Thesis Mid Term for the Degree of Master of Computer Information System

Nepali Text Part Of Speech Tagging Using Different Deep Learning Algorithms

Tika Ram Khojwar
(2020-2-92-0018)

Nepal College of Information Technology
Faculty of Management Studies
Pokhara University, Nepal

September, 2023

ACKNOWLEDGEMENTS

First of all, I would like to express my deep gratitude to Dr. Pradip Paudyal, my supervisor, for his unwavering help and insightful comments during my research work. His constructive feedback was the reason to improve my ideas and increase the overall quality of the work.

I'd also want to thank Mr. Saroj Shakya, the master's Program coordinator, for his amazing coordination and support. He provided me with tremendous possibilities to further my studies and academic aspirations.

As an NCIT student, I owe a debt of gratitude to the entire faculty for their accomplishments constant availability, and advice. Their provision of the right tools and advice contributed significantly to the successful outcome of my work dissertation.

My heartfelt thanks go to my parents for their constant advice, encouragement, and support in difficult times. Their belief in me has never wavered and I owe my success to their constant motivation.

Finally, I would like to thank my friend Mr. Sujan Paudel. His camaraderie, support, and assistance played a crucial role in getting me through this thesis journey.

In summary, the road to completing this thesis would not be long possible without the generous help and support of my mentors, educators, parents, and colleagues. I express my deep gratitude to you to all of them.

ABSTRACT

POS tagging is an essential and foundational task in numerous natural language processing (NLP) applications, such as machine translation, text-to-speech conversion, question answering, speech recognition, word sense disambiguation, information retrieval, text summarization, Named Entity Recognition, and sentiment analysis, among others. POS tagging entails assigning the correct tag to each token in the corpus, considering its context and the language's syntax. An optimal part-of-speech tagger plays a crucial role in computational linguistics. Its importance cannot be emphasized enough because inaccuracies in tagging can greatly affect the performance of complex natural language processing systems. This work shows that deep learning algorithms are great for fine grained Nepali POS-tagging. We experiment with three DL algorithms GRU, LSTM, BiLSTM and mBERT. We trained all the models in this work using genism's Word2Vec word embedding technique. We found that among the tested models, BiLSTM model performed the best and achieved F1 score of 99.76 % for fine grained Nepali text POS tagging, while using 37 XML files with 583,028 words as a sample dataset and 98 tags. XML are books, journals and web.

Keywords: POS Tagging, Nepali Text, Natural Language Processing, GRU, LSTM, BiLSTM, mBERT.

Table of content

List of table

List of figures

List of abbreviation/acronyms

NLP	Natural Language Processing
POS	Part Of Speech
HMM	Hidden Markov Model
SVM	Support Vector Machine
ANN	Artificial Neural Network
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
BiLSTM	Bi-directional Long Short-Term Memory
mBERT	Multilingual Bidirectional Encoder Representations from Transformers

CHAPTER 1

INTRODUCTION

1.1 Background

Natural Language Processing is a field of artificial intelligence that focuses on the interaction between computers and human language. NLP involves the development of algorithms and techniques to enable computers to understand, interpret, and generate human language in a way that is meaningful and useful.

In NLP, Part-of-Speech tagging is a fundamental task. It involves assigning POS tags (Noun, pronoun, verb, adjective, adverb, preposition etc.) to each word in a sentence of a natural language. The input for the algorithm consists of a sequence of words in a natural language sentence and a predefined set of POS tags. The output is the most suitable POS tag assigned to each word in the sentence. POS tagging provides valuable information about a word and its neighboring words, which proves beneficial for various advanced NLP tasks like speech and natural language processing applications, semantic analysis, machine translation, text-to-speech conversion, question answering, speech recognition, word sense disambiguation and information retrieval, text summarization, Named entity recognition and more.

Nepali is a morphologically rich language. One of the characteristics features of the Nepali language is its rich inflectional system, especially the verbal inflection system. A verb in Nepali can easily display more than 20 inflectional forms while encoding different morphological features, including aspect, mood, tense, gender, number, honorifics, and person.

1.2 Statement of the problem

Nepali is a morphologically rich language. Several POS tagging models for the Nepali language have been attempted in the past. Rule-based and statistical techniques are not showing significant results as context and sequence are not taken care of. Deep learning approaches found best in the context of morphologically rich languages like Nepali. There is also a constraint in automatically tagging "Unknown" words with a high false positive rate. There is also found, there have been very limited efforts in fine grained Nepali POS-tagging. This research study focused on evaluate the performance of different models and algorithms to find the optimal fine grained Nepali POS-tagging. The models are supervised deep learning models such as GRU, LSTM, BiLSTM and mBERT. Because Deep Learning oriented methodologies improves the efficiency and effectiveness of POS tagging in terms of accuracy and reduction in false-positive rate. These models trained with the available tagged dataset and tested to compare the performance measures of each classification algorithm.

1.3 Research questions

Some research questions are:

- Which supervised classification model demonstrates the highest accuracy in assigning tags to Nepali text, considering factors such as training data availability?
- How does this research contribute to improving word sense disambiguation in Nepali text?
- What are the potential practical implications and benefits of implementing the findings of this research study for various advanced NLP tasks in the Nepali language?

1.4 Research objectives

The best model depends on various factors, including the availability of training data, Language, computational resources, and the specific requirements of the application. So, experiment with different models and compare their performance on the specific task or dataset to determine the most suitable model.

The main objective of this paper is:

- Train and compare between different deep learning algorithms such as GRU, LSTM, BiLSTM and mBERT for nepali text POS tagging.
- Find out which algorithm is best suited for the process of POS tagging for Nepali text.

1.5 Significance/Rationale of the study

The main significance of this research is to assign the correct tag to the word of text. Only correct assignment of the tag gives correct sense of the words. Which proves beneficial for various advanced NLP tasks like speech and natural language processing applications, semantic analysis, machine translation, text-to-speech conversion, question answering, speech recognition, word sense disambiguation and information retrieval, text summarization, Named entity recognition.

CHAPTER 2

LITERATURE REVIEW

2.1 Literature Review

There are only few researches have been done in the field of POS tagging for Nepali language. Some of them used statistical model (HMM) for identifying the tags while some used supervised machine learning model and some used supervised deep learning model to train the model.

Ingroj Shrestha, Shreeya Singh Dhakal (2021):

This article applied three deep learning models: BiLSTM, BiGRU, and BiLSTM-CRF for fine-grain POS tagging for the Nepali language. It uses Nepali National Corpus (NNC). It has 17 million manually and semi-manually words tagged with 112 POS-tags. Results show that deep learning models could capture fine-grained morphological features like gender, person, number, and honorifics that are encoded within words in highly inflectional languages like Nepali with a large enough dataset. This study trained all the models using two embedding: pre-trained multi-lingual BERT and randomly initialized Bare embedding. It found that training a randomly initialized Bare embedding is better than the ones trained using large pre-trained multi-lingual BERT embedding for downstream tasks in Nepali like POS tagging. Among the tested models, the BiLSTM-CRF with the Bare embedding performed the best and achieved a new state-of-the-art F1 score of 98.51% for fine-grained Nepali POS tagging. This research contributes to the advancement of NLP techniques tailored specifically for the Nepali language.

Sarbin Sayami, Tej Bahadur Shahi and Subarna Shakya (2019):

This paper addresses the implementation and comparison of various deep learning-based POS taggers for Nepali text. The examined approaches include LSTM, GRU, BiLSTM and mBERT. These models are trained and evaluated using Nepali English parallel corpus annotated with 43 POS tag and contains nearly 88000 words which is collected from m Madan Puraskar Pustakalaya. The design of this Nepali POS Tag-set is inspired by the PENN Treebank POS Tag-set. The data set is divided into three sections i.e. training, development and testing. The accuracy obtained for simple RNN, LSTM, GRU and

Bidirectional LSTM are 96.84%, 96.48%, 96.86% and 97.27% respectively. Therefore, Bi-directional LSTM outperformed all other three variants of RNN.

Greeshma Prabha, Jyothsna P V, Shahina kk, Premjith B, Soman K P (2018):

This paper proposed a deep learning based POS tagger for Nepali text which is built using Recurrent Neural Network (RNN), Long Short Term Memory Networks (LSTM), Gated Recurrent Unit (GRU) and their bidirectional variants. It uses POS Tagged Nepali Corpus generated by translating 4325 English sentences from the PENN Treebank corpus tagged with 43 POS tags. The results demonstrate that the proposed model outperforms existing state-of-the-art POS taggers with an accuracy rate exceeding 99%. This research contributes to the field by showcasing the effectiveness of deep learning techniques in improving POS tagging for Nepali text.

Ashish Pradhan, Archit Yajnik (2021):

This article presents a comprehensive study and comparing two techniques, HMM and GRNN, for POS Tagging in Nepali text. The POS taggers aim to address the issue of ambiguity in Nepali text. Evaluation of the taggers is performed using corpora from TDIL (Technology Development for Indian Languages) which contains a total of 424716 tagged words with 39 tags, tags follow the guidelines of ILCI (Indian Languages Corpora Initiative), BSI (Bureau of Indian Standard), with implementation carried out using Python and Java programming languages, along with the NLTK Toolkit library. The achieved accuracy rates are as follows: 100% for known words (without ambiguity), 58.29% for ambiguous words (HMM), 60.45% for ambiguous words (GRNN), and 85.36% for non-ambiguous unknown words (GRNN). Although the GRNN tagger achieves the accuracy as high as the HMM Tagger, it fails or becomes unstable when the training data set is greater than 7000 words, while the HMM Tagger is trained with more than 400000 words with corresponding tags. A total of 4000 words are used for testing on both HMM and GRNN taggers.

Archit Yajnik (2018):

This article focuses on POS tagging for Nepali text using the GRNN. Because GRNN is less expensive as compared to standard algorithms viz. Back propagation, Radial basis function, support vector machine etc. And also neural network is usually much faster to train than the traditional multilayer perceptron network. The corpus has total 7873 Nepali words with 41 tags. Out of which 5373 samples are used for training and the remaining 2500 samples for testing. The results show that 96.13% of words are correctly tagged on the training set, while 74.38% are accurately tagged on the testing set using GRNN. To compare the performance, the traditional Viterbi algorithm based on HMM is also evaluated. The Viterbi algorithm achieves classification accuracies of 97.2% and 40% on the training and testing datasets, respectively. The study concludes that the GRNN-based POS tagger demonstrates more consistency compared to the traditional Viterbi decoding technique. The GRNN approach yields a higher accuracy on the testing dataset, suggesting its potential for improved POS tagging in Nepali text compared to the Viterbi algorithm.

Archit Yajnik (2018):

The article that introduces POS tagging for Nepali text using three Artificial Neural Network (ANN) techniques. A novel algorithm is proposed, extracting features from the marginal probability of the Hidden Markov Model. These features are used as input vectors for Radial Basis Function (RBF) network, General Regression Neural Networks (GRNN), and Feed forward neural network. The training database contains 42100 words whereas the testing set consists of 6000 words with 41 tags. The GRNN-based POS tagging technique outperforms the others, achieving 100% accuracy for training and 98.32% accuracy for testing. This research contributes to Nepali POS tagging by presenting a novel algorithm and highlighting the effectiveness of the GRNN approach.

Archit Yajnik (2017):

This article focuses on POS tagging for Nepali text using the HMM and Viterbi algorithm. The study reveals that the Viterbi algorithm outperforms HMM in terms of computational efficiency and accuracy. Database is generated from NELRALEC Tagset with 41 tags. A report on Nepali Computational Grammar is made available by Prajwal Rupakheti et al. Database contains 45000 Nepali words with corresponding Tag, out of which 15005 samples are randomly collected for testing purpose. The Viterbi algorithm achieves an accuracy rate of 95.43%. The article also provides a detailed discussion of error analysis, specifically examining the instances where mismatches occurred during the POS tagging process.

Abhijit Paul, Bipul Syam Purkayastha, Sunita Sakar (2015):

This paper discusses HMM based POS tagging for the Nepali language. The study evaluates the HMM tagger using corpora from Technology Development for Indian Languages (TDIL) which contains around 1,50,839 tagged words and tagset consists of 42 tags including generic attributes and language specific attribute values. It has been followed the guidelines of ILCI (Indian Languages Corpora Initiative), BSI (Bureau of Indian Standard). The implementation is done using Python and the NLTK library. The HMM-based tagger achieves an accuracy of over 96% for known words but the system is not performing well for the text with unknown words yet. Overall, the paper provides insights into the effectiveness of HMM for Nepali POS tagging and highlights areas for future improvement.

Tej Bahadur Shahi, Tank Nath Dhamala, Bikash Balami (2013):

This paper focuses on SVM based POS tagger for Nepali language which uses the dictionary as a primary resources. This dictionary is collected from the FinalNepaliCorpus which contains only 11147 unique words. The POS tagging approaches like rule-based and HMM cannot handle many features that would generally be required for modeling a morphologically rich language like Nepali. SVM is efficient, portable, scalable and trainable. So, this paper proposes a SVM based tagger. The SVM tagger constructs feature vectors for each word in the input and classifies them into one of two classes using a One Vs Rest approach. The SVM algorithm achieves an accuracy rate of 96.48% for known words, 90.06% for unknown words and 93.27% in overall. That means SVM tagger demonstrates strong performance for known words. In comparison to rule-based and Hidden Markov Model (HMM) approaches, the SVM-based tagger exhibits a slightly higher overall accuracy.

CHAPTER 3

METHODOLOGY

In this chapter, we present the proposed method for determining the POS tags of the provided text. The following figure provides an overview of the tagging process.

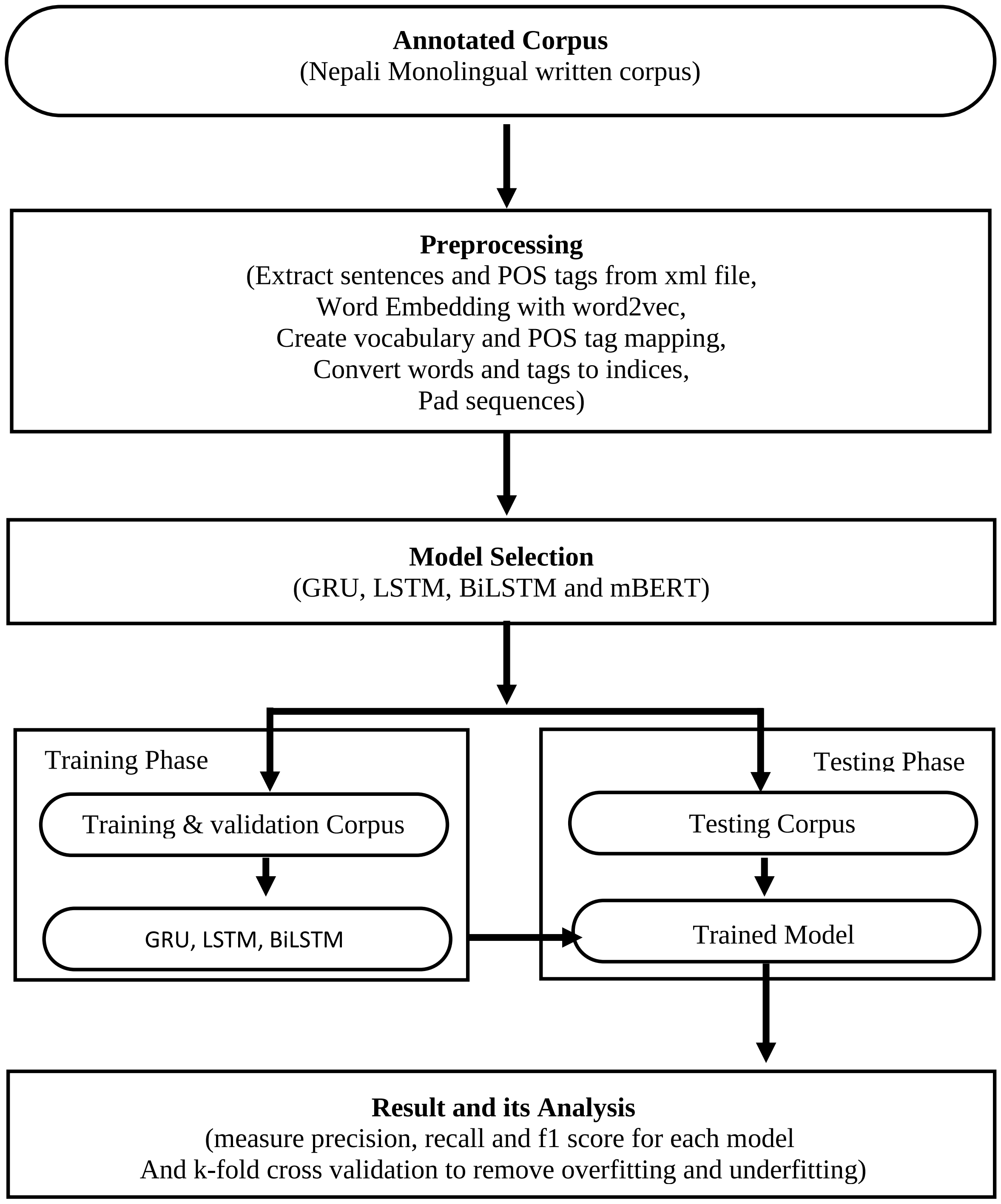
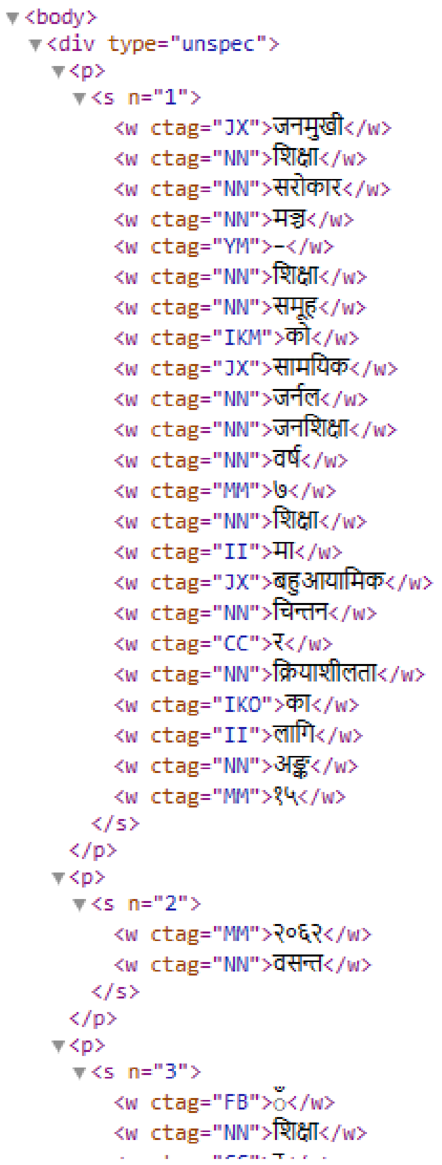

Annotated Corpus
(Nepali Monolingual written corpus)
Preprocessing
(Extract sentences and POS tags from xml file,
Word Embedding with word2vec,
Create vocabulary and POS tag mapping,
Convert words and tags to indices,
Pad sequences)
Model Selection
(GRU, LSTM, BiLSTM and mBERT)
Training Phase
Training & validation Corpus
GRU, LSTM, BiLSTM
Testing Phase
Testing Corpus
Trained Model
Result and its Analysis
(measure precision, recall and f1 score for each model
And k-fold cross validation to remove overfitting and underfitting)

Fig 3.1: Proposed methodology for POS tagging

3.1 Data Collection

This research study has implemented Nepali National Corpus (NNC). It has two main parts: the core corpus (core sample) and the general corpus. The core sample (CS) is a compilation of Nepali written texts from 15 diverse genres, with each text containing 2000 words. These texts were published between 1990 and 1992. On the other hand, the general corpus (GC) comprises written texts gathered from various sources, including the internet, newspapers, books, publishers, and authors, opportunistically collected without a specific sampling criteria. The corpus has over 14 million words. It is a morphologically annotated corpus. A parts-of-speech tagset has been produced within the project: the Nelralec Tagset. Nelralec tagset has 112 tags. Following is a sample of the training data:



```
▼<body>
  ▼<div type="unspec">
    ▼<p>
      ▼<s n="1">
          <w ctag="JX">जनमुखी</w>
          <w ctag="NN">शिक्षा</w>
          <w ctag="NN">सरोकार</w>
          <w ctag="NN">मञ्च</w>
          <w ctag="YM">-</w>
          <w ctag="NN">शिक्षा</w>
          <w ctag="NN">समूह</w>
          <w ctag="IKM">को</w>
          <w ctag="JX">सामयिक</w>
          <w ctag="NN">जर्नल</w>
          <w ctag="NN">जनशिक्षा</w>
          <w ctag="NN">वर्ष</w>
          <w ctag="MM">७</w>
          <w ctag="NN">शिक्षा</w>
          <w ctag="II">मा</w>
          <w ctag="JX">बहुआयामिक</w>
          <w ctag="NN">चिन्तन</w>
          <w ctag="CC">र</w>
          <w ctag="NN">क्रियाशीलता</w>
          <w ctag="IKO">का</w>
          <w ctag="II">लागि</w>
          <w ctag="NN">अङ्क</w>
          <w ctag="MM">१५</w>
        </s>
      </p>
    ▼<p>
      ▼<s n="2">
          <w ctag="MM">२०६२</w>
          <w ctag="NN">वसन्त</w>
        </s>
      </p>
    ▼<p>
      ▼<s n="3">
          <w ctag="FB">ँ</w>
          <w ctag="NN">शिक्षा</w>
```

Fig. 3.2: Sample of training dataset

The tag distribution in the tag-set is depicted by the bar graph below:

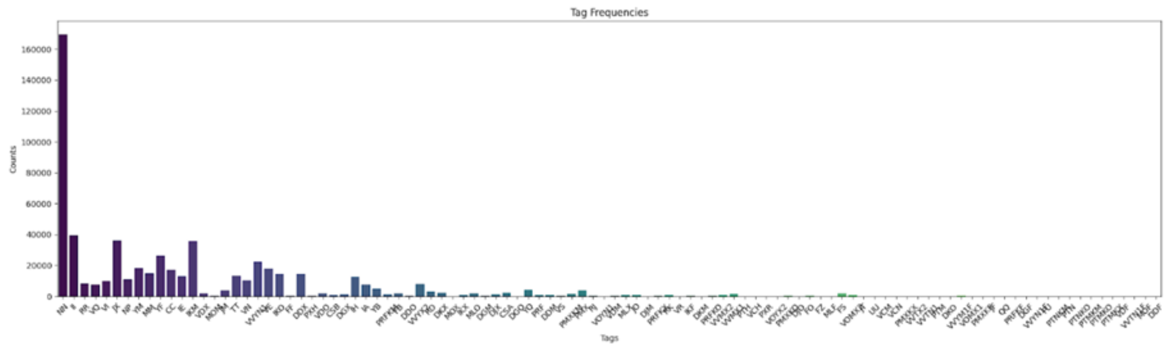


Fig. 3.3: Tags distribution in the corpus

The tag distribution chart shows and uneven distribution of the tags in the corpus. The first tag 'NN' has the highest number of occurrence totaling to 169767 whereas the second highest number of occurrence is for 'II' totaling 39387. There are tags with only a few hundred or tens of occurrences with few tags having 1 occurrences. This shows a very non-uniform distribution of the tags in the dataset.

3.2 Pre-processing

Only good dataset can give good output. To make good dataset, we need to transform the text into something meaningful that the algorithm can use. Preprocessing includes the extract sentences and POS tags from xml file, word embedding with word2vec, create vocabulary and POS tag mapping, convert words and tags to indices and pad sequences.

3.2.1 Extract Sentences and POS Tags from XML files

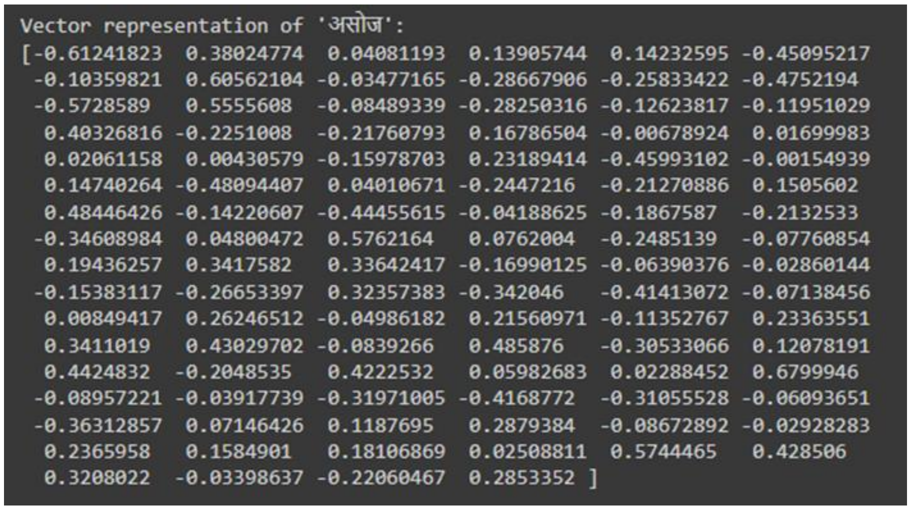
Nepali monolingual written corpus files are in XML format. We cannot use XML files as input to train a deep learning model. Therefore, we first extract sentences and their corresponding parts-of-speech tags in list form. For example:

all_sentences = [['सत्तापक्ष', 'सित', 'नजीक', 'रहेर', 'काम', 'गर्न', 'विपक्षीदल', 'प्रतिवद्ध'], ['काठमाडौं', ',', 'असोज', '१६', '।'], …]

all_pos_tags = [['NN', 'II', 'RR', 'VQ', 'NN', 'VI', 'NN', 'JX'], ['NP', 'YM', 'NN', 'MM', 'YF'], ...]

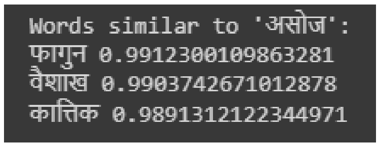
3.2.2 Word Embedding with Word2Vec

"Word2Vec is a popular word embedding technique in the fields of NLP and ML. It is used to represent words as dense vectors in a continuous vector space, where words with similar meanings are located closer to each other in this space. Word2Vec models are trained on large text corpora and are capable of capturing semantic relationships between words. To achieve this, we first created a Word2Vec model using Gensim in Python. We then passed all_sentences as an argument to the created model, resulting in the embedding_model. Let's examine the vector representation of a 'असोज' word and find similar words using the embedding_model."



```
Vector representation of 'असोज':
[-0.61241823  0.38024774  0.04081193  0.13905744  0.14232595 -0.45095217
 -0.10359821  0.60562104 -0.03477165 -0.28667906 -0.25833422 -0.4752194
 -0.5728589   0.5555608  -0.08489339 -0.28250316 -0.12623817 -0.11951029
  0.40326816 -0.2251008  -0.21760793  0.16786504 -0.00678924  0.01699983
  0.02061158  0.00430579 -0.15978703  0.23189414 -0.45993102 -0.00154939
  0.14740264 -0.48094407  0.04010671 -0.2447216  -0.21270886  0.1505602
  0.48446426 -0.14220607 -0.44455615 -0.04188625 -0.1867587  -0.2132533
 -0.34608984  0.04800472  0.5762164   0.0762004  -0.2485139  -0.07760854
  0.19436257  0.3417582   0.33642417 -0.16990125 -0.06390376 -0.02860144
 -0.15383117 -0.26653397  0.32357383 -0.342046   -0.41413072 -0.07138456
  0.00849417  0.26246512 -0.04986182  0.21560971 -0.11352767  0.23363551
  0.3411019   0.43029702 -0.0839266   0.485876   -0.30533066  0.12078191
  0.4424832  -0.2048535   0.4222532   0.05982683  0.02288452  0.6799946
 -0.08957221 -0.03917739 -0.31971005 -0.4168772  -0.31055528 -0.06093651
 -0.36312857  0.07146426  0.1187695   0.2879384  -0.08672892 -0.02928283
  0.2365958   0.1584901   0.18106869  0.02508811  0.5744465   0.428506
  0.3208022  -0.03398637 -0.22060467  0.2853352 ]
```

Fig. 3.4: Vector representation of 'असोज'



```
Words similar to 'असोज':
फागुन 0.9912300109863281
वैशाख 0.9903742671012878
कात्तिक 0.9891312122344971
```

Fig. 3.5: Words similar to 'असोज'

3.2.3 Create Vocabulary and POS Tag Mapping

Vocabulary and POS tag mappings convert words and tags into numerical indices for deep learning models. For vocabulary and POS tag mappings, we firstly create vocab set and pos_tags_set set for unique words and unique POS tags respectively. Then create word_to_idx dictionary to maps each unique word in the vocabulary to a unique index. And same for the tag_to_idx. Then create idx_to_tag dictionary. It is reverse of the tag_to_idx dictionary. We also use 'UNK' token to handle out-of-vocabulary words or tags that are not present in the initial training data. Following are sample of the word_to_idx and tag_to_idx dictionaries.

word_to_idx = {'समर्थित': 1, 'व्यक्तिविशेष': 2, 'खुलेछन्': 3, 'चर्कँदो': 4, …}

tag_to_idx = {'IE': 1, 'VVTN1': 2, 'DGF': 3, 'PTH': 4, 'YQ': 5, …}

3.2.4 Convert Words and Tags to Indices

It is the process where we converted words and POS tags to their corresponding numerical indices using the mappings created earlier (word_to_idx and tag_to_idx). Each inner list represents a sentence, and it contains numerical indices corresponding to the words or POS tags in that sentence. Following are the sample of sentences_indices and pos_tags_indices list of lists.

sentences_indices = [[24283, 33809, 29171, 17993, 31405, 11612, 27897, 29351],

[39860, 14035, 16097, 37307, 10206], …]

pos_tags_indices = [[20, 89, 25, 24, 20, 43, 20, 88], [44, 39, 20, 55, 46], …]

3.2.5 Pad Sequences

When training neural networks algorithms that expect fixed-length or uniform-length input sequences. The purpose of padding is to ensure all sequences have the same length. Padding is often necessary when working with sequences of varying lengths, which is common in NLP tasks.

3.3 Model

There are several models have been widely used and achieved good performance in POS tagging tasks. Different models have their own different features and specific task. There is no single "best" model for POS tagging, as the effectiveness of a model can vary depending on factors such as the dataset, language, and specific requirements of the task. According to previous research Deep Learning algorithms based models gives better accuracy in testing dataset and can deal with ambiguous, unknown words as compare to rule based, statistical and machine learning algorithms for POS tagging.

3.3.1 RNN

Recurrent Neural Network (RNN) is a type of neural network specifically designed to handle sequential data. It has feedback connections that enable it to maintain and utilize information from previous steps in the sequence. RNNs are effective in capturing dependencies over time and are commonly used in tasks such as natural language processing, speech recognition, and time series analysis. They can learn patterns and make predictions based on the context of the sequence. Overall, RNNs are powerful tools for modeling and processing sequential data.

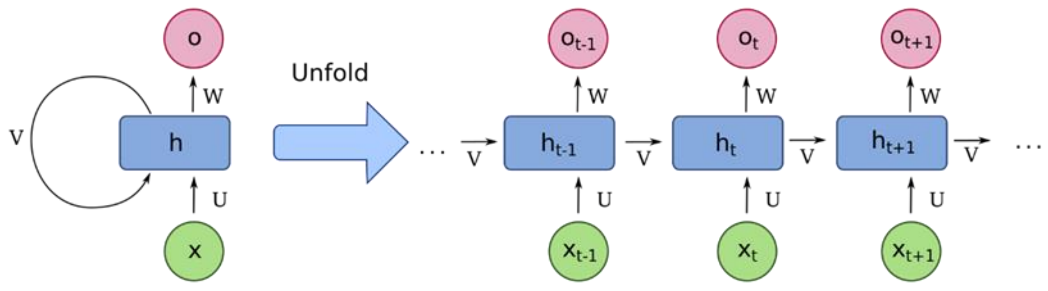


Fig. 3.6: Compressed (left) and unfolded (right) basic recurrent neural network

3.3.2 GRU

A Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) used in deep learning. It is designed to address the vanishing gradient problem by incorporating gating mechanisms. GRU has two gates, an update gate and a reset gate, which control information flow in and out of the hidden state. The update gate helps retain relevant information from the past while the reset gate helps update the hidden state with new information. GRUs are computationally efficient and often outperform traditional RNNs in tasks involving sequential data, such as natural language processing and speech recognition.

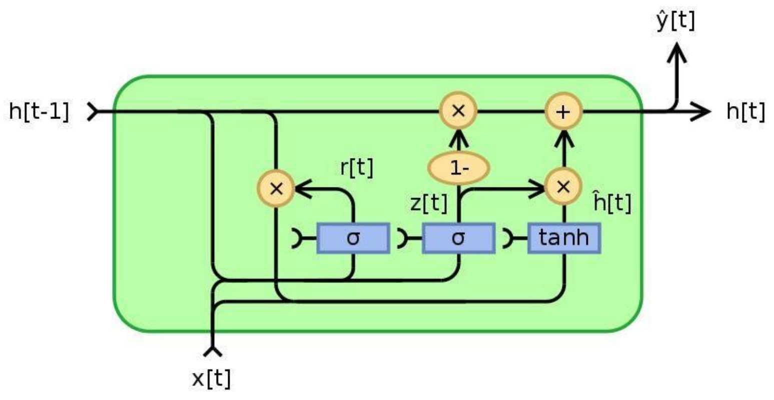


Fig. 3.7: GRU model [12]

3.3.3 LSTM

Long Short-Term Memory (LSTM) is a type of RNN architecture. It can deal with capturing long-term dependencies in sequential data. Because LSTMs utilize a memory cell, along with input, forget, and output gates, to selectively retain or discard information based on context. This helps them overcome the vanishing gradient problem and effectively learn from sequences of varying lengths. LSTMs have been successfully applied to tasks such as natural language processing, speech recognition, machine translation, and time series analysis. They are known for their ability to capture long-term dependencies and have become popular for modeling sequential data.

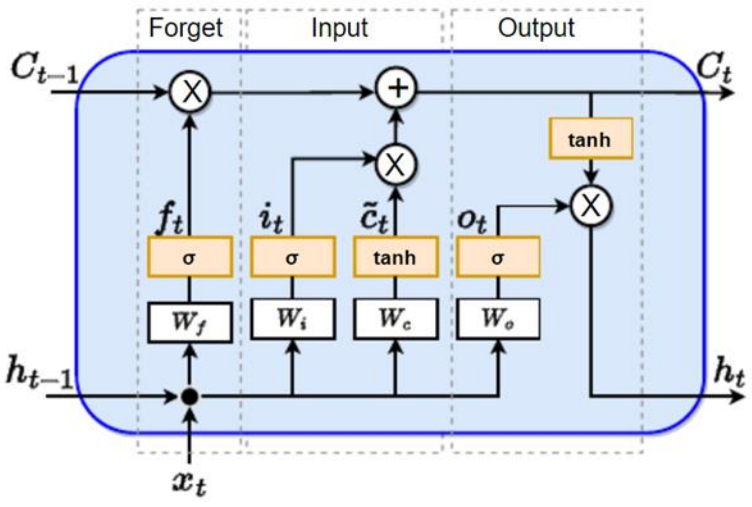


Fig. 3.8: LSTM cell with gating controls [13]

3.3.4 BiLSTM

Bidirectional Long Short-Term Memory (BiLSTM) is a variant of the LSTM model that processes input sequences in both forward and backward directions. By capturing context from both past and future elements, Bi-LSTM can model long-range dependencies and excel in tasks that require complete context understanding. It is widely used in NLP tasks, enabling improved performance and accuracy in analyzing sequential data with bidirectional context.

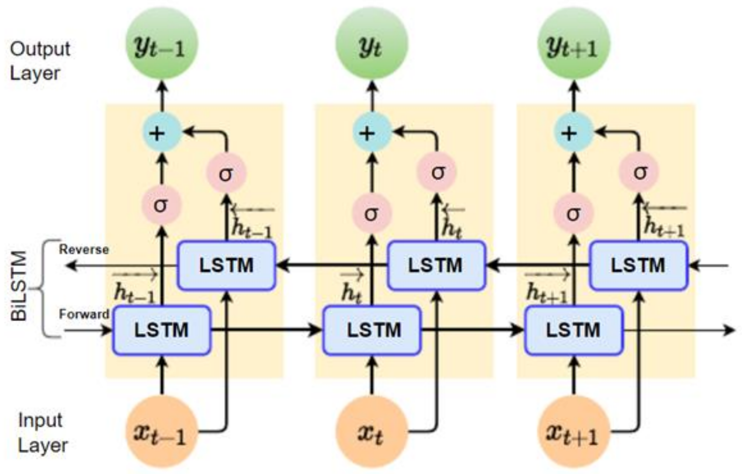


Fig. 3.9: Architecture of the BiLSTM network [13]

3.3.5 mBERT

Multilingual BERT (mBERT) is a BERT-based model trained on text from various languages, making it versatile for multilingual natural language processing (NLP). It learns shared language representations, enabling cross-lingual understanding and generation. With fine-tuning, it's adaptable to specific NLP tasks. Accessible through libraries like Hugging Face Transformers, mBERT supports a wide range of languages and is valuable for low-resource languages. Its universality simplifies NLP tasks in multilingual contexts.

3.4 Validation Criteria

Once a model is developed, it is very important to check the performance of the model. To measure the performance of a predictor, there are commonly used performance metrics such as confusion matrix. In classification problems, the primary source of performance measurements is confusion matrix.

3.4.1 Confusion Matrix

Confusion Matrix is a performance evaluation metric which provides a summary of the predictions made by a classification model, highlighting the correct and incorrect classifications across different classes. It is typically represented as a table with rows and columns corresponding to the predicted and actual classes, respectively. It helps in assessing the model's accuracy and identifying common types of errors.

Actual class

Predicted class		Positive	Negative
	Positive	TP	FP
	Negative	FN	TN

Fig. 3.10: Confusion Matrix

3.4.1.1 Accuracy

The overall accuracy of the model, calculated as

Accuracy = (TP + TN) / (TP + TN + FP + FN).

3.4.1.2 Precision

The ability of the model to correctly identify positive instances, calculated as Precision = TP / (TP + FP).

3.4.1.3 Recall

The proportion of actual positive instances correctly identified by the model, calculated as
Recall (Sensitivity or True Positive Rate) = TP / (TP + FN).

3.4.1.4 F1 Score

A combined metric that considers both precision and recall, calculated as

F1 Score = 2 * (Precision * Recall) / (Precision + Recall).

3.4.2 K-fold Cross Validation

K-fold cross-validation is a technique used for model evaluation and performance estimation in machine learning. It involves dividing the dataset into k equal-sized folds and iteratively training and testing the model k times. In each iteration, a different fold is used as the testing set while the remaining folds are combined as the training set. The model's performance is evaluated on each iteration, and the performance metrics are averaged to provide an overall estimate of the model's performance. K-fold cross-validation allows for better utilization of the data, reduces the risk of overfitting or underfitting, and provides insights into the model's generalization performance. Stratified k-fold cross-validation can be used to preserve the class distribution in each fold, especially for imbalanced datasets. Overall, k-fold cross-validation is a widely used technique for reliable model evaluation and selection

3.5 Coding-wise Methodology

The following figure shows the detailed coding-wise implementation of this research.

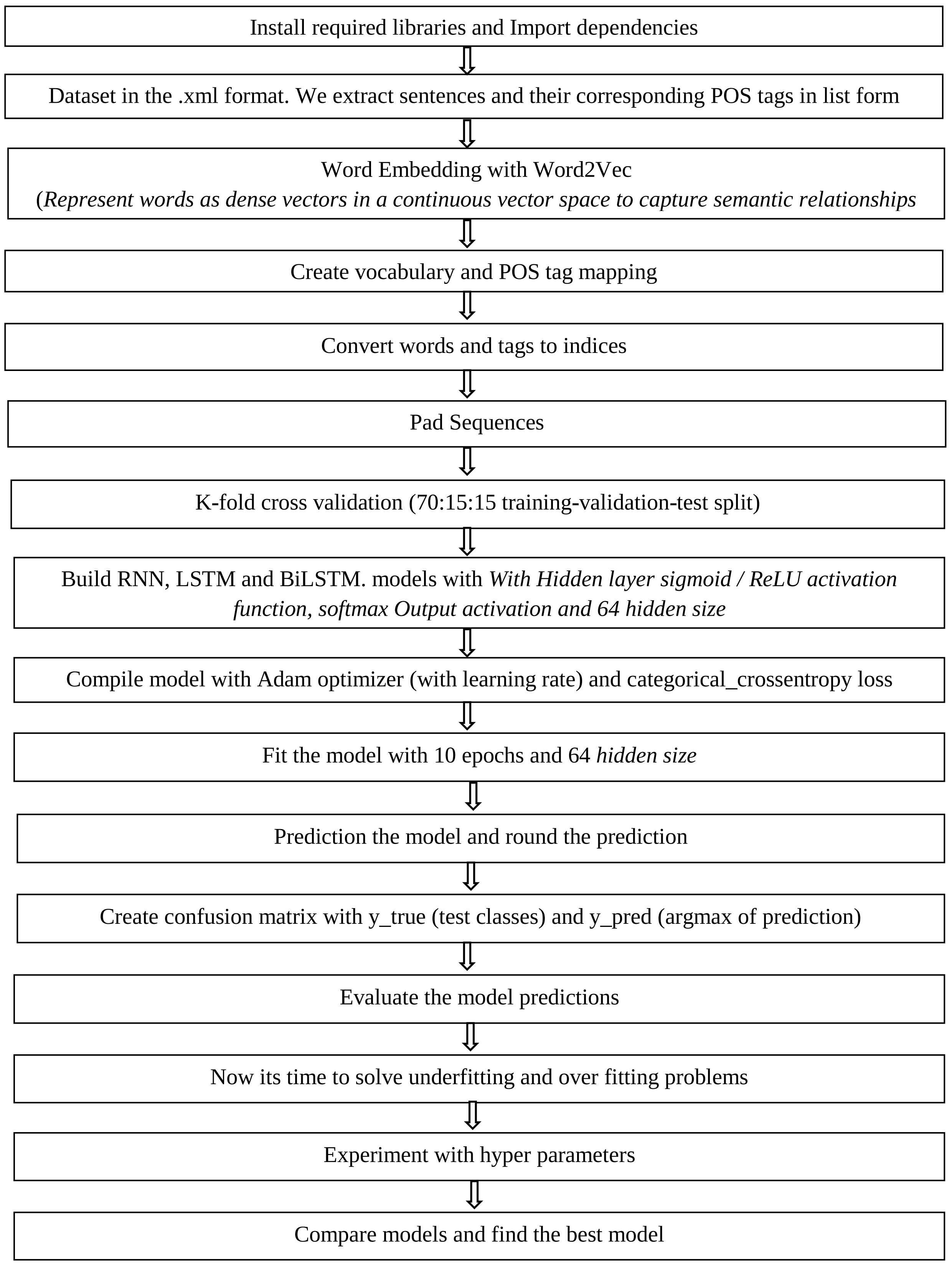

Install required libraries and Import dependencies
Dataset in the .xml format. We extract sentences and their corresponding POS tags in list form
Word Embedding with Word2Vec
(*Represent words as dense vectors in a continuous vector space to capture semantic relationships*
Create vocabulary and POS tag mapping
Convert words and tags to indices
Pad Sequences
K-fold cross validation (70:15:15 training-validation-test split)
Build RNN, LSTM and BiLSTM. models with *With Hidden layer sigmoid / ReLU activation function, softmax Output activation and 64 hidden size*
Compile model with Adam optimizer (with learning rate) and categorical_crossentropy loss
Fit the model with 10 epochs and 64 *hidden size*
Prediction the model and round the prediction
Create confusion matrix with y_true (test classes) and y_pred (argmax of prediction)
Evaluate the model predictions
Now its time to solve underfitting and over fitting problems
Experiment with hyper parameters
Compare models and find the best model

Fig. 3.11: Coding-wise Methodology

3.6 Tools and Environment

The proposed programming language for the entire project is Python. The following tools and libraries needs to be used:

a. Python: Python is a high-level, versatile programming language known for its simplicity and readability. It features a rich standard library, supports multiple programming paradigms, and is widely used in web development, data analysis, machine learning, and scientific computing. Python's extensive community and ecosystem make it a popular choice for a broad range of applications. The thesis is implemented on python 3.10.9 version.

b. Anaconda: Anaconda is an open-source platform and distribution for data science and machine learning that simplifies the management of Python and R libraries, environments, and dependencies. It provides a user-friendly interface for creating isolated environments and installing packages, making it a popular choice for data scientists and developers to streamline project setups and maintain version control of libraries. Anaconda also includes a package manager called conda, which facilitates package installation and environment management.

c. Tensorflow: TensorFlow is an open-source machine learning framework developed by Google that facilitates building, training, and deploying deep learning models. It offers a comprehensive ecosystem for various machine learning tasks, including neural networks, natural language processing, and computer vision. TensorFlow is known for its flexibility, scalability, and support for both research experimentation and production-level deployment.

d. Keras: Keras is a high-level deep learning framework for Python that simplifies the creation and training of neural networks. It offers an intuitive API for building complex models, supports multiple backends (e.g., TensorFlow), and is widely used for various machine learning tasks, including image recognition, natural language processing, and reinforcement learning. Its modularity, ease of use, and extensive community support make it a popular choice among researchers and practitioners.

e. NumPy: It is a fundamental Python library for numerical computations, providing support for multidimensional arrays and matrices along with a vast collection of mathematical functions. It is widely used in scientific computing, data analysis, and machine learning due to its efficiency and versatility. NumPy serves as a foundation for many other Python libraries and tools in the data science ecosystem.

f. Pandas: Pandas is a Python library for data manipulation and analysis, offering data structures like DataFrames and Series. It simplifies tasks like data cleaning, transformation, and exploration. Pandas is widely used in data science for handling and analyzing structured data efficiently.

g. Matplotlib: Matplotlib is a versatile Python library for creating static, animated, or interactive visualizations and plots. It provides a wide range of customizable options for creating charts, graphs, and other data visualizations. Matplotlib is commonly used in scientific computing, data analysis, and data presentation.

h. Seaborn: Seaborn is a Python data visualization library built on top of Matplotlib, designed to create informative and attractive statistical graphics. It simplifies the process of creating aesthetically pleasing visualizations for data analysis and exploration. Seaborn offers a high-level interface for creating various types of plots, including scatter plots, bar charts, and heatmaps, making it a valuable tool for data visualization in Python.

i. Scikit-learn (sklearn): It is a widely-used machine learning library in Python, known for its user-friendly API and comprehensive set of tools for classification, regression, clustering, dimensionality reduction, and more. In this thesis, sklearn used for splitting the dataset into training, validation and test sets by using train_test_split function. And plotting confusion matrix by using confusion_matrix function.

j. NLTK (Natural Language Toolkit): It is a Python library for natural language processing and text analysis. It provides a wide range of tools, resources, and corpora for tasks such as tokenization, stemming, part-of-speech tagging, sentiment analysis, and more. NLTK is widely used in research and education to work with textual data and develop NLP applications.

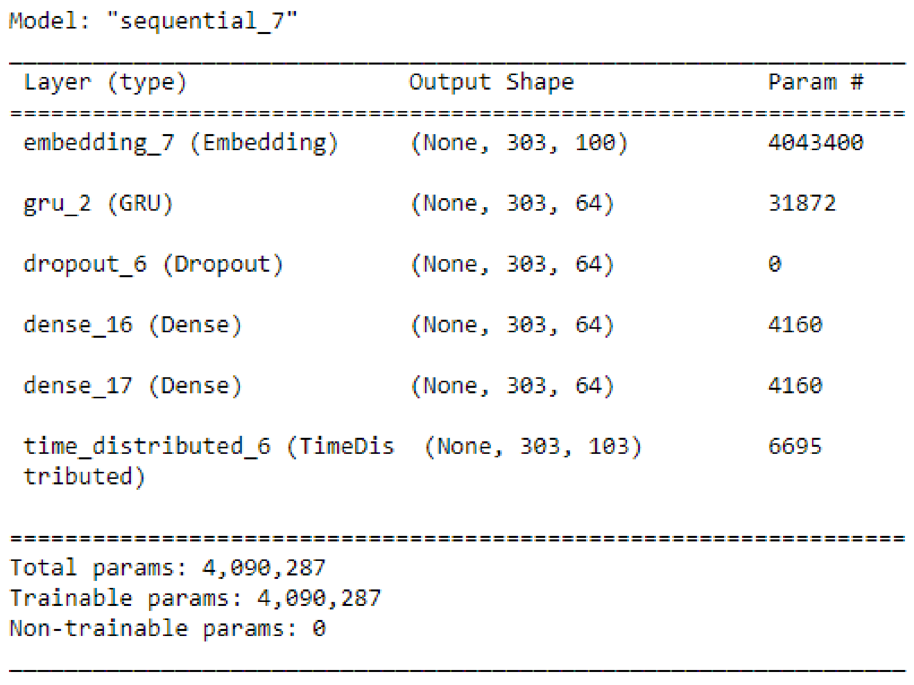
CHAPTER 4

RESULTS AND DISCUSSION

This chapter deals with analysis of results of 3 models GRU, LSTM and BiLSTM.

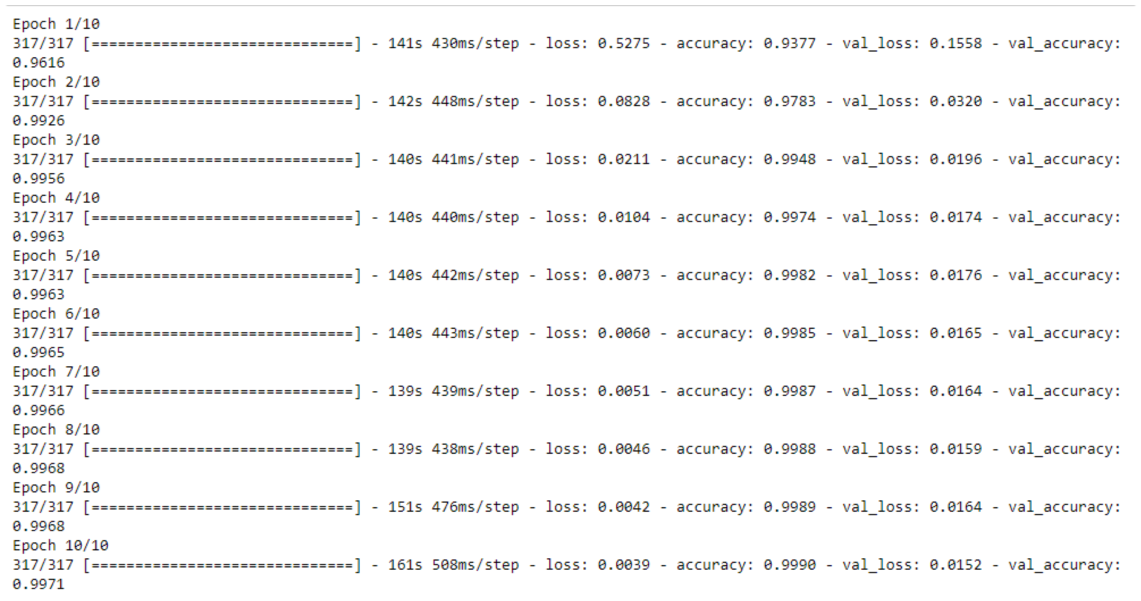
4.1 Experiment using GRU

GRU model with over 580k words, 10 epochs, 64 batch size, 0.3 dropout, Adam optimizer, 0.001 learning rate, sparse_categorical_crossentropy loss function gave 99.71 % F1-score and 99.69 % accuracy.



```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_7 (Embedding)     (None, 303, 100)          4043400

 gru_2 (GRU)                 (None, 303, 64)           31872

 dropout_6 (Dropout)         (None, 303, 64)           0

 dense_16 (Dense)            (None, 303, 64)           4160

 dense_17 (Dense)            (None, 303, 64)           4160

 time_distributed_6 (TimeDis  (None, 303, 103)         6695
 tributed)

=================================================================
Total params: 4,090,287
Trainable params: 4,090,287
Non-trainable params: 0
_________________________________________________________________
```

Fig. 4.1: Summary of GRU Model



```
Epoch 1/10
317/317 [==============================] - 141s 430ms/step - loss: 0.5275 - accuracy: 0.9377 - val_loss: 0.1558 - val_accuracy:
0.9616
Epoch 2/10
317/317 [==============================] - 142s 448ms/step - loss: 0.0828 - accuracy: 0.9783 - val_loss: 0.0320 - val_accuracy:
0.9926
Epoch 3/10
317/317 [==============================] - 140s 441ms/step - loss: 0.0211 - accuracy: 0.9948 - val_loss: 0.0196 - val_accuracy:
0.9956
Epoch 4/10
317/317 [==============================] - 140s 440ms/step - loss: 0.0104 - accuracy: 0.9974 - val_loss: 0.0174 - val_accuracy:
0.9963
Epoch 5/10
317/317 [==============================] - 140s 442ms/step - loss: 0.0073 - accuracy: 0.9982 - val_loss: 0.0176 - val_accuracy:
0.9963
Epoch 6/10
317/317 [==============================] - 140s 443ms/step - loss: 0.0060 - accuracy: 0.9985 - val_loss: 0.0165 - val_accuracy:
0.9965
Epoch 7/10
317/317 [==============================] - 139s 439ms/step - loss: 0.0051 - accuracy: 0.9987 - val_loss: 0.0164 - val_accuracy:
0.9966
Epoch 8/10
317/317 [==============================] - 139s 438ms/step - loss: 0.0046 - accuracy: 0.9988 - val_loss: 0.0159 - val_accuracy:
0.9968
Epoch 9/10
317/317 [==============================] - 151s 476ms/step - loss: 0.0042 - accuracy: 0.9989 - val_loss: 0.0164 - val_accuracy:
0.9968
Epoch 10/10
317/317 [==============================] - 161s 508ms/step - loss: 0.0039 - accuracy: 0.9990 - val_loss: 0.0152 - val_accuracy:
0.9971
```

Fig. 4.2: History of training GRU model

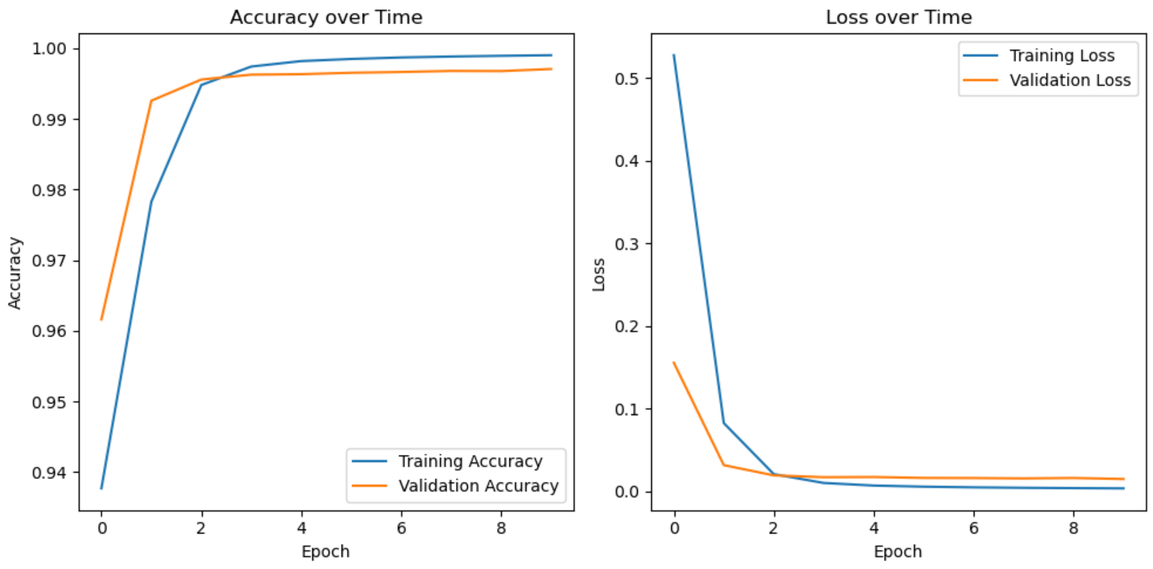
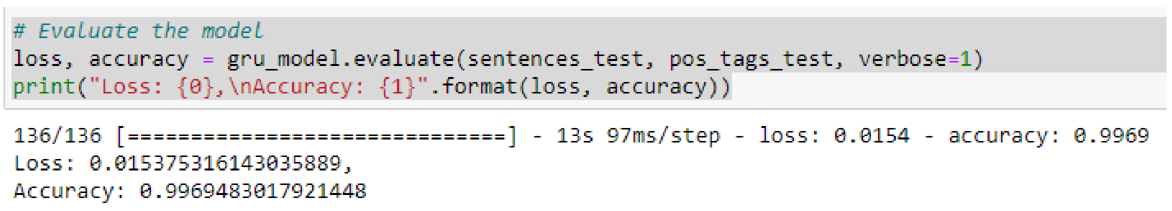


Fig. 4.3: Accuracy and Loss over time of GRU model



```
# Evaluate the model
loss, accuracy = gru_model.evaluate(sentences_test, pos_tags_test, verbose=1)
print("Loss: {0},\nAccuracy: {1}".format(loss, accuracy))
```

```
136/136 [==============================] - 13s 97ms/step - loss: 0.0154 - accuracy: 0.9969
Loss: 0.015375316143035889,
Accuracy: 0.9969483017921448
```

Fig. 4.4: Evaluate GRU model using test dataset

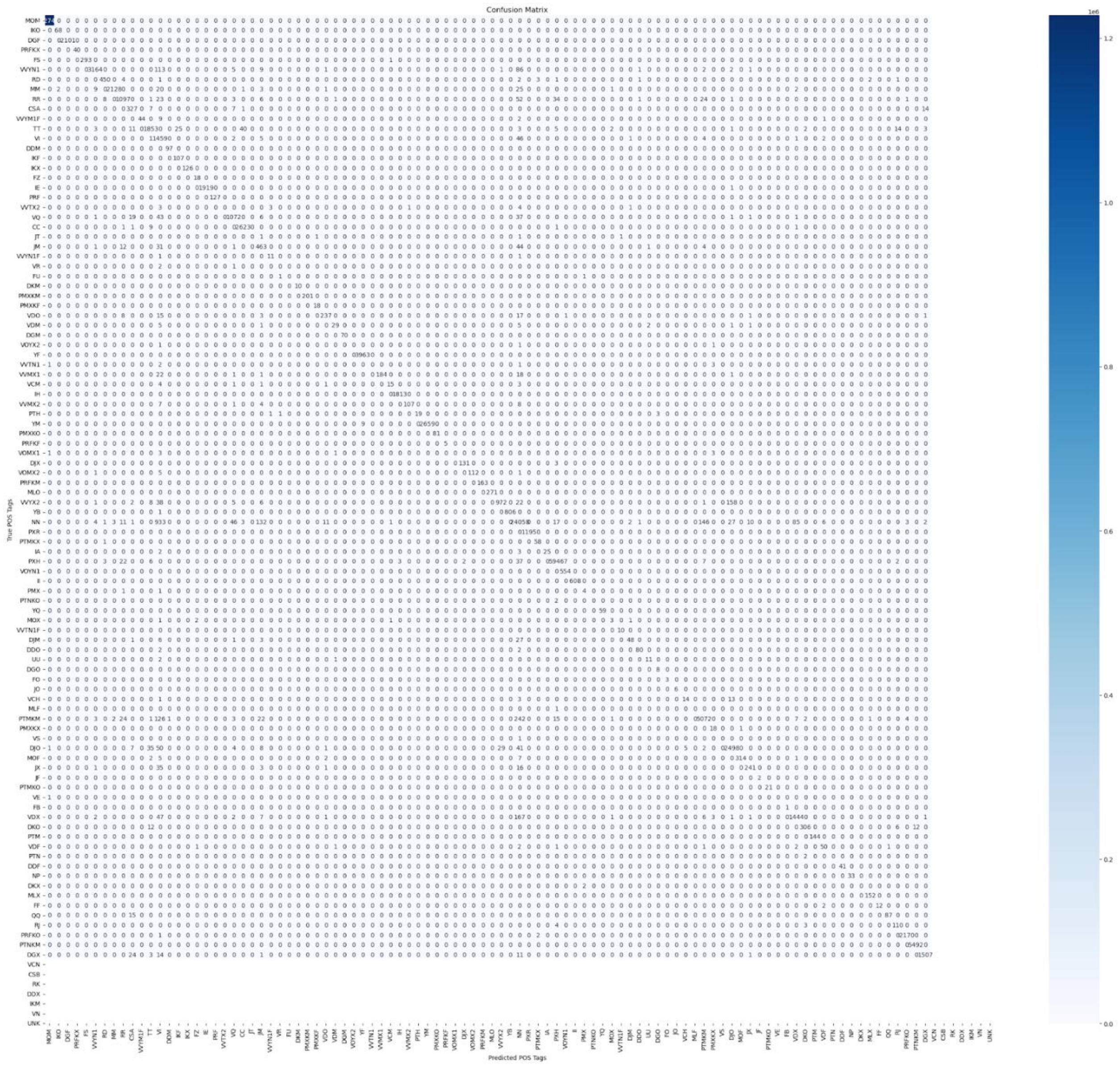
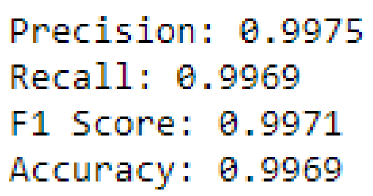
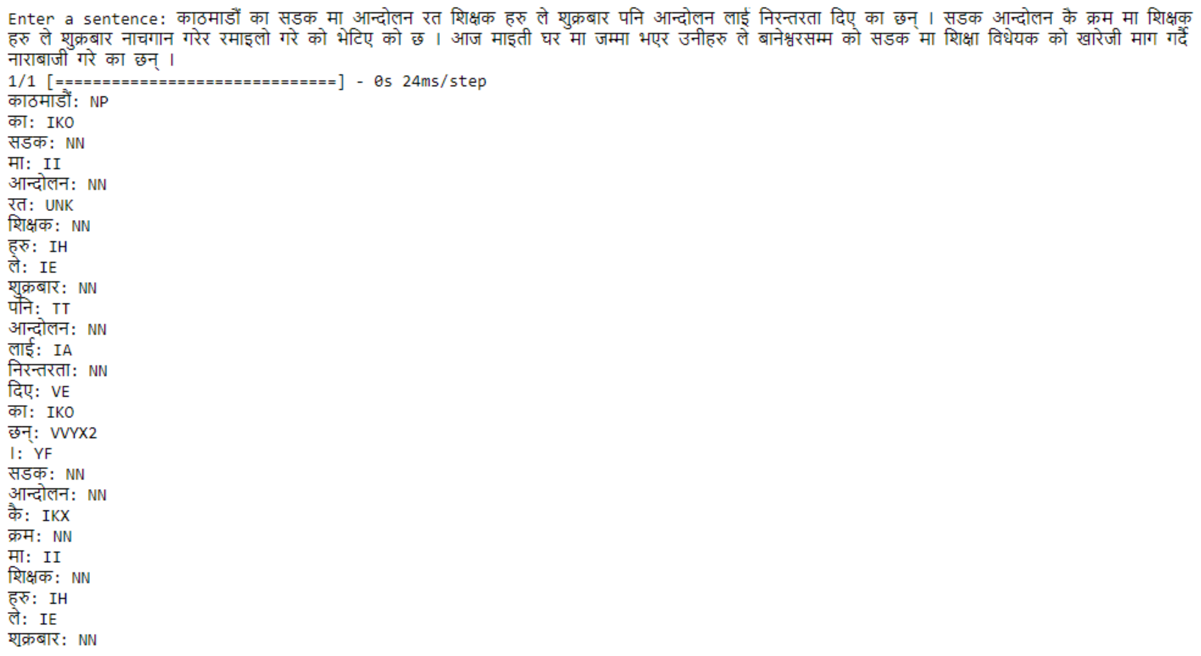


Fig. 4.5: Confusion matrix for GRU model



```
Precision: 0.9975
Recall: 0.9969
F1 Score: 0.9971
Accuracy: 0.9969
```

Fig. 4.6: Precision, Recall, F1 score and Accuracy

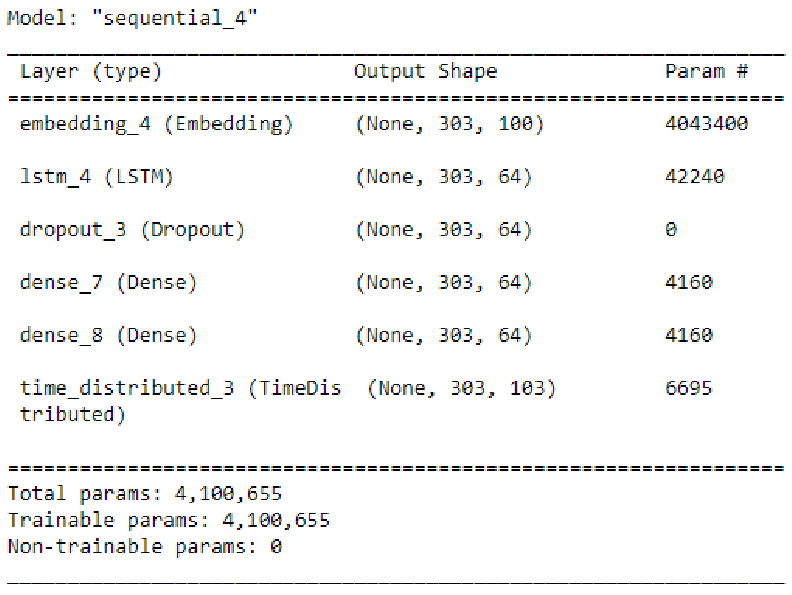


```
Enter a sentence: काठमाडौं का सडक मा आन्दोलन रत शिक्षक हरु ले शुक्रबार पनि आन्दोलन लाई निरन्तरता दिए का छन् । सडक आन्दोलन के क्रम मा शिक्षक
हरु ले शुक्रबार नाचगान गरेर रमाइलो गरे को भेटिए को छ । आज माइती घर मा जम्मा भएर उनीहरु ले बानेश्वरसम्म को सडक मा शिक्षा विधेयक को खारेजी माग गर्दै
नाराबाजी गरे का छन् ।
1/1 [==============================] - 0s 24ms/step
काठमाडौं: NP
का: IKO
सडक: NN
मा: II
आन्दोलन: NN
रत: UNK
शिक्षक: NN
हरु: IH
ले: IE
शुक्रबार: NN
पनि: TT
आन्दोलन: NN
लाई: IA
निरन्तरता: NN
दिए: VE
का: IKO
छन्: VVYX2
।: YF
सडक: NN
आन्दोलन: NN
के: IKX
क्रम: NN
मा: II
शिक्षक: NN
हरु: IH
ले: IE
शुक्रबार: NN
```

Fig. 4.7: Output of user input sentences using GRU

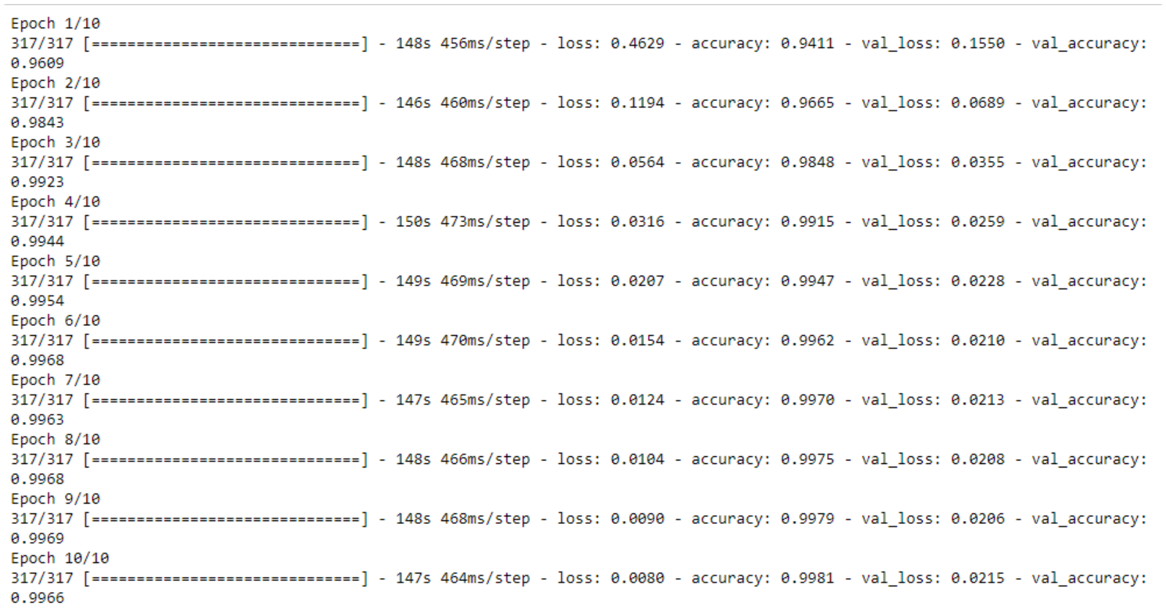
4.2 Experiment using LSTM

LSTM model with over 580k words, 10 epochs, 64 batch size, 0.3 dropout, Adam optimizer, 0.001 learning rate, sparse_categorical_crossentropy loss function gave 99.65 % F1-score and 99.64 % accuracy.



```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 303, 100)          4043400

 lstm_4 (LSTM)               (None, 303, 64)           42240

 dropout_3 (Dropout)         (None, 303, 64)           0

 dense_7 (Dense)             (None, 303, 64)           4160

 dense_8 (Dense)             (None, 303, 64)           4160

 time_distributed_3 (TimeDis  (None, 303, 103)         6695
 tributed)

=================================================================
Total params: 4,100,655
Trainable params: 4,100,655
Non-trainable params: 0
_________________________________________________________________
```

Fig. 4.8: Summary of LSTM Model



```
Epoch 1/10
317/317 [==============================] - 148s 456ms/step - loss: 0.4629 - accuracy: 0.9411 - val_loss: 0.1550 - val_accuracy:
0.9609
Epoch 2/10
317/317 [==============================] - 146s 460ms/step - loss: 0.1194 - accuracy: 0.9665 - val_loss: 0.0689 - val_accuracy:
0.9843
Epoch 3/10
317/317 [==============================] - 148s 468ms/step - loss: 0.0564 - accuracy: 0.9848 - val_loss: 0.0355 - val_accuracy:
0.9923
Epoch 4/10
317/317 [==============================] - 150s 473ms/step - loss: 0.0316 - accuracy: 0.9915 - val_loss: 0.0259 - val_accuracy:
0.9944
Epoch 5/10
317/317 [==============================] - 149s 469ms/step - loss: 0.0207 - accuracy: 0.9947 - val_loss: 0.0228 - val_accuracy:
0.9954
Epoch 6/10
317/317 [==============================] - 149s 470ms/step - loss: 0.0154 - accuracy: 0.9962 - val_loss: 0.0210 - val_accuracy:
0.9968
Epoch 7/10
317/317 [==============================] - 147s 465ms/step - loss: 0.0124 - accuracy: 0.9970 - val_loss: 0.0213 - val_accuracy:
0.9963
Epoch 8/10
317/317 [==============================] - 148s 466ms/step - loss: 0.0104 - accuracy: 0.9975 - val_loss: 0.0208 - val_accuracy:
0.9968
Epoch 9/10
317/317 [==============================] - 148s 468ms/step - loss: 0.0090 - accuracy: 0.9979 - val_loss: 0.0206 - val_accuracy:
0.9969
Epoch 10/10
317/317 [==============================] - 147s 464ms/step - loss: 0.0080 - accuracy: 0.9981 - val_loss: 0.0215 - val_accuracy:
0.9966
```

Fig. 4.9: History of training LSTM model

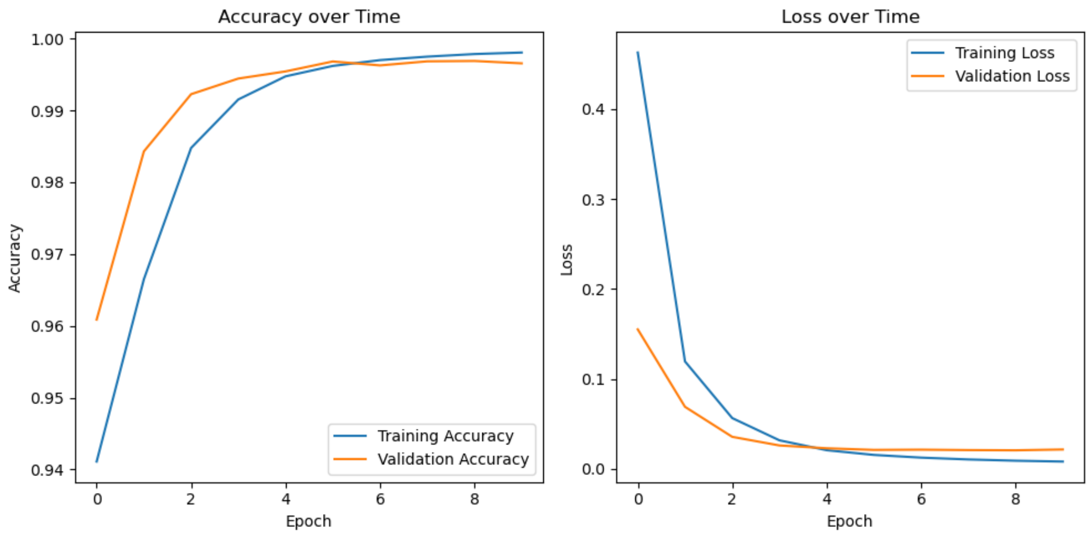
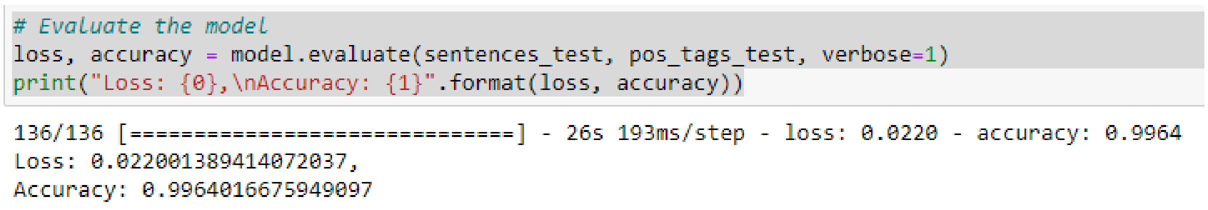


Fig. 4.10: Accuracy and Loss over time of LSTM model



```
# Evaluate the model
loss, accuracy = model.evaluate(sentences_test, pos_tags_test, verbose=1)
print("Loss: {0},\nAccuracy: {1}".format(loss, accuracy))
```

```
136/136 [==============================] - 26s 193ms/step - loss: 0.0220 - accuracy: 0.9964
Loss: 0.022001389414072037,
Accuracy: 0.9964016675949097
```

Fig. 4.11: Evaluate LSTM model using test dataset

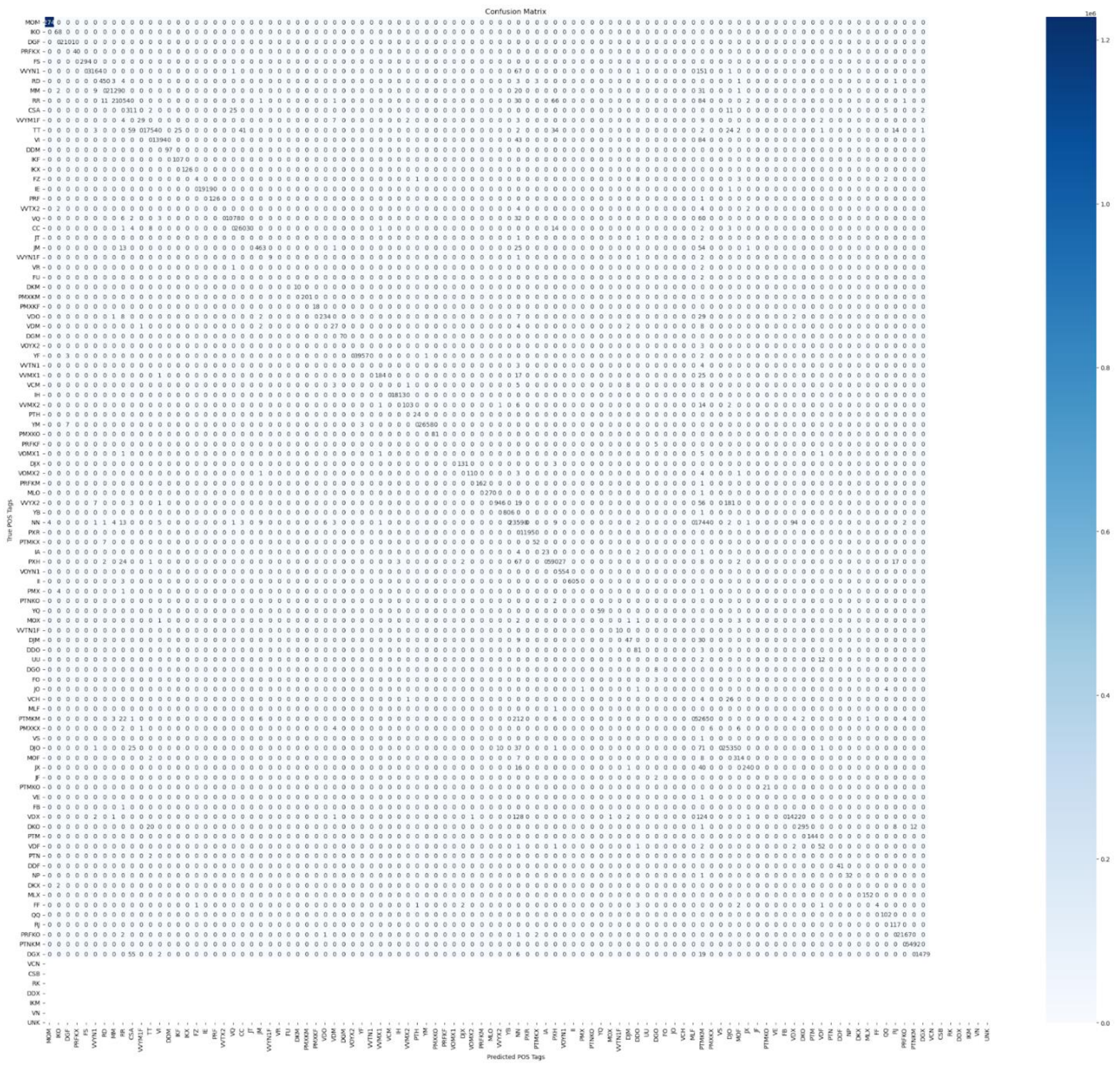
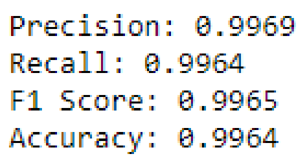
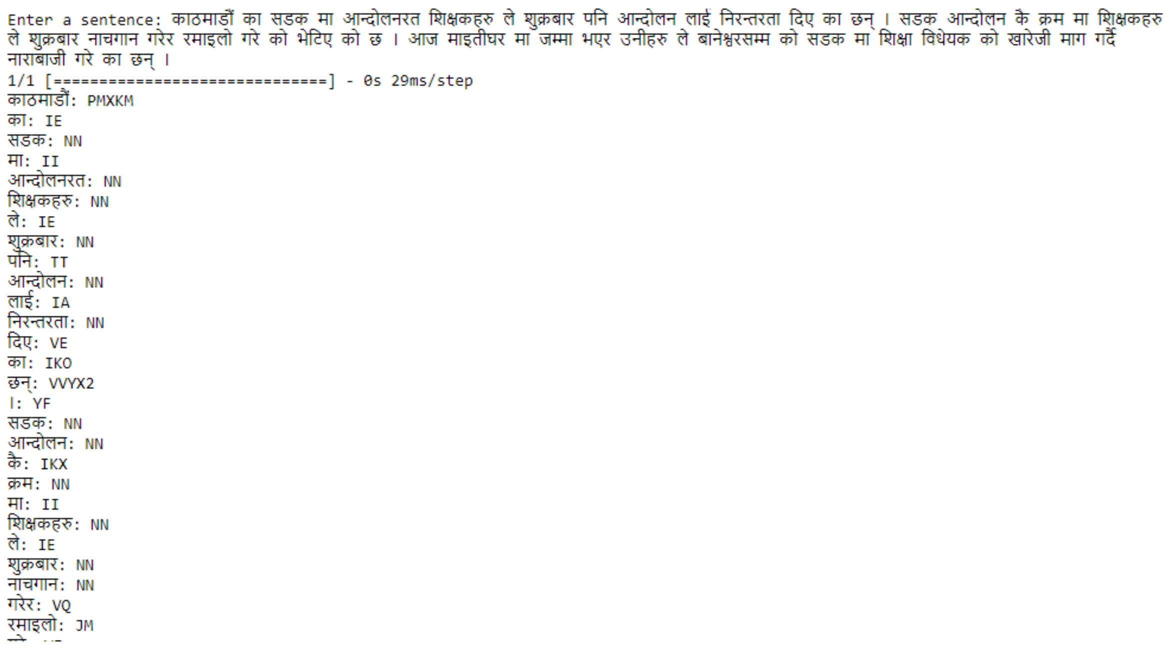


Fig. 4.12: Confusion matrix for LSTM model



```
Precision: 0.9969
Recall: 0.9964
F1 Score: 0.9965
Accuracy: 0.9964
```

Fig. 4.13: Precision, Recall, F1 score and Accuracy

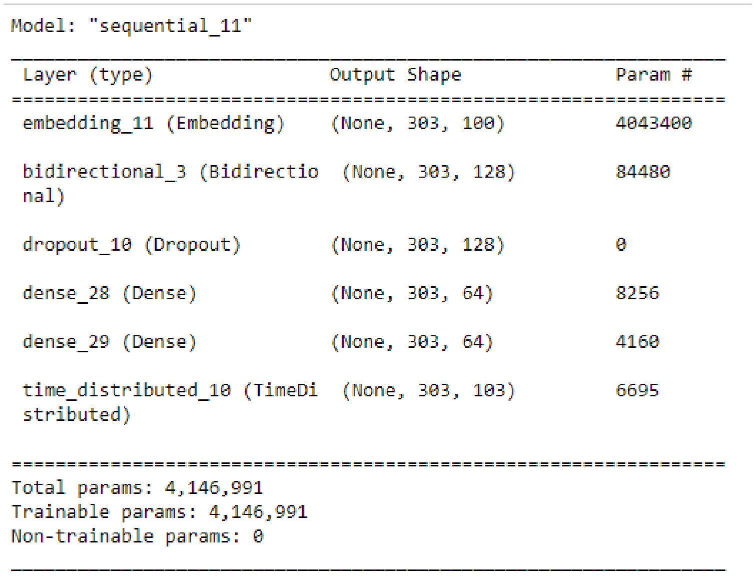


```
Enter a sentence: काठमाडौं का सडक मा आन्दोलनरत शिक्षकहरु ले शुक्रबार पनि आन्दोलन लाई निरन्तरता दिए का छन् । सडक आन्दोलन कै क्रम मा शिक्षकहरु
ले शुक्रबार नाचगान गरेर रमाइलो गरे को भेटिए को छ । आज माइतीघर मा जम्मा भएर उनीहरु ले बानेश्वरसम्म को सडक मा शिक्षा विधेयक को खारेजी माग गर्दै
नाराबाजी गरे का छन् ।
1/1 [==============================] - 0s 29ms/step
काठमाडौं: PMXKM
का: IE
सडक: NN
मा: II
आन्दोलनरत: NN
शिक्षकहरु: NN
ले: IE
शुक्रबार: NN
पनि: TT
आन्दोलन: NN
लाई: IA
निरन्तरता: NN
दिए: VE
का: IKO
छन्: VVYX2
।: YF
सडक: NN
आन्दोलन: NN
कै: IKX
क्रम: NN
मा: II
शिक्षकहरु: NN
ले: IE
शुक्रबार: NN
नाचगान: NN
गरेर: VQ
रमाइलो: JM
```

Fig. 4.14: Output of user input sentences using LSTM model

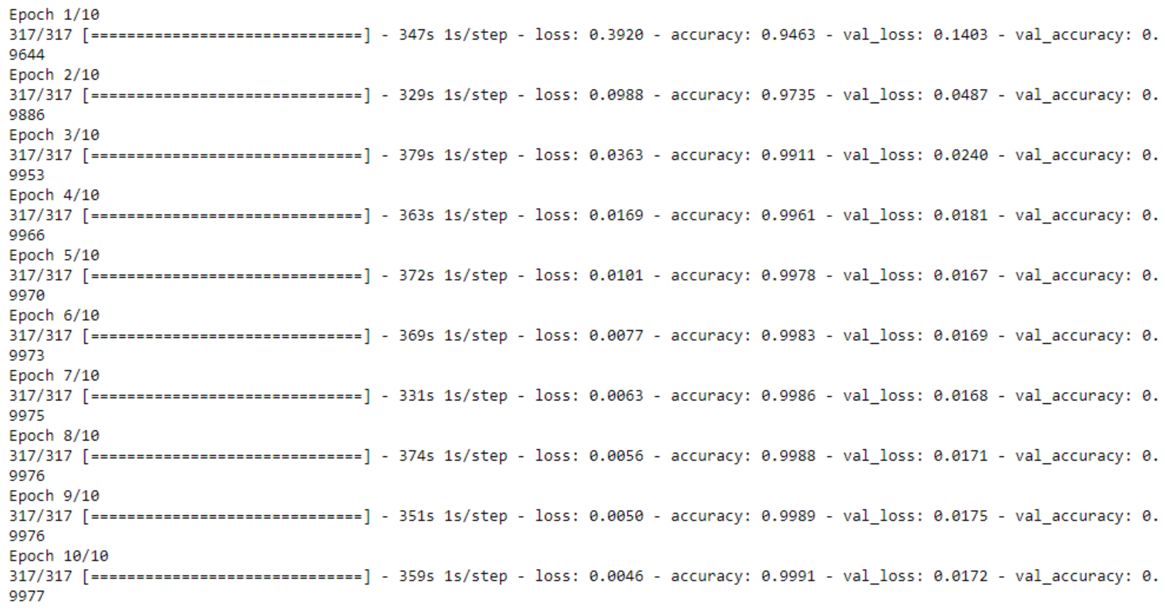
4.3 Experiment using BiLSTM

BiLSTM model with over 580k words, 10 epochs, 64 batch size, 0.3 dropout, Adam optimizer, 0.001 learning rate, sparse_categorical_crossentropy loss function gave 99.76 % F1-score and 99.77 % accuracy.



```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_11 (Embedding)    (None, 303, 100)          4043400

 bidirectional_3 (Bidirectio  (None, 303, 128)         84480
 nal)

 dropout_10 (Dropout)        (None, 303, 128)          0

 dense_28 (Dense)            (None, 303, 64)           8256

 dense_29 (Dense)            (None, 303, 64)           4160

 time_distributed_10 (TimeDi  (None, 303, 103)         6695
 stributed)

=================================================================
Total params: 4,146,991
Trainable params: 4,146,991
Non-trainable params: 0
_________________________________________________________________
```

Fig. 4.15: Summary of BiLSTM Model



```
Epoch 1/10
317/317 [==============================] - 347s 1s/step - loss: 0.3920 - accuracy: 0.9463 - val_loss: 0.1403 - val_accuracy: 0.
9644
Epoch 2/10
317/317 [==============================] - 329s 1s/step - loss: 0.0988 - accuracy: 0.9735 - val_loss: 0.0487 - val_accuracy: 0.
9886
Epoch 3/10
317/317 [==============================] - 379s 1s/step - loss: 0.0363 - accuracy: 0.9911 - val_loss: 0.0240 - val_accuracy: 0.
9953
Epoch 4/10
317/317 [==============================] - 363s 1s/step - loss: 0.0169 - accuracy: 0.9961 - val_loss: 0.0181 - val_accuracy: 0.
9966
Epoch 5/10
317/317 [==============================] - 372s 1s/step - loss: 0.0101 - accuracy: 0.9978 - val_loss: 0.0167 - val_accuracy: 0.
9970
Epoch 6/10
317/317 [==============================] - 369s 1s/step - loss: 0.0077 - accuracy: 0.9983 - val_loss: 0.0169 - val_accuracy: 0.
9973
Epoch 7/10
317/317 [==============================] - 331s 1s/step - loss: 0.0063 - accuracy: 0.9986 - val_loss: 0.0168 - val_accuracy: 0.
9975
Epoch 8/10
317/317 [==============================] - 374s 1s/step - loss: 0.0056 - accuracy: 0.9988 - val_loss: 0.0171 - val_accuracy: 0.
9976
Epoch 9/10
317/317 [==============================] - 351s 1s/step - loss: 0.0050 - accuracy: 0.9989 - val_loss: 0.0175 - val_accuracy: 0.
9976
Epoch 10/10
317/317 [==============================] - 359s 1s/step - loss: 0.0046 - accuracy: 0.9991 - val_loss: 0.0172 - val_accuracy: 0.
9977
```

Fig. 4.16: History of training BiLSTM model

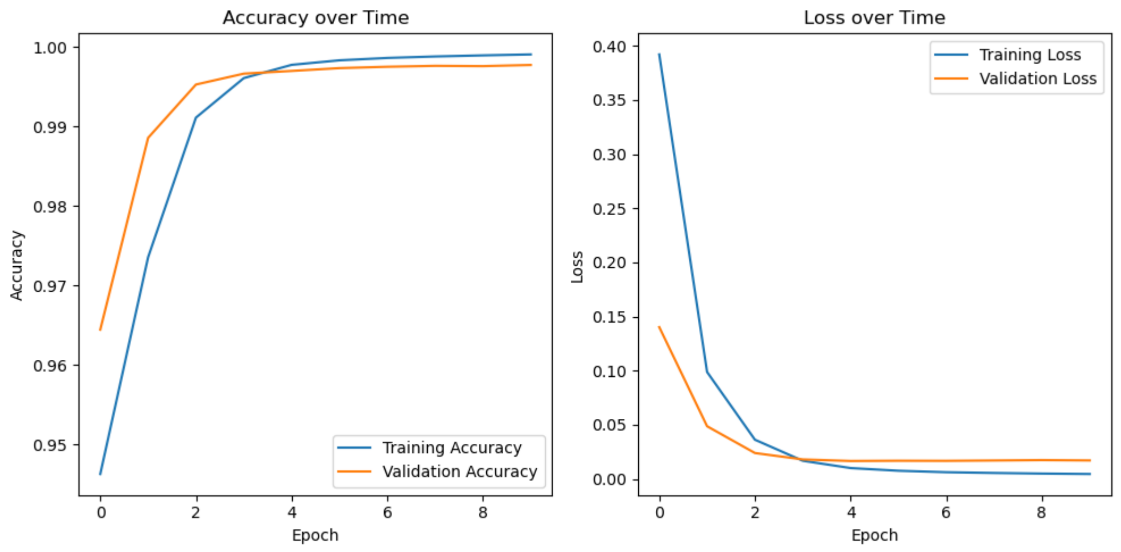
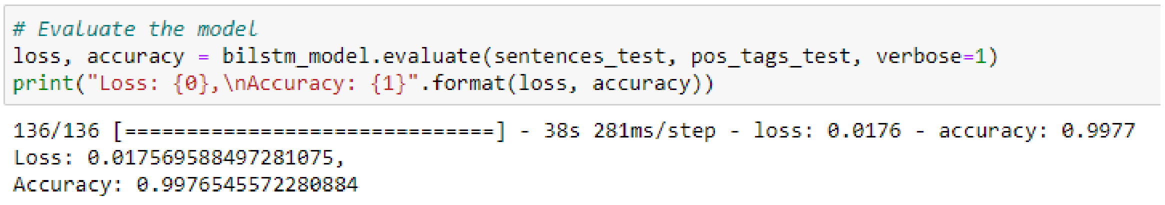


Fig. 4.17: Accuracy and Loss over time of BiLSTM model



```
# Evaluate the model
loss, accuracy = bilstm_model.evaluate(sentences_test, pos_tags_test, verbose=1)
print("Loss: {0},\nAccuracy: {1}".format(loss, accuracy))
```

```
136/136 [==============================] - 38s 281ms/step - loss: 0.0176 - accuracy: 0.9977
Loss: 0.017569588497281075,
Accuracy: 0.9976545572280884
```

Fig. 4.18: Evaluate BiLSTM model using test dataset

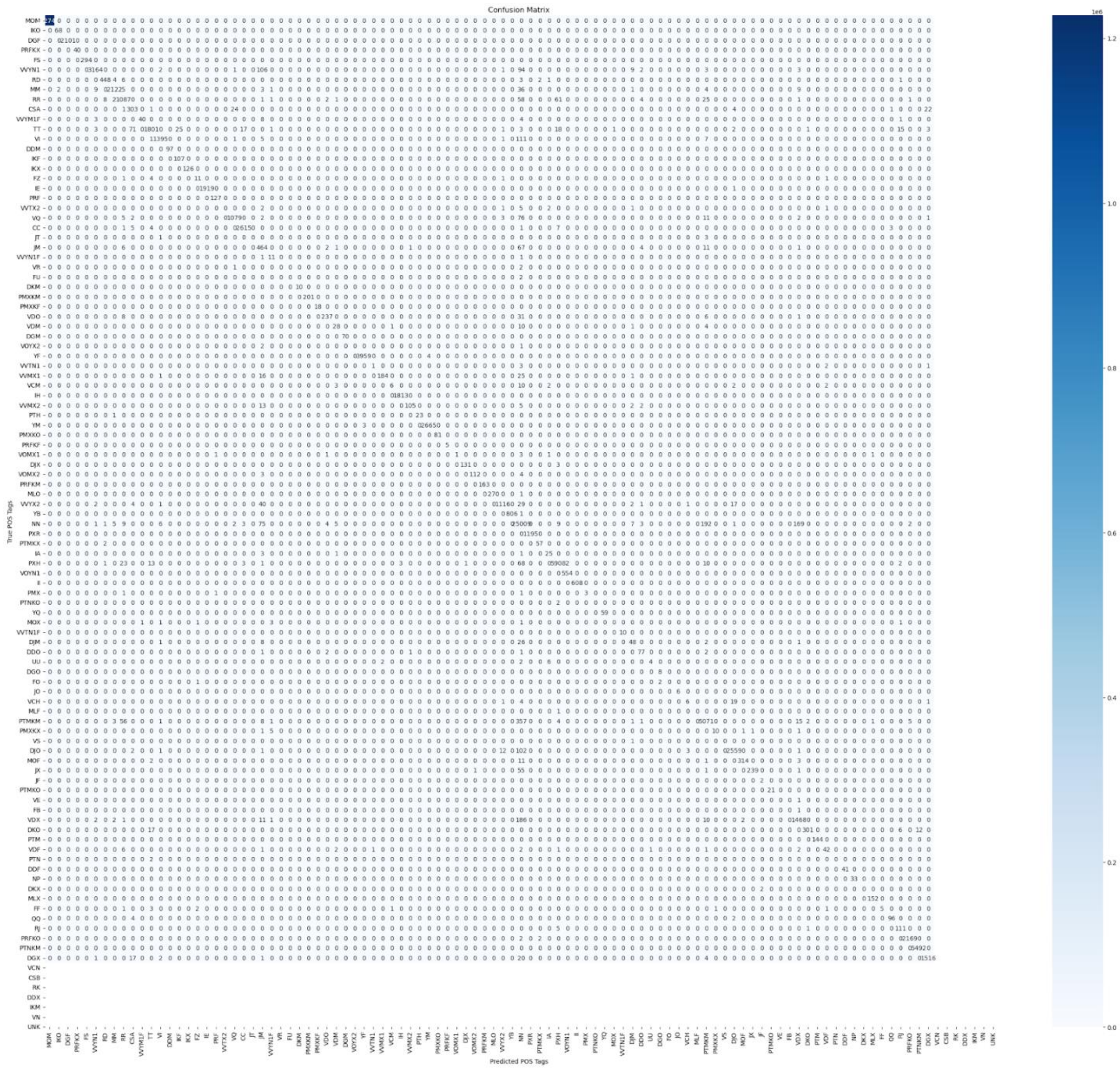
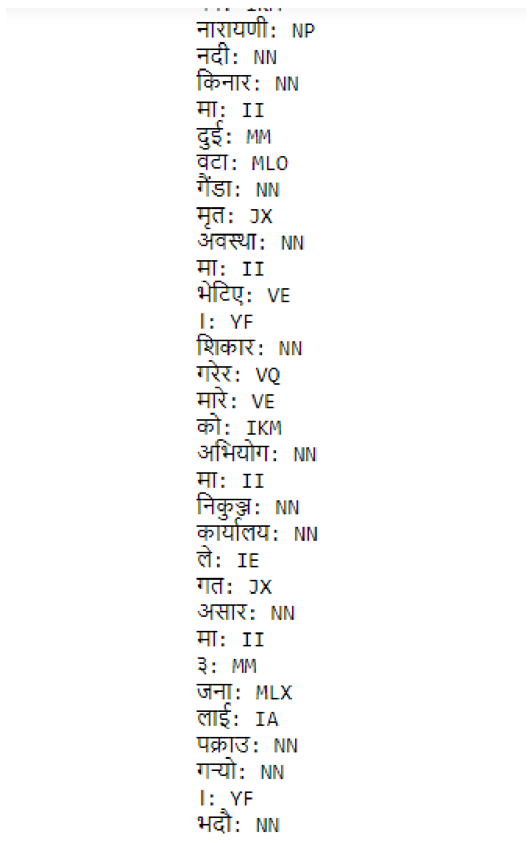


Fig. 4.19: Confusion matrix for BiLSTM model

```
Precision: 0.9977
Recall: 0.9977
F1 Score: 0.9976
Accuracy: 0.9977
```

Fig. 4.20: Precision, Recall, F1 score and Accuracy



```
नारायणी: NP
नदी: NN
किनार: NN
मा: II
दुई: MM
वटा: MLO
गैंडा: NN
मृत: JX
अवस्था: NN
मा: II
भेटिए: VE
।: YF
शिकार: NN
गरेर: VQ
मारे: VE
को: IKM
अभियोग: NN
मा: II
निकुञ्ज: NN
कार्यालय: NN
ले: IE
गत: JX
असार: NN
मा: II
३: MM
जना: MLX
लाई: IA
पक्राउ: NN
गऱ्यो: NN
।: YF
भदौ: NN
```

Fig. 4.21: Output of user input sentences using BiLSTM model

4.4 Comparing Results between GRU, LSTM, BiLSTM

There are two tables which show the performance of the three models.

Models	**Loss**	**Accuracy**	**Val_loss**	**Val_accuracy**	**Evaluation Loss**	**Evaluation Accuracy**
LSTM	0.0080	0.9981	0.0215	0.9966	0.0220	0.9964
GRU	**0.0039**	0.9990	**0.0152**	0.9971	**0.0153**	0.9969
BiLSTM	0.0046	**0.9991**	0.0172	**0.9977**	0.0175	**0.9976**

Table 4.1: *Comparison between 3 Deep Learning Models based on Loss, Accuracy, Val_loss, Val_accuracy*

Models	**Precision**	**Recall**	**F1 Score**	**Accuracy**
LSTM	0.9969	0.9964	0.9965	0.9964
GRU	0.9975	0.9969	0.9971	0.9969
BiLSTM	**0.9977**	**0.9977**	**0.9976**	**0.9977**

Table 4.2: *Comparison between 3 Deep Learning Models based on Precision, Recall, F1 score and Accuracy*

Models	**Loss**	**Accurcy**
LSTM	0.0220	0.9964
GRU	**0.0153**	0.9969
BiLSTM	0.0175	**0.9976**

Table 4.3: *Comparison between 3 Deep Learning Models based on Testing Dataset*

According to above tables, BiLSTM achieved the maximum performance scores as compare to other models. Other models also performed good score. There is very small difference in the scores of models.

CHAPTER 5

CONCLUSION AND TASK TO BE COMPLETED

4.1 Conclusion

In this work, we applied three deep learning models for fine-grain POS tagging for the Nepali language. We showed that deep learning models could capture fine-grained morphological features like gender, person, number, and honorifics that are encoded within words in highly inflectional languages like Nepali. POS Tagged Nepali National Corpus (NNC) with over 14 million words and NELRALEC tag sets with 112 fine-grained POS Tags was used for modelling the POS tagger. BiLSTM model achieved best F1 score with sparse categorical cross entropy as the loss function. It is also able to tag unknown words most of the cases too. This research also showed that If we extract xml files like we mentioned in above preprocessing topic, it reduces model training time.

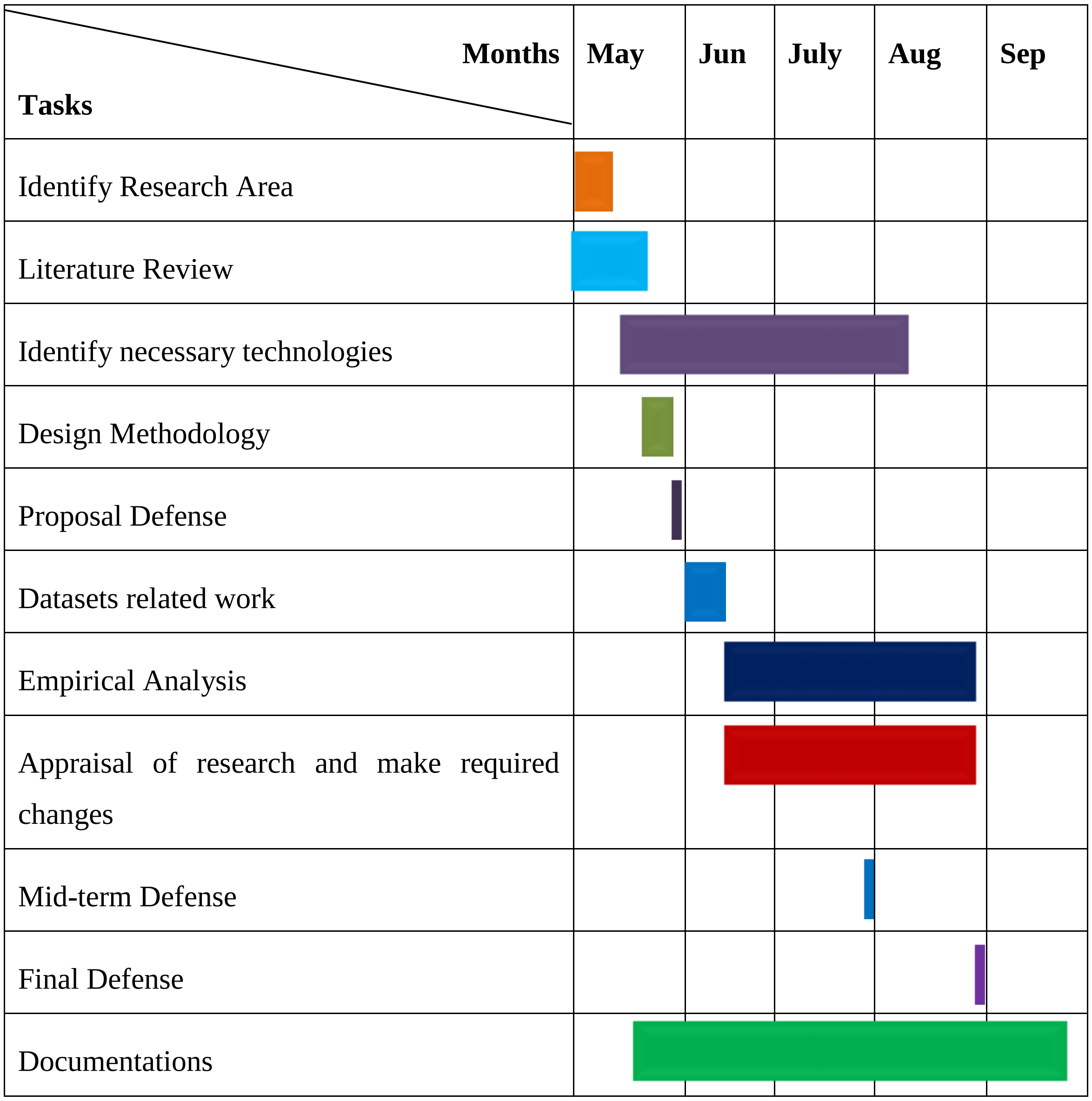
4.2 Task to be Completed

The following tasks need to be carried for completion of work:

a. Train three models with all dataset.
b. Fine tune pre-trained mBERT model with this dataset
c. Calculate the performance metrics.

APPENDIX A: GANTT CHART

The project will be five months long and the works are divided accordingly. The planned schedule for the project are illustrated in Gantt Chart below:



Tasks / Months	May	Jun	July	Aug	Sep
Identify Research Area	■				
Literature Review	■				
Identify necessary technologies	■	■	■	■	
Design Methodology	■				
Proposal Defense	■				
Datasets related work		■			
Empirical Analysis		■	■	■	
Appraisal of research and make required changes		■	■	■	
Mid-term Defense			■		
Final Defense				■	
Documentations	■	■	■	■	■

REFERENCES

[1] I. Shrestha, S. S. Dhakal, "Fine-grained part-of-speech tagging in Nepali text," *Procedia Computer Science,* vol. 189, PP 300-311, 2021

[2] A. Pradhan, A. Yajnik, "Probabilistic and Neural Network Based POS Tagging of Ambiguous Nepali text: A Comparative Study," *ISEEIE 2021: 2021 International Symposium on Electrical, Electronics and Information Engineering,* PP. 249–253, feb 2021

[3] A. Yajnik, "Part of Speech Tagging Using Statistical Approach for Nepali Text," *International Scholarly and Scientific Research & Innovation,* vol. 11, no. 1, 2017

[4] S. Sayami, T. B. Shahi and S. Shakya, "Nepali POS Tagging using Deep Learning Approaches," *NU. International Journal of Science*, Dec 2019.

[5] A. Paul, B. S. Purkayastha, S. Sakar, "Hidden Markov Model Based Part of Speech Tagging for Nepali Language," *International Symposium on Advanced Computing and Communication (lSACC),* Sep 2015.

[6] A. Yajnik, "GENERAL REGRESSION NEURAL NETWORK BASED POS TAGGING FOR NEPALI TEXT," 4th *International Conference on Natural Language*, pp. 35–40, 2018.

[7] Tej Bahadur Shahi, Tank Nath Dhamala, Bikash Balami, "Support Vector Machines based Part of Speech Tagging for Nepali Text," *International Journal of Computer Applications* (0975 – 8887), Volume 70– No.24, May 2013.

[8] A. Yajnik, "ANN Based POS Tagging For Nepali Text," *International Journal on Natural Language Computing (IJNLC),* Vol.7, No.3, June 2018.

[9] Greeshma Prabha, P. V. Jyothsna, K. K. Shahina, B. Premjith, K. P. Soman, "A Deep Learning Approach for Part-of-Speech Tagging in Nepali Language," *International Conference on Advances in Computing, Communications and Informatics (ICACCI),* Sep 2018.

[10] M. Jayaweera, N. G. J Dias, “HIDDEN MARKOV MODEL BASED PART OF SPEECH TAGGER FOR SINHALA LANGUAGE,” *International Journal on Natural Language Computing (IJNLC),* Vol. 3, No.3, June 2014.

[11] P. Sinha, N. M. Veyie, B. S. Purkayastha, “Enhancing the Performance of Part of Speech tagging of Nepali language through Hybrid approach,” *International Journal of Emerging Technology and Advanced Engineering,* Vol. 5, Issue 5, May 2015.

[12] “Gated Recurrent Unit,” commons.wikimedia.org.
https://commons.wikimedia.org/wiki/File:Gated_Recurrent_Unit,_type_1.svg (accessed Sep 23, 2023)

[13] B. Bohara, Raymond I. Fernandez, V. Gollapudi, Xingpeng Li, “Short-Term Aggregated Residential LoadForecasting using BiLSTM and CNN-BiLSTM,” *International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, 2022.