NLP_Assignment2_Report
Name: Udrasht Pal
Roll NO: 2022201020

Question 3 and 4 answer combine below

3.1

I explored various hyper-parameter and model architecture options. Multiple approaches were used to optimize hyper-parameters and model architecture.

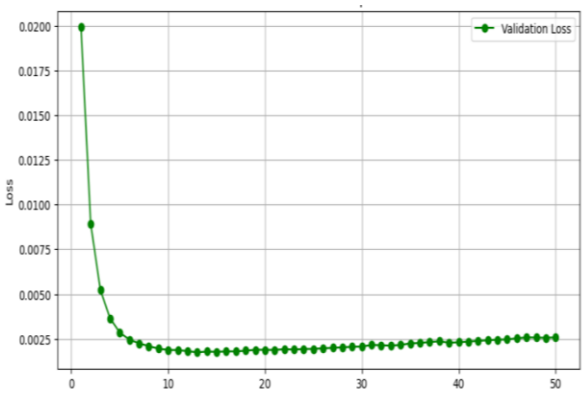
I experiment the model by changing the dimension of hidden layer, learning rate, loss function

A few of the possible analyses are displayed below

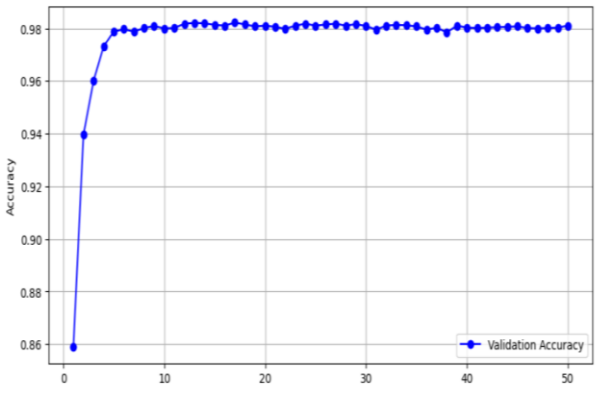
Hyper-parameters Tuning

```
embedding_dim = 601

hidden_dim = 128
output_dim = 13  # Number of POS tags
batch_size = 32
num_epochs = 50
```



Loss vs epoch graph for dev set



Accuracy vs epoch graph for dev set

On test data set

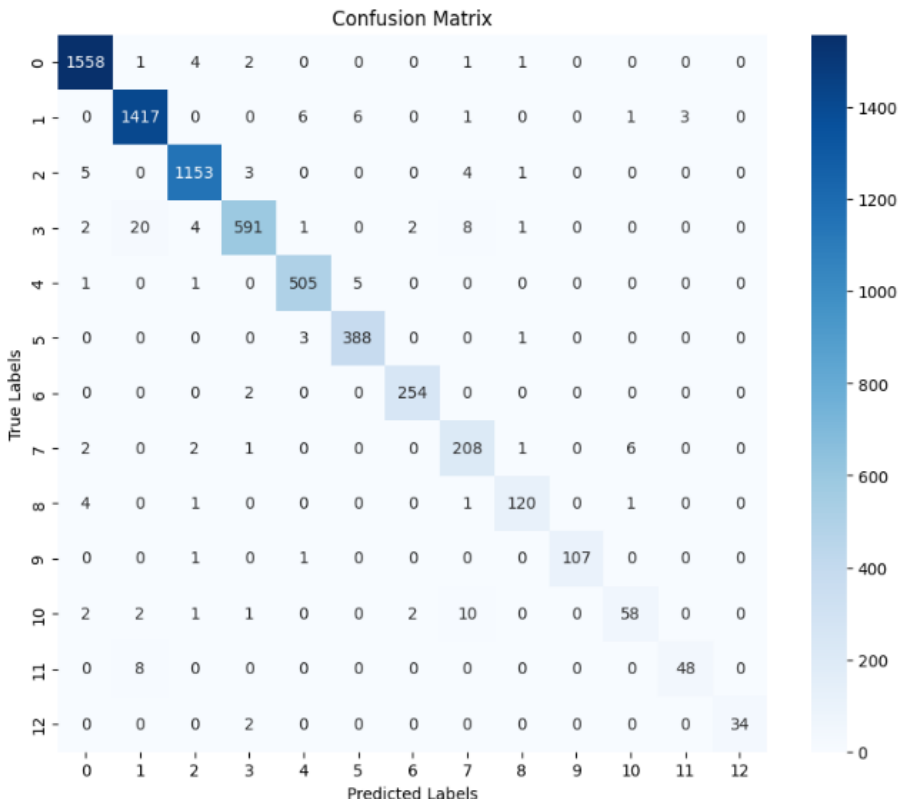
Classification Report :

	precision	recall	f1-score	support
0	0.988571	0.993618	0.991088	1567.000000
1	0.993702	0.990237	0.991966	1434.000000
2	0.984629	0.988851	0.986735	1166.000000
3	0.974359	0.966614	0.970471	629.000000
4	0.986275	0.982422	0.984344	512.000000
5	0.965261	0.992347	0.978616	392.000000
6	0.988235	0.984375	0.986301	256.000000
7	0.912664	0.950000	0.930958	220.000000
8	0.968000	0.952756	0.960317	127.000000
9	1.000000	0.990826	0.995392	109.000000
10	0.887097	0.723684	0.797101	76.000000
11	0.964286	0.964286	0.964286	56.000000
12	1.000000	0.916667	0.956522	36.000000
accuracy	0.982067	0.982067	0.982067	0.982067
macro avg	0.970237	0.953591	0.961085	6580.000000
weighted avg	0.981990	0.982067	0.981893	6580.000000

class wise accuracy on te

	Class	Accuracy
0	PROPN	0.993618
1	ADP	0.990237
2	NOUN	0.988851
3	VERB	0.966614
4	DET	0.982422
5	PRON	0.992347
6	AUX	0.984375
7	ADJ	0.950000
8	NUM	0.952756
9	CCONJ	0.990826
10	ADV	0.723684
11	PART	0.964286
12	INTJ	0.916667

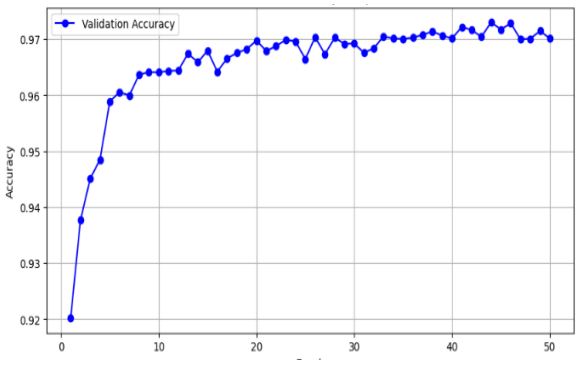
r



```
embedding_dim = 601

hidden_dim = 64
output_dim = 13  # Number of POS tags
batch_size = 32
num_epochs = 50
L_R=0.0001
```

Loss vs epoch graph for dev set



Accuracy vs epoch graph for dev

This model perform better as compare to other model in FFNN

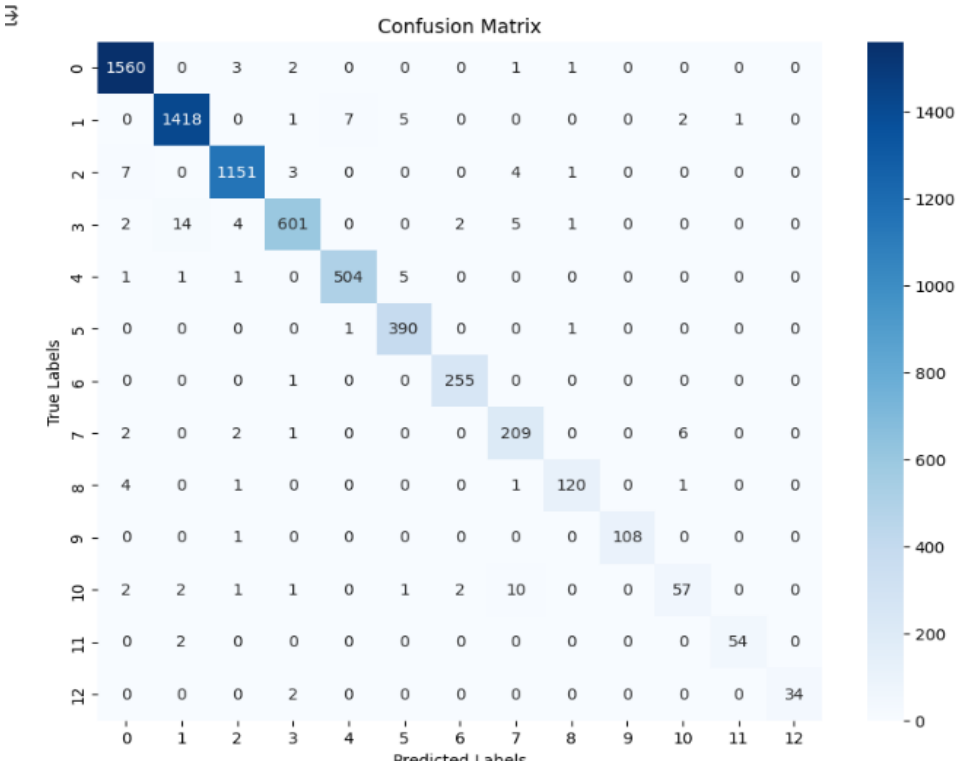
On test set

Classification Report in Color:

	precision	recall	f1-score	support
0	0.987967	0.995533	0.991736	1567.000000
1	0.956960	0.992329	0.974324	1434.000000
2	0.987179	0.990566	0.988870	1166.000000
3	0.991103	0.885533	0.935348	629.000000
4	0.980769	0.996094	0.988372	512.000000
5	0.997442	0.994898	0.996169	392.000000
6	0.965779	0.992188	0.978805	256.000000
7	0.963134	0.950000	0.956522	220.000000
8	0.951220	0.921260	0.936000	127.000000
9	1.000000	1.000000	1.000000	109.000000
10	0.895522	0.789474	0.839161	76.000000
11	0.964912	0.982143	0.973451	56.000000
12	1.000000	0.972222	0.985915	36.000000
accuracy	0.977812	0.977812	0.977812	0.977812
macro avg	0.972461	0.958634	0.964975	6580.000000
weighted avg	0.977973	0.977812	0.977476	6580.000000

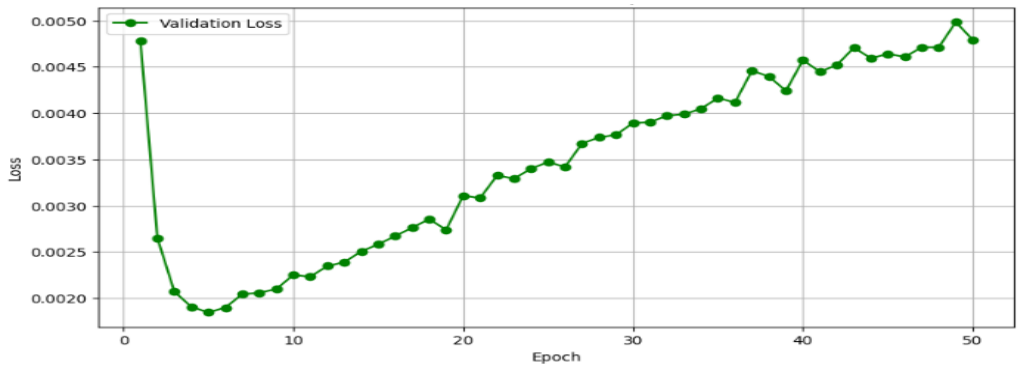
class wise accuracy on test data

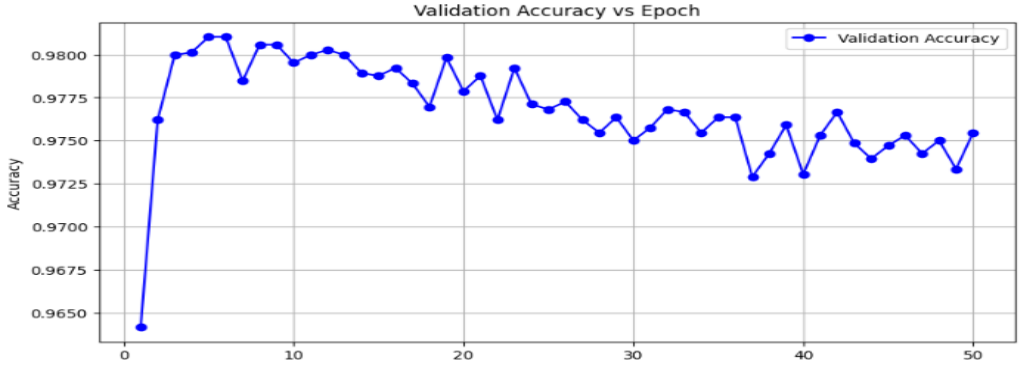
	Class	Accuracy
0	PROPN	0.995533
1	ADP	0.988842
2	NOUN	0.987136
3	VERB	0.955485
4	DET	0.984375
5	PRON	0.994898
6	AUX	0.996094
7	ADJ	0.950000
8	NUM	0.944882
9	CCONJ	0.990826
10	ADV	0.750000
11	PART	0.964286
12	INTJ	0.944444




```
embedding_dim = 601

hidden_dim = 90
output_dim = 13  # Number of POS tags
batch_size = 32
num_epochs = 50
L_R=0.0004
```





This model perform worst hear the loss value is increase and accuracy decrease

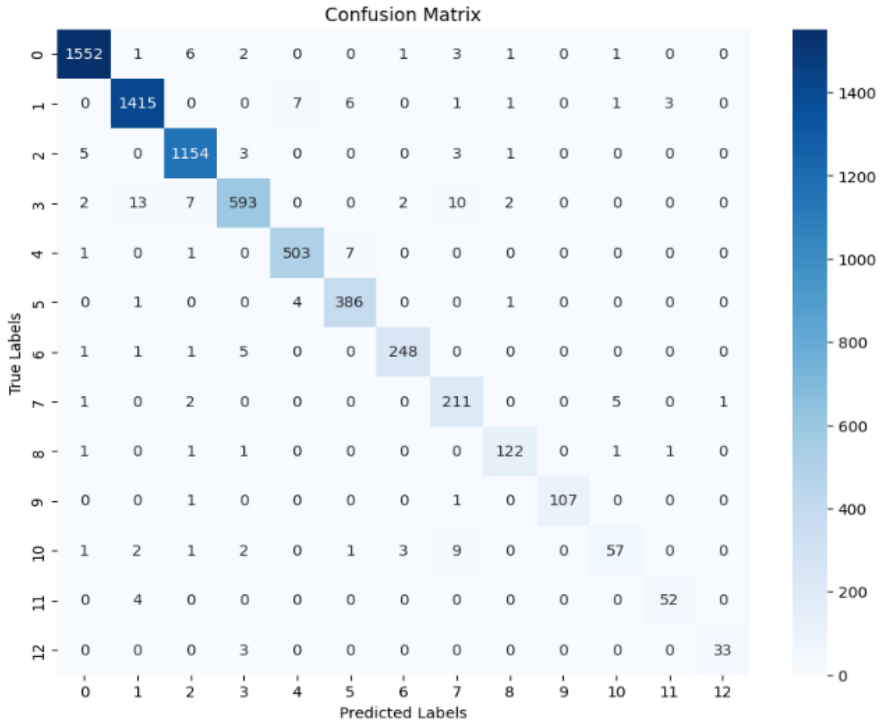
On test data

Classification Report :

	precision	recall	f1-score	support
0	0.992327	0.990428	0.991377	1567.000000
1	0.984690	0.986750	0.985719	1434.000000
2	0.982964	0.989708	0.986325	1166.000000
3	0.973727	0.942766	0.957997	629.000000
4	0.978599	0.982422	0.980507	512.000000
5	0.965000	0.984694	0.974747	392.000000
6	0.976378	0.968750	0.972549	256.000000
7	0.886555	0.959091	0.921397	220.000000
8	0.953125	0.960630	0.956863	127.000000
9	1.000000	0.981651	0.990741	109.000000
10	0.876923	0.750000	0.808511	76.000000
11	0.928571	0.928571	0.928571	56.000000
12	0.970588	0.916667	0.942857	36.000000
accuracy	0.977660	0.977660	0.977660	0.977660
macro avg	0.959188	0.949395	0.953705	6580.000000
weighted avg	0.977749	0.977660	0.977560	6580.000000

class wise accuracy on test dat

	Class	Accuracy
0	PROPN	0.990428
1	ADP	0.986750
2	NOUN	0.989708
3	VERB	0.942766
4	DET	0.982422
5	PRON	0.984694
6	AUX	0.968750
7	ADJ	0.959091
8	NUM	0.960630
9	CCONJ	0.981651
10	ADV	0.750000
11	PART	0.928571
12	INTJ	0.916667



Conclusion: On below parameters my model perform best in my best knowledge with test set accuracy 97.5% accuracy on test set and 97% on dev set

```
embedding_dim = 601

hidden_dim = 64
output_dim = 13  # Number of POS tags
batch_size = 32
num_epochs = 50
L_R=0.0001
```

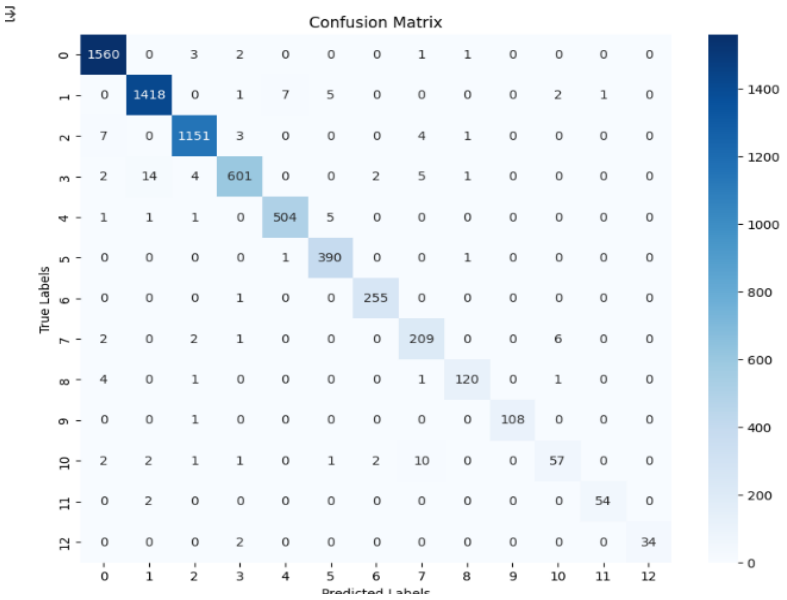
On test set

Classification Report in Color:

	precision	recall	f1-score	support
0	0.987967	0.995533	0.991736	1567.000000
1	0.956960	0.992329	0.974324	1434.000000
2	0.987179	0.990566	0.988870	1166.000000
3	0.991103	0.885533	0.935348	629.000000
4	0.980769	0.996094	0.988372	512.000000
5	0.997442	0.994898	0.996169	392.000000
6	0.965779	0.992188	0.978805	256.000000
7	0.963134	0.950000	0.956522	220.000000
8	0.951220	0.921260	0.936000	127.000000
9	1.000000	1.000000	1.000000	109.000000
10	0.895522	0.789474	0.839161	76.000000
11	0.964912	0.982143	0.973451	56.000000
12	1.000000	0.972222	0.985915	36.000000
accuracy	0.977812	0.977812	0.977812	0.977812
macro avg	0.972461	0.958634	0.964975	6580.000000
weighted avg	0.977973	0.977812	0.977476	6580.000000

class wise accuracy on test data

	Class	Accuracy
0	PROPN	0.995533
1	ADP	0.988842
2	NOUN	0.987136
3	VERB	0.955485
4	DET	0.984375
5	PRON	0.994898
6	AUX	0.996094
7	ADJ	0.950000
8	NUM	0.944882
9	CCONJ	0.990826
10	ADV	0.750000
11	PART	0.964286
12	INTJ	0.944444

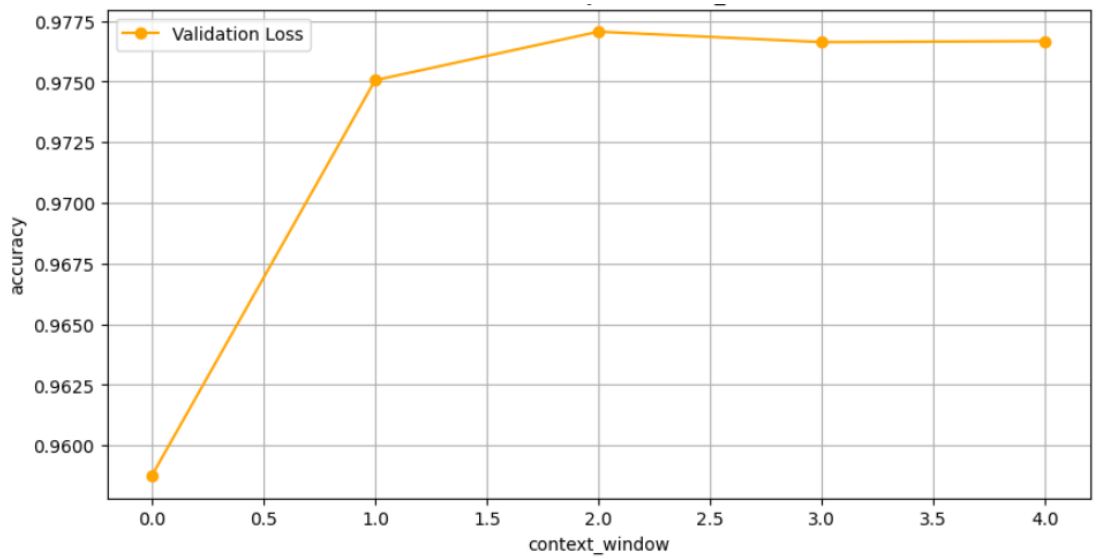


So I draw accuracy vs context_window graph on dev set

On context_window size =2 (p=2 , s=2) model perform best

```
embedding_dim = 601

hidden_dim = 64
output_dim = 13  # Number of POS tags
batch_size = 32
num_epochs = 50
L_R=0.0001
```



Accuracy vs context_window graph

3.2

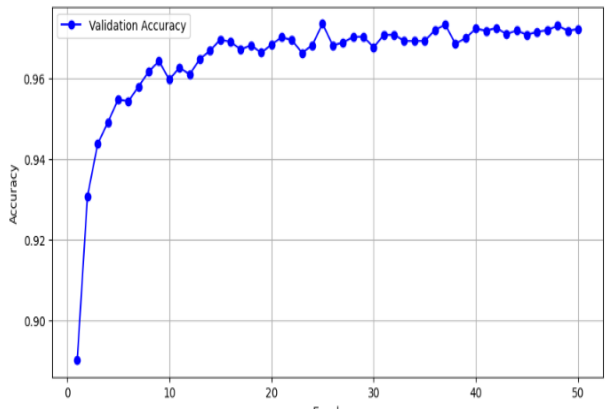
RNN→ LSTM

I explored various hyper-parameter and model architecture options. Multiple approaches were used to optimize hyper-parameters and model architecture.

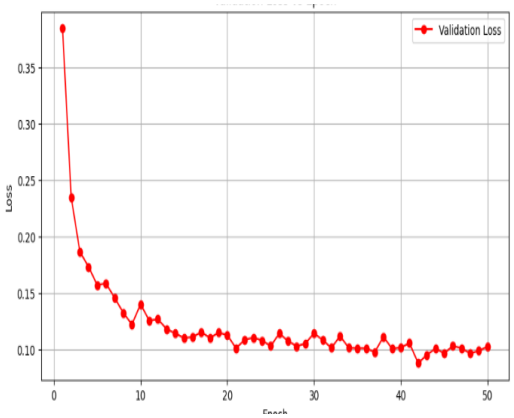
A few of the possible analyses are displayed below

Hyper-parameters Tuning

```
input_dim = 32
output_dim = 13
hidden_dim = 32
DROPOUT=0.5
lr=0.005
num_epochs = 50
Bidirectional=True
Layer_number=2
```



Accuracy vs epoch graph for dev set



Loss vs epoch graph for dev set

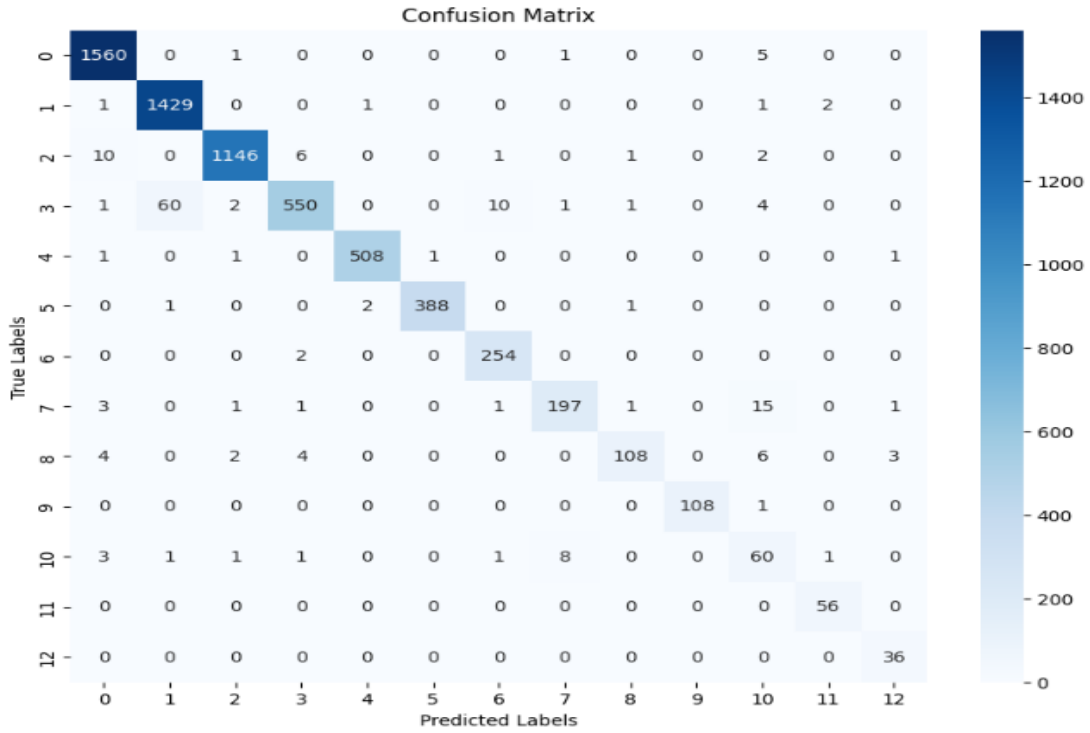
On test set

Classification Report in Color:

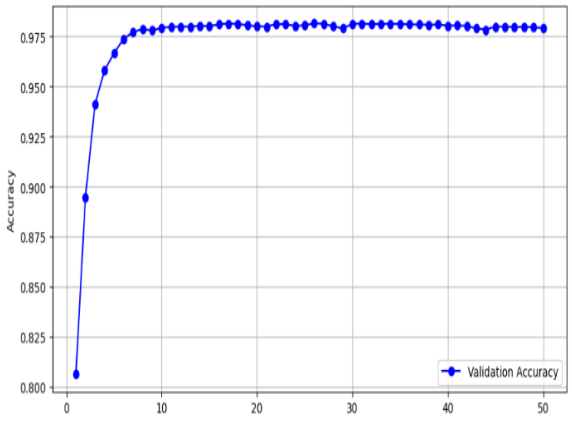
	precision	recall	f1-score	support
0	0.983617	0.996171	0.989854	1567.000000
1	0.957191	0.997908	0.977125	1434.000000
2	0.985395	0.983705	0.984549	1166.000000
3	0.987676	0.891892	0.937343	629.000000
4	0.984556	0.996094	0.990291	512.000000
5	0.992308	0.987245	0.989770	392.000000
6	0.980695	0.992188	0.986408	256.000000
7	0.949772	0.945455	0.947608	220.000000
8	0.957627	0.889764	0.922449	127.000000
9	1.000000	0.981651	0.990741	109.000000
10	0.935484	0.763158	0.840580	76.000000
11	0.982456	1.000000	0.991150	56.000000
12	1.000000	1.000000	1.000000	36.000000
accuracy	0.977052	0.977052	0.977052	0.977052
macro avg	0.976675	0.955787	0.965221	6580.000000
weighted avg	0.977200	0.977052	0.976659	6580.000000

class wise accuracy on test data

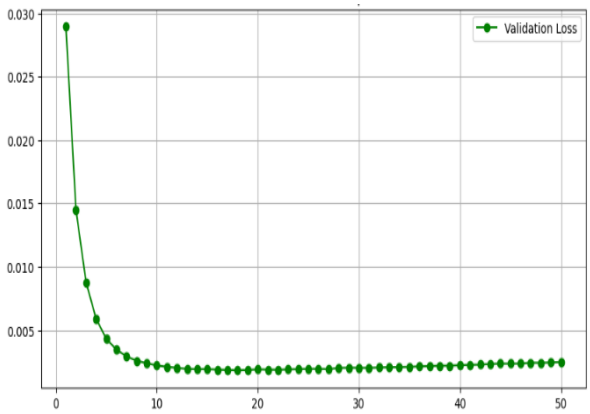
	Class	Accuracy
0	PROPN	0.995533
1	ADP	0.996513
2	NOUN	0.982847
3	VERB	0.874404
4	DET	0.992188
5	PRON	0.989796
6	AUX	0.992188
7	ADJ	0.895455
8	NUM	0.850394
9	CCONJ	0.990826
10	ADV	0.789474
11	PART	1.000000
12	INTJ	1.000000



```
input_dim = 64
output_dim = 13
hidden_dim = 64
DROPOUT=0.5
lr=0.005
num_epochs = 50
Bidirectional=True
Layer_number=2
```



Accuracy vs epoch graph for dev set



Loss vs epoch graph for dev set

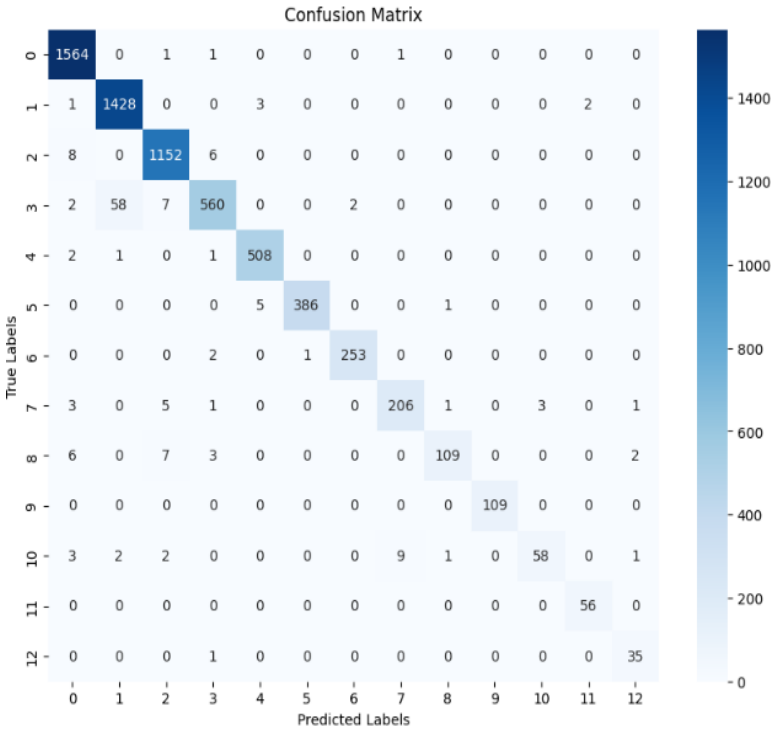
On test Data

Classification Report :

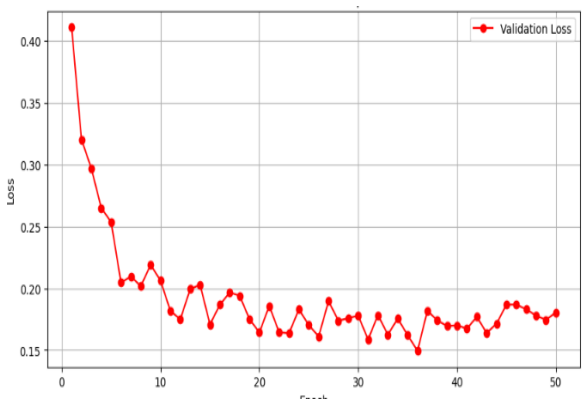
	precision	recall	f1-score	support
0	0.988593	0.995533	0.992051	1567.000000
1	0.986778	0.988842	0.987809	1434.000000
2	0.988832	0.987136	0.987983	1166.000000
3	0.982026	0.955485	0.968574	629.000000
4	0.984375	0.984375	0.984375	512.000000
5	0.972569	0.994898	0.983607	392.000000
6	0.984556	0.996094	0.990291	256.000000
7	0.908696	0.950000	0.928889	220.000000
8	0.967742	0.944882	0.956175	127.000000
9	1.000000	0.990826	0.995392	109.000000
10	0.863636	0.750000	0.802817	76.000000
11	0.981818	0.964286	0.972973	56.000000
12	1.000000	0.944444	0.971429	36.000000
accuracy	0.981915	0.981915	0.981915	0.981915
macro avg	0.969971	0.957446	0.963259	6580.000000
weighted avg	0.981849	0.981915	0.981783	6580.000000

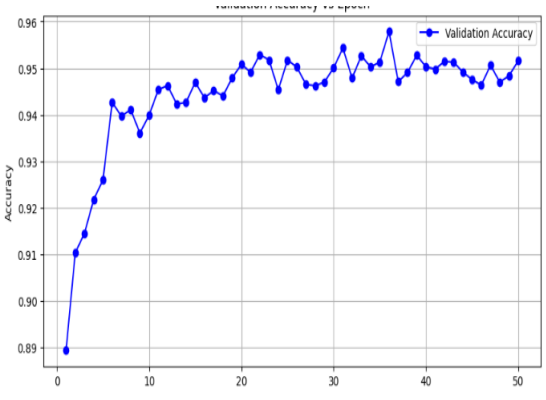
class wise accuracy on test data

	Class	Accuracy
0	PROPN	0.998086
1	ADP	0.995816
2	NOUN	0.987993
3	VERB	0.890302
4	DET	0.992188
5	PRON	0.984694
6	AUX	0.988281
7	ADJ	0.936364
8	NUM	0.858268
9	CCONJ	1.000000
10	ADV	0.763158
11	PART	1.000000
12	INTJ	0.972222




```
input_dim = 80
output_dim = 13
hidden_dim = 100
DROPOUT=0.65
lr=0.006
num_epochs = 50
Bidirectional=True
Layer_number=3
```





Accuracy vs epoch graph for dev set

Loss vs epoch graph for dev set

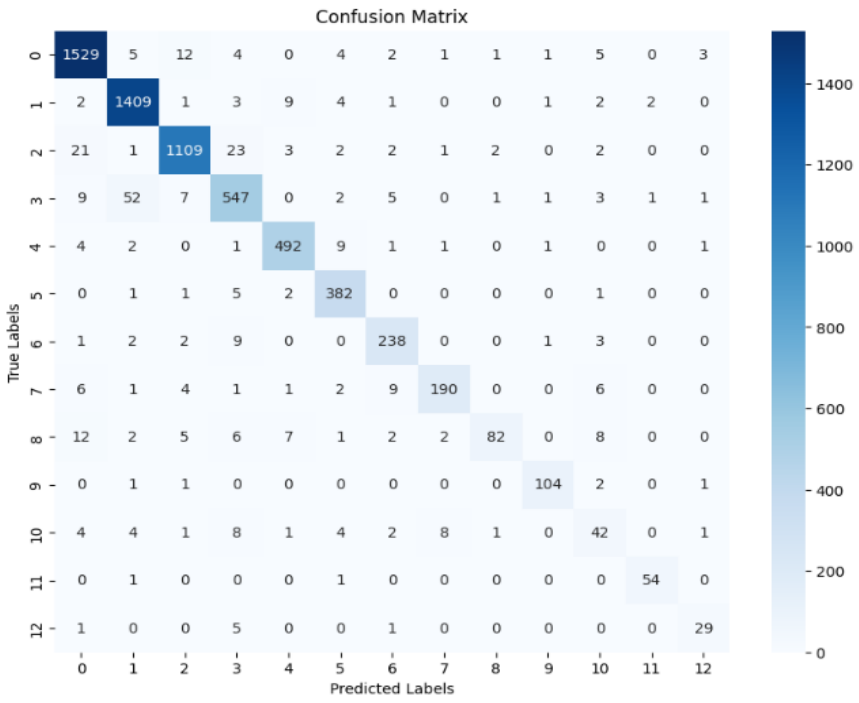
On test data

Classification Report in Color:

	precision	recall	f1-score	support
0	0.962240	0.975750	0.968948	1567.000000
1	0.951384	0.982566	0.966724	1434.000000
2	0.970254	0.951115	0.960589	1166.000000
3	0.893791	0.869634	0.881547	629.000000
4	0.955340	0.960938	0.958130	512.000000
5	0.929440	0.974490	0.951432	392.000000
6	0.904943	0.929688	0.917148	256.000000
7	0.935961	0.863636	0.898345	220.000000
8	0.942529	0.645669	0.766355	127.000000
9	0.954128	0.954128	0.954128	109.000000
10	0.567568	0.552632	0.560000	76.000000
11	0.947368	0.964286	0.955752	56.000000
12	0.805556	0.805556	0.805556	36.000000
accuracy	0.943313	0.943313	0.943313	0.943313
macro avg	0.901577	0.879237	0.888050	6580.000000
weighted avg	0.943095	0.943313	0.942481	6580.000000

class wise accuracy on test c

	Class	Accuracy
0	PROPN	0.992980
1	ADP	0.995816
2	NOUN	0.981990
3	VERB	0.872814
4	DET	0.986328
5	PRON	0.979592
6	AUX	0.949219
7	ADJ	0.895455
8	NUM	0.685039
9	CCONJ	0.944954
10	ADV	0.644737
11	PART	1.000000
12	INTJ	0.944444



Result:

After following the approach discussed above , I was able to achieve accuracy above 98.5% on test set.

Below are the classification report on test set

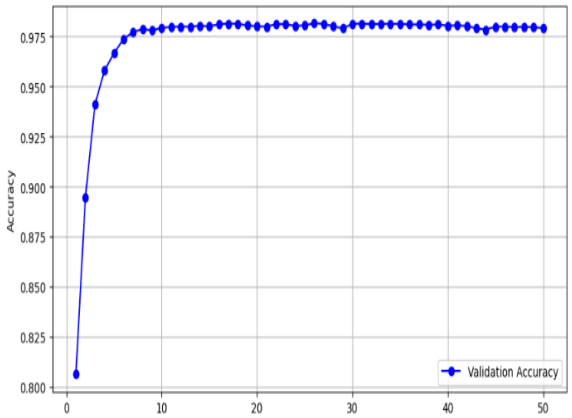
Classification Report :

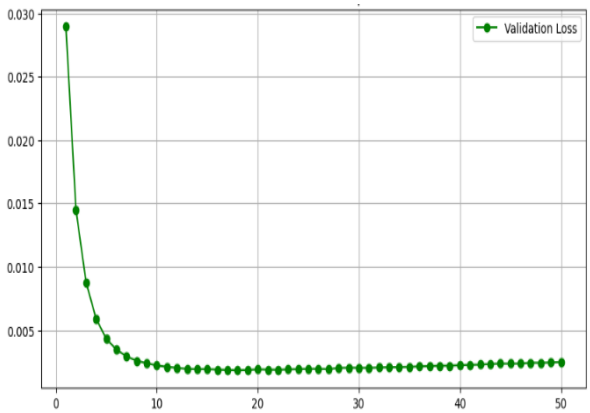
	precision	recall	f1-score	support
0	0.988593	0.995533	0.992051	1567.000000
1	0.986778	0.988842	0.987809	1434.000000
2	0.988832	0.987136	0.987983	1166.000000
3	0.982026	0.955485	0.968574	629.000000
4	0.984375	0.984375	0.984375	512.000000
5	0.972569	0.994898	0.983607	392.000000
6	0.984556	0.996094	0.990291	256.000000
7	0.908696	0.950000	0.928889	220.000000
8	0.967742	0.944882	0.956175	127.000000
9	1.000000	0.990826	0.995392	109.000000
10	0.863636	0.750000	0.802817	76.000000
11	0.981818	0.964286	0.972973	56.000000
12	1.000000	0.944444	0.971429	36.000000
accuracy	0.981915	0.981915	0.981915	0.981915
macro avg	0.969971	0.957446	0.963259	6580.000000
weighted avg	0.981849	0.981915	0.981783	6580.000000

class wise accuracy on test data

	Class	Accuracy
0	PROPN	0.998086
1	ADP	0.995816
2	NOUN	0.987993
3	VERB	0.890302
4	DET	0.992188
5	PRON	0.984694
6	AUX	0.988281
7	ADJ	0.936364
8	NUM	0.858268
9	CCONJ	1.000000
10	ADV	0.763158
11	PART	1.000000
12	INTJ	0.972222

The above figures show different measurements like precision, recall, and F1-score, which balance precision and recall by calculating their harmonic mean. Below, there are plots that show the validation accuracy and loss during training. These plots analyze how well the model performs with each training epoch.





Accuracy vs epoch graph for dev set

Loss vs epoch graph for dev set

The results demonstrate that as the number of epochs increase, there is a reduction in overall loss and an increase in accuracy. This indicates that the model has been effectively trained.

5 Analysis:

According to my analysis the RNN(LSTM) perform better Post tagging then FFNN because it can capture sequential dependencies in the input data, which is crucial for understanding language structure.

LSTM model: test set accuracy 98.5% on

```
input_dim = 64
output_dim = 13
hidden_dim = 64
DROPOUT=0.5
lr=0.005
num_epochs = 50
Bidirectional=True
Layer_number=2
```

FFNN model: test set accuracy 97.5% approx. on

```
embedding_dim = 601

hidden_dim = 64
output_dim = 13  # Number of POS tags
batch_size = 32
num_epochs = 50
L_R=0.0001
```

Some input output sample for my FFNN model

```
Your Sentense: i want an early upgrade
word --> tag
i --> PRON
want --> VERB
an --> DET
early --> ADJ
upgrade --> NOUN
```

Below are the some input output result of my model on RNN LSTM

```
Your Sentense: i want an early upgrade
word --> tag
i --> PRON
want --> VERB
an --> DET
early --> ADJ
upgrade --> NOUN
```

```
Your Sentense: An apple a day keeps the doctor away
word --> tag
an --> DET
apple --> ADJ
a --> DET
day --> NOUN
keeps --> VERB
the --> DET
doctor --> NOUN
away --> PROPN
the --> DET
doctor --> PROPN
away --> PROPN
```