NAME - YUG GARG
ENTRY NO.- 2019EE30607

ELL 881

Assignment - 2

Methodology

1) Data Preprocessing

Before Data Preprocessing, we would be fetching data from the text files and convert it into arrays for our model. The following code is used for this.

```
def read_file(filename):
    with open(filename, "r") as file:
        text = file.readlines()
    return text

def process_text(text):
    X = []
    Y = []
    sentenceX = []
    sentenceY = []
    for line in text:
        split = line.split(" ")
        if len(split) > 1:
            sentenceX.append(split[0])
            sentenceY.append(split[1].replace("\n", ""))
        else:
            X.append(sentenceX)
            Y.append(sentenceY)
            sentenceX = []
            sentenceY = []
    return X, Y
text = read_file("/kaggle/input/ner-a2/train.txt")
X, Y = process_text(text)
```

In order to make our operations easily, we have made the following dataloader class to fetch the sentences in the form of a tuple i.e of the form (word,label).

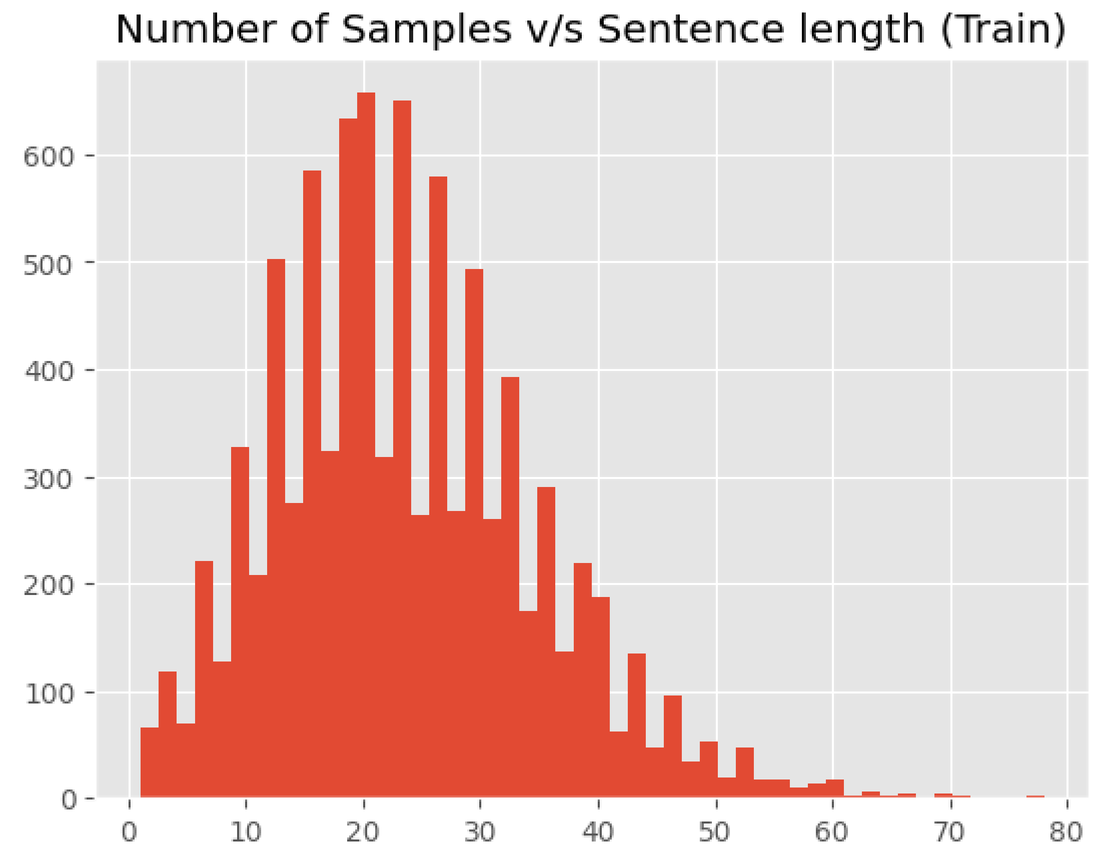
```
class SentenceGetter(object):
    def __init__(self, X,Y):
        self.n_sent = 1
        self.empty = False
        self.sentences = []
        for i in range(len(X)):
            sentence = []
            for word,tag in zip(X[i],Y[i]):
                sentence.append((word,tag))
            (self.sentences).append(sentence)

    def get_next(self):
        try:
            s = self.grouped["Sentence: {}".format(self.n_sent)]
            self.n_sent += 1
            return s
        except:
            return None
```

Now, a dictionary corresponding to all the unique labels and words is made in which each label/words is assigned with some id for the tokenization of the sentences. Special tokens like "UNK" for unknown words,"PAD" for padding token are used.In one of the experiments a third token named " NUM" is used for labelling numbers. Following code is used for the same.

```
unique = {}
for i in range(len(Y)):
    for j in range(len(Y[i])):
        if Y[i][j] not in unique:
            unique[Y[i][j]] = len(unique)
words = list(np.concatenate(X))
words = list(set(words))
tags = list(set(unique.keys()))
tags.append('PAD')
word2idx = {w: i+2 for i, w in enumerate(words)}
word2idx['PAD'] = 0
word2idx['UNK'] = 1
# word2idx["NUM"] = 2
idx2word = {w : i for i, w in word2idx.items()}
words.append("PAD")
words.append("UNK")
tag2idx = {t: i+1 for i, t in enumerate(tags)}
tag2idx['PAD'] = 0
idx2tag = {w : i for i, w in tag2idx.items()}
```

We have plotted a histogram to find out the distrubution of the length of sentences.
From the below distribution we can say that the max length of the sentences is around 80(which come

out to be 78 after finding from the dataset) and most of the sentences have length less than 40.



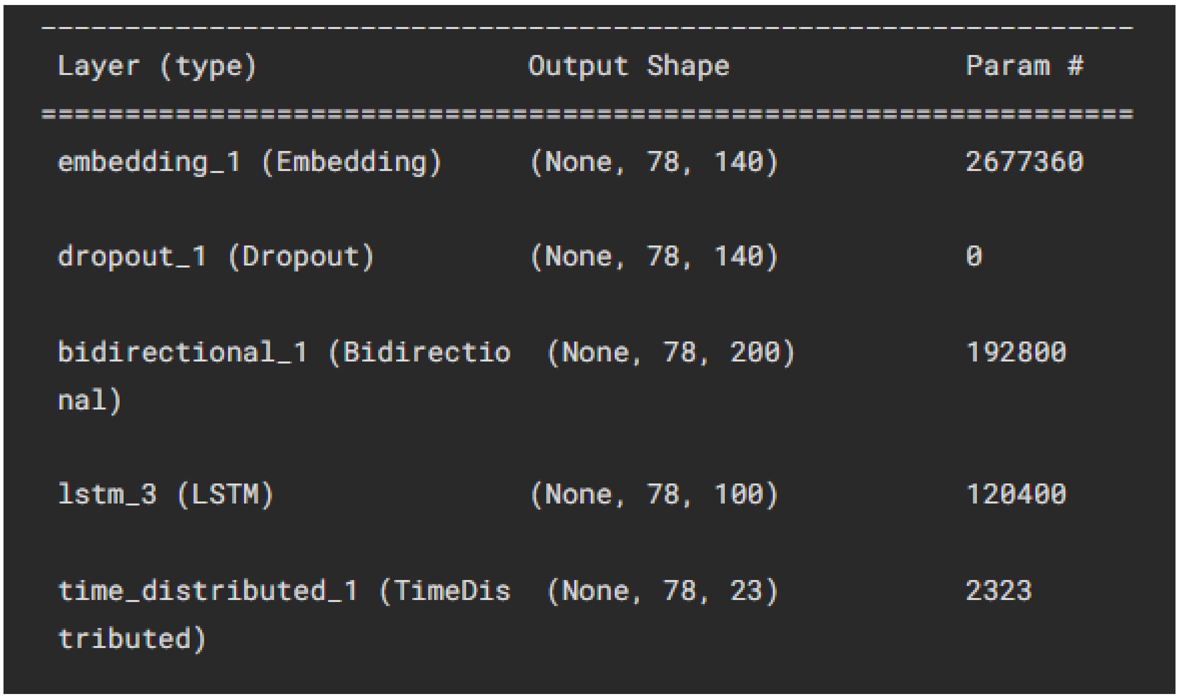
Padding and encoding labels

At last, after tokenization we would be padding sequences to make the length of sequences equal and one hot encoding the labels for the embedding of the words which would be later used in the model.

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
X_train= pad_sequences(maxlen=maxlen, sequences =
X_train,padding="post",value=word2idx['PAD'])
X_test= pad_sequences(maxlen=maxlen, sequences =
X_test,padding="post",value=word2idx['PAD'])
Y_train = pad_sequences(maxlen=maxlen, sequences=Y_train, padding="post",
value=tag2idx["PAD"])
print(np.array(Y_train).shape)
Y_test = pad_sequences(maxlen=maxlen, sequences=Y_test, padding="post",
value=tag2idx["PAD"])
from keras.utils import to_categorical
y_train = [to_categorical(i, num_classes=n_tags) for i in Y_train]
y_test = [to_categorical(i, num_classes=n_tags) for i in Y_test]
```

2) Model Building

We would be using **BiLSTM** state of the art model in keras for our task in this assignment.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 78, 140)           2677360

 dropout_1 (Dropout)         (None, 78, 140)           0

 bidirectional_1 (Bidirectio  (None, 78, 200)          192800
 nal)

 lstm_3 (LSTM)               (None, 78, 100)           120400

 time_distributed_1 (TimeDis  (None, 78, 23)           2323
 tributed)
```

We would be following the above model structure for our model building. Same hypeparameters are used for the baseline models.
Some important points regarding the model:

1. Here, **Embedding Layer** from keras module is used for embedding and no pre-trained word vectors are used.
2. **Bidirectional LSTM** is used to get the more in sentence information like contextual and syntactical information.
3. **LSTM** layer is used further to gain more features and information.
4. At last since we are using a many to many RNN architecture **time distribution layer** is used for the entity tagging task. Since, we are not using any probabilistic layer like CRF we have used "softmax" activation function to get the probability distribution of tags for the words.

3) Training

Now, comes the most important step in this methodolgy i.e. training the model. The code for the same can be found in the notebook attached.
I have used the following hyperparameters for baseline model training:
embedding output dimension =140
batch_size = 32
epochs = 10
For backpropogation, I have used Adam optimizer and cross entropy loss for updation of parameters.
Below code is used for the same

```
model = build_model(n_words, EMBEDDING, MAX_LEN, n_tags)
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=
["accuracy"])
model.summary()
```

```
history = model.fit(X_train,np.array(y_train),
                    batch_size=BATCH_SIZE, epochs=EPOCHS,
validation_split=0.1, verbose=1)
```

The final step after training is **evaluation** of the model on the test dataset, the results of which can be found in the next section and below code is used:

```
from seqeval.metrics import classification_report, accuracy_score
pred_tag = [[idx2tag[i] for i in row] for row in pred]
y_true_tag = [[idx2tag[i] for i in row] for row in y_test_true]

report = classification_report(y_true_tag,pred_tag)
print(report)

def get_scores(predY, trueY):
    from seqeval.metrics import f1_score
    trueY_O = [i for i, x in enumerate(y_true_tag) if x == "O"]
    predY = [predY[i] for i in range(len(predY)) if i not in trueY_O]
    trueY = [trueY[i] for i in range(len(trueY)) if i not in trueY_O]

    print("Micro F1 score: ", f1_score(trueY, predY, average="micro"))
    print("Macro F1 score: ", f1_score(trueY, predY, average="macro"))
    print("Average F1 score: ", (f1_score(trueY, predY, average="micro") +
f1_score(trueY, predY, average="macro")) / 2)

get_scores(pred_tag, y_true_tag)
```

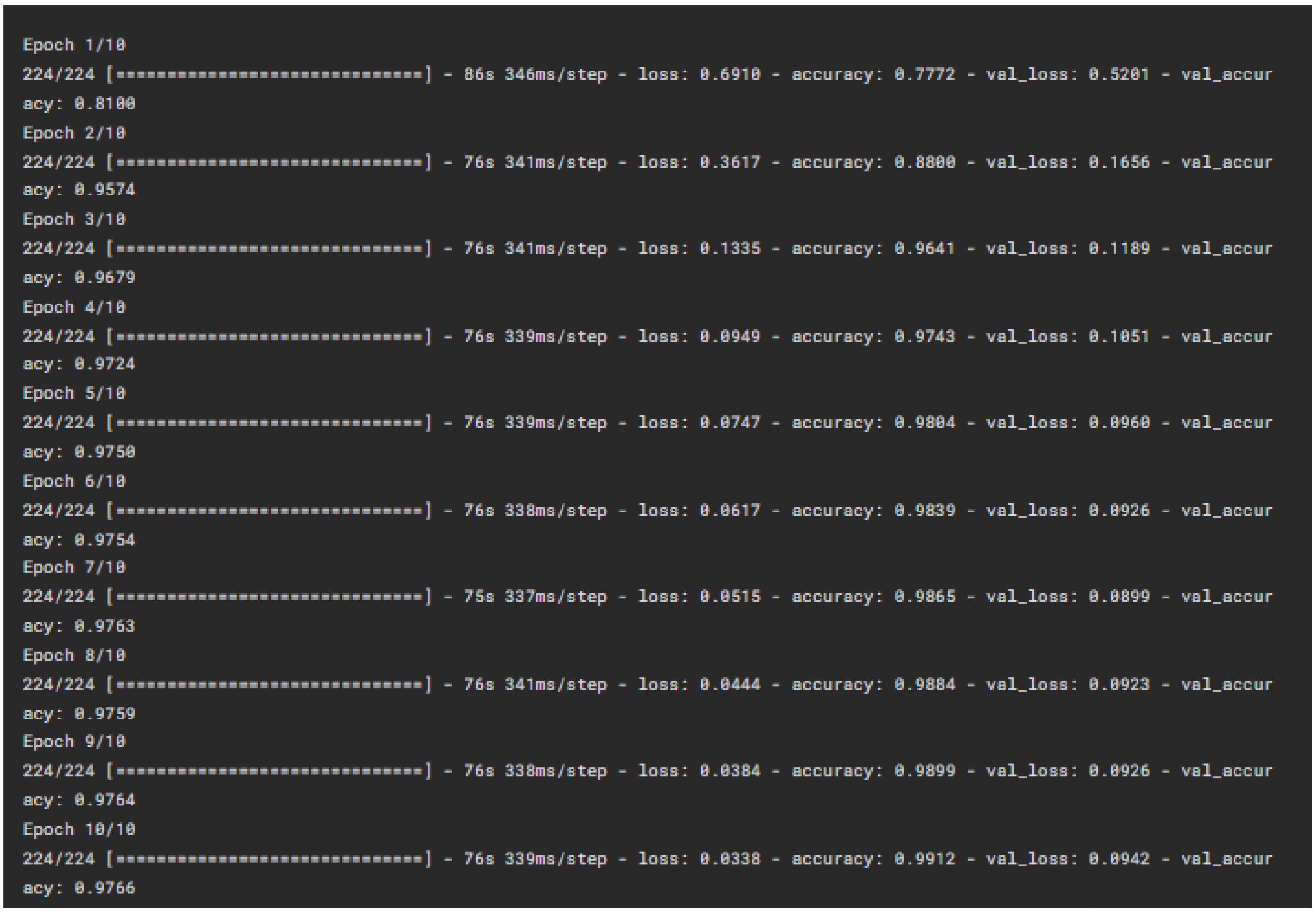
Experimental Results

.We have done three experiments as follows:

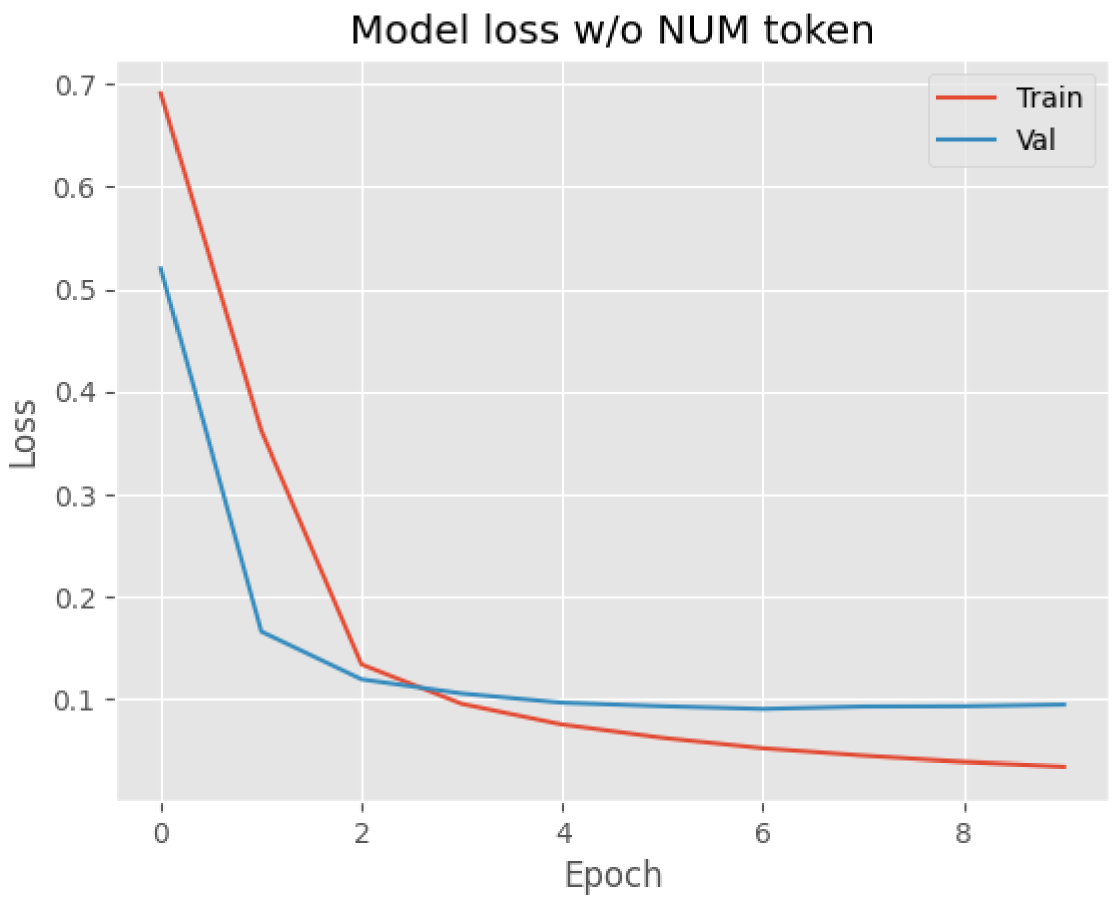
a) Experiment - I

In this experiment we have used the **baseline model** as discussed above.Following are the training and test results from it:

Training



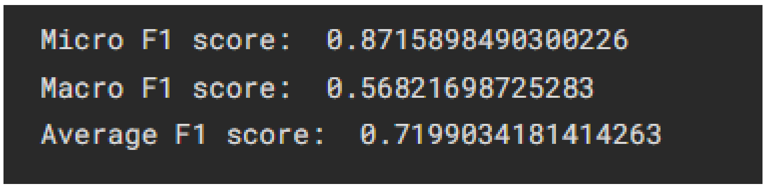
```
Epoch 1/10
224/224 [==============================] - 86s 346ms/step - loss: 0.6910 - accuracy: 0.7772 - val_loss: 0.5201 - val_accur
acy: 0.8100
Epoch 2/10
224/224 [==============================] - 76s 341ms/step - loss: 0.3617 - accuracy: 0.8800 - val_loss: 0.1656 - val_accur
acy: 0.9574
Epoch 3/10
224/224 [==============================] - 76s 341ms/step - loss: 0.1335 - accuracy: 0.9641 - val_loss: 0.1189 - val_accur
acy: 0.9679
Epoch 4/10
224/224 [==============================] - 76s 339ms/step - loss: 0.0949 - accuracy: 0.9743 - val_loss: 0.1051 - val_accur
acy: 0.9724
Epoch 5/10
224/224 [==============================] - 76s 339ms/step - loss: 0.0747 - accuracy: 0.9804 - val_loss: 0.0960 - val_accur
acy: 0.9750
Epoch 6/10
224/224 [==============================] - 76s 338ms/step - loss: 0.0617 - accuracy: 0.9839 - val_loss: 0.0926 - val_accur
acy: 0.9754
Epoch 7/10
224/224 [==============================] - 75s 337ms/step - loss: 0.0515 - accuracy: 0.9865 - val_loss: 0.0899 - val_accur
acy: 0.9763
Epoch 8/10
224/224 [==============================] - 76s 341ms/step - loss: 0.0444 - accuracy: 0.9884 - val_loss: 0.0923 - val_accur
acy: 0.9759
Epoch 9/10
224/224 [==============================] - 76s 338ms/step - loss: 0.0384 - accuracy: 0.9899 - val_loss: 0.0926 - val_accur
acy: 0.9764
Epoch 10/10
224/224 [==============================] - 76s 339ms/step - loss: 0.0338 - accuracy: 0.9912 - val_loss: 0.0942 - val_accur
acy: 0.9766
```



Test

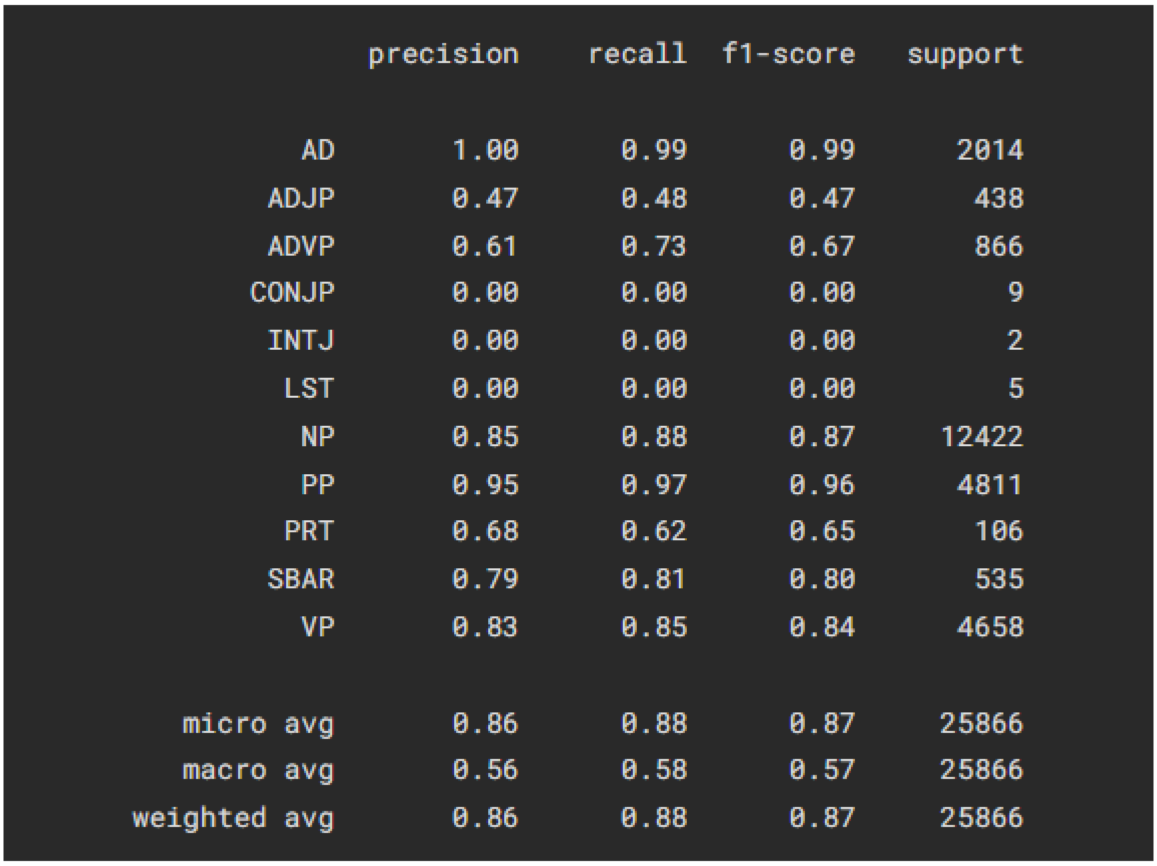
Loss= 0.0994

Accuracy= 0.9760



```
Micro F1 score:  0.8715898490300226
Macro F1 score:  0.56821698725283
Average F1 score:  0.7199034181414263
```

Classification Report ->

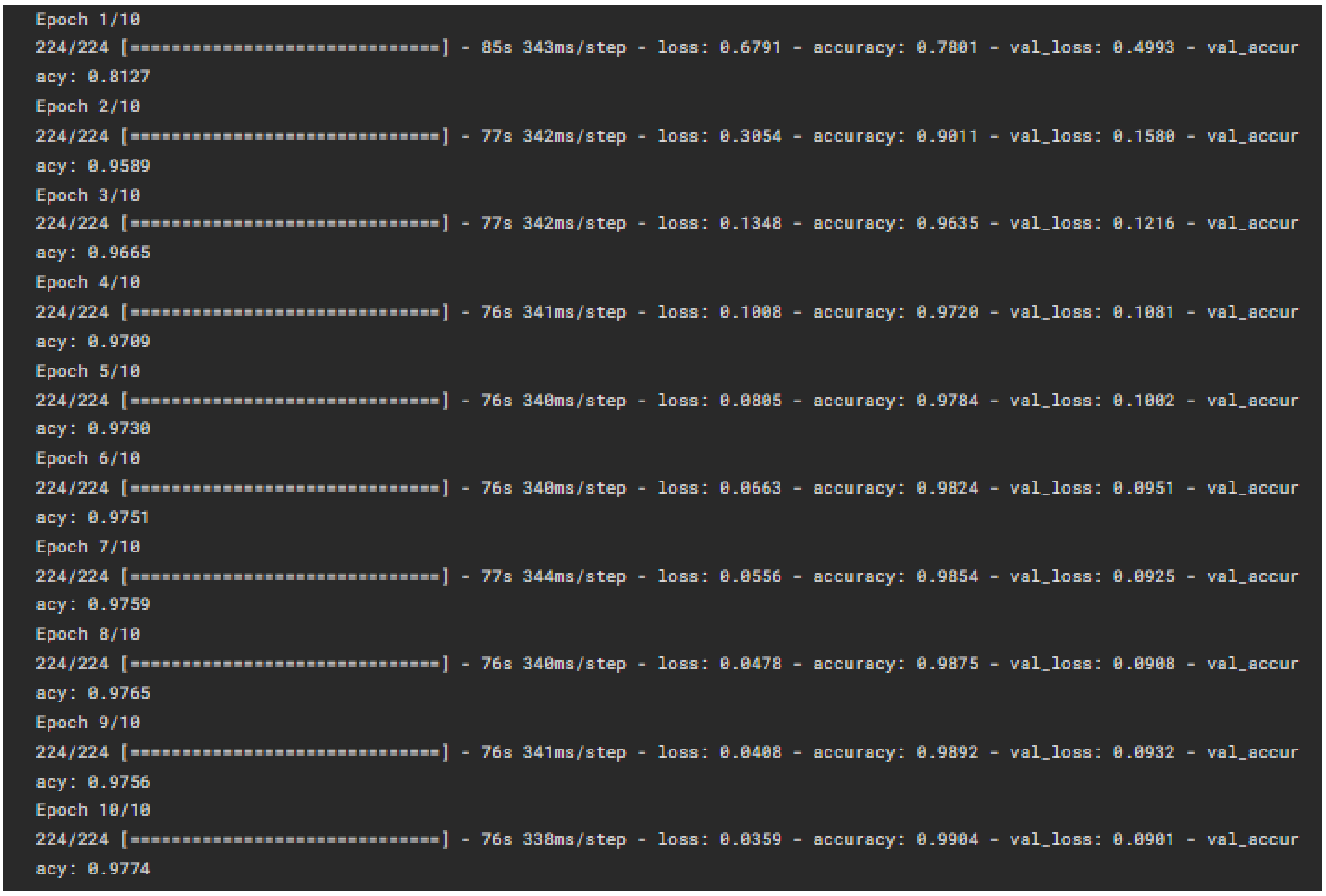


```
              precision    recall  f1-score   support

          AD       1.00      0.99      0.99      2014
        ADJP       0.47      0.48      0.47       438
        ADVP       0.61      0.73      0.67       866
       CONJP       0.00      0.00      0.00         9
        INTJ       0.00      0.00      0.00         2
         LST       0.00      0.00      0.00         5
          NP       0.85      0.88      0.87     12422
          PP       0.95      0.97      0.96      4811
         PRT       0.68      0.62      0.65       106
        SBAR       0.79      0.81      0.80       535
          VP       0.83      0.85      0.84      4658

   micro avg       0.86      0.88      0.87     25866
   macro avg       0.56      0.58      0.57     25866
weighted avg       0.86      0.88      0.87     25866
```

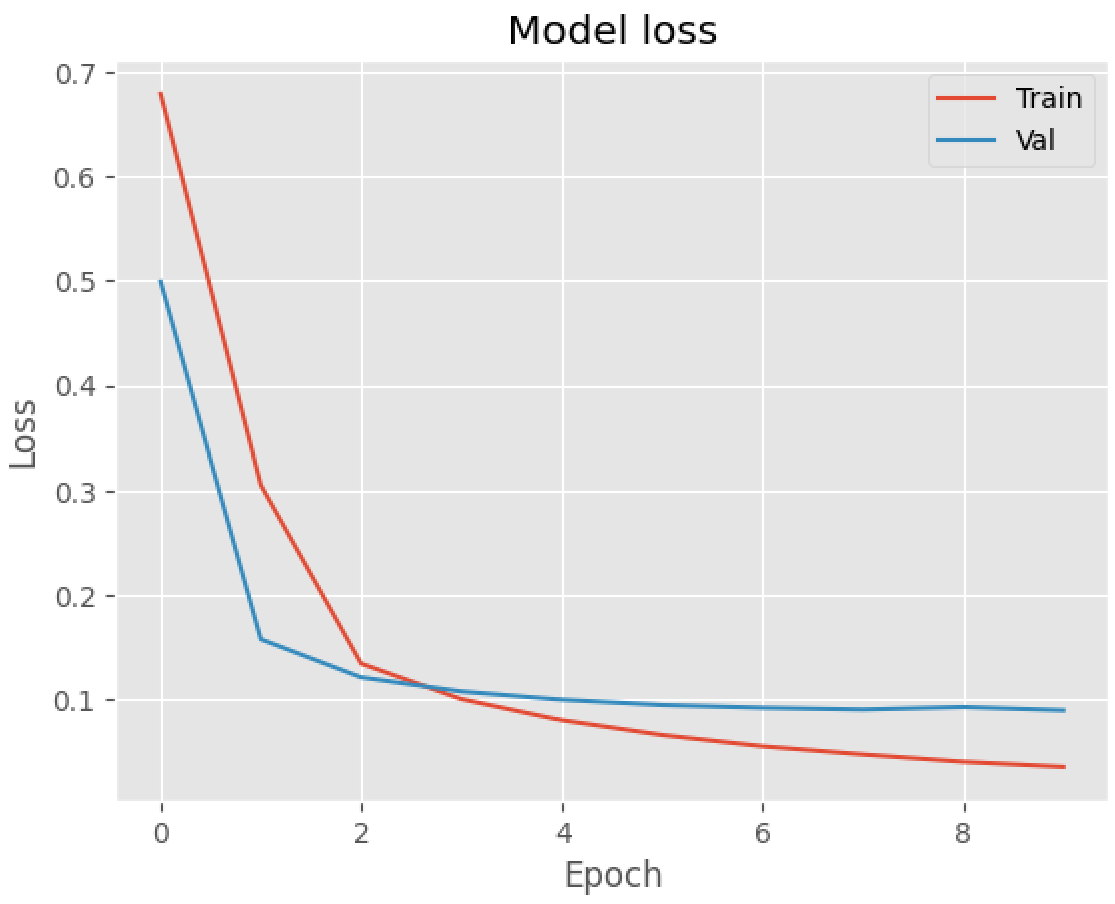
b) Experiment - II

In this experiment we have used **additional features** by adding a special token "NUM" for numerical tokens. Following are the training and test results from it. Same **baseline model** is used here too. Following are the training and test results from it:

Training



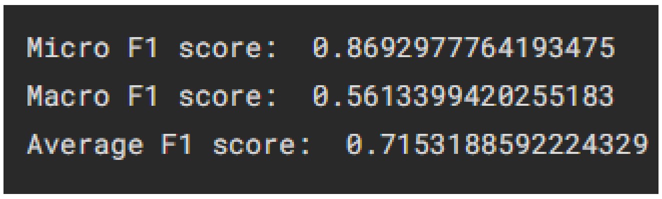
```
Epoch 1/10
224/224 [==============================] - 85s 343ms/step - loss: 0.6791 - accuracy: 0.7801 - val_loss: 0.4993 - val_accur
acy: 0.8127
Epoch 2/10
224/224 [==============================] - 77s 342ms/step - loss: 0.3054 - accuracy: 0.9011 - val_loss: 0.1580 - val_accur
acy: 0.9589
Epoch 3/10
224/224 [==============================] - 77s 342ms/step - loss: 0.1348 - accuracy: 0.9635 - val_loss: 0.1216 - val_accur
acy: 0.9665
Epoch 4/10
224/224 [==============================] - 76s 341ms/step - loss: 0.1008 - accuracy: 0.9720 - val_loss: 0.1081 - val_accur
acy: 0.9709
Epoch 5/10
224/224 [==============================] - 76s 340ms/step - loss: 0.0805 - accuracy: 0.9784 - val_loss: 0.1002 - val_accur
acy: 0.9730
Epoch 6/10
224/224 [==============================] - 76s 340ms/step - loss: 0.0663 - accuracy: 0.9824 - val_loss: 0.0951 - val_accur
acy: 0.9751
Epoch 7/10
224/224 [==============================] - 77s 344ms/step - loss: 0.0556 - accuracy: 0.9854 - val_loss: 0.0925 - val_accur
acy: 0.9759
Epoch 8/10
224/224 [==============================] - 76s 340ms/step - loss: 0.0478 - accuracy: 0.9875 - val_loss: 0.0908 - val_accur
acy: 0.9765
Epoch 9/10
224/224 [==============================] - 76s 341ms/step - loss: 0.0408 - accuracy: 0.9892 - val_loss: 0.0932 - val_accur
acy: 0.9756
Epoch 10/10
224/224 [==============================] - 76s 338ms/step - loss: 0.0359 - accuracy: 0.9904 - val_loss: 0.0901 - val_accur
acy: 0.9774
```



Test

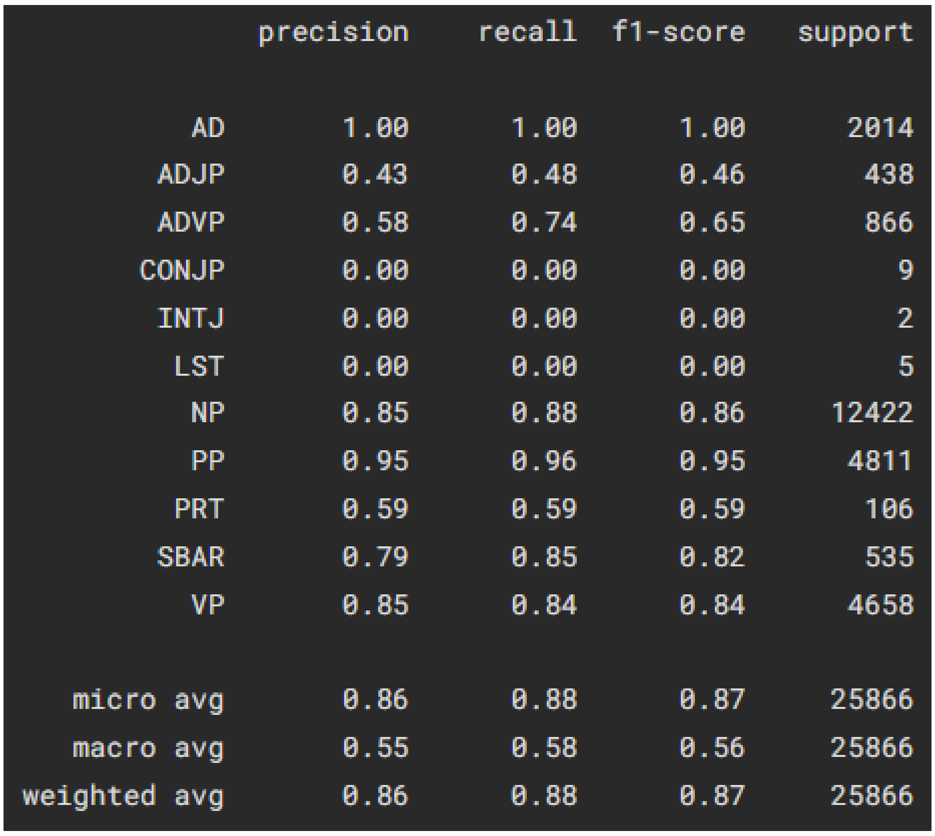
Loss = 0.0964

Accuracy= 0.9757



```
Micro F1 score:  0.8692977764193475
Macro F1 score:  0.5613399420255183
Average F1 score:  0.7153188592224329
```

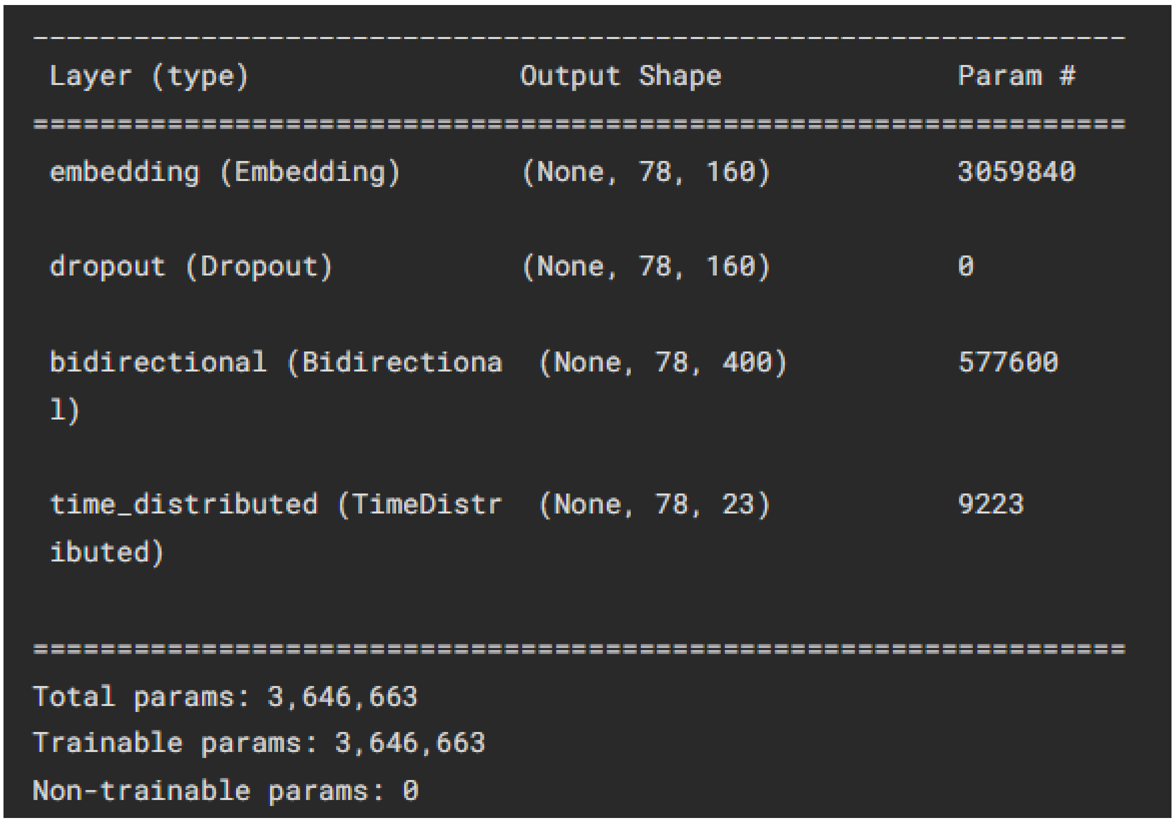
Classification Report ->



```
              precision    recall  f1-score   support

          AD       1.00      1.00      1.00      2014
        ADJP       0.43      0.48      0.46       438
        ADVP       0.58      0.74      0.65       866
       CONJP       0.00      0.00      0.00         9
        INTJ       0.00      0.00      0.00         2
         LST       0.00      0.00      0.00         5
          NP       0.85      0.88      0.86     12422
          PP       0.95      0.96      0.95      4811
         PRT       0.59      0.59      0.59       106
        SBAR       0.79      0.85      0.82       535
          VP       0.85      0.84      0.84      4658

   micro avg       0.86      0.88      0.87     25866
   macro avg       0.55      0.58      0.56     25866
weighted avg       0.86      0.88      0.87     25866
```

c) Experiment - III

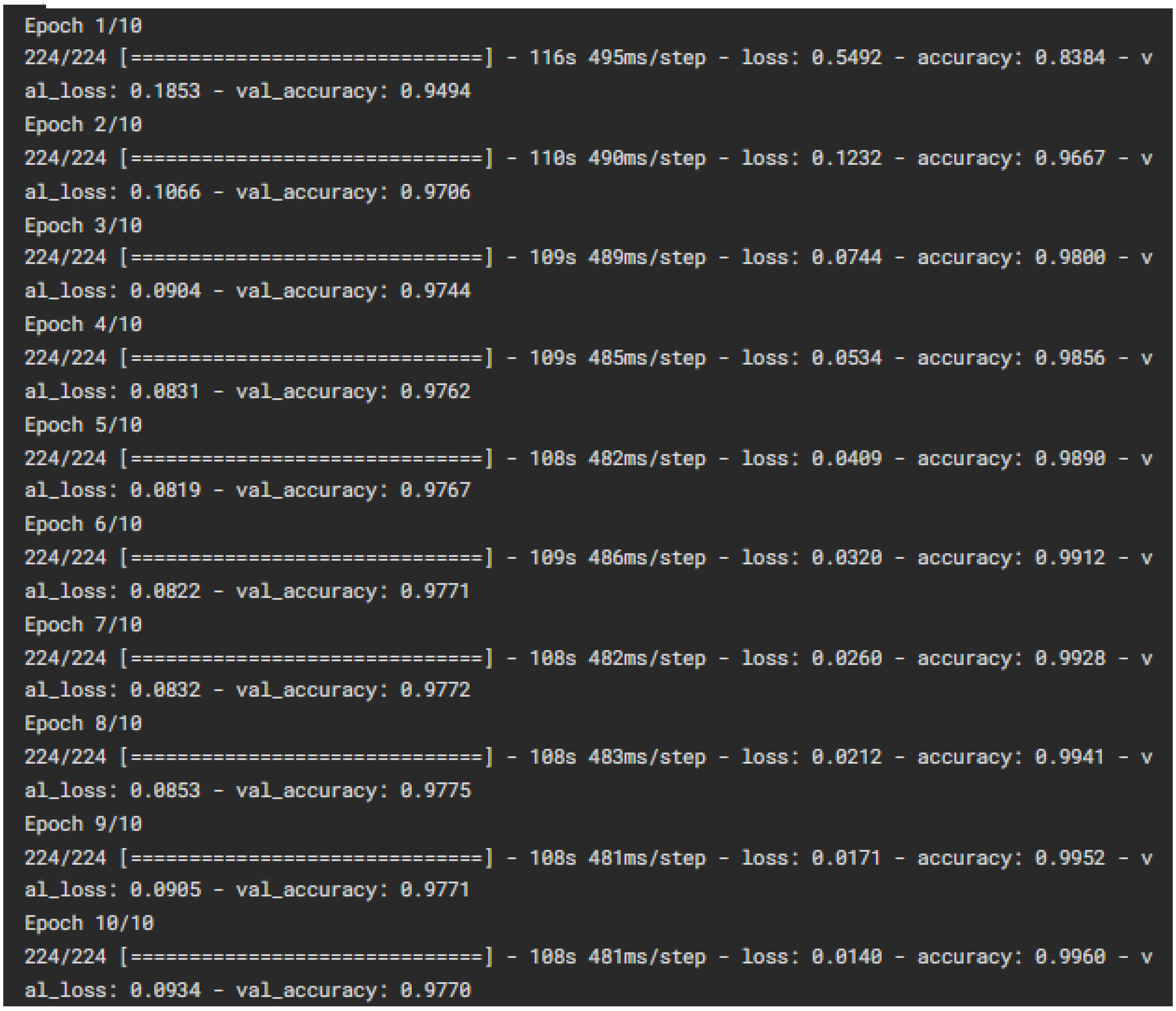
In this experiment **model tuning and hyperparameter tuning** is done on the baseline model. We have removed the last LSTM layer and changed the embedding and BiLSTM layer dimensions to 160 and 200 respectively. Model structure is as below:



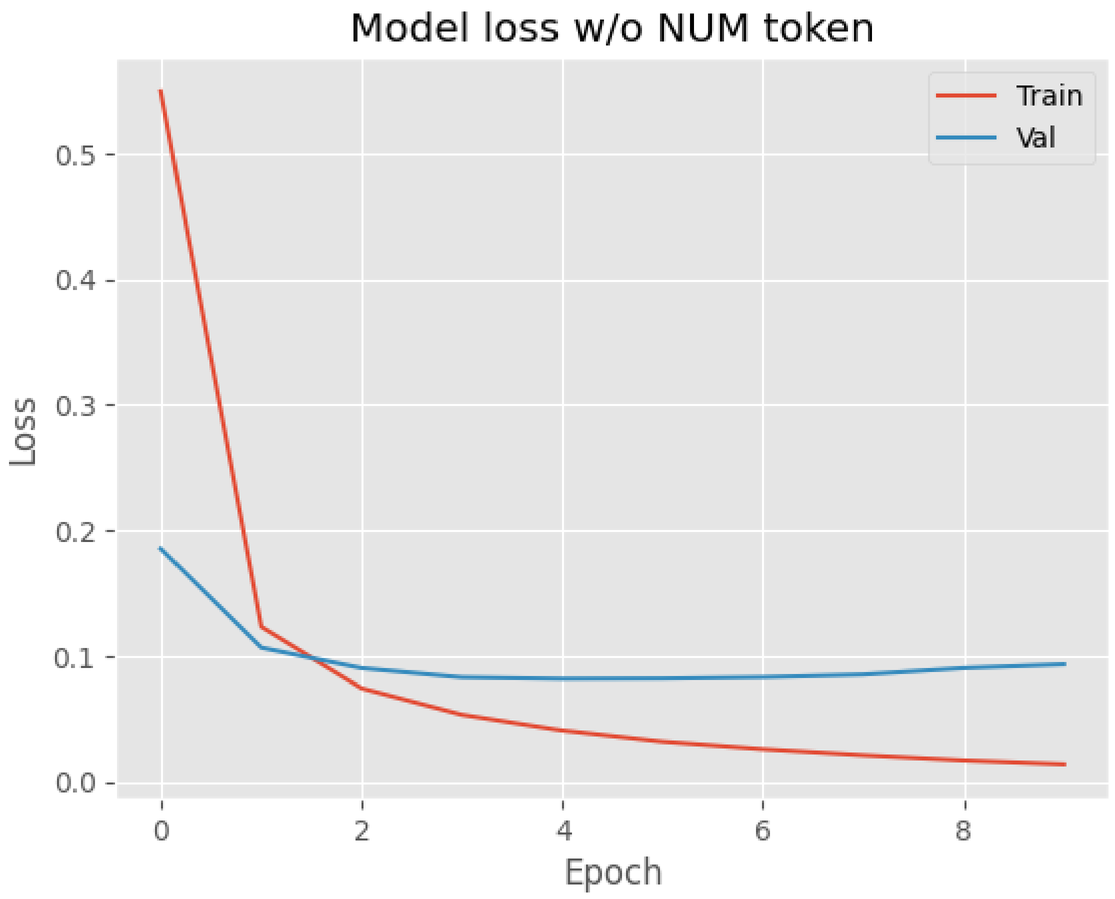
```
-----------------------------------------------------------------
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 78, 160)           3059840

 dropout (Dropout)           (None, 78, 160)           0

 bidirectional (Bidirectiona  (None, 78, 400)          577600
 l)

 time_distributed (TimeDistr  (None, 78, 23)           9223
 ibuted)

=================================================================
Total params: 3,646,663
Trainable params: 3,646,663
Non-trainable params: 0
```

Following are the training and test results from it:

Training



```
Epoch 1/10
224/224 [==============================] - 116s 495ms/step - loss: 0.5492 - accuracy: 0.8384 - v
al_loss: 0.1853 - val_accuracy: 0.9494
Epoch 2/10
224/224 [==============================] - 110s 490ms/step - loss: 0.1232 - accuracy: 0.9667 - v
al_loss: 0.1066 - val_accuracy: 0.9706
Epoch 3/10
224/224 [==============================] - 109s 489ms/step - loss: 0.0744 - accuracy: 0.9800 - v
al_loss: 0.0904 - val_accuracy: 0.9744
Epoch 4/10
224/224 [==============================] - 109s 485ms/step - loss: 0.0534 - accuracy: 0.9856 - v
al_loss: 0.0831 - val_accuracy: 0.9762
Epoch 5/10
224/224 [==============================] - 108s 482ms/step - loss: 0.0409 - accuracy: 0.9890 - v
al_loss: 0.0819 - val_accuracy: 0.9767
Epoch 6/10
224/224 [==============================] - 109s 486ms/step - loss: 0.0320 - accuracy: 0.9912 - v
al_loss: 0.0822 - val_accuracy: 0.9771
Epoch 7/10
224/224 [==============================] - 108s 482ms/step - loss: 0.0260 - accuracy: 0.9928 - v
al_loss: 0.0832 - val_accuracy: 0.9772
Epoch 8/10
224/224 [==============================] - 108s 483ms/step - loss: 0.0212 - accuracy: 0.9941 - v
al_loss: 0.0853 - val_accuracy: 0.9775
Epoch 9/10
224/224 [==============================] - 108s 481ms/step - loss: 0.0171 - accuracy: 0.9952 - v
al_loss: 0.0905 - val_accuracy: 0.9771
Epoch 10/10
224/224 [==============================] - 108s 481ms/step - loss: 0.0140 - accuracy: 0.9960 - v
al_loss: 0.0934 - val_accuracy: 0.9770
```



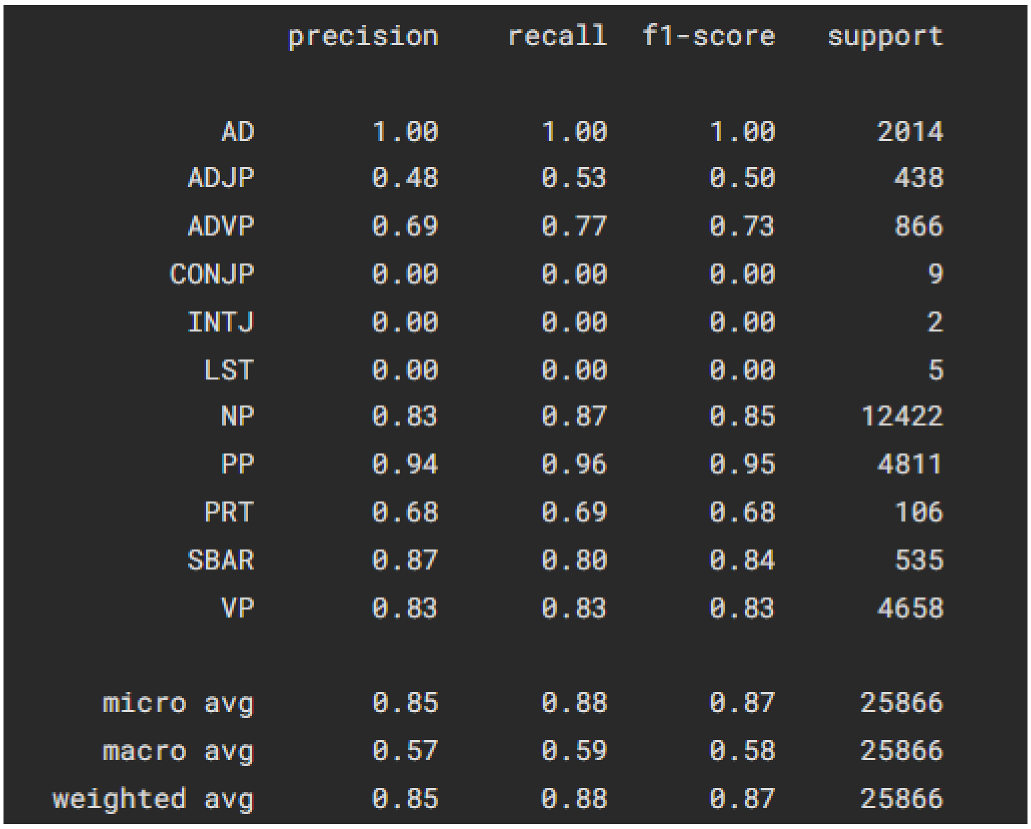
Test

Loss = 0.1022

Accuracy = 0.9753

```
  Micro F1 score:  0.8653092293054234
  Macro F1 score:  0.5798167167880058
  Average F1 score:  0.7225629730467147
```

Classification Report ->

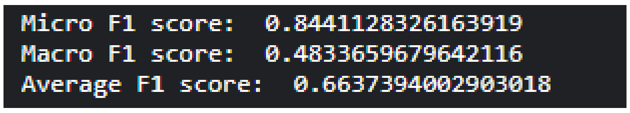


```
              precision    recall  f1-score   support

          AD       1.00      1.00      1.00      2014
        ADJP       0.48      0.53      0.50       438
        ADVP       0.69      0.77      0.73       866
       CONJP       0.00      0.00      0.00         9
        INTJ       0.00      0.00      0.00         2
         LST       0.00      0.00      0.00         5
          NP       0.83      0.87      0.85     12422
          PP       0.94      0.96      0.95      4811
         PRT       0.68      0.69      0.68       106
        SBAR       0.87      0.80      0.84       535
          VP       0.83      0.83      0.83      4658

   micro avg       0.85      0.88      0.87     25866
   macro avg       0.57      0.59      0.58     25866
weighted avg       0.85      0.88      0.87     25866
```

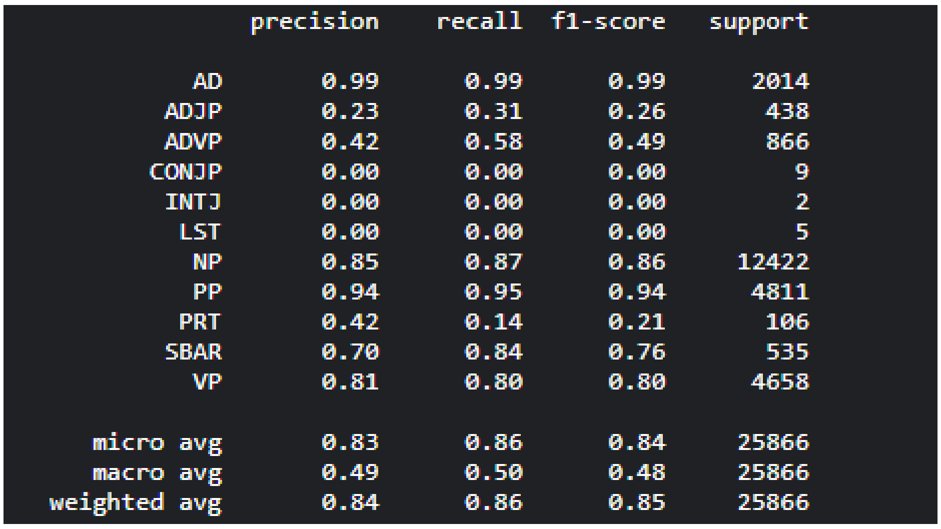
Analysis

The following observations can be analysed from the results:

1. According to the accuracy score model with experiment 1 came out to be best but the average F1 score for experiment 3 is the best which can be due to the increase in the **embedding dimension** i.e. more information is learned in that case.
2. We can see that model performs slightly better if we haven't added additional numerical token. But overall, much effect is not seen.
3. **Training loss** in all the cases is decreasing and tend to decrease more as can be seen from the graphs. This concludes that model can be trained for more epochs. **Validation loss** is almost constant in the case of first two experiments but in experiment 3 it starts to diverge a bit upwards.
4. Macro average of **recall** and **precision** in experiment 3 came out to be best i.e model is recalling more true positives than the other two cases. This can be due to the same reason as explained in point 1.
5. Test Loss is maximum in the case of experiment 3. This can be due to the fact that model is still undertrained due to an increase in features and can be trained for more epochs
6. We have done one more experiment taking the same baseline model in experiment but the only change is the embedding dimension which is reduced to 20.
   Loss = 0.1011 and Accuracy = 0.9717

   ```
   Micro F1 score:  0.8441128326163919
   Macro F1 score:  0.4833659679642116
   Average F1 score:  0.6637394002903018
   ```

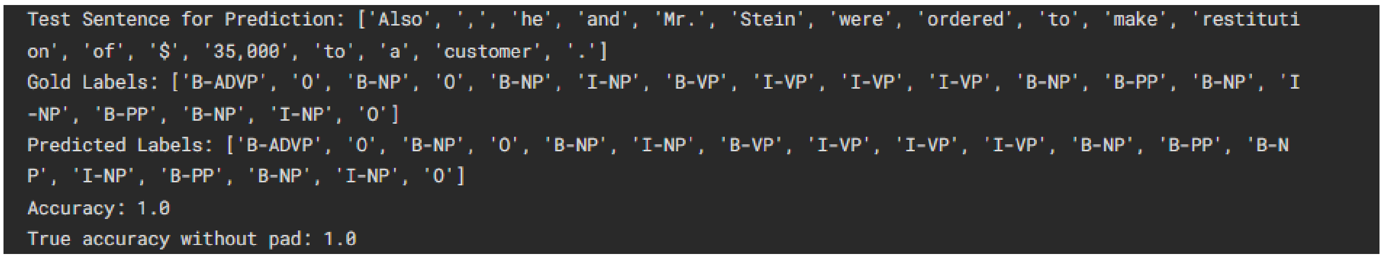


```
              precision    recall  f1-score   support

          AD       0.99      0.99      0.99      2014
        ADJP       0.23      0.31      0.26       438
        ADVP       0.42      0.58      0.49       866
       CONJP       0.00      0.00      0.00         9
        INTJ       0.00      0.00      0.00         2
         LST       0.00      0.00      0.00         5
          NP       0.85      0.87      0.86     12422
          PP       0.94      0.95      0.94      4811
         PRT       0.42      0.14      0.21       106
        SBAR       0.70      0.84      0.76       535
          VP       0.81      0.80      0.80      4658

   micro avg       0.83      0.86      0.84     25866
   macro avg       0.49      0.50      0.48     25866
weighted avg       0.84      0.86      0.85     25866
```

We can see a clear effect due to a decrease in the embedding dimension i.e. loss increases and average F1 score declines drastically.

Conclusion

NER using neural networks is successfully done in this assignment.
We can see that validation loss is almost same for every epoch, this can be due to the smaller number of epochs model has been trained on. As seen in the last point from the analysis on the reduction of embedding dimension model performance weakens due to a decrease in learnable embedding parameters.
We have also tested our baseline model(Experiment-I) on a random sentences from test sets and most of time it predicts accurately inspite of unknown words. Following is one of the example



```
Test Sentence for Prediction: ['Also', ',', 'he', 'and', 'Mr.', 'Stein', 'were', 'ordered', 'to', 'make', 'restituti
on', 'of', '$', '35,000', 'to', 'a', 'customer', '.']
Gold Labels: ['B-ADVP', 'O', 'B-NP', 'O', 'B-NP', 'I-NP', 'B-VP', 'I-VP', 'I-VP', 'I-VP', 'B-NP', 'B-PP', 'B-NP', 'I
-NP', 'B-PP', 'B-NP', 'I-NP', 'O']
Predicted Labels: ['B-ADVP', 'O', 'B-NP', 'O', 'B-NP', 'I-NP', 'B-VP', 'I-VP', 'I-VP', 'I-VP', 'B-NP', 'B-PP', 'B-N
P', 'I-NP', 'B-PP', 'B-NP', 'I-NP', 'O']
Accuracy: 1.0
True accuracy without pad: 1.0
```

Jupyter Notebook Link

https://www.kaggle.com/yuggarg/ell881a2