, tar, training, enc_padding_mask,
             look_ahead_mask, dec_padding_mask):
        enc_output = self.encoder(inp, training, enc_padding_mask)  #
(batch_size, inp_seq_len, d_model)

        # dec_output.shape == (batch_size, tar_seq_len, d_model)
        dec_output, attention_weights = self.decoder(
            tar, enc_output, training, look_ahead_mask,
dec_padding_mask)

        final_output = self.final_layer(dec_output)  # (batch_size,
tar_seq_len, target_vocab_size)

        return final_output, attention_weights
```

A.2 CODE FOR TRAINING THE MODEL

```
# Get the datasets
dataset = data.Data(cl_args.lang_code, cl_args.reverse)
train_dataset, test_dataset, val_dataset =
dataset.get_dataset(cl_args.short_test)

# Transformer network
transformer = model.Transformer(param.NUM_LAYERS, param.D_MODEL,
param.NUM_HEADS, param.DFF,
    input_vocab_size = dataset.inp_vocab_size,
    target_vocab_size = dataset.tar_vocab_size,
    pe_input = param.PAD_SIZE,
    pe_target = param.PAD_SIZE,
    rate=param.DROPOUT
)

train_loss = tf.metrics.Mean(name='train_loss')
optimizer = utils.optimizer
train_step_signature = [
    tf.TensorSpec(shape=(None, None), dtype=tf.float32),
    tf.TensorSpec(shape=(None, None), dtype=tf.float32),
    tf.TensorSpec(shape=(None, None), dtype=tf.float32)]
```

```
@tf.function(input_signature = train_step_signature)
def train_step(inp, tar_inp, tar_real):
    enc_padding_mask, combined_mask, dec_padding_mask =
utils.create_masks(inp, tar_inp)
    # shape(inp) = (batch_size, pad_size)
    # shape(predictions) = (batch_size, pad_size, tar_vocab_size)
    with tf.GradientTape() as tape:
        predictions, _ = transformer(inp, tar_inp,
                                     True,
                                     enc_padding_mask,
                                     combined_mask,
                                     dec_padding_mask)
        loss = metrics.loss_function(tar_real, predictions)

    gradients = tape.gradient(loss, transformer.trainable_variables)
    optimizer.apply_gradients(zip(gradients,
transformer.trainable_variables))
    train_loss(loss)
```

A.3 CODE FOR EVALUATION

```
def eval_step(transformer, test_dataset, eval_file):
    eval_perf = metrics.PerformanceMetrics()

    test_dataset = utils.mk_eval_dataset(test_dataset)

    i = 0
    for inp, ref_real in test_dataset.items():
        i += 1
        # shape(pred) = (pad_size, tar_vocab_size)
        # shape(ref_real) = (ref_size, pad_size)
        pred = utils.predict(inp, transformer)

        eval_perf(ref_real, pred)

        pred = tf.argmax(pred, axis = -1)
        tr_inp  = dataset.tokenizer.inp_decode(inp) #.numpy())
        tr_pred = dataset.tokenizer.tar_decode(pred.numpy())
        for real in ref_real:
            tr_real = dataset.tokenizer.tar_decode(real.numpy())
            eval_file.write('{}, {}, {}, {}\n'.format(tr_inp, tr_real,
tr_pred, str(tr_real == tr_pred)))

        if (i + 1) % (100) == 0:
            loss, cer, wer = eval_perf.result()
            print ('\tEvaluation update\t Loss: {:.2f}\t CER: {:.2f}\t
WER: {:.2f}'.format(
                loss, cer, wer))
    eval_loss, cer, wer = eval_perf.result()
    eval_file.write('test_dataset size: {}\n'.format(i))
    eval_file.write('Loss: {:.4f}\tCER: {:.4f}\tWER:
{:.4f}'.format(eval_loss, cer, wer))
    return eval_loss, cer, wer
```

```
if __name__ == '__main__':
    cl_args = parse_cl_args()

    train_details_path = 'records/active/'
    checkpoint_path = 'records/active/checkpoints'
    best_checkpoint_path = 'records/active/checkpoints/best'

    # Get the datasets
    dataset = data.Data(cl_args.lang_code, cl_args.reverse)
    _, test_dataset, _ = dataset.get_dataset(cl_args.short_test)

    # Transformer network
    transformer_network = model.Transformer(param.NUM_LAYERS,
param.D_MODEL, param.NUM_HEADS, param.DFF,
        input_vocab_size = dataset.inp_vocab_size,
        target_vocab_size = dataset.tar_vocab_size,
        pe_input = param.PAD_SIZE,
        pe_target = param.PAD_SIZE,
        rate=param.DROPOUT
    )

    # Restoring from best checkpoint for evaluating
    ckpt = tf.train.Checkpoint(transformer=transformer_network,
optimizer=utils.optimizer)
    ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path,
max_to_keep=15)
    try:
        with open(best_checkpoint_path, 'r') as file:
            best_checkpoint = file.read().split(' ')[1]
    except:
        print('Cannot open best checkpoint file: ',
best_checkpoint_path)
        exit()

    ckpt.restore(best_checkpoint)
    print ('\nBest checkpoint restored: ', best_checkpoint)

    try:
        eval_file = open('./records/active/eval', 'w')
    except:
        print('Cannot open eval file: /records/active/eval')
        exit()

    start = time.time()
    print('\nEvaluating ...\n')
    eval_loss, cer, wer = eval_step(transformer_network, test_dataset,
eval_file)
    print('\nAfter evaluation:\tLoss: {:.4f}\tCER: {:.4f}\t WER:
{:.4f}'.format(eval_loss, cer, wer))
    eval_time_taken = time.time() - start
    print ('\nTime taken for evaluation: {}\
n'.format(utils.get_time(eval_time_taken)))
```

A.4 CODE TO DRAW EPOCHS VS ERROR GRAPH

```
graph_file = cl_args.record_dir + '/graph'
    try:
        file = open(graph_file, 'r')
    except FileNotFoundError:
        print('Metric file {} not found'.format(graph_file))
        exit()
    epochs = []
    t_loss_list = []
    v_loss_list = []
    cer_list = []
    wer_list = []
    cnt = 0
    line = file.readline()
    while line:
        cnt += 1
        line = line.strip()
        metrics = line.split(',')
        try:
            t_loss = float(metrics[0].strip())
            v_loss = float(metrics[1].strip())
            cer = float(metrics[2].strip())
            wer = float(metrics[3].strip())
        except:
            print('Invalid metric: {}'.format(metrics))
            exit()
        epochs.append(cnt)
        t_loss_list.append(t_loss)
        v_loss_list.append(v_loss)
        cer_list.append(cer)
        wer_list.append(wer)
        line = file.readline()
    # end of while
    file.close()
    plt.figure(figsize=(12, 5))
    plt.subplot(121)
    t_handle, = plt.plot(epochs, t_loss_list)
    v_handle, = plt.plot(epochs, v_loss_list)
    plt.xticks(np.arange(0, cnt, 5.0))
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Epochs vs Loss')
    plt.grid(True)
    plt.legend([t_handle, v_handle], ['training', 'validation'], loc =
'upper right')

    plt.subplot(122)
    cer_handle, = plt.plot(epochs, cer_list)
    wer_handle, = plt.plot(epochs, wer_list)
    plt.xticks(np.arange(0, cnt, 5.0))
    plt.xlabel('Epochs')
    plt.ylabel('Error rate')
    plt.title('Epoch vs Error rate')
    plt.grid(True)
    plt.legend([cer_handle, wer_handle], ['CER', 'WER'], loc = 'upper
right')
```

```
plt.subplots_adjust(bottom=0.2)

if not cl_args.dont_save:
    img = cl_args.record_dir + "/graph.png"
    plt.savefig(img) # save to file
if cl_args.on_screen:
    plt.show()
```