CS671 - Deep Learning and Applications

Recurrent Neural Networks and Long Short Term Memory

ASSIGNMENT 4 REPORT

submitted by

Aditya Sarkar	Aaron Thomas Joseph	Srishti Ginjala
IIT Mandi	IIT Mandi	IIT Mandi
B19003	B19128	B19084

under the supervision of

Dr. Dileep AD

INDIAN INSTITUTE OF TECHNOLOGY, MANDI

May 2022

Abstract

In this programming assignment, we have improved our understanding of the recurrence operation, recurrent neural networks and long short term memory. The report starts by briefly explaining the datasets used in the experiment. Then the first chapter talks about RNN and the recurrence it uses to perform classification. We also discussed the problems related to RNN. The next chapter deals with LSTMs which partially solved the problems faced by RNNs. However, it also has a bunch of issues which was later solved by having an attention mechanism. A lot of plots have been used to explain the concepts.
This report is made in LaTeX

Contents

Chapter 1

Recurrence for learning

1.1 Brief description of the dataset

We were given two datasets in this assignment :

1. **Handwritten character dataset** : This dataset consists of subset of handwritten characters from Kannada/Telugu script. Each character is a sequence of 2-dimensional points (x and y coordinates) of one stroke of character (pen down to pen up). Each data file included an array of elements. The numbers in each file was read as follows:

   (a) First element indicated the number of 2-d sequential points in that file.

   (b) Second element onwards corresponded to the 2-d sequential data points. They need to be considered in pairs as follows: first 2 numbers (i.e., 2nd and 3rd elements) are corresponding to first sequential point, next 2 numbers (i.e., 4th and 5th elements) are corresponding to second sequential point and so on.

   To ensure that all the characters are in same scale, x and y coordinates in each file was normalized to the range of 0 to 1.

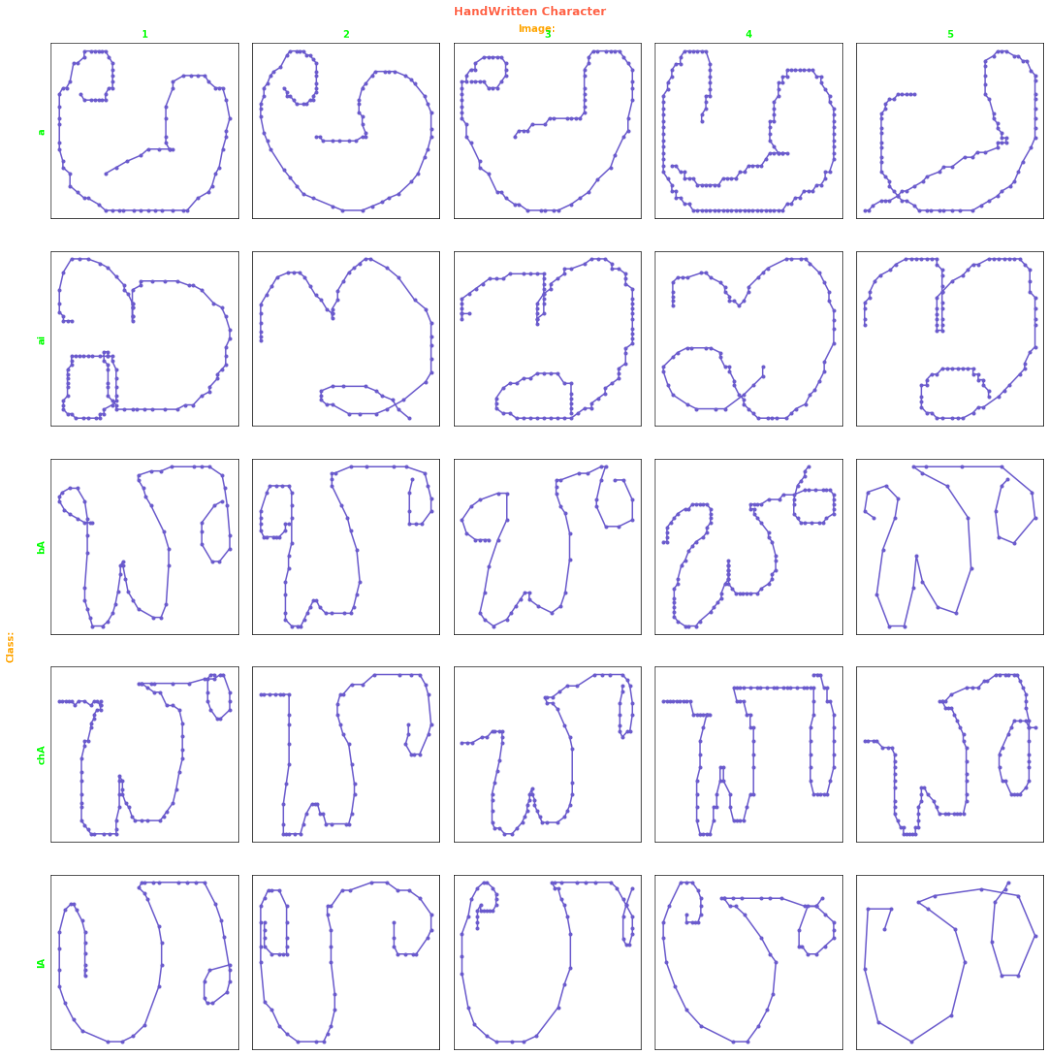

Fig. 1.1: HW dataset

2. **Consonant Vowel (CV) segment dataset** : This dataset consists of subset of CV segments from a conversational speech data spoken in Hindi language. Training and test data are separated and are provided inside the respective CV segment folder. The 39-dimensional Mel Frequency Cepstral Coefficient (MFCC) features extracted frame-wise from utterances of a particular CV segment uttered by multiple people were provided in separate files inside the respective folders corresponding to each class. Each data file was considered as one sample. Each row in a data file indicated one 39-dimensional MFCC feature vector. The number of such feature vectors (rows) depended on the duration of speech segment.

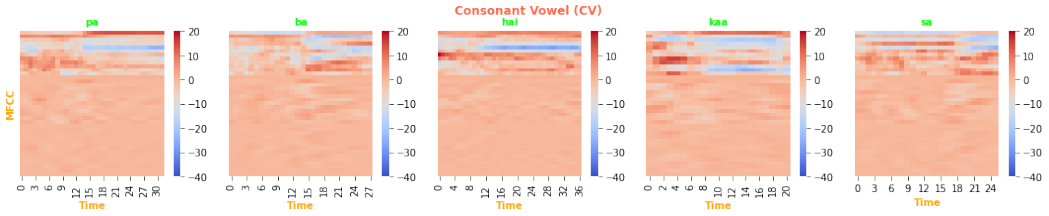

Fig. 1.2: CV dataset

1.2 Feedback networks

In this section, we will be discussing about feedback networks and recurrent neural networks.

1.2.1 Feedback

In control theory, if either the output or some part of the output is returned to the input side and utilized as part of the system input, then it is known as feedback. Feedback brings in stabilisation in a system. For example, an autonomous car without any feedback loop is moving on a road and suppose the traffic light turns red, then the car will keep on moving. This is not desired because the car should stop on red traffic light. Consider the same example, but now, we have placed a camera in the front window. Once the traffic is red, the camera will note it down and send a feedback so as to apply the brakes. As a result, the car

will stop. Thus we can see that the feedback is helpful in cases where we are relying on the past information or knowledge. Feedback is well known for bringing in stabilisation in control systems, but it is also notorious for bringing in complexity. Thus, one should wisely use these concepts for handling any sitation or problem.

1.2.2 Memory

On closely examining the concept of feedback above, one can also view the past information. So the camera is continuously taking pictures while the car is moving. When the picture of red traffic light is passed onto the controller, the brakes are applied. The pictures that the camera is taking can be viewed a history information. This history can be short or can be big depending on the problem. Suppose only one image of the camera is being used by the controller to take decision. Now instead of a real traffic light, if a photo of red traffic light is passed as feedback, the car will again stop. This is again undesirable, because we are not supposed to stop on seeing a photo. Thus, we need a series of images to be passed as feedback, for the controller to make a correct decision. This series is also called a sequence. The length of the sequence can be considered a hyperparameter and varies from problem to problem. This sequence of images, which is being passed to the controller, is also called as memory. The goodness of any feedback model depends on what data and how much data it is storing as memory.

1.2.3 Feedback networks

Now coming back to deep learning, the recurrent blocks in any RNN based architecture is trying to store history information and before predicting any new outcome, this history information is passed as a feedback. Networks which rely on history to make prediction are called feedback networks. RNN is an example of feedback network. It is widely used in cases where we are having sequential data or any data can be converted to a sequential form.

1.2.4 Types of RNN networks

RNNs have their inputs and outputs of various lengths. The RNNs are basically classified into 4 types -

1. One to One - One input and one output
2. One to Many - One input and multiple outputs
3. Many to One - Multiple outputs and single output
4. Many to Many - Multiple inputs and multiple outputs

In RNNs, we prefer hyperbolic tangent as activation function to avoid the exploding and vanishing gradient problems.

1.3 Results and Observations

In this section, we will be focusing on the various experiments that we have conducted and their outcomes. **PLEASE NOTE** that the batch size was taken as 1 and no parity bits was added to the input in all our experiments.

1.3.1 Number of layers in RNN

RNNs are usually preferred in cases where the length of input is not too long due to its inherent problem of long term dependency. In our case, the length is long, so the RNNs were not able to perform well. In praxis, one needs to choose hyperparameters like number of layers wisely. 2 layers are enough to learn any complicated data. Adding more layers bring in less improvement, but it is adding up in computational cost. Thus we have three cases -

1. 1 layer RNN
2. 2 layers RNN
3. Multiple layer RNN

1.3.1.1 One layer RNN

The performance is given below. One layer RNN is not able to handle complex data which can be seen from the loss vs epochs plot.

1.3.1.2 Two layers RNN

Two layers RNN's performance is also bad, due to long term dependency problem. Number of nodes in first layer was 16 and number of layers in second layer was 5. Optimizer is SGD. Performance is -

RNN Performance (HW dataset)		
Operation	HW Data	CV Data
Training	0.64	0.64
Testing	0.49	0.43

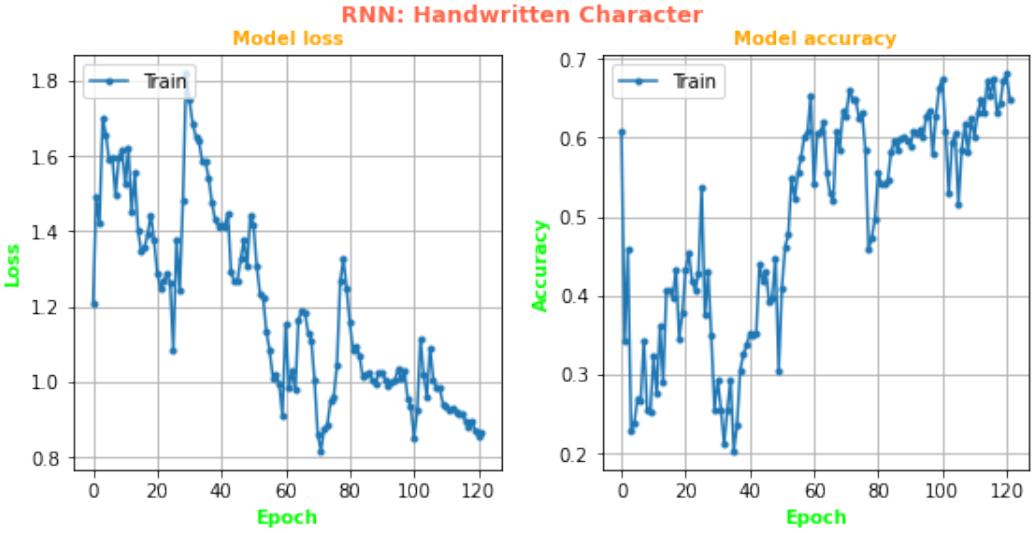

Fig. 1.3: Two layer RNN - Loss vs Epoch - HW dataset

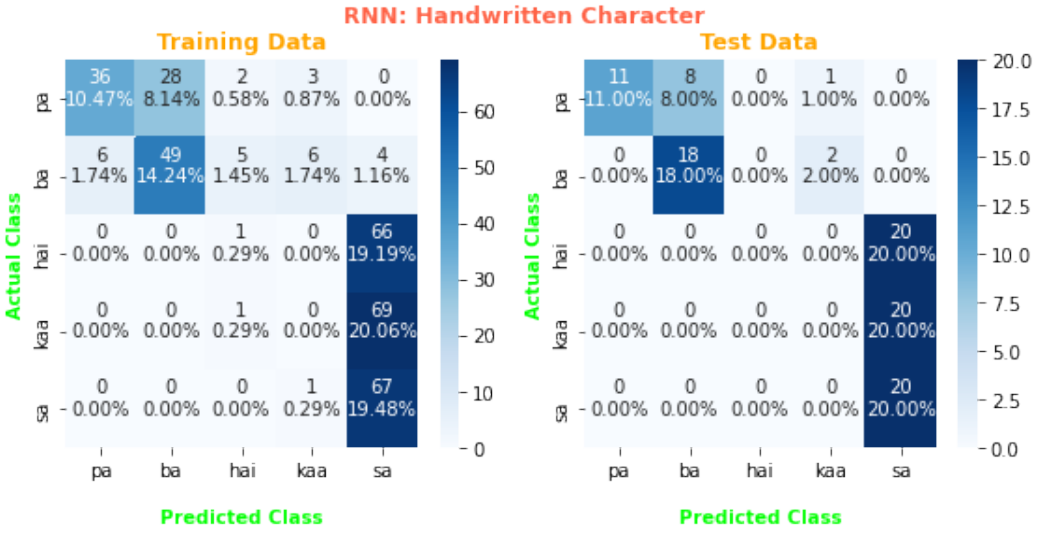

Fig. 1.4: Two layer RNN - CF - HW dataset

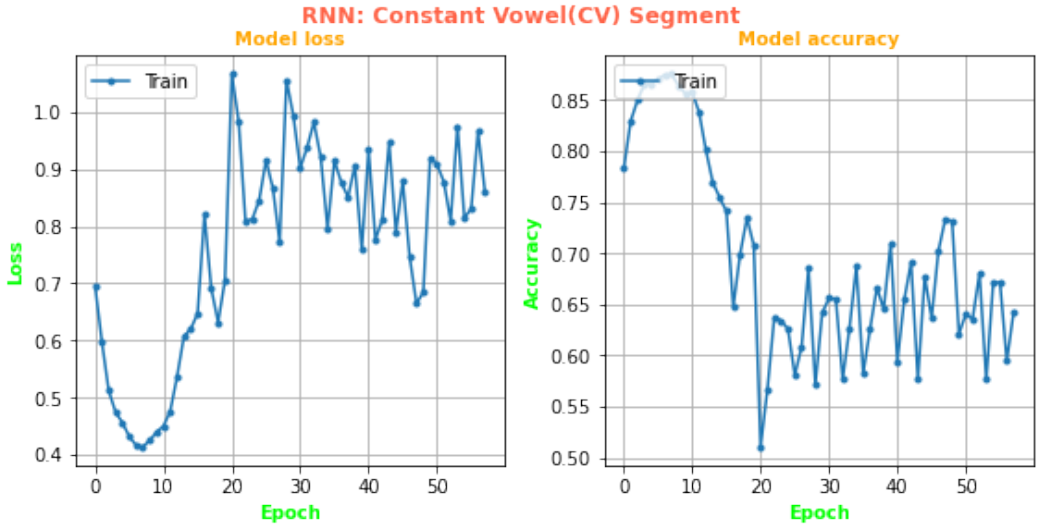

Fig. 1.5: Two layer RNN - Loss vs Epoch - CV dataset

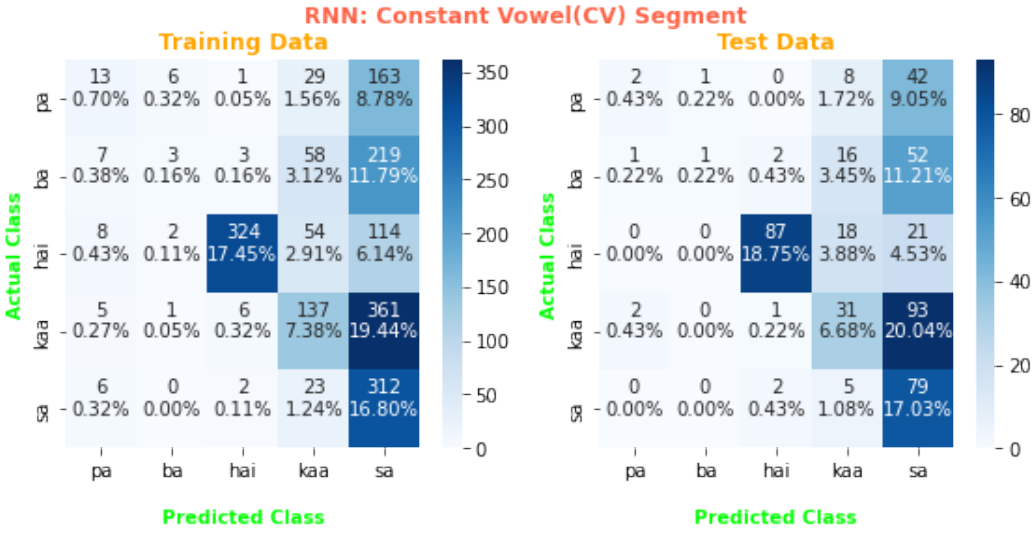

Fig. 1.6: Two layer RNN - CF - CV dataset

1.3.1.3 Multiple layers RNN

Adding multiple layers, as mentioned earlier, doesn't help much. The training accuracy is increasing but validation accuracy is still 20-21 %, that means no learning is taking place. The model is getting overfit.

RNN Performance (HW dataset)		
Number of layers	Training Accuracy	Testing Accuracy
3	0.45	0.21
8	0.55	0.20
16	0.59	0.19
64	0.71	0.20
128	0.89	0.21

1.3.2 Number of nodes in each layer

Again, there is no rule of thumb. In praxis, we considered this -

$$N_h = \frac{N_s}{(\alpha * (N_i + N_0))}$$

where -

N_h = number of neurons

N_i = number of input neurons.

N_0 = number of output neurons.

N_s = number of samples in training data set.

α = an arbitrary scaling factor usually 2-10.

For our experiments, we have varied the alpha from 2 to 10. Moreover, we have also considered other values of N_h. The table illustrating the same is provided below. The bold letter show the best model. The error vs epoch for 16 nodes and 32 nodes are provided below, for both the datasets.

RNN Performance (HW Dataset)		
Number of Nodes	Training Accuracy	Testing Accuracy
4	0.56	0.53
8	0.49	0.44
16	**0.64**	**0.49**
32	0.55	0.41
64	0.71	0.20
128	0.89	0.21

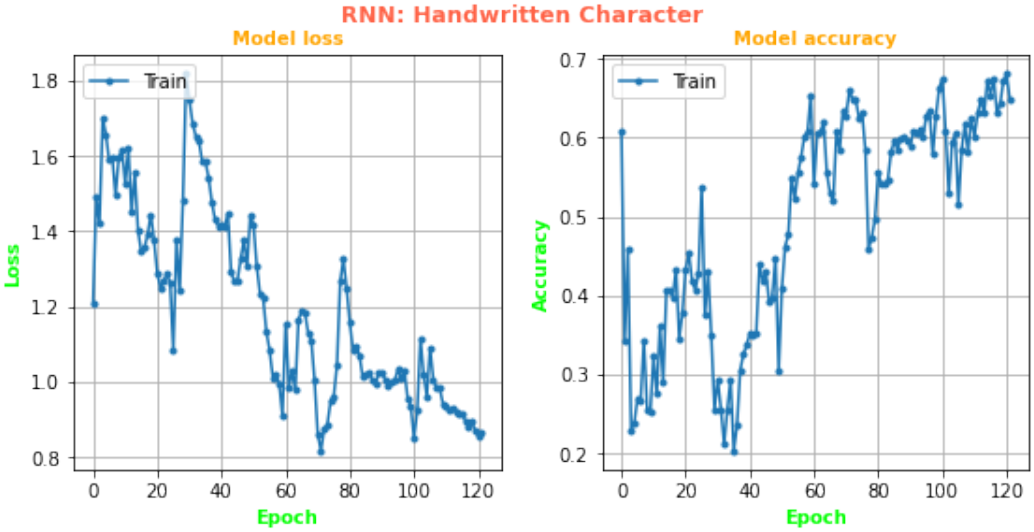

Fig. 1.7: Nodes number = 16 - Loss vs Epoch - HW dataset

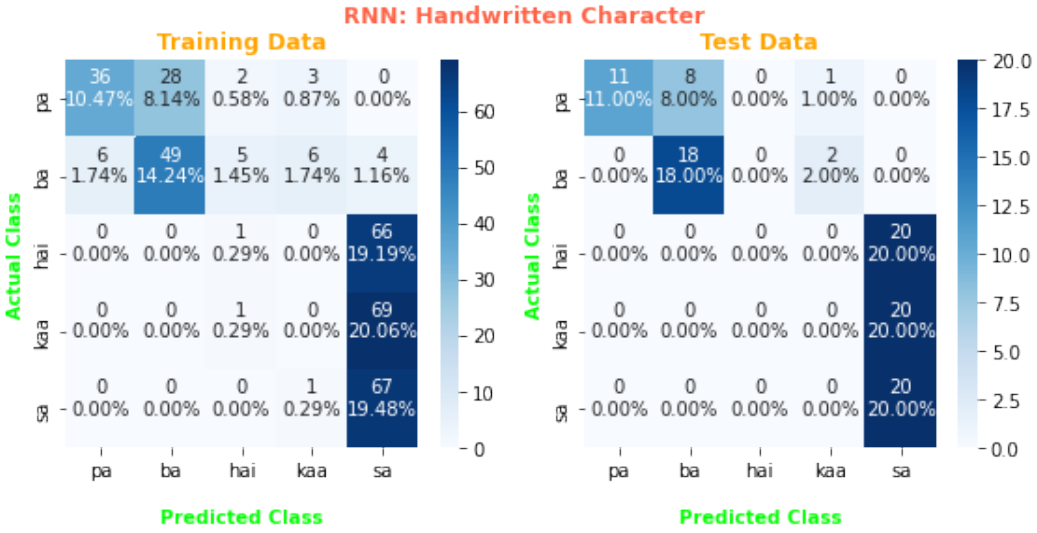

Fig. 1.8: Nodes number = 16 - CF - HW dataset

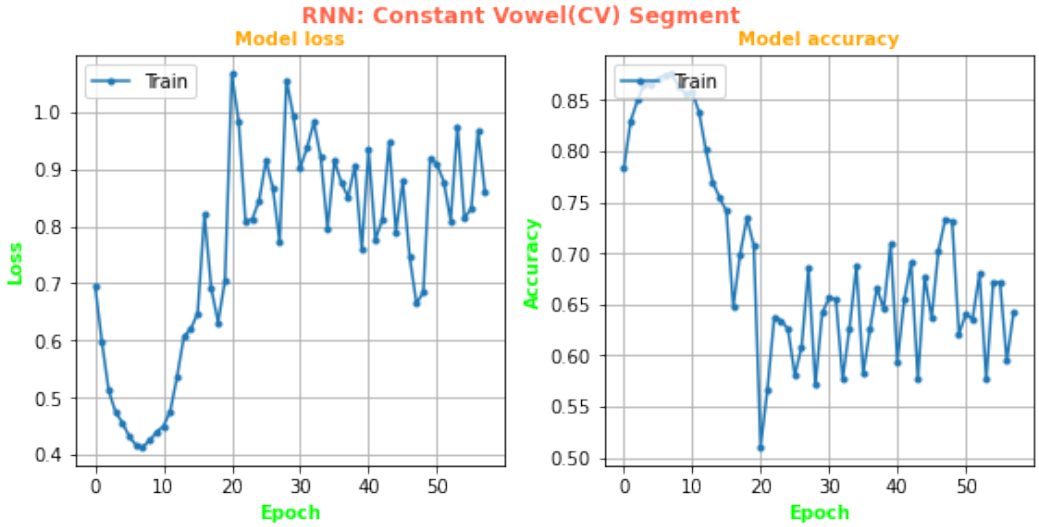

Fig. 1.9: Nodes number = 16 - Loss vs Epoch - CV dataset

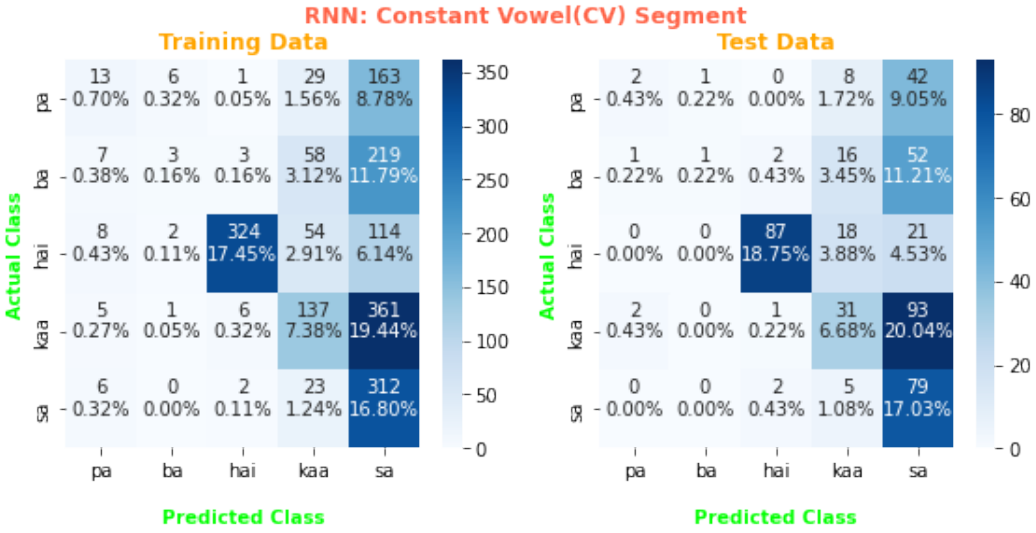

Fig. 1.10: Nodes number = 16 - CF - CV dataset

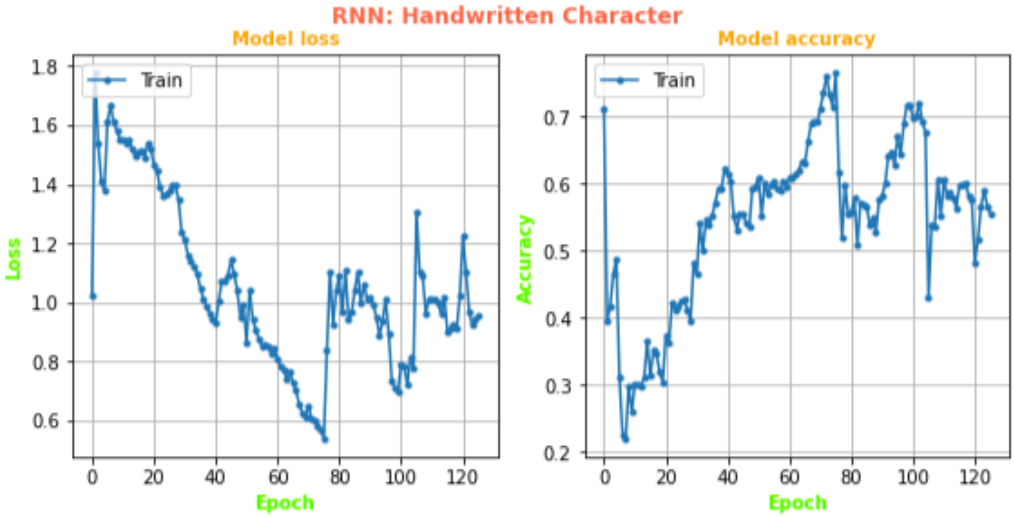

Fig. 1.11: Nodes number = 32 - Loss vs Epoch - HW dataset

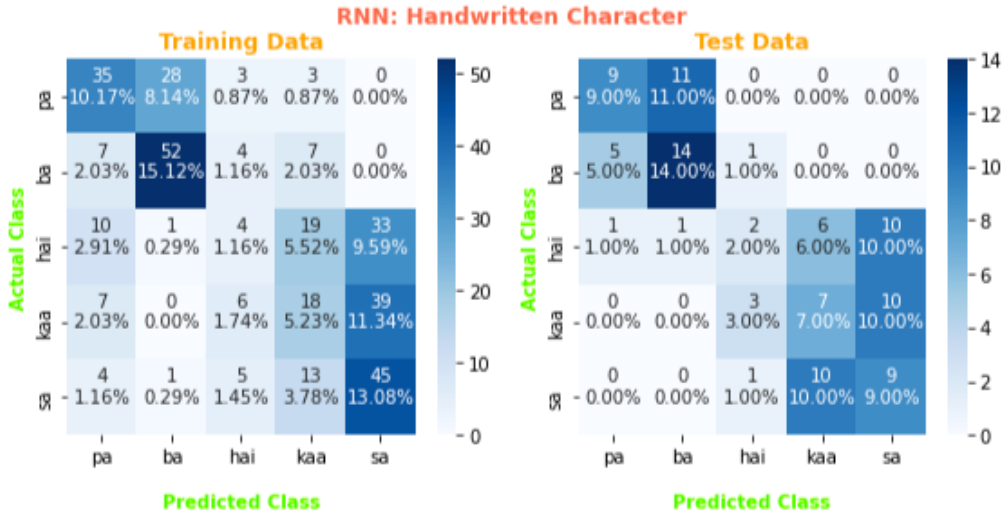

Fig. 1.12: Nodes number = 32 - CF - HW dataset

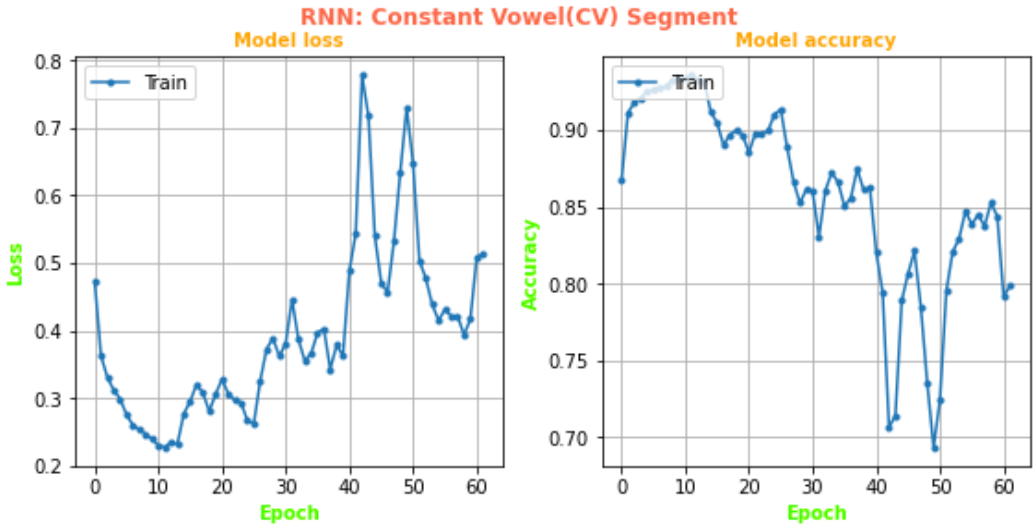

Fig. 1.13: Nodes number = 32 - Loss vs Epoch - CV dataset

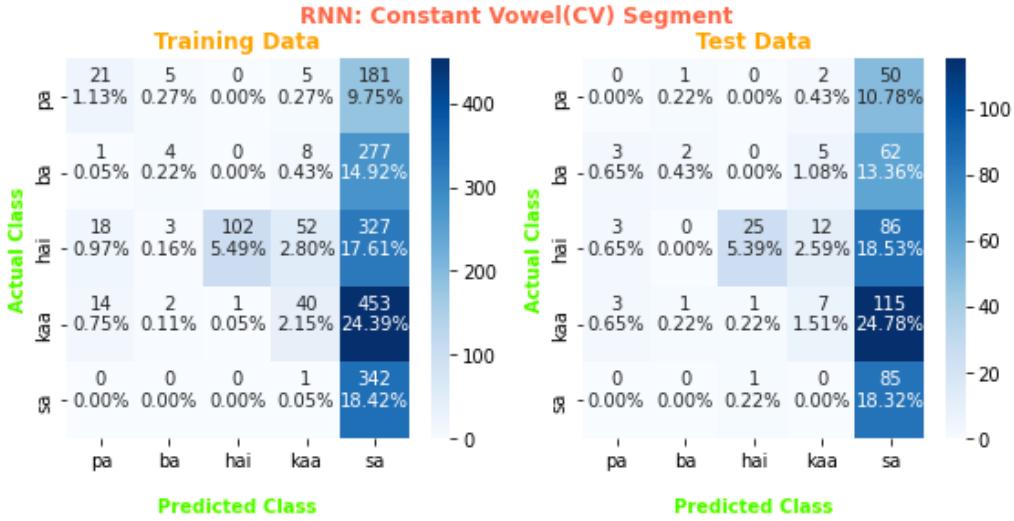

Fig. 1.14: Nodes number = 32 - CF - CV dataset

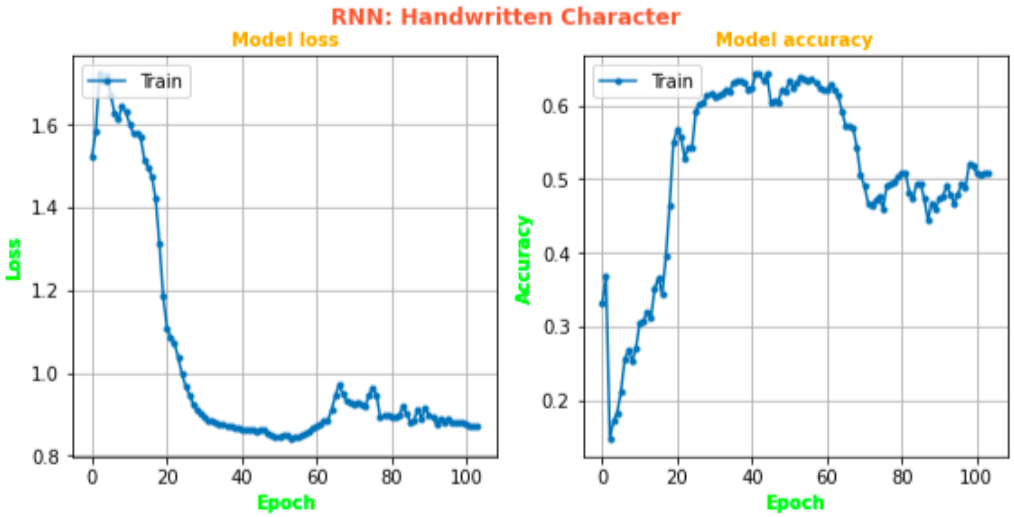

Fig. 1.15: Nodes number = 8 - Loss vs Epoch - HW dataset

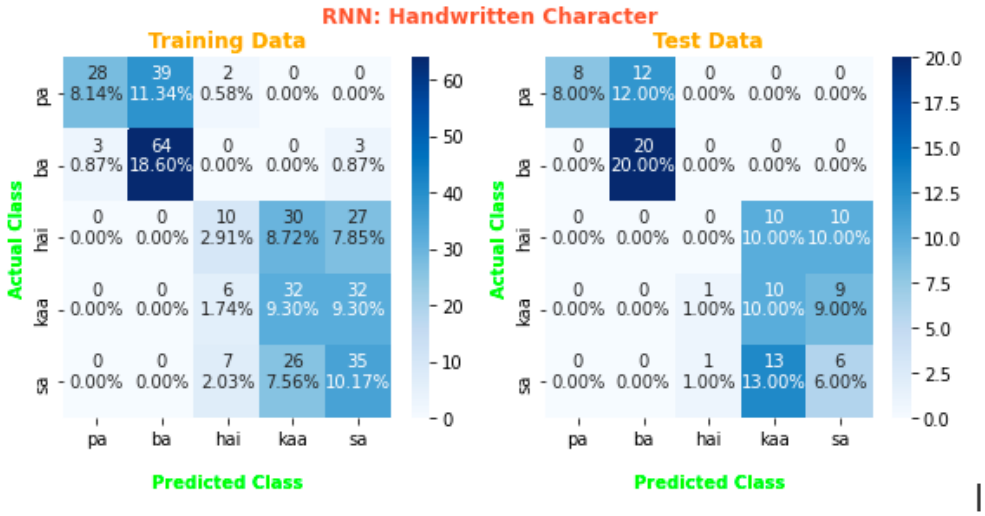

Fig. 1.16: Nodes number = 8 - CF - HW dataset

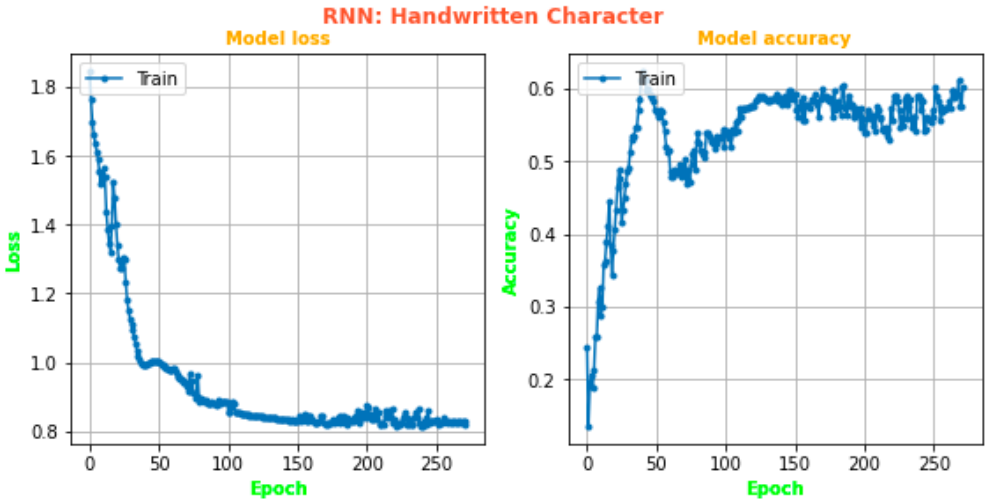

Fig. 1.17: Nodes number = 4 - Loss vs Epoch - HW dataset

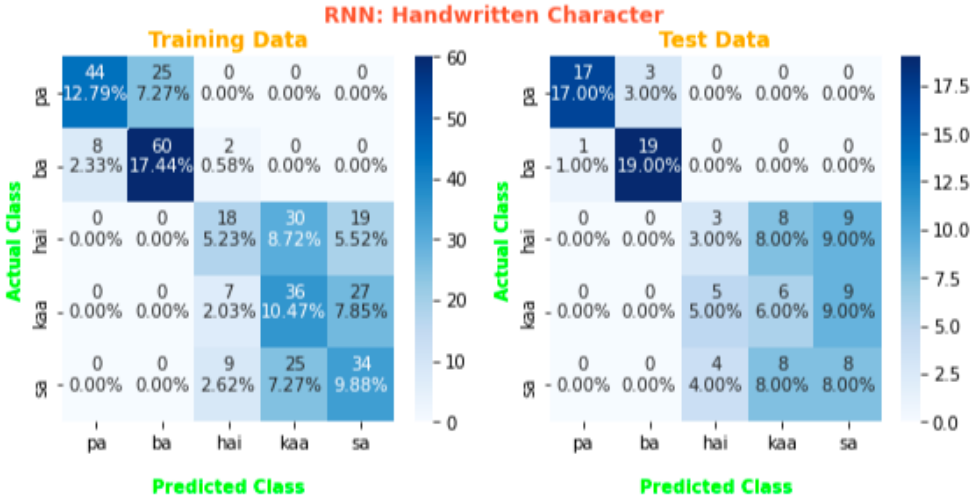

Fig. 1.18: Nodes number = 4 - CF - HW dataset

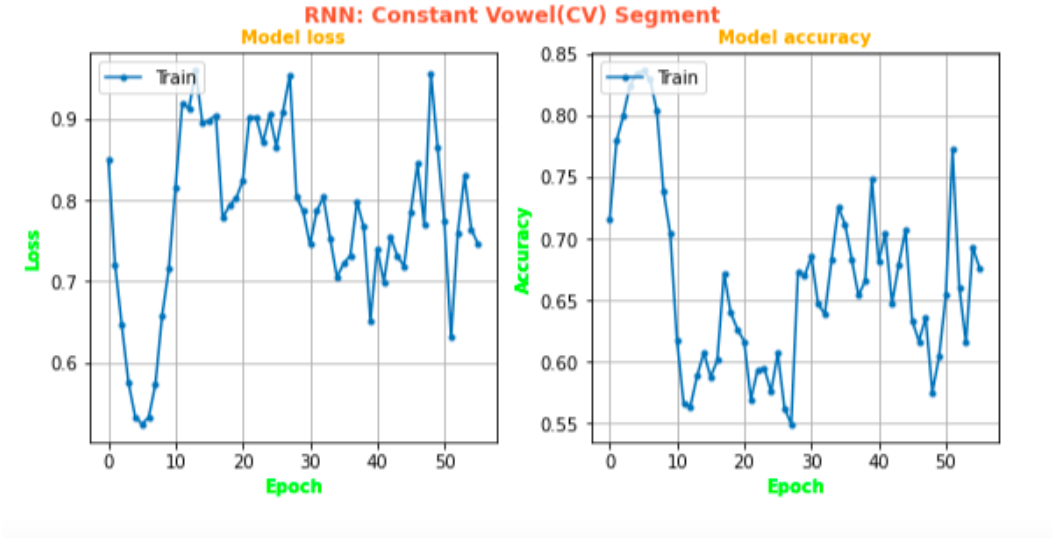

Fig. 1.19: Nodes number = 8 - Loss vs Epoch - CV dataset

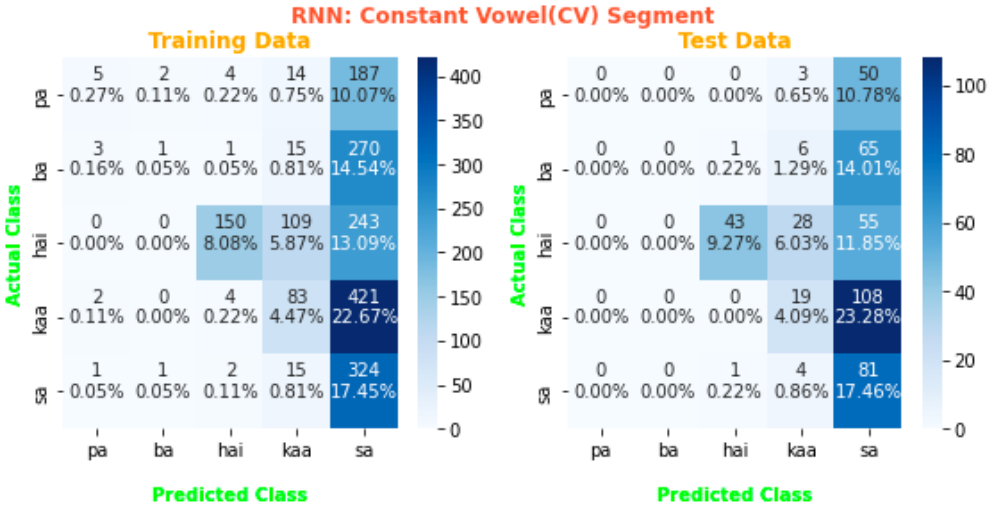

Fig. 1.20: Nodes number = 8 - CF - CV dataset

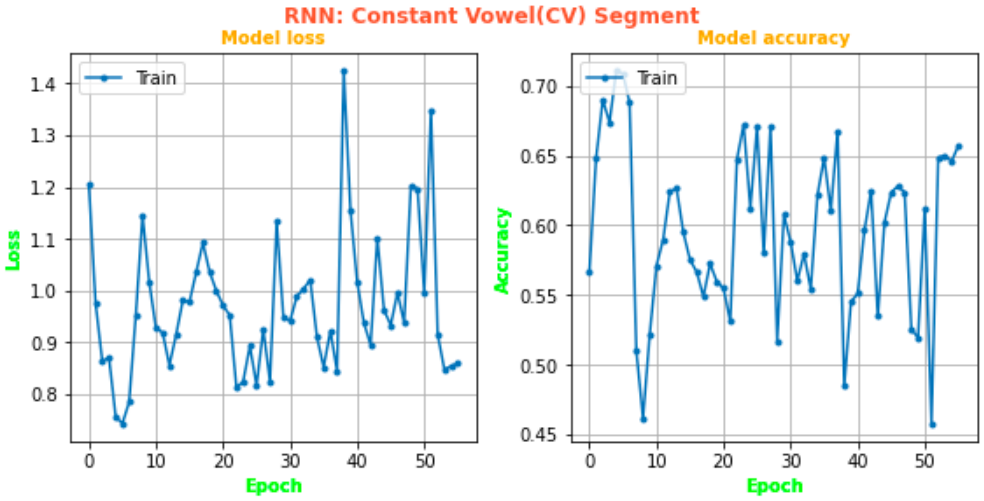

Fig. 1.21: Nodes number = 4 - Loss vs Epoch - CV dataset

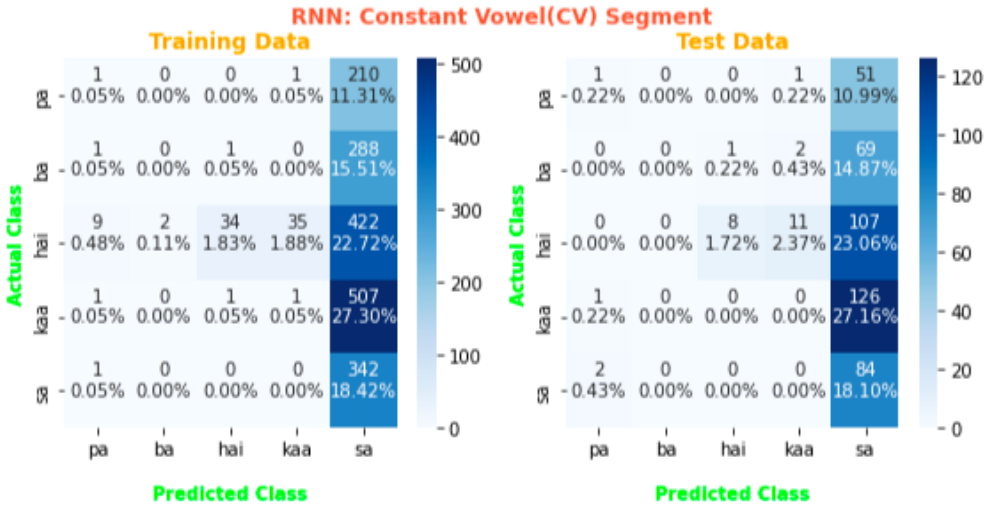

Fig. 1.22: Nodes number = 4 - CF - CV dataset

1.3.3 Dropout fraction

RNNs have a tendency to overfit, due to a large number of parameters being learnt. Dropout was added after each RNN layer. We tried different fractions of dropout which acted like regularization. We used a threshold of 100 epochs here. Dropout wasn't able to improve the performance as seen in the above models. The performance was more or less the same.

1.3.4 Best architecture

We used K-fold cross validation to find the best architecture. Here k is the number of nodes in each layer. Number of layers was fixed to 2. The hyperparameters for the best architecture are presented below -

RNN Performance	
Feature	Value
Number of layers	2
Number of Nodes in Layer 1	16
Number of Nodes in Layer 2	5
Optimizer	SGD
Patience	25
Threshold	10^{-4}

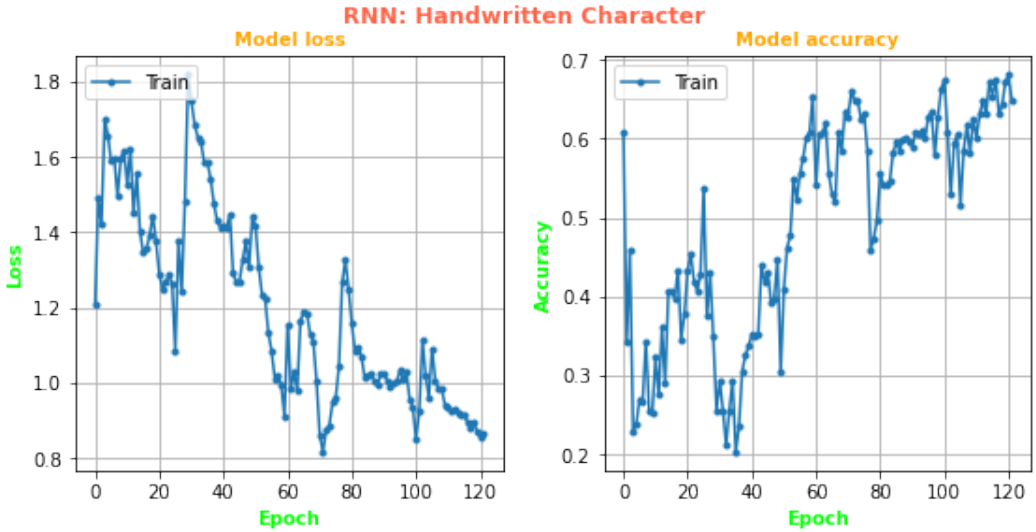

Fig. 1.23: Two layer RNN - Loss vs Epoch - HW dataset

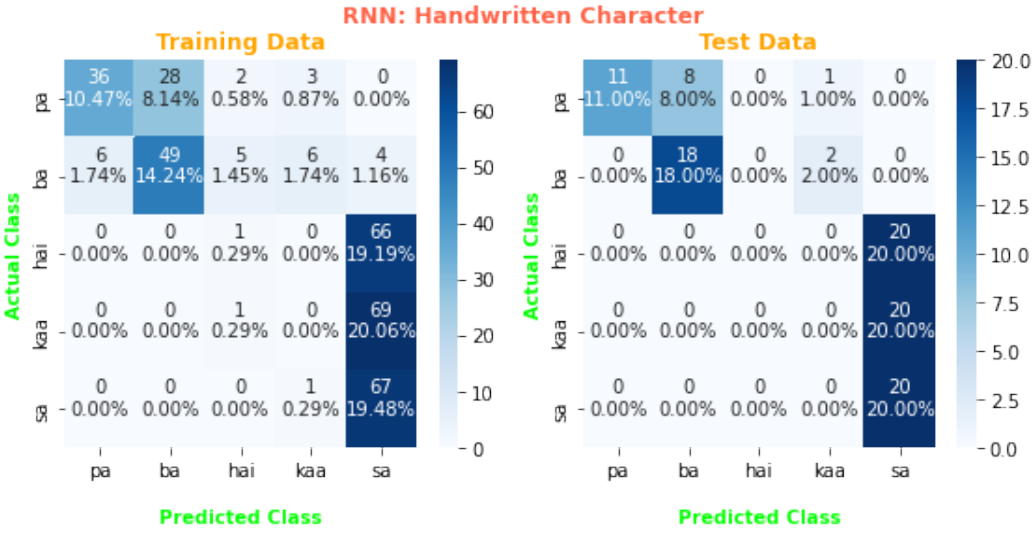

Fig. 1.24: Two layer RNN - CF - HW dataset

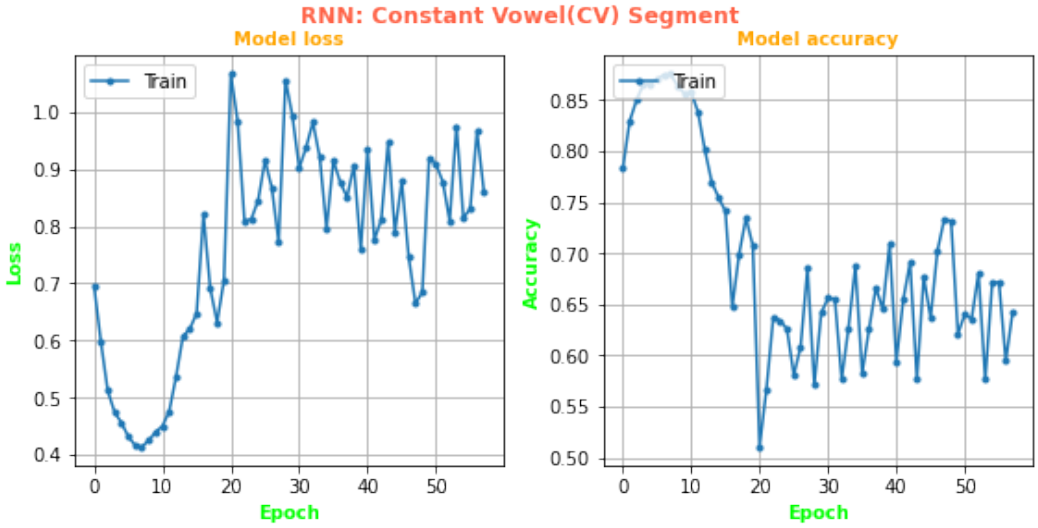

Fig. 1.25: Two layer RNN - Loss vs Epoch - CV dataset

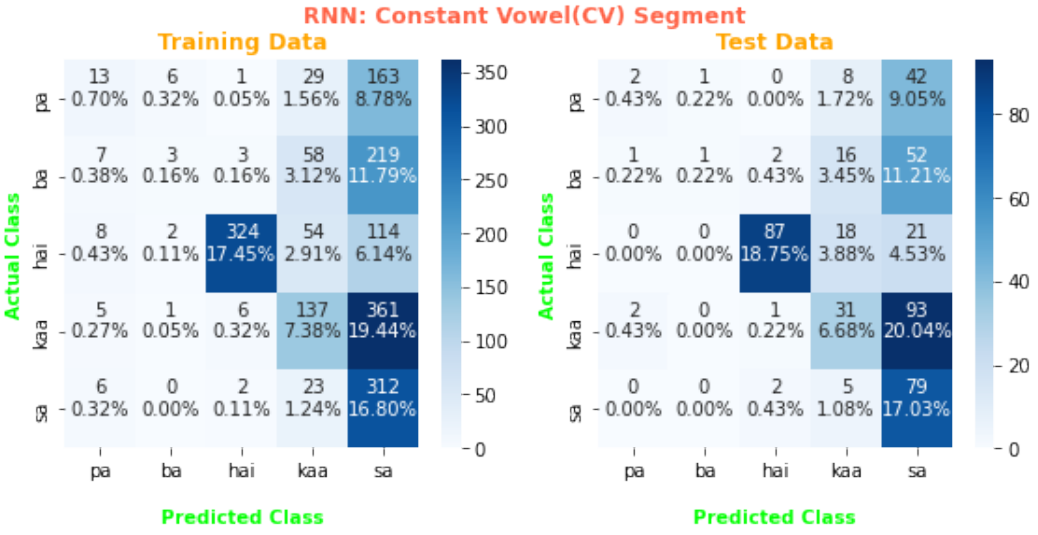

Fig. 1.26: Two layer RNN - CF - CV dataset

Chapter 2

Long Short Term Memory

2.1 Brief description of the dataset

We were given two datasets in this assignment :

1. **Handwritten character dataset** : This dataset consists of subset of handwritten characters from Kannada/Telugu script. Each character is a sequence of 2-dimensional points (x and y coordinates) of one stroke of character (pen down to pen up). Each data file included an array of elements. The numbers in each file was read as follows:

   (a) First element indicated the number of 2-d sequential points in that file.

   (b) Second element onwards corresponded to the 2-d sequential data points. They need to be considered in pairs as follows: first 2 numbers (i.e., 2nd and 3rd elements) are corresponding to first sequential point, next 2 numbers (i.e., 4th and 5th elements) are corresponding to second sequential point and so on.

   To ensure that all the characters are in same scale, x and y coordinates in each file was normalized to the range of 0 to 1.

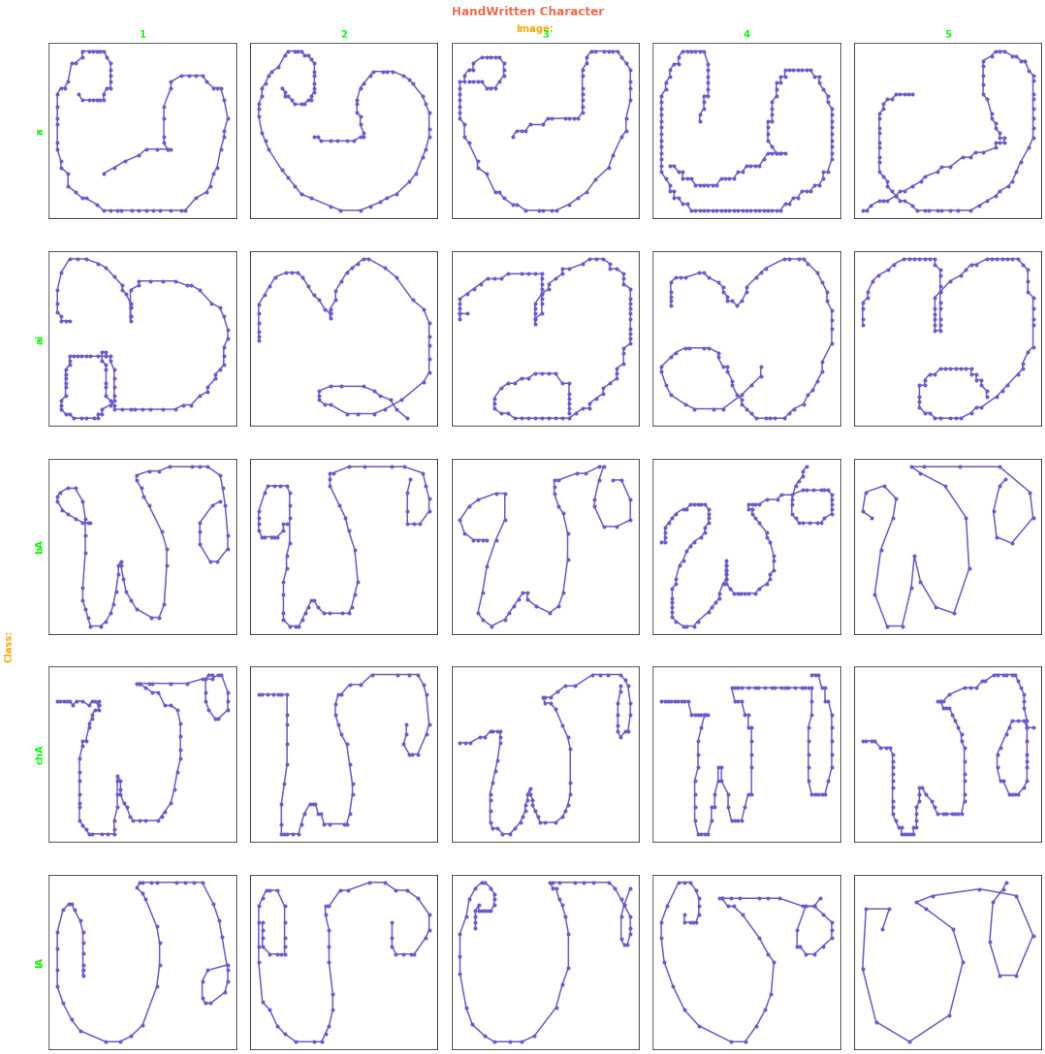

Fig. 2.1: HW dataset

2. **Consonant Vowel (CV) segment dataset** : This dataset consists of subset of CV segments from a conversational speech data spoken in Hindi language. Training and test data are separated and are provided inside the respective CV segment folder. The 39-dimensional Mel Frequency Cepstral Coefficient (MFCC) features extracted frame-wise from utterances of a particular CV segment uttered by multiple people were provided in separate files inside the respective folders corresponding to each class. Each data file was considered as one sample. Each row in a data file indicated one 39-dimensional MFCC feature vector. The number of such feature vectors (rows) depended on the duration of speech segment.

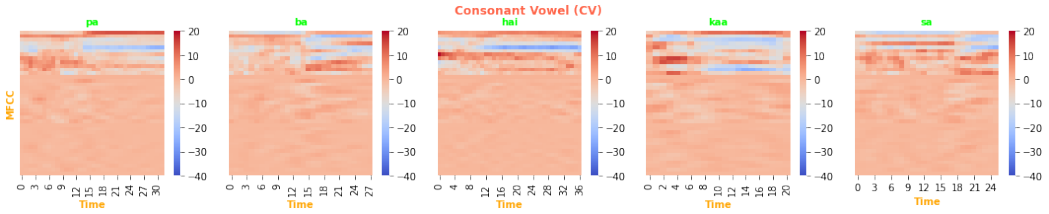

Fig. 2.2: CV dataset

2.2 Long Term Dependency Problem

As discussed above, RNNs are able to connect previous information to the present task. Suppose we have to predict the future action of Akshay Kumar for a fighting scene in a film. In order to do so, we need previous video frames which will act as history and help us predict if the actor will punch, kick or drop back. In this, one should note that the history might have other actions as well. For example, previous video frames might contain an action where Akshay is punching the villain. But the most recent video frame contains a shot where the actor is lifting his leg. So the point is that apart from the history, we need to have some kind of mechanism which will tell me which information I should select to predict the future action. In this case, we will focus on the most nearest frame, but this might not be the case at all times.

Now consider this dialogue from a film : *Ek do teen char paanch chah saat aath nau das*

gyara baarah terah. Suppose the task to predict the last word of this dialogue. It is easy to do so, because one doesn't need any context to predict. After 12, 13 will definitely come. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information. Now consider this statement : *Khaak mazaa hai jeene mein ... jab tak aag na lagi ho apne dushman ke seene mein*. Here the task to predict the second last word. It is hard to do so, why? Because one needs a context for this. Where will there be a fire? It can be in enemy's house, or maybe enemy's office. We need some kind of context. Similar is case with this dialogue : *Boss ka khoon bolta nahi, khaulta hai ... aur jab yeh khaulta hai ... toh yeh boss ek ek ko phodta hai* and task is predicting the last word. No clue again! In the above two cases, the RNN will fail. It is able to not able to memorize the older history if the length of the sequence is too long. Moreover, it is not able to select those words which help in predicting the future word, which is necessary in the above two cases. This problem is well known as Long Term Dependency Problem and has been first illustrated by Bengio in 1994.

2.3 The Theory of Gate and Cells

LSTMs were purposefully designed to handle this. The concept they developed was called gate and cell state. In Electronics, we have something called gates. Gates actually control the flow of electric current through a circuit. Current is like an information which is flowing through the circuit. In LSTMs, gate is used to manage history and present state, in such a way, that they pick only the very important message, that is flowing on a conveyor belt like structure, called cell, which is required for predicting the future. This was able to partially solve the above problem of long term dependency, but it was having a lot of problems. These are -

1. Representation bottleneck.
2. High Memory requirements.
3. Regularization is tough.
4. Exploding and Vanishing gradient problem

5. Recurrence

6. No attention to the words

Some of the problems (2, 4) are easy to understand, as it was taught in the lectures. I will focus more on 1,3,5 and 6.

2.4 LSTM is dead!

Here are detailed explanations for some of the problems because of which the focus of DL community shifted from LSTMs to attention based networks-

2.4.1 Representation Bottleneck

Consider the sentence : *Boss ka khoon bolta nahi, khaulta hai ... aur jab yeh khaulta hai ... toh yeh boss ek ek ko phodta hai*. In this sentence, what is *ek ek* ? Does it mean 11? Does it mean 2? Does it mean each person? This text can have more than one meanings, or in linguistic terms, representations. LSTMs rigidly focus on only 1 type of representation. We should try to see all the variants and then pick up that one which is the most appropriate. And this problem is also called representation bottleneck.

2.4.2 Hard regularization

Batch Normalization does not make any sense for any time series data. We need something called layer normalization which is not present in the LSTM networks. This leads to over-fitting because of the large number of parameters it has to learn. Same problem exists with transformer which uses the concept of attention to resolve long term dependency problem.

2.4.3 Recurrence problem

Both LSTMs and RNNs have the feature of getting unrolled in a recursive fashion. This actually makes the computation for backpropagation algorithm (BPTT) harder when the

sentences are long, and hence one need more resources. Moreover, this reduces any scope of parallelization.

2.4.4 Attention to Words

Every word should be seen with different attention. Each word has some kind of weightage towards a prediction. Consider the sentence : *Boss ka khoon bolta nahi, khaulta hai ... aur jab yeh khaulta hai ... toh yeh boss ek ek ko phodta hai* and task is to predict the second last word. Suppose equal attention to each word, then the predicted word can be *bolta* or *khaulta* or any word related to this. But this is not the case. We need to give more attention to *ek ek*. *ek ek* and *khaulta* does not make any sense, and it should be understood by the architecture. LSTMs fail to understand this, as it has a very primitive way of providing attention, by selecting features.

2.5 Results and Observations

In this section, we will be focusing on the various experiments that we have conducted and their outcomes. **PLEASE NOTE** that the batch size was taken as 1 and no parity bits was added to the input in all our experiments.

2.5.1 Number of layers in LSTM

LSTMs are usually preferred in cases where the length of input is long, but not too long. In our case, the length is decently long, so the LSTMs were able to perform ”good”, but it was *not that good*. In praxis, one needs to choose hyperparameters like number of layers wisely. 2 layers are enough to learn any complicated data. Adding more layers bring in less improvement, but it is adding up in computational cost. Thus we have three cases -

1. 1 layer LSTM
2. 2 layers LSTM
3. Multiple layers LSTM

2.5.1.1 One layer LSTM

The performance is given below. One layer LSTM is not able to handle complex data which can be seen from the loss vs epochs plot.

2.5.1.2 Two layers LSTM

Two layers LSTM's performance is decent, due to long term dependency problem.

LSTM Performance		
Operation	HW Data	CV Data
Training	0.70	0.84
Testing	0.74	0.431

2.5.1.3 Multiple layers LSTM

Adding multiple layers, as mentioned earlier, doesn't help much. This is evident from the below plots. LSTMs take time to run, so we fixed the number of epochs to 120.

LSTM Performance (HW dataset)		
Number of layers	Training Accuracy	Testing Accuracy
3	0.56	0.21
8	0.55	0.20
16	0.65	0.21
64	0.81	0.21
128	0.94	0.21

2.5.2 Number of nodes in each layer

Again, there is no rule of thumb. In praxis, we considered this -

$$N_h = \frac{N_s}{(\alpha * (N_i + N_0))}$$

where -

N_h = number of neurons

N_i = number of input neurons.

N_0 = number of output neurons.

N_s = number of samples in training data set.

α = an arbitrary scaling factor usually 2-10.

For our experiments, we have varied the alpha from 2 to 10. Moreover, we have also considered other values of N_h. The table illustrating the same is provided below.

LSTM Performance (HW Dataset)		
Number of Nodes	Training Accuracy	Testing Accuracy
4	**0.70**	**0.74**
8	0.689	0.66
16	0.64	0.55
32	0.21	0.25
64	0.28	0.21
128	0.29	0.21

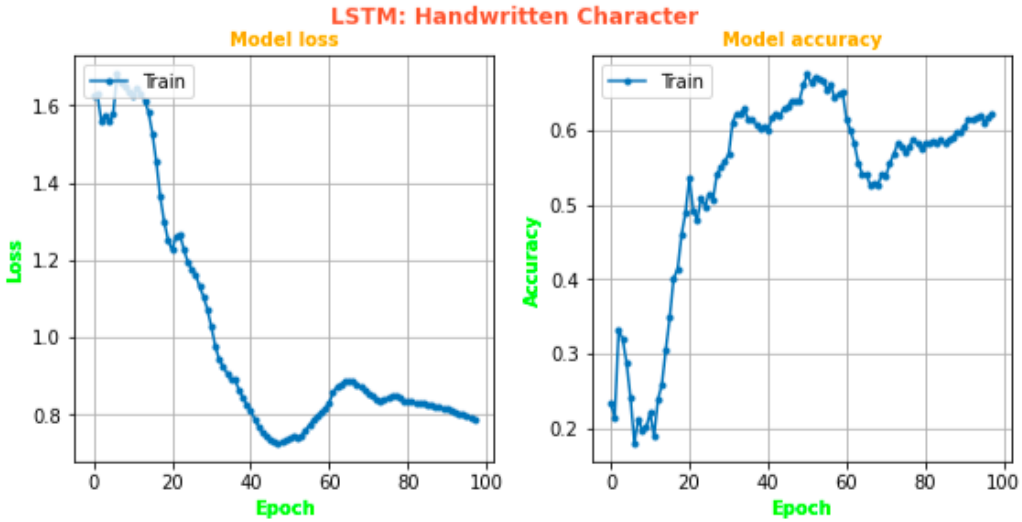

Fig. 2.3: 4 node LSTM - Loss vs Epoch - HW dataset

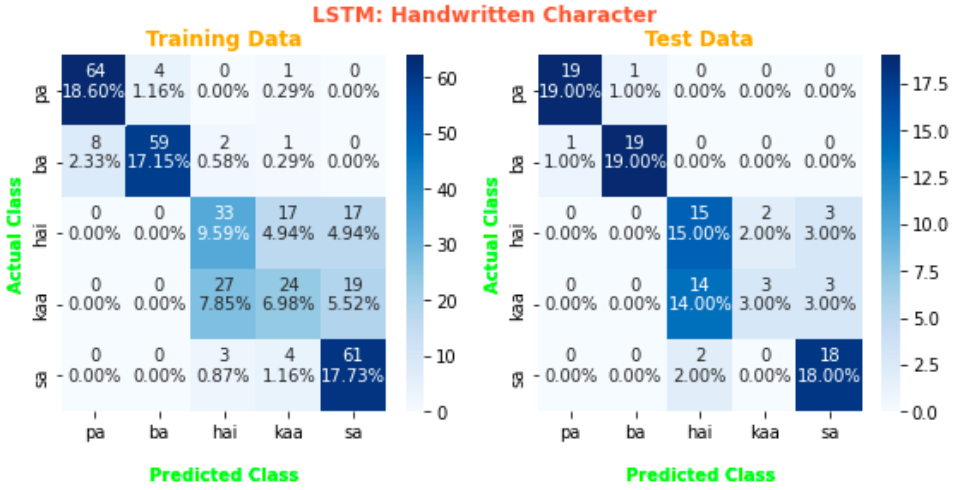

Fig. 2.4: 4 node LSTM - CF - HW dataset

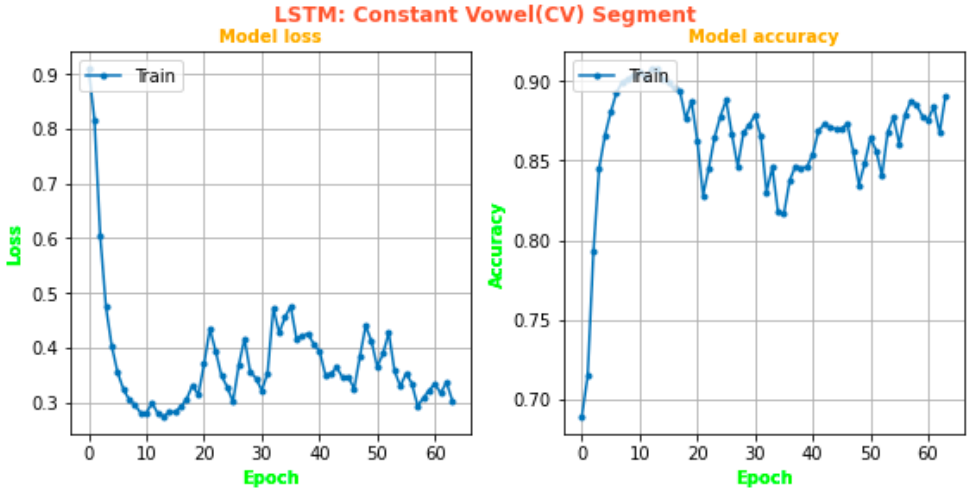

Fig. 2.5: 4 node LSTM - Loss vs Epoch - CV dataset

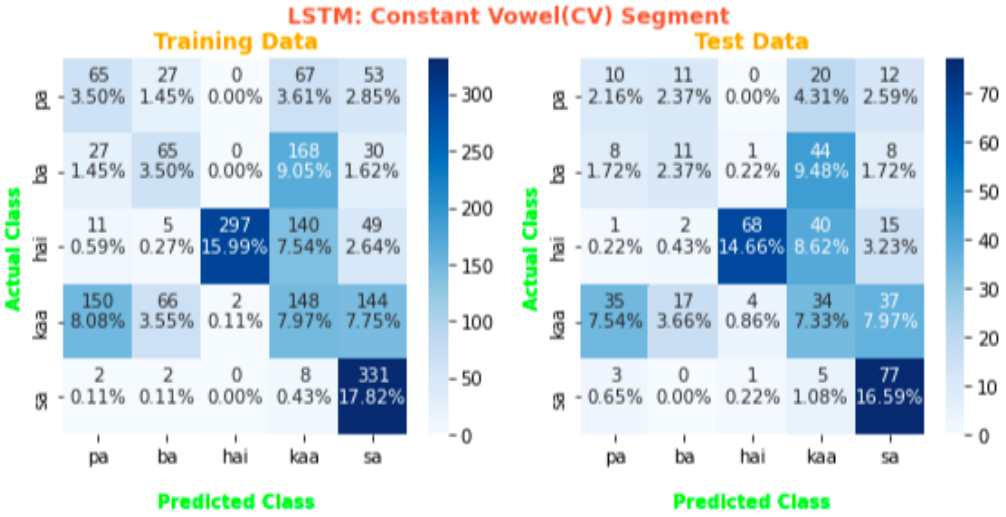

Fig. 2.6: 4 node LSTM - CF - CV dataset

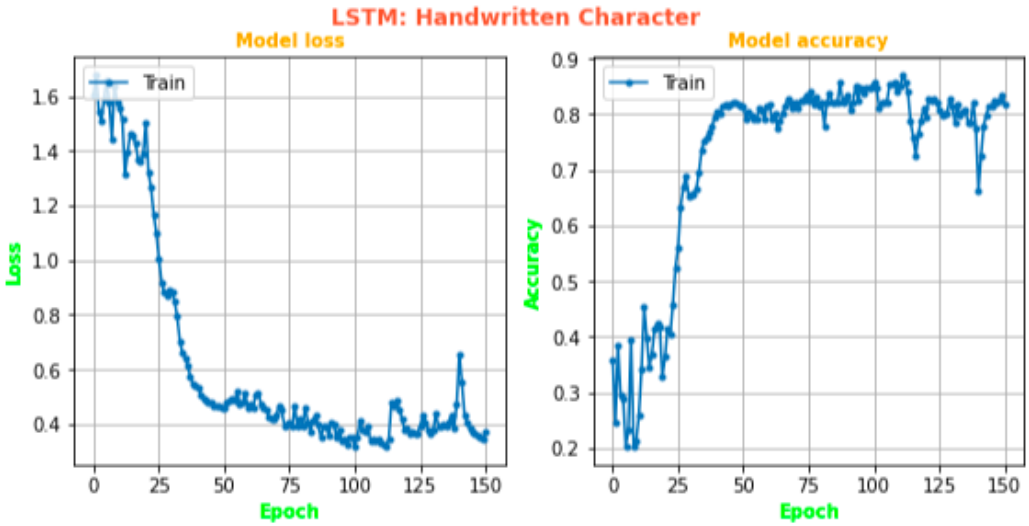

Fig. 2.7: 8 node LSTM - Loss vs Epoch - HW dataset

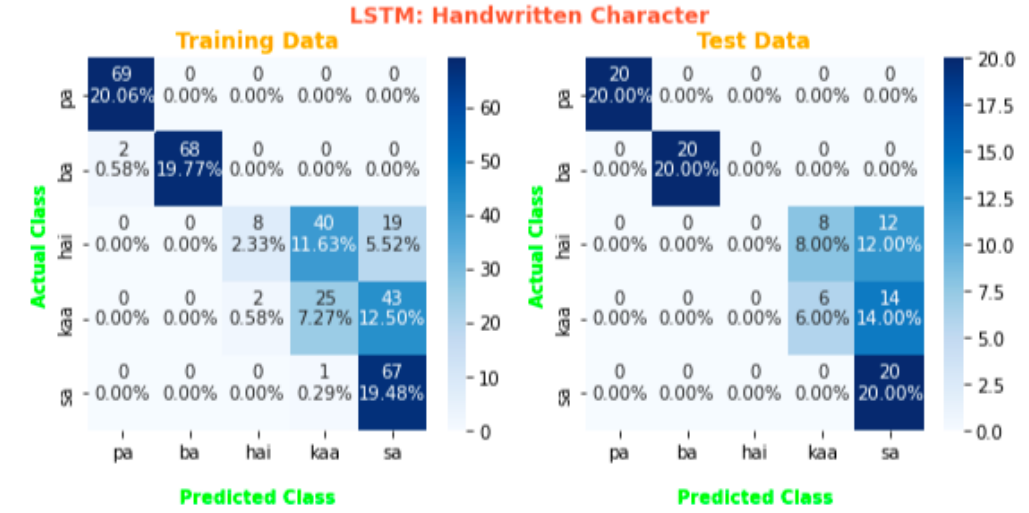

Fig. 2.8: 8 node LSTM - CF - HW dataset

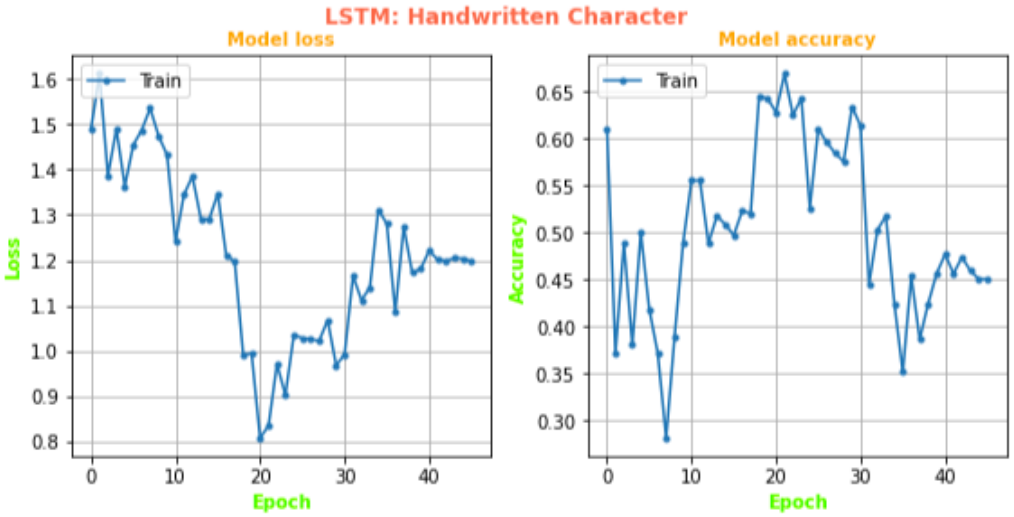

Fig. 2.9: 32 node LSTM - Loss vs Epoch - HW dataset

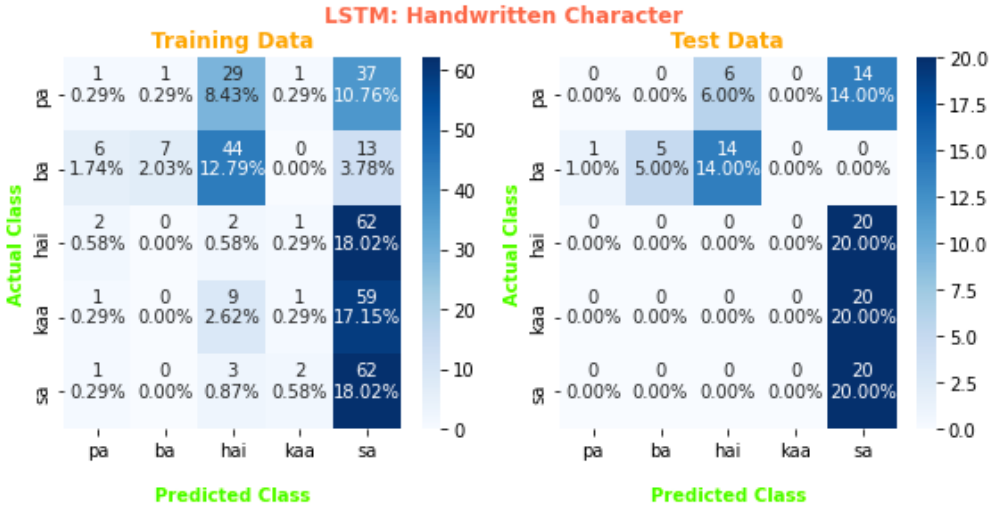

Fig. 2.10: 32 node LSTM - CF - HW dataset

2.5.3 Dropout fraction

LSTMs have a tendency to overfit, due to a large number of parameters being learnt. Dropout was added after each LSTM layer. We tried different fractions of dropout which acted like regularization. We used a threshold of 100 epochs here. Dropout wasn't able to improve the performance as seen in the above models. The performance was more or less the same.

2.5.4 Best architecture

We used K-fold cross validation to find the best architecture. Here k is the number of nodes in each layer. Number of layers was fixed to 2. The hyperparameters for the best architecture are presented below -

LSTM Performance	
Feature	Value
Number of layers	2
Number of Nodes in Layer 1	4
Number of Nodes in Layer 2	5
Optimizer	SGD
Patience	25
Threshold	10^{-4}

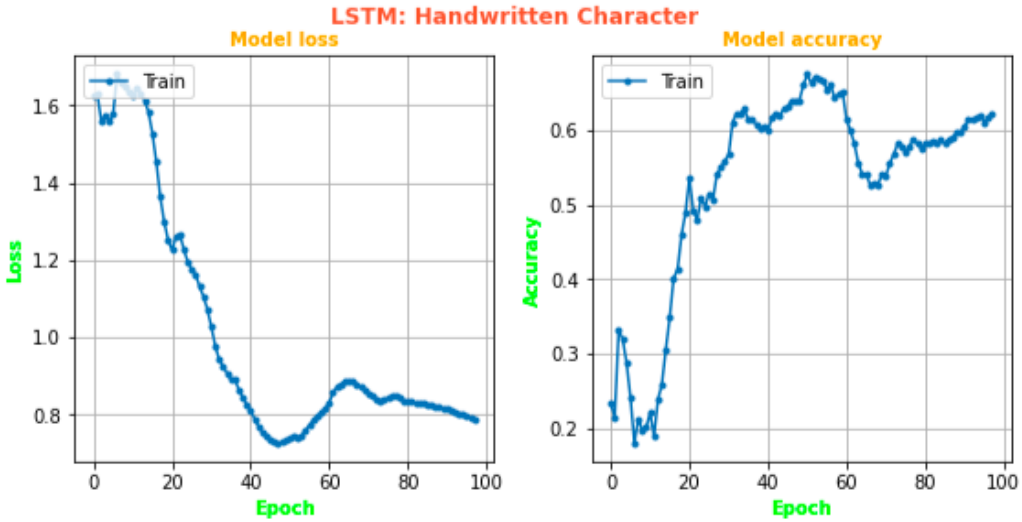

Fig. 2.11: Two layer LSTM - Loss vs Epoch - HW dataset

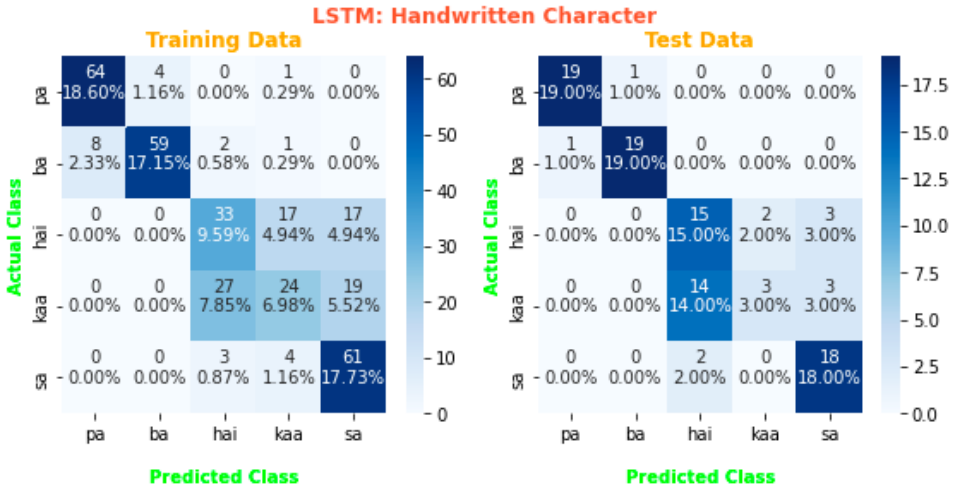

Fig. 2.12: Two layer LSTM - CF - HW dataset

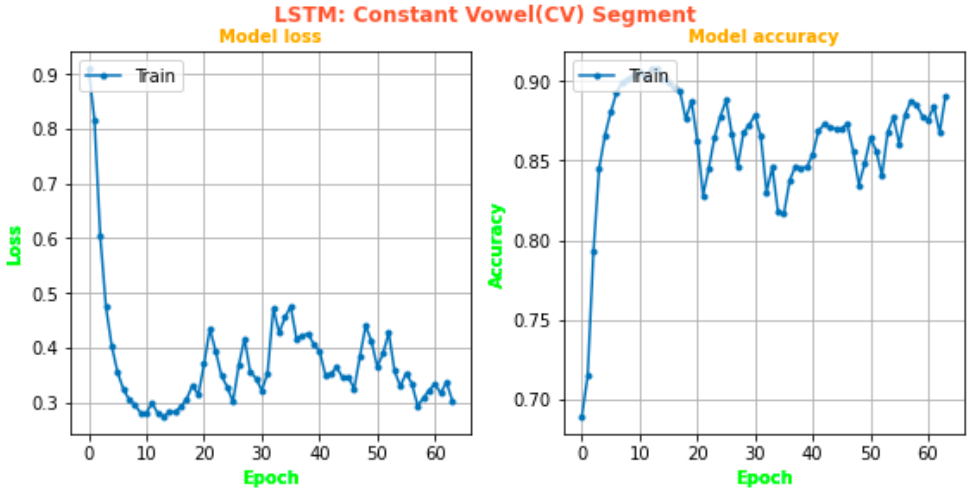

Fig. 2.13: Two layer LSTM - Loss vs Epoch - CV dataset

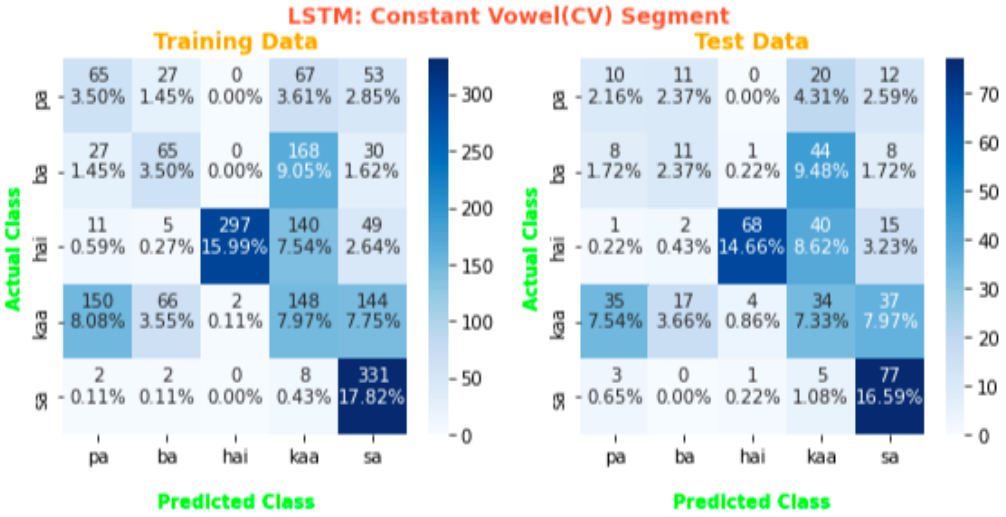

Fig. 2.14: Two layer LSTM - CF - CV dataset

2.5.5 LSTMs vs RNNs

1. Both LSTMs and RNNs solves the problem of sequential learning. The major drawback of RNN is that it is not able to handle long sentences, which reduces its applications in the real life. LSTMs are primarily designed to attack this problem and it has indeed solved this problem using gates but upto a certain level. On increasing the length of the sequence, RNNs perform very badly. For the HW dataset, on comparing the RNN and the LSTM, we can clearly see that LSTM outperforms RNN. Same is the case with CV dataset. This is because both these datasets have decently long sequences.

2. Both models take a good amount of time and resources for getting compiled.

References