TEXT CLASSIFICATION: COMPARATIVE ANALYSIS OF DIFFERENT DEEP NEURAL NETWORK ARCHITECTURES

OVERVIEW

- Text Classification: Comparative Analysis of Different Deep Neural Network Architectures
  - CNN: Convolutional Neural Networks
  - RNN: Recurrent Neural Networks
  - HAN: Hierarchical Attention Network
- Underlying data model provided by GloVe word vectors
- Dataset
  - IMDB Movie Reviews
- Goal: Understand audience sentiment (positive/negative)

DATASET

- IMDB Movie Reviews
  - “The labeled data set consists of 50,000 IMDB movie reviews, specially selected for sentiment analysis. The sentiment of reviews is binary, meaning the IMDB rating < 5 results in a sentiment score of 0, and rating >=7 have a sentiment score of 1.”
- https://www.kaggle.com/c/word2vec-nlp-tutorial/data

HAN FOR DOCUMENT CLASSIFICATION

- Hierarchical Attention Networks for Document Classification (Yang 2016)
- 4 main components to HAN architecture
  - Word sequence encoder
  - Word-level attention layer
  - Sentence encoder
  - Sentence-level attention layer
- Implemented using Keras
  - Specifically, Keras' ability to create custom layers was utilized to create the word and sentence-level attention layers for the model.

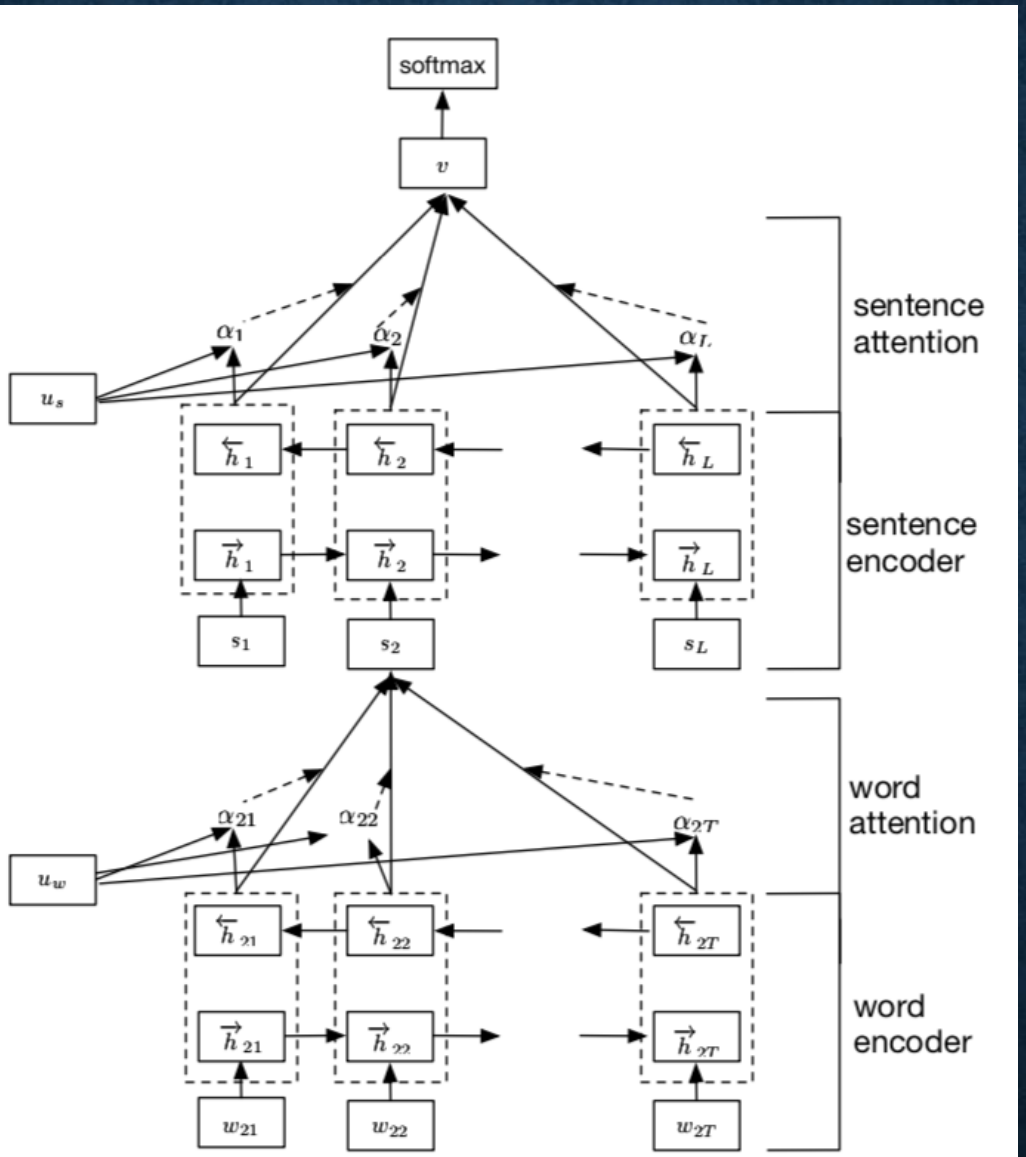
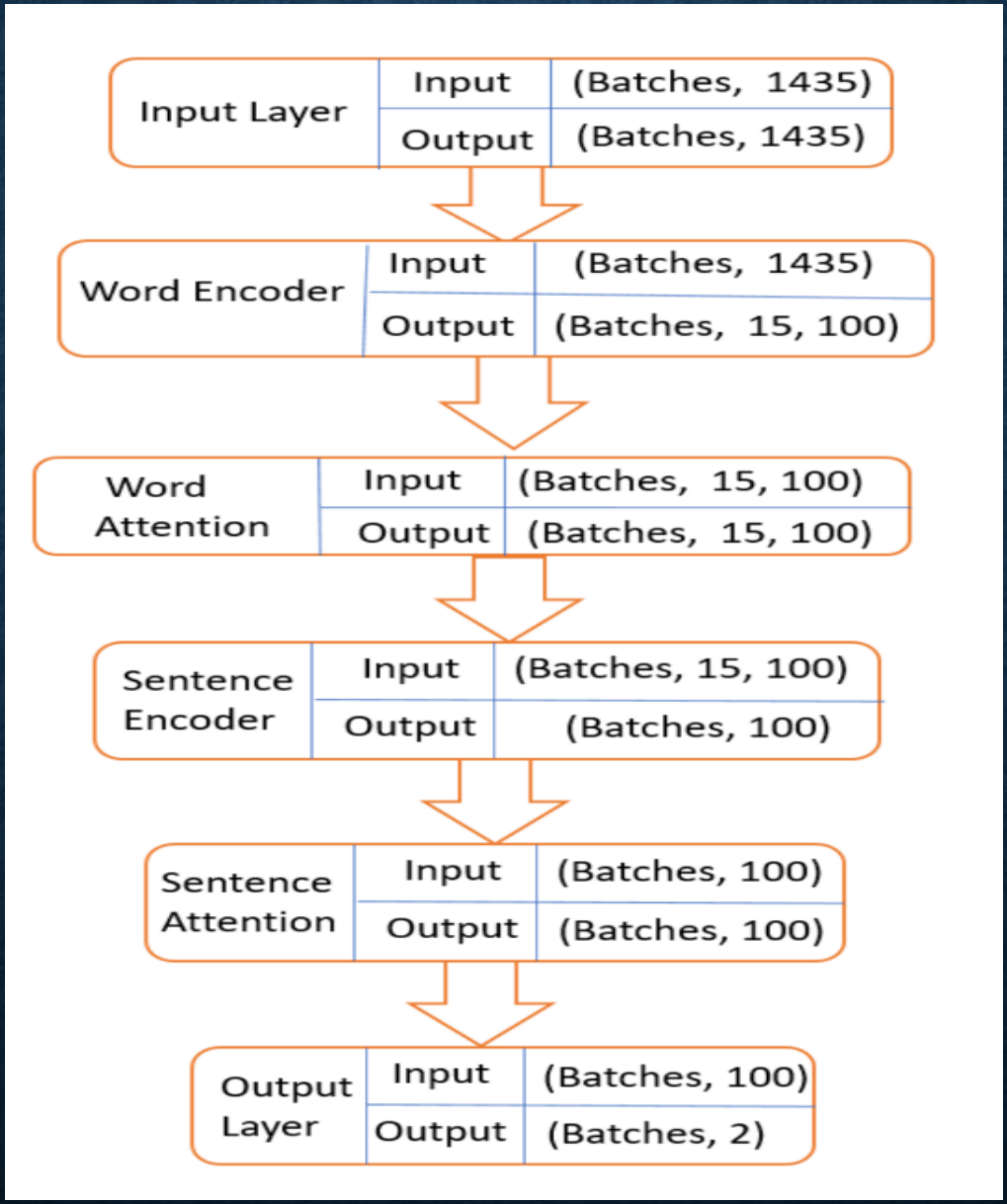


Figure 2: Hierarchical Attention Network.

HAN ARCHITECTURAL DIAGRAM



HAN IMPLEMENTATION

- Word Encoder – Utilizing GloVe (Global Vectors for Word Representation), embed words to vectors through an embedding matrix. From here use a bidirectional gated recurrent unit (GRU) to to get annotations of words by summarizing information from both directions for words, and therefore incorporate the contextual information in the annotation.
  - Keras Functions: Embedding, Input, Bidirectional, GRU, Model

- Word Attention – Introduce word attention layer in order to compute sentence vectors by extracting words that are important to the meaning of the sentence and aggregate the representation of those informative words to form a sentence vector.
  - Keras Functions: TimeDistributed

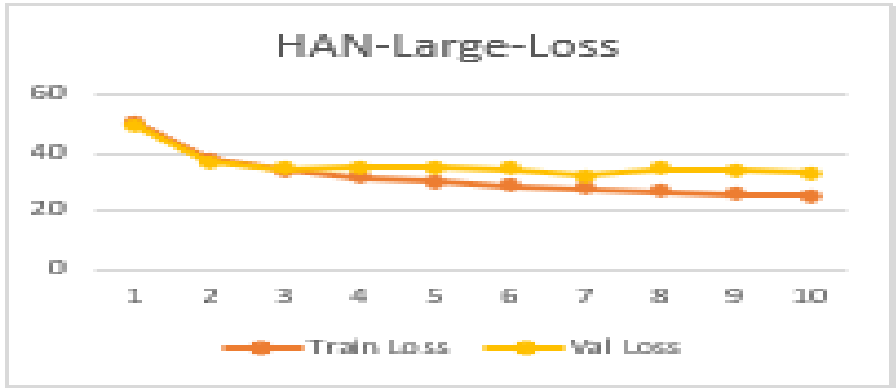
HAN IMPLEMENTATION

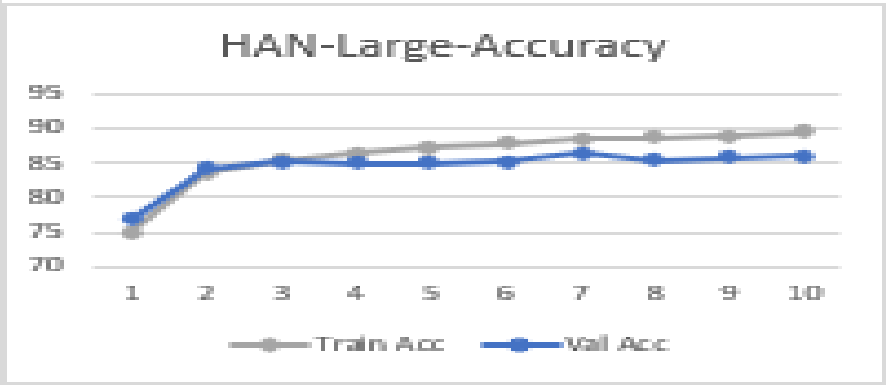
- Sentence Encoder – Given the sentence vectors, again use a bidirectional GRU to encode the sentences.
  - Keras Functions: Input, Bidirectional, GRU, Model

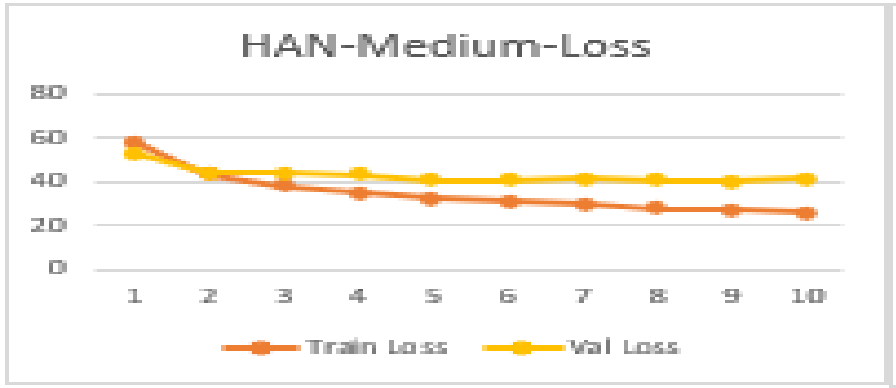
- Sentence Attention - We introduce a second attention mechanism and introduce a sentence level context vector to measure the importance of the sentences. This yields a document vector that summarizes all the information of sentences in a document.
  - Keras Functions: TimeDistributed, Custom Keras Attention Layer

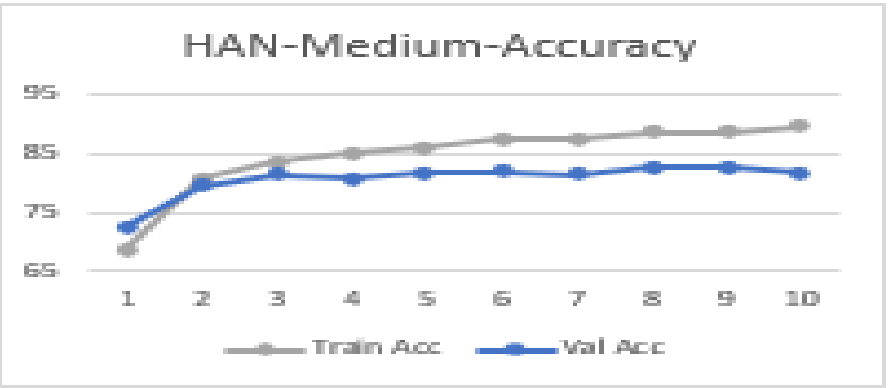
- Document Classification – The document vector is a high level representation of the document and can be used as features for document classification.
  - Keras Functions: Dense, Custom Keras Attention Layer

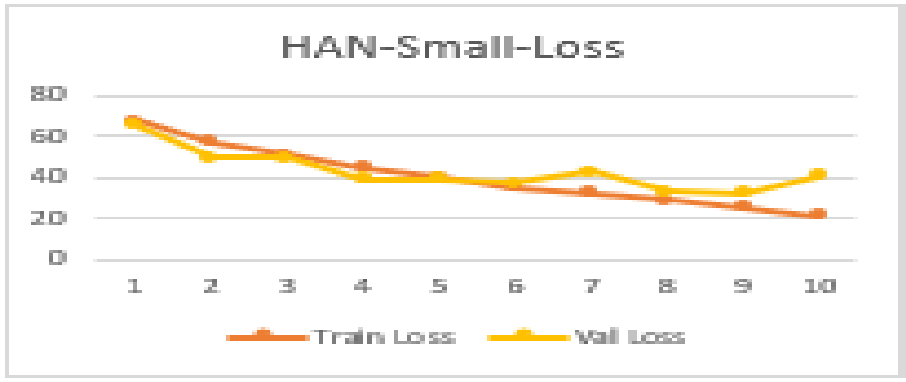
HAN GRAPHS

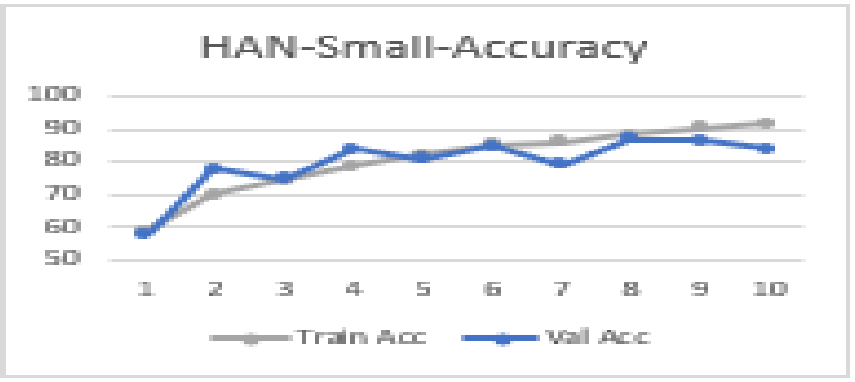

HAN-Large-Loss
60
40
20
0
1 2 3 4 5 6 7 8 9 10
Train Loss
Val Loss


HAN-Large-Accuracy
95
90
85
80
75
70
1 2 3 4 5 6 7 8 9 10
Train Acc
Val Acc


HAN-Medium-Loss
80
60
40
20
0
1 2 3 4 5 6 7 8 9 10
Train Loss
Val Loss


HAN-Medium-Accuracy
95
85
75
65
1 2 3 4 5 6 7 8 9 10
Train Acc
Val Acc


HAN-Small-Loss
80
60
40
20
0
1 2 3 4 5 6 7 8 9 10
Train Loss
Val Loss


HAN-Small-Accuracy
100
90
80
70
60
50
1 2 3 4 5 6 7 8 9 10
Train Acc
Val Acc

HAN RESULTS

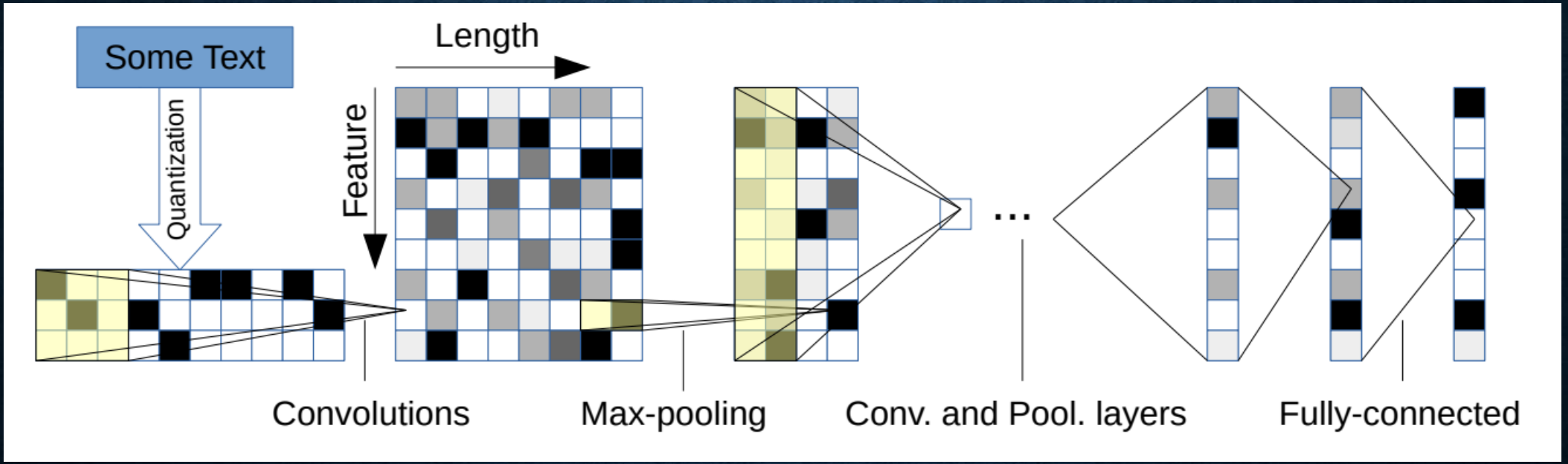
- Maximum accuracy of 90.2% and 87.3% validation accuracy

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 262s 13ms/step - loss: 0.4569 - acc: 0.7842 -
val_loss: 0.3860 - val_acc: 0.8314
Epoch 2/10
20000/20000 [==============================] - 246s 12ms/step - loss: 0.3490 - acc: 0.8482 -
val_loss: 0.3577 - val_acc: 0.8496
Epoch 3/10
20000/20000 [==============================] - 245s 12ms/step - loss: 0.3195 - acc: 0.8647 -
val_loss: 0.3550 - val_acc: 0.8502
Epoch 4/10
20000/20000 [==============================] - 245s 12ms/step - loss: 0.3016 - acc: 0.8736 -
val_loss: 0.3393 - val_acc: 0.8618
Epoch 5/10
20000/20000 [==============================] - 244s 12ms/step - loss: 0.2874 - acc: 0.8800 -
val_loss: 0.3355 - val_acc: 0.8616
Epoch 6/10
20000/20000 [==============================] - 243s 12ms/step - loss: 0.2772 - acc: 0.8849 -
val_loss: 0.3468 - val_acc: 0.8572
Epoch 7/10
20000/20000 [==============================] - 242s 12ms/step - loss: 0.2667 - acc: 0.8912 -
val_loss: 0.3226 - val_acc: 0.8688
Epoch 8/10
20000/20000 [==============================] - 257s 13ms/step - loss: 0.2588 - acc: 0.8951 -
val_loss: 0.3276 - val_acc: 0.8674
Epoch 9/10
20000/20000 [==============================] - 274s 14ms/step - loss: 0.2495 - acc: 0.8977 -
val_loss: 0.3257 - val_acc: 0.8728
Epoch 10/10
20000/20000 [==============================] - 245s 12ms/step - loss: 0.2418 - acc: 0.9020 -
val_loss: 0.3262 - val_acc: 0.8728
```

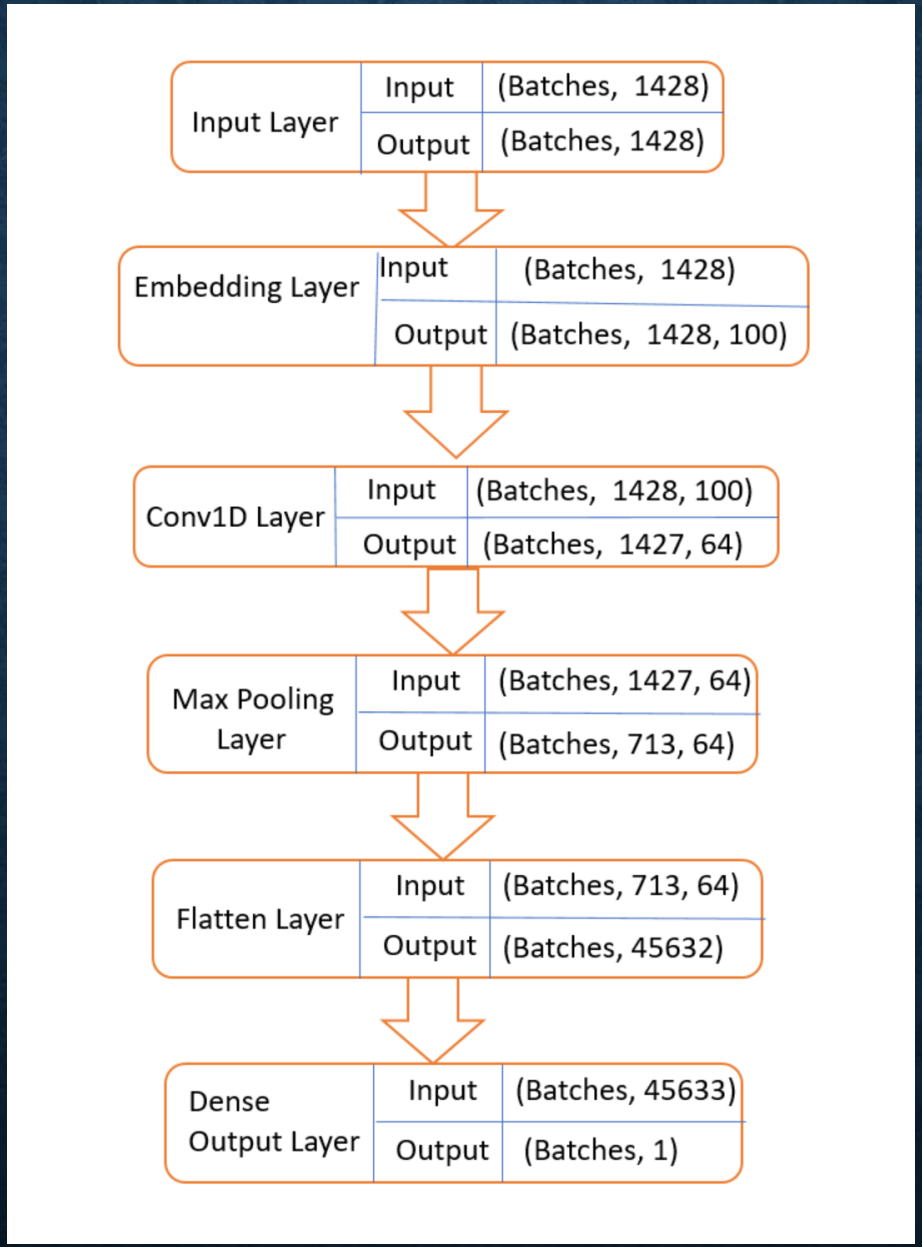
CNN FOR TEXT CLASSIFICATION

- CNN is a class of deep, feed-forward artificial neural network
- Initially developed for Image Processing
- Involves 2 major operations – Convolution and Pooling
- Output from Convolution and Pooling Layers is given to Fully Connected Layer which is in principle the same as traditional Multilayer Perceptron
- High success in Image Recognition led to CNN being used in NLP tasks

CNN GENERAL ARCHITECTURE



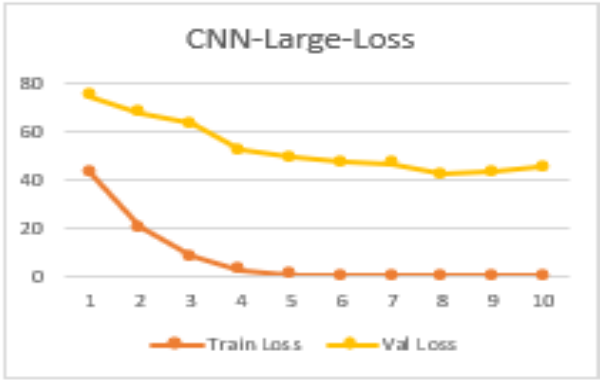
CNN ARCHITECTURAL DIAGRAM

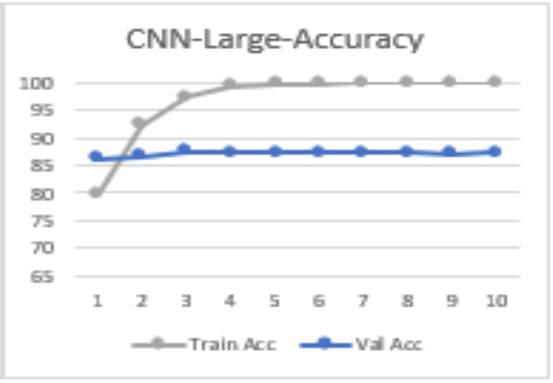


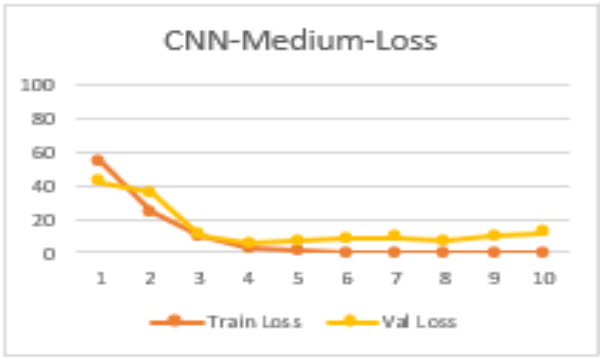
CNN IMPLEMENTATION

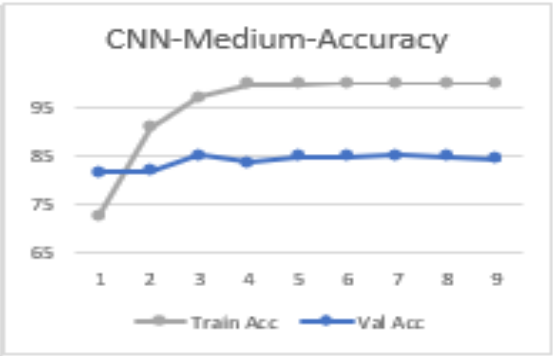
- The input data is fed into the embedding layer with 100d size.
- Using Conv1D function, the model is filtered with optimal filter size.
- MaxPooling1D is followed by this process.
- Flatten Layer is used to flatten the output and finally a Dense layer with 1 neuron and sigmoid activation is added to get the output as binary classification.
- The model is complied using standard adam optimizer and the loss is calculated as binary crossentropy.

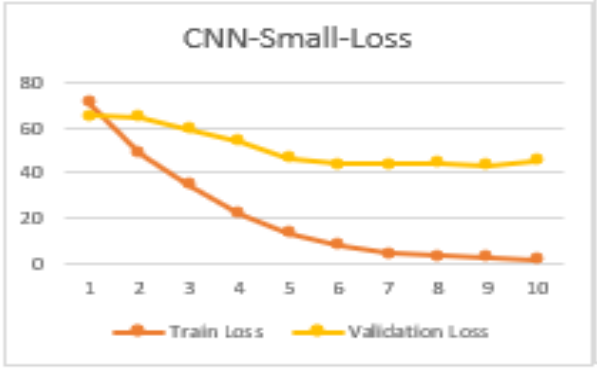
CNN GRAPH

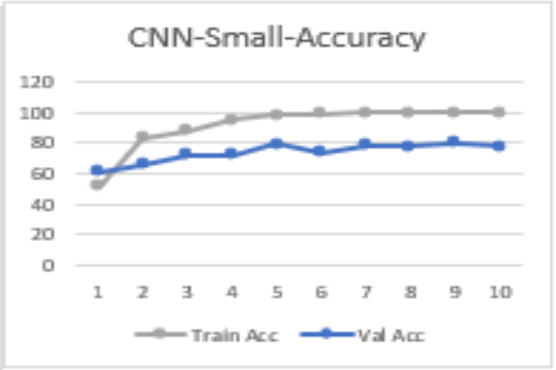

CNN-Large-Loss
80
60
40
20
0
1 2 3 4 5 6 7 8 9 10
Train Loss
Val Loss


CNN-Large-Accuracy
100
95
90
85
80
75
70
65
1 2 3 4 5 6 7 8 9 10
Train Acc
Val Acc


CNN-Medium-Loss
100
80
60
40
20
0
1 2 3 4 5 6 7 8 9 10
Train Loss
Val Loss


CNN-Medium-Accuracy
95
85
75
65
1 2 3 4 5 6 7 8 9
Train Acc
Val Acc


CNN-Small-Loss
80
60
40
20
0
1 2 3 4 5 6 7 8 9 10
Train Loss
Validation Loss


CNN-Small-Accuracy
120
100
80
60
40
20
0
1 2 3 4 5 6 7 8 9 10
Train Acc
Val Acc

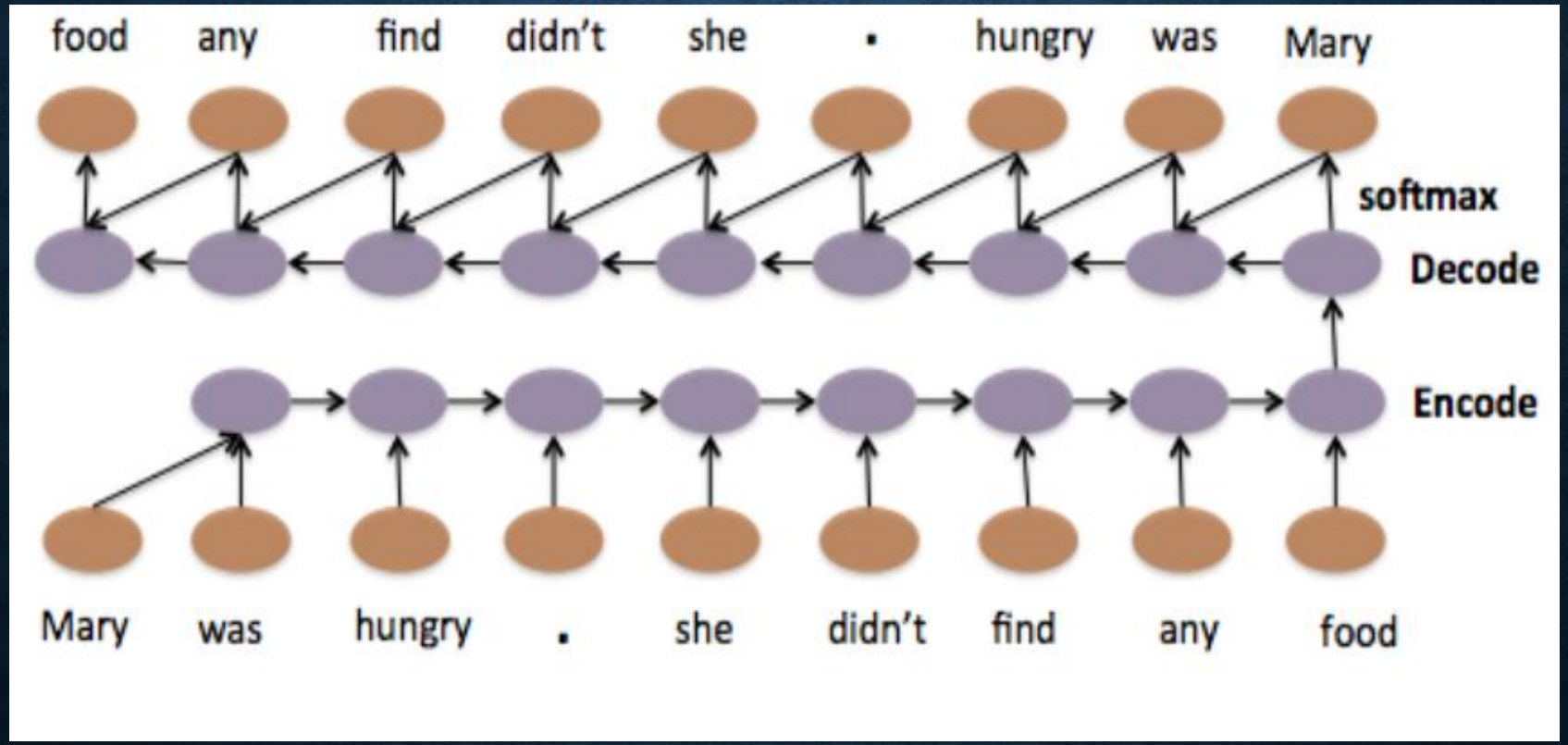
CNN RESULT

Dataset	Time/Epoch(s)	Validation Accuracy
Small	5	79.5
Medium	143	85.2
Large	203	87.6

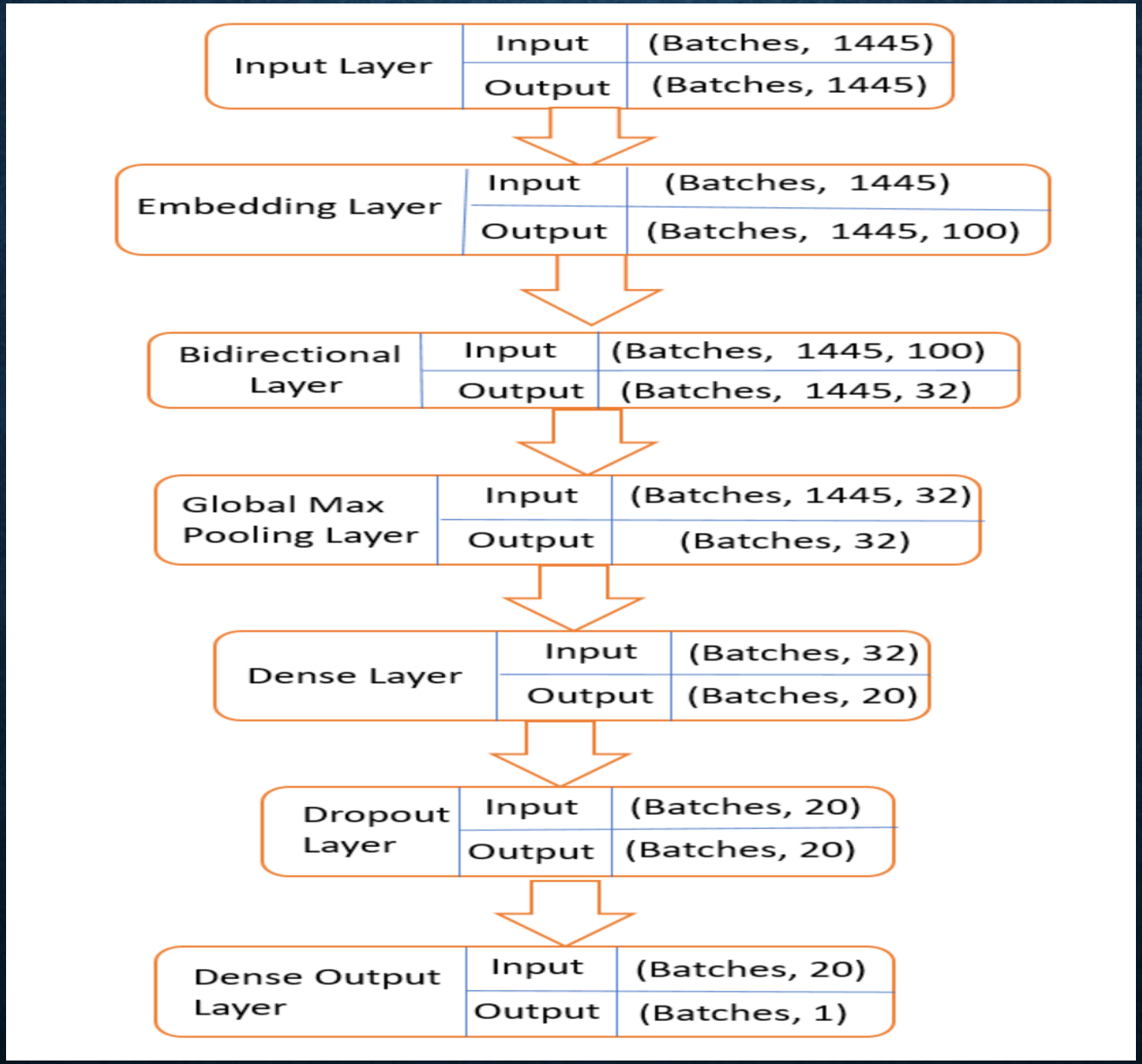
RNN FOR TEXT CLASSIFICATION

- A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a sequence.
- Using the knowledge from an external embedding can enhance the precision of your RNN because it integrates new information (lexical and semantic) about the words, an information that has been trained and distilled on a very large corpus of data.
- A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.

RNN GENERAL DIAGRAM



RNN ARCHITECTURAL DIAGRAM

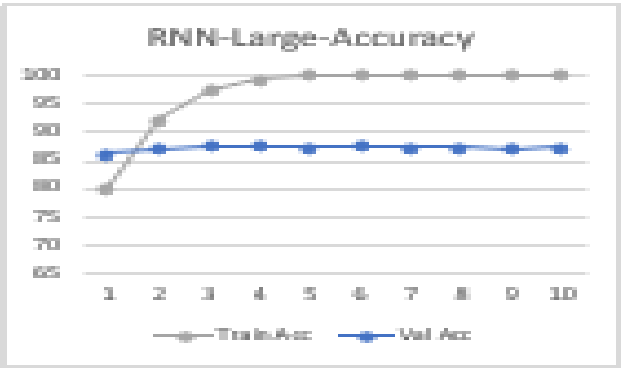


RNN IMPLEMENTATION

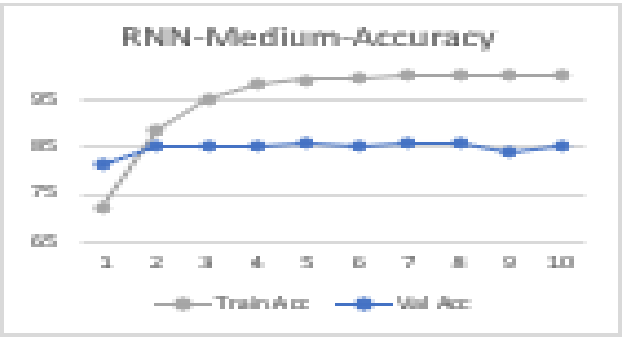
- The input is fed to the sequential layer which is then fed to the embedding layer.
- From embedding layer, the Bidirectional function takes over and LSTM units are given to the model.
- Furthermore, couple of dense layers are added to the model with a Dropout function in between them.
- Finally the model is compiled using the standard optimizer – adam.

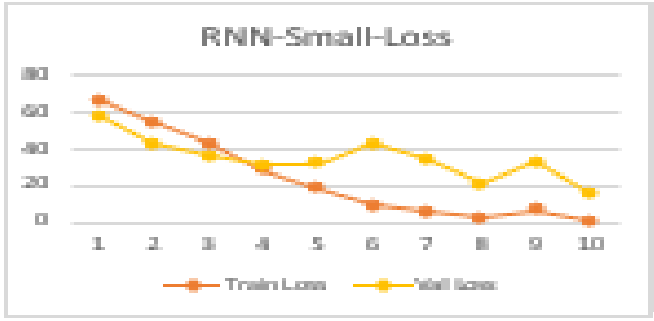
RNN GRAPH

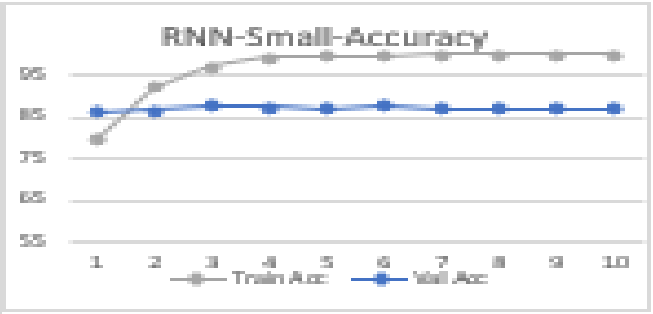

RNN-Large-Loss
80
70
60
50
40
30
20
10
0
1 2 3 4 5 6 7 8 9 10
Train Loss
Val Loss


RNN-Large-Accuracy
100
95
90
85
80
75
70
65
1 2 3 4 5 6 7 8 9 10
Train Acc
Val Acc


RNN-Medium-Loss
80
60
40
20
0
1 2 3 4 5 6 7 8 9 10
Train Loss
Val Loss


RNN-Medium-Accuracy
95
85
75
65
1 2 3 4 5 6 7 8 9 10
Train Acc
Val Acc


RNN-Small-Loss
80
60
40
20
0
1 2 3 4 5 6 7 8 9 10
Train Loss
Val Loss


RNN-Small-Accuracy
95
85
75
65
55
1 2 3 4 5 6 7 8 9 10
Train Acc
Val Acc

RNN RESULTS

DATASET	TIME/EPOCH(S)	VALIDATION ACCURACY
Small	30	89.5
Medium	283	86
Large	486	88.3

HAN HYPERPARAMETER TUNING

Hyper-parameter	Values
Batch size	32,64,128
Encoding dimensions	100,200,300

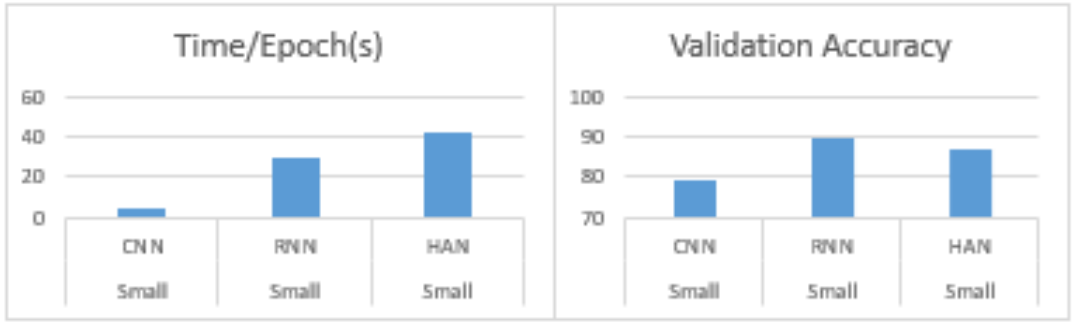
CNN HYPERPARAMETER TUNING

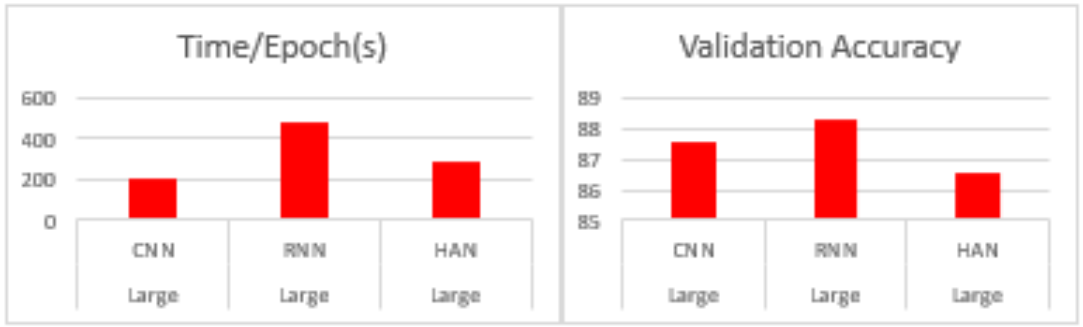
Hyper-parameter	Values
Activation-Type	Relu, Leaky relu
Pool size	2,3,4
Kernel size	2,3,5
Filter size	64,128,256

RNN HYPERPARAMETER TUNING

Hyper-parameter	Values
LSTM units	16,32,64
Dense units	20,40,60
Dropout rate	0.2,0.5,0.8

COMPARATIVE STUDY OF THE MODELS





From the graphs on the left, we observed the following:

- CNN model has outperformed the other two models (RNN & HAN) in terms of training time
- RNN has consistently performed better than CNN and HNN and has shown consistent higher validation accuracies for all the three datasets.
- For different sized datasets, RNN seems to perform better for the given dataset, but in general, it depends on the dataset used and how we design the model.

REFERENCES

01

https://www.nltk.org/api/nltk.tokenize.html

02

https://keras.io/layers/writing-your-own-keras-layers/

03

https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/

04

https://github.com/FlorisHoogenboom/keras-han-for-docla

05

https://richliao.github.io/supervised/classification/2016/12/26/textclassifier-HATN/

06

https://medium.com/jatana/report-on-text-classification-using-cnn-rnn-han-f0e887214d5f

THANK YOU