Project Report
(Comparative Analysis of CNN, RNN and HAN for Text Classification with GloVe Data Model)

Rachit Shah (rshah25), Sourabh Sandanshi (ssandan)
CSC 591: Algorithms for Data-Guided Business Intelligence
May 03, 2019

BI Use Case and Dataset

We are using these DNNs to classify News articles into different categories. The goal of our project is to predict/classify categories of news based on the content of news articles from the BBC website. To do this we are using the BBC news articles dataset from University College Dublin's Insight project. The dataset consists of 2225 documents from the BBC news website corresponding to stories in five topical categories (business, entertainment, politics, sports, tech) from 2004-2005. We will be performing stratified sampling to prepare a test data set with 20% of the 2225 records (i.e. 445 records). The remaining records are then shuffled and split into training and validation sets with 80% and 20% proportion respectively. Hence, our training set consists of 64% of records, validation 16% and test 20% of the overall data. We have also used another dataset which is larger and has higher number of classes to compare the results of each model for different varied datasets. The other dataset we have used is in-built in scikit-learn's datasets called "20_newsgroups". It consists of 18846 records with 20 class labels like atheism, electronics, politics, graphics, etc.

Table 1 Dataset description

Set	Train	Validation	Test	Labels
Dataset 1 (BBC)	1424	356	445	5
Dataset 2 (20 Newsgroup)	9051	2263	7532	20

Solution Framework

We will use a standard solution framework typically used in NLP problems namely, inputting data, preprocessing, model architecture, training, hyperparameter tuning and prediction. The figure below shows an overview of the steps we have taken to build the model and comparing the output of different models on both of our datasets. As you can see, we have used Google Colaboratory to train our models on cloud using fast GPUs. We first load the data into pandas dataframes and apply preprocessing like removing punctuations, stopwords, lemmatization, etc. Afterwards we tokenize the data using word_index which is fit on the train data. For CNN and RNN, we build a 2D data of (articles, words) while for HAN we build a 3D data of (articles, sentences, words) using the tokenizer. We set hyperparameters like dropout, embedding dimensions of glove model, trainable parameter of embedding layer, bidirectional lstm or simple lstm, etc. We then use hyperparameter tuning to find the best parameters by comparing the validation loss of each model. We build the model architecture corresponding to each of our model and the set hyperparameters. We will expand on the hyperparameter tuning later. After training using early stopping and checkpointing, we predict on the test set to find our test accuracies. We then compare test accuracies for each of our 3 models on both datasets to compare them.

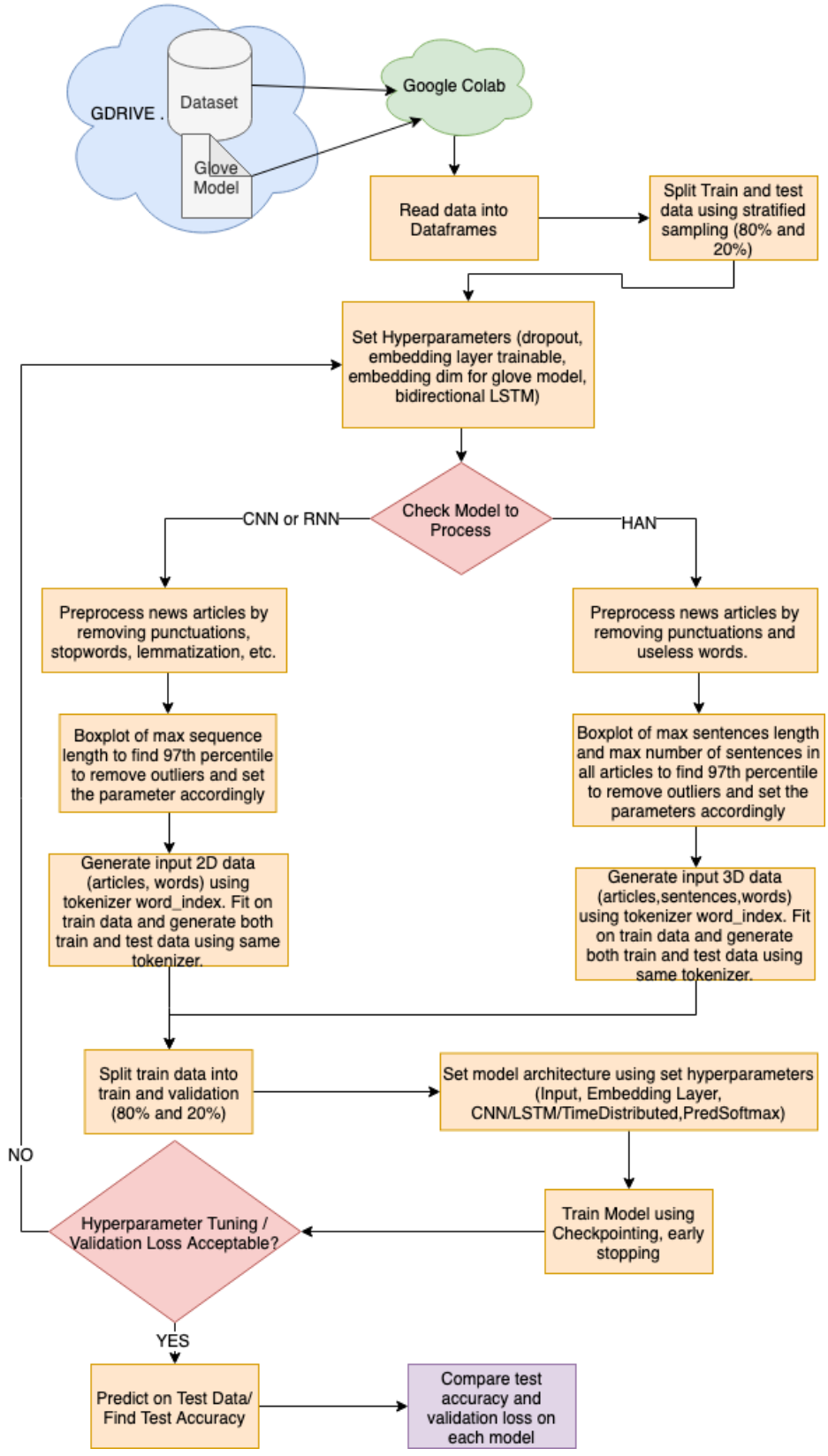


Figure 1 Solution Framework

Model Architectures and Hyperparameters

RNN

RNN is a sequence of neural network blocks that are linked to each other like a chain. Each one is passing a message to a successor. They are networks with loops in them, allowing information to persist. Long Short-Term Memory networks, usually just called “LSTMs” are a special kind of RNN, capable of learning long-term dependencies. We have built our RNN architecture referring to this site. We have a input and embedding layer, a LSTM layer with dropout and a prediction softmax layer at the end.

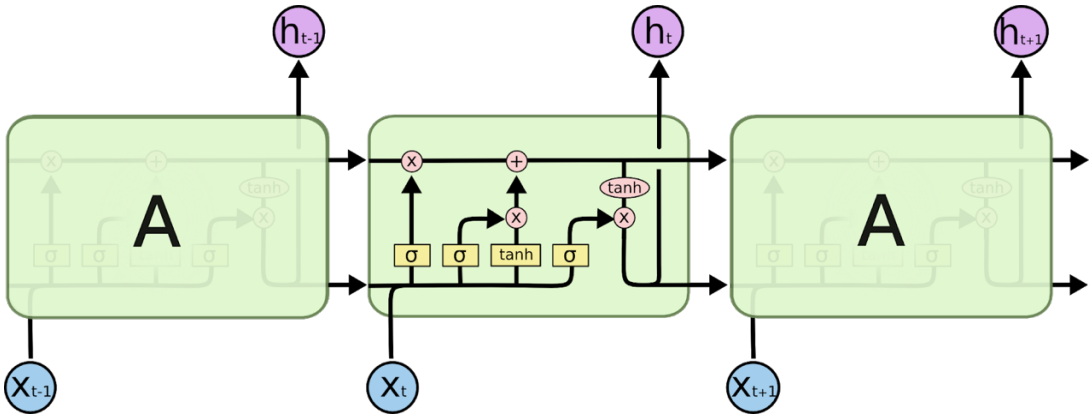


Figure 2 The repeating module in an LSTM. Image Reference : http://colah.github.io/posts/2015-08-Understanding-LSTMs/

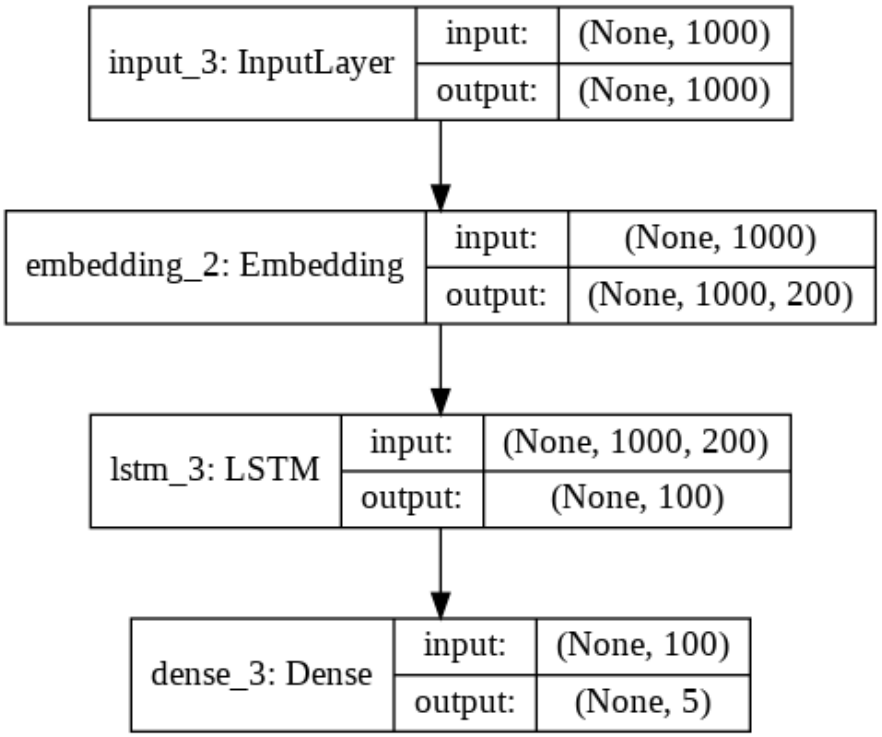


Figure 3 RNN architecture

HAN

HAN has a hierarchical structure that mirrors the hierarchical structure of documents. It has two levels of attention mechanisms applied at the word and sentence-level, enabling it to attend differentially to more and less important content when constructing the document representation. The architecture allows the neural network to understand the meaning of sentences which is a combination of meaning of words. Hence, unlike CNN and RNNs which only considers the words separately, HAN considers the meaning of both sentences and words. Our architecture consists of input and glove embedding layer, time distributed layer, bidirectional lstm layer and a prediction softmax layer. We have referred this site to build our model architecture.

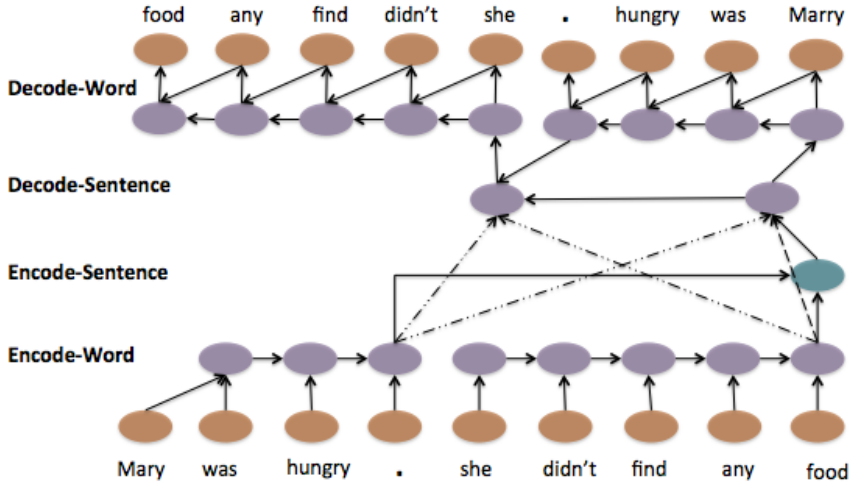


Figure 4 Hierarchical Sequence to Sequence Model with Attention. Image Reference :https://arxiv.org/pdf/1506.01057v2.pdf

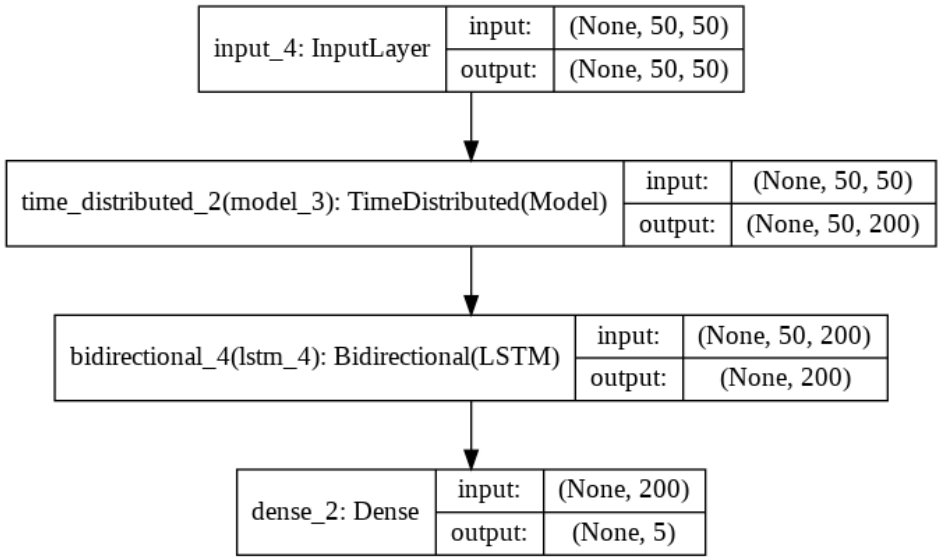


Figure 5 HAN Architecture

CNN

CNNs were initially used for Image processing and computer vision applications. The network consists of series of convolutions (extracting high level features from pixels of images) and pooling (down sampling to capture representative values). In context of Text classification, pixels can be replaced by words. We try to extract and understand the pattern on sentences which help us classify them. In our architecture, we have an input and glove embedding layer, 3 blocks of 1D convolutions, maxpooling and dropout, flatten layer, a fully connected dense layer and a prediction softmax layer at end. The parameters like dropout percentage and embedding dimensions are tuned using hyperparameter tuning. We have used the keras tutorial here to build our CNN architecture.

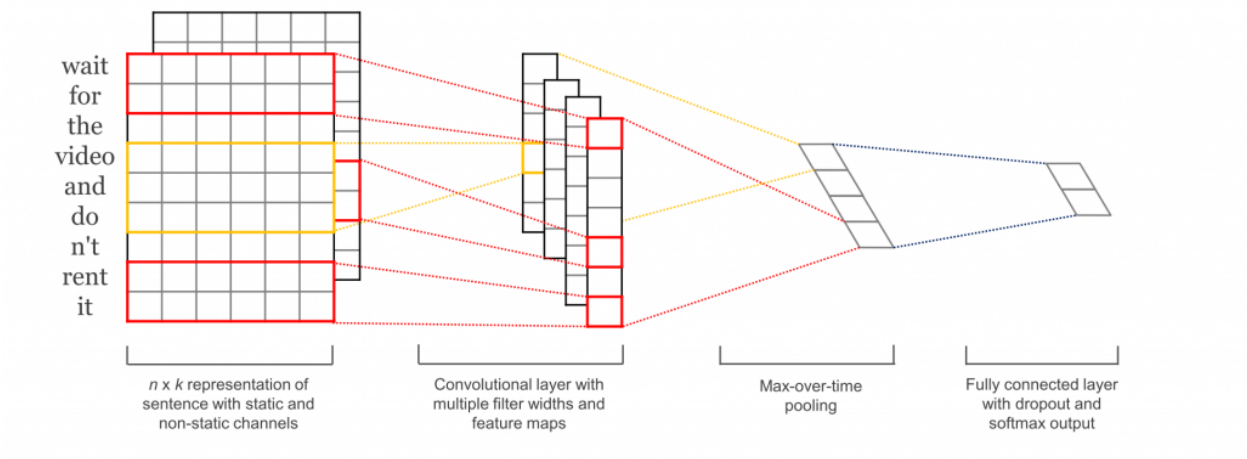


Figure 6 CNN workflow Image Reference : http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/

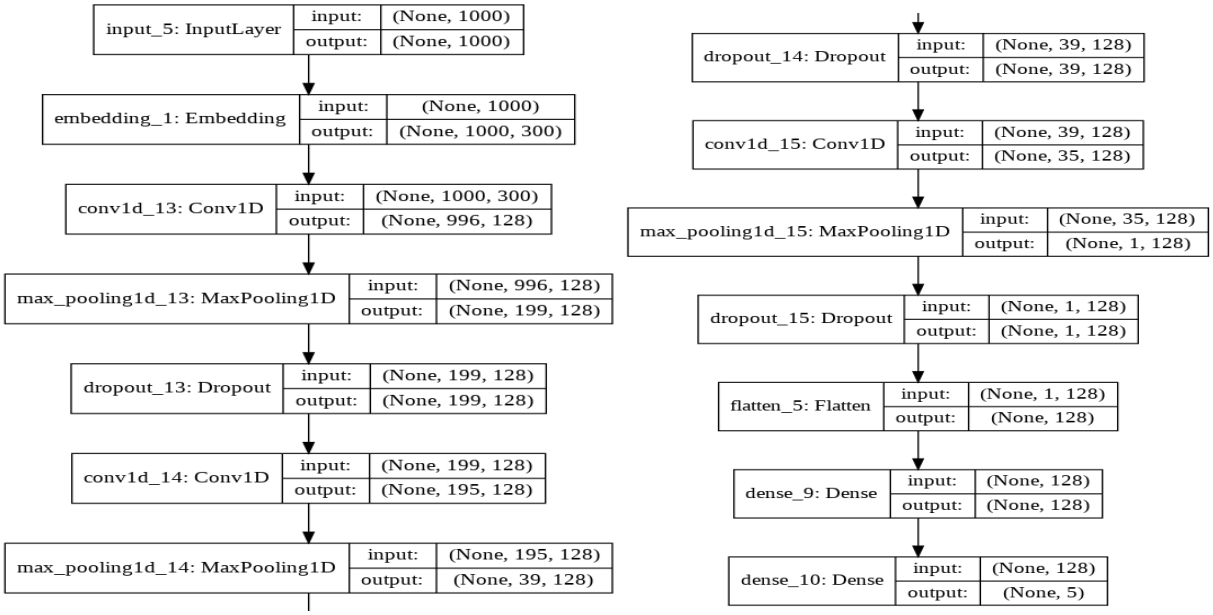


Figure 7 CNN Architecture

Glove Embeddings

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. We used the Wikipedia 2014 + Gigaword 5 pre-trained vectors (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors). We have used this as an embedding layer after input data. We have determined which dimensions of embedding to use for each model using hyperparameter tuning.

Hyperparameter Tuning and Final Parameters

CNN

Table 2 CNN Hyperparameter Tuning

CNN			
Embedding Trainable	**Embedding Dim**	**Dropout**	**Validation Loss**
TRUE	100D	0.2	0.04462
FALSE	100D	0.2	0.06694
TRUE	300D	0.2	0.03009
TRUE	300D	0.4	**0.01323**
TRUE	300D	0.5	0.02558

RNN

Table 3 RNN Hyperparameter Tuning

RNN				
Embedding Trainable	**Embedding Dim**	**Dropout**	**Bidirectional**	**Validation Loss**
TRUE	100D	0.2	FALSE	0.10813
FALSE	100D	0.2	FALSE	0.1482
TRUE	300D	0.2	FALSE	0.13059
TRUE	200D	0.2	FALSE	0.05499
TRUE	200D	0.4	FALSE	0.05803
TRUE	200D	0.2	TRUE	0.06209
TRUE	200D	0.4	TRUE	0.0731
TRUE	200D	0.3	FALSE	**0.04299**

HAN

Table 4 HAN Hyperparameter Tuning

HAN			
Embedding Trainable	**Embedding Dim**	**Dropout**	**Validation Loss**
TRUE	100D	0.2	0.03365
FALSE	100D	0.2	0.06856

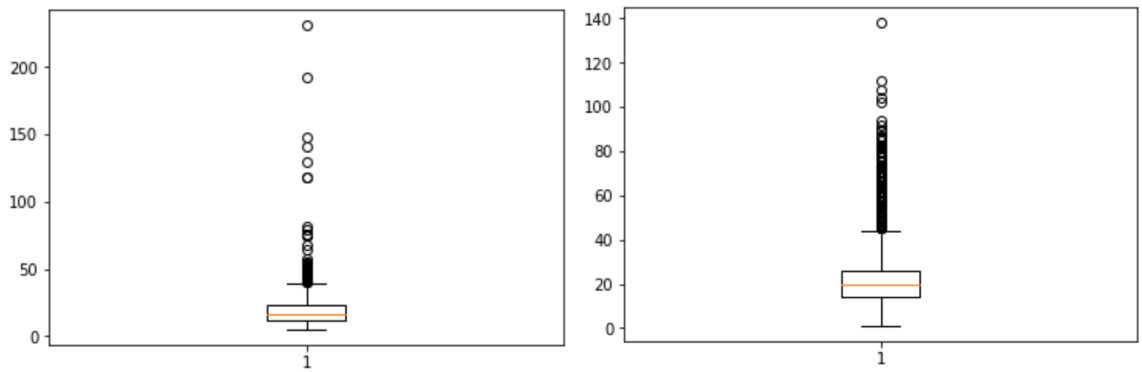
TRUE	300D	0.2	0.02858
TRUE	300D	0.4	0.03002
TRUE	300D	0.3	0.02612

Hyperparameter Rationale and Results

As you can see in the above tables, we have done hyperparameter tuning for the following parameters:

1) **Embedding Layer Trainable** – Since we are using the Glove embedding layer which has pre-trained weights on a different dataset, we have the option to either freeze the pre-trained weights or allow the weights to be retrained in the training process. From the results of our tuning, we can see that for all of our models, retraining the weights resulted in a better validation loss (lower).
2) **Embedding Dimensions** – We had different dimensions of the Glove embedding model pre-trained on the Wikipedia corpus namely 50d,100d,200d and 300d. Higher dimensions will lengthen the training time but also provide a complex model. From our tuning, we found that only RNN had better results with 200d, while CNN and HAN performed better with 300d.
3) **Dropout** – We used a variety of different dropout rates like 0.2,0.3,0.4 and 0.5 to reduce overfitting of our trained model. From our tuning process, we found CNN needed 0.4 dropout while RNN and HAN needed 0.3 dropout for best results.
4) **Bidirectional LSTM or Unidirectional LSTM (for RNN)** – While unidirectional LSTM only considers the past input, bidirectional layer considers both the past and future inputs. While this may help uncover the context better, it will also make the model complex. From our results we found that unidirectional LSTM performed better on RNN while bidirectional performed better on RNN.
5) **MAX_SEQUENCE_LENGTH** – This parameter is used while generating the final input for CNN and RNN using tokenizer word_index and embedding index. It is essentially the input dimension (i.e, the number of words it will consider for each article). It will pad the articles which has less words and cut the articles which has more words. We generated a boxplot to visualize the outliers and the 97th percentile of sequence length for all articles. This was around 500.
6) **MAX_SENTENCES** – This parameter along with MAX_SENT_LENGTH determines the final input dimension for HAN model using tokenizer word_index and embedding index. Using a similar method to find the 97th percentile before, we found that the optimal number of sentences for each article was 50 for dataset 1 and 143 for dataset 2.
7) **MAX_SENT_LENGTH** – Similar to MAX_SENTENCES, this parameter considers the maximum word length to take for each sentence in each article. This was found to be 50 for dataset 1 and 15 for dataset 2.

Figure 7 Boxplot of MAX_SENT and MAX_SENT_LENGTH

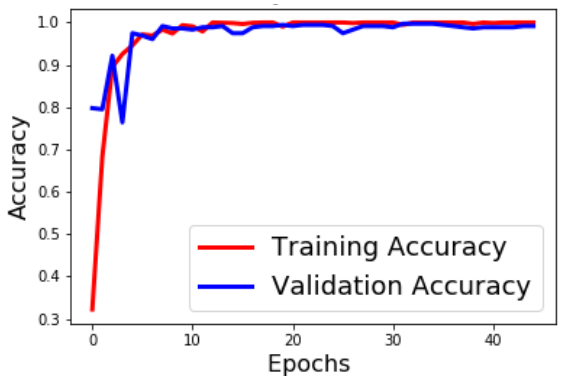


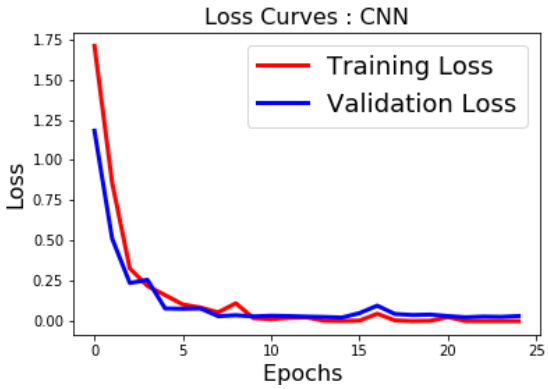
Best Parameters on Dataset 1

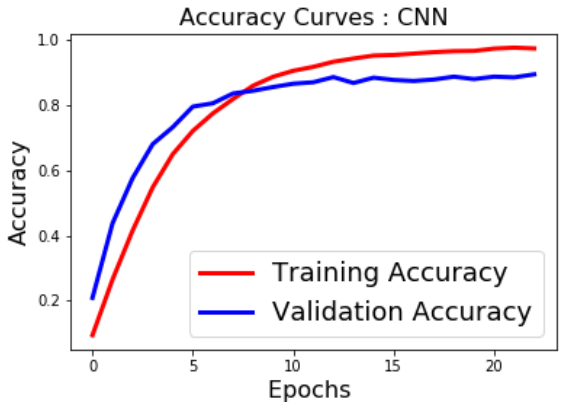
Model	CNN	RNN	HAN
Embedding Layer Trainable	TRUE	TRUE	TRUE
Embedding Dimension	300D	200D	300D
Dropout	0.4	0.3	0.3
Bidirectional LSTM	N/A	FALSE	TRUE
MAX_SEQUENCE_LENGTH (for CNN/RNN) or MAX_SENT_LENGTH (for HAN)	1000	1000	50
MAX_SENTS (for HAN)			50
MAX_NB_WORDS	20000	20000	20000
LSTM units		100	100
Optimizer	rmsprop	rmsprop	rmsprop
Activation (CNN)	relu	N/A	N/A

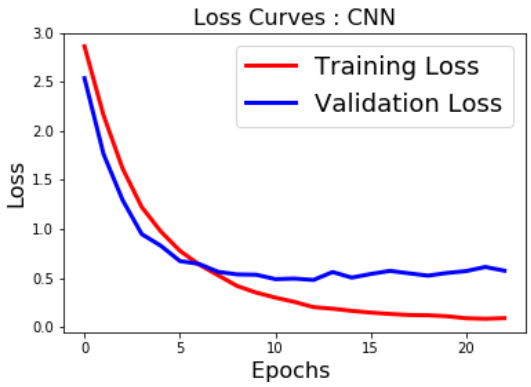
Results: Training and Validation Loss and Accuracy Plots

CNN (Dataset1 on first row and Dataset 2 on second row)

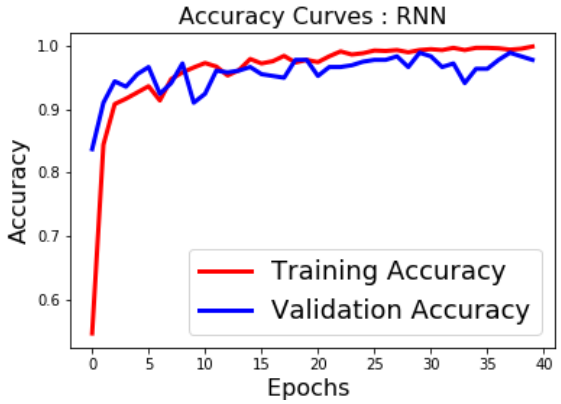


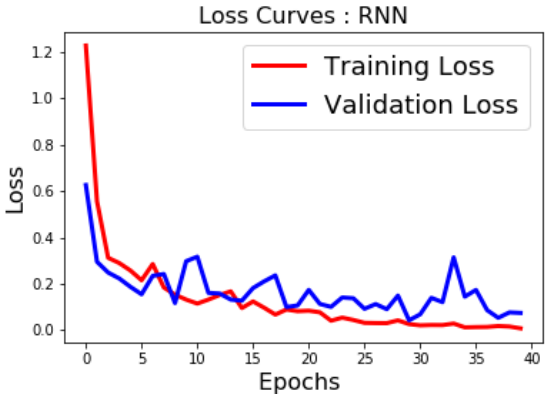


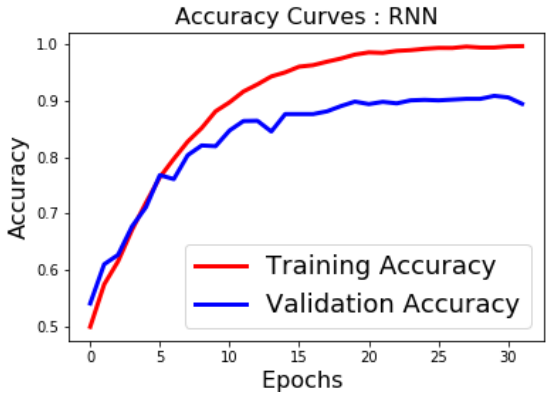


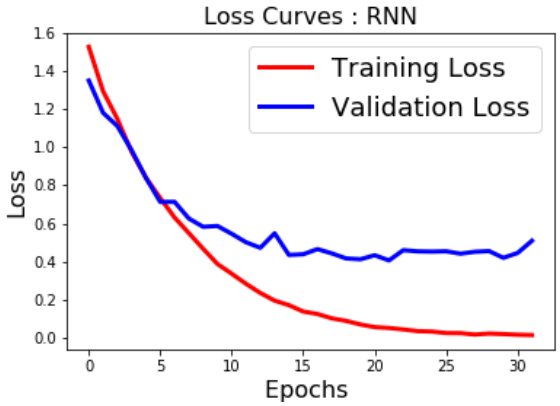


RNN (Dataset1 on first row and Dataset 2 on second row)

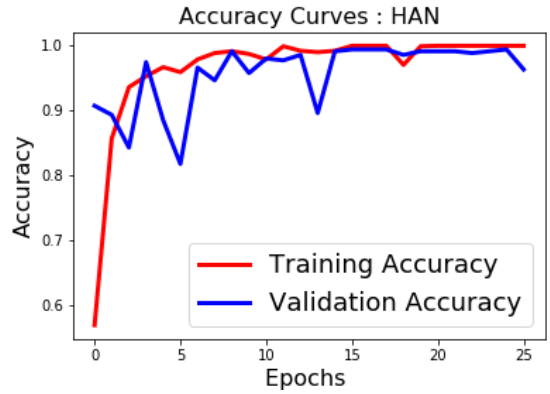


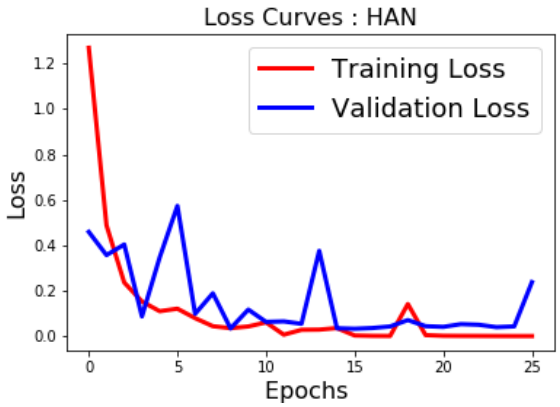


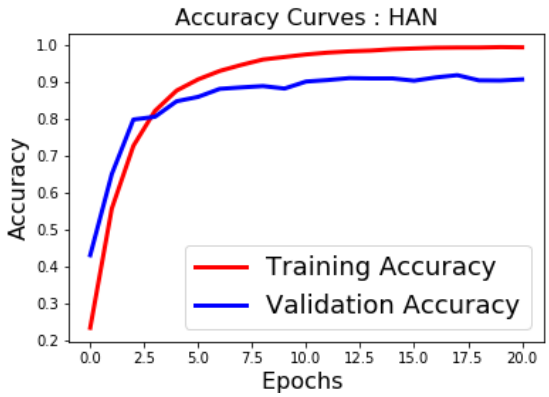


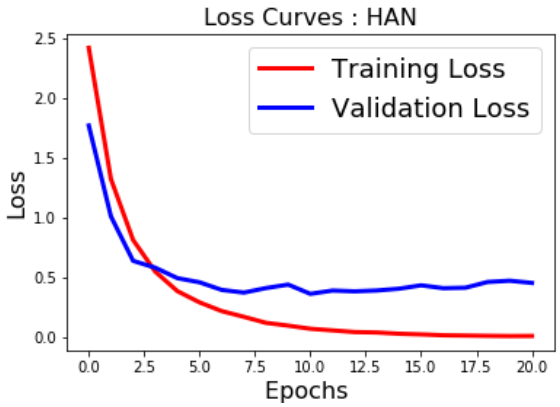


HAN (Dataset1 on first row and Dataset 2 on second row)









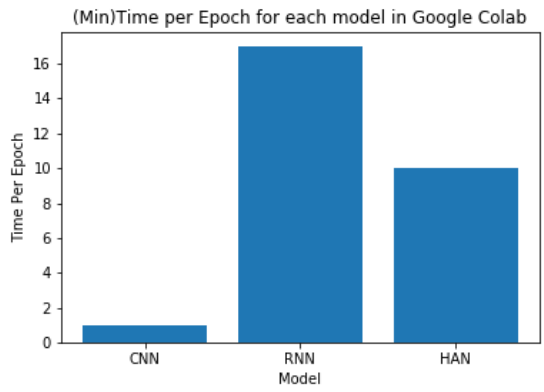
Final Accuracy on Test Data (Dataset 1)

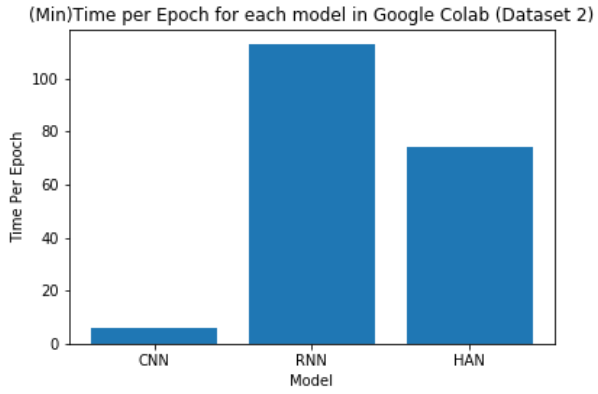
Model	CNN	RNN	HAN
Validation Loss	0.01323	0.04299	0.02612
Test Accuracy	96.63	95.73	97.07

Final Accuracy on Test Data and corresponding parameters (Dataset 2)

Model	CNN	RNN	HAN
Validation Loss	0.48017	0.40511	0.36618
Test Accuracy	79.82	82.63	83.4
Dropout	0.5	0.3	0.3
Trainable	TRUE	TRUE	TRUE
Embedding DIM	300D	200D	300D
MAX_SEQ_LEN	1000	1000	N/A
MAX_SENT_LEN	N/A	N/A	15
MAX_SENT	N/A	N/A	143

Time Per Epoch





Conclusion

From our experiment, we found that while CNN had the lowest validation loss on dataset 1, the test accuracy of HAN was highest even though it had lower validation loss compared to CNN. We can see that the difference between the test accuracies is very less. Hence, if someone needs to train a model faster, they could choose CNN over HAN.

For dataset 2, we had considered a data with higher number of records and more classes. HAN performed the best on both validation loss and test accuracy while CNN performed the worst on both. This may have been because of larger dataset; HAN was able to retrieve a deeper understanding/context of the data.

Overall, HAN performed consistently better for both types of datasets and it also took average time to train compared to CNN and RNN.

References

1. http://mlg.ucd.ie/datasets/bbc.html
2. https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html
3. https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html
4. https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17

5. https://medium.com/jatana/report-on-text-classification-using-cnn-rnn-han-f0e887214d5f
6. http://www.wildml.com/2015/11/understanding-convolutional-neural-netwo rks-for-nlp/
7. https://machinelearningmastery.com/cnn-long-short-term-memory-networ ks/
8. https://www.cs.cmu.edu/~diyiy/docs/naacl16.pdf
9. https://arxiv.org/pdf/1506.01057v2.pdf
10. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
11. https://arxiv.org/abs/1408.5882