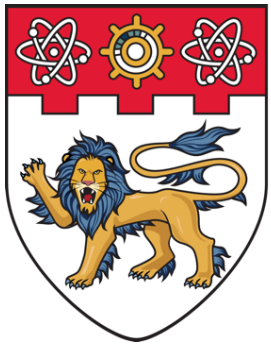


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CE/CZ 4042 NEURAL NETWORKS

Assignment 2

Date of Last Update: 15 November 2019

Group Members: Ang Qi Xuan (U1621610H)
Lim Zheng Min (U1622040D)

Part A Object Recognition

1.1 Introduction

The CIFAR-10 dataset contains RGB images of 10 different classes, each with image size of 32 x 32. The objective of this section is to design a Convolutional Neural Network (CNN) architecture to train on the examples from *data_batch_1,* and test the performance of the CNN on *test_batch_trim.* Grid search is implemented for hyperparameter tuning, and various optimizers will be explored to compare their performances.

1.2 Methods

From the questions, we realised that certain functions are being called regularly with the addition of new parameters to modify the architecture. Hence, we have tried to modularised the whole process with the use of 3 main functions; loading of data, designing of CNN model, and main function for training and validation.

1.2.1 Functions

- **load_data(*file*):** Function that extracts input features and labels from the given file. Arguments: file path.
- **cnn():** Function for designing the CNN architecture
- **main():** Main function for training and validation of model performance.

In every part of the question, we will be creating a set of cnn() and main() with different parameters based on the objectives of the question part.

1.2.2 Image Format

From our research, we found that to pass in the images in the correct format for training, we will need to first reshape into the following format.

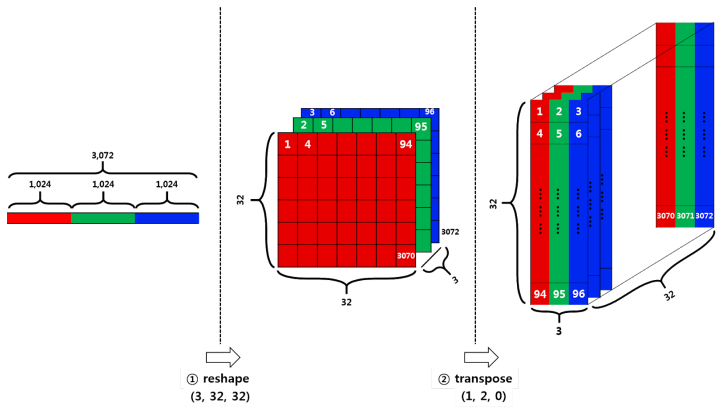
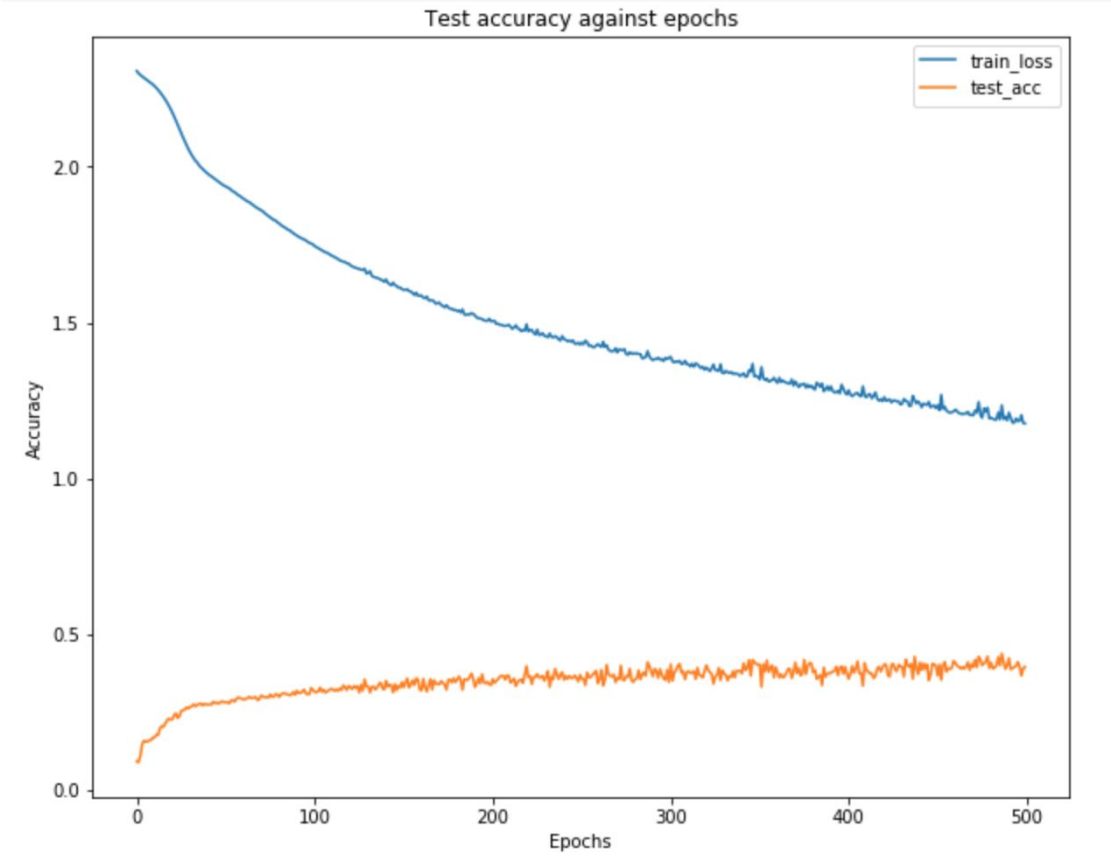


Image extracted from https://towardsdatascience.com/cifar-10-image-classification-in-tensorflow-5b501f7dc77c

This ensures that the image format we passed in as our input layer will correctly represent the position of each RGB pixel in the image, which is essential in the later part when we try to conceive the patterns observed from the feature maps in the convolution layer.

1.3 Experiments & Results

1(a). Training Loss and Test Accuracy against learning epochs



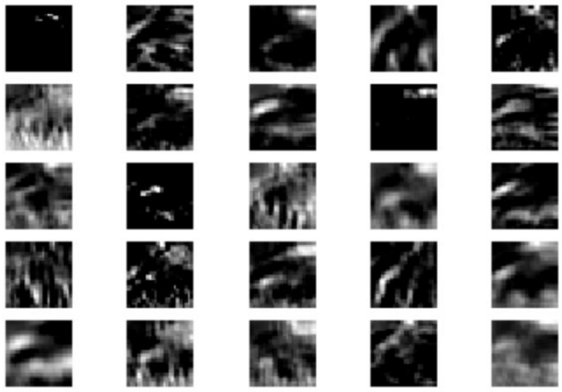
We can see that the training loss is decreasing significantly in the initial few epochs. As the model is being trained with more epochs, the training loss decrease in a linear fashion.

Despite the drop in loss function for the model, we can see that the test accuracy converges after around 500 epochs, with an accuracy of ~35%. Further training does not improve the test accuracy any further. This starting architecture has high bias, given its low accuracy of ~35%, performing worse than random guess.

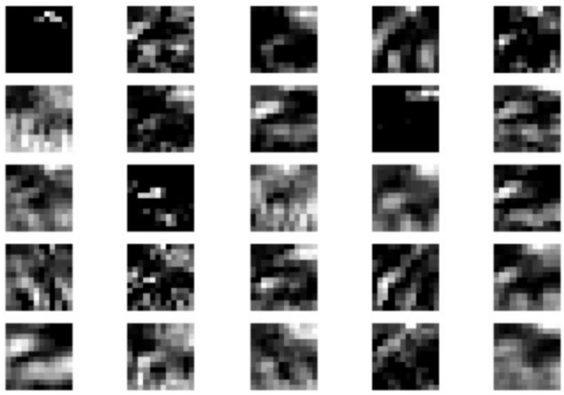
1(b). **Test Example 1**

From the images shown below, we can observe that the feature map in the first convolution layer has resemblance of the original image. This is because the filters at this layer will capture fine details of the image like edges etc. The second convolution layer takes the features from the pooling layer 1 and capture any general features found in the image. Hence, these abstract features does not resemble the image compared to the feature maps from convolution layer 1. The final pool layer will reduce the dimensions of the feature maps, to reduce the number of neurons used in the fully connected layer, to reduce computational time.

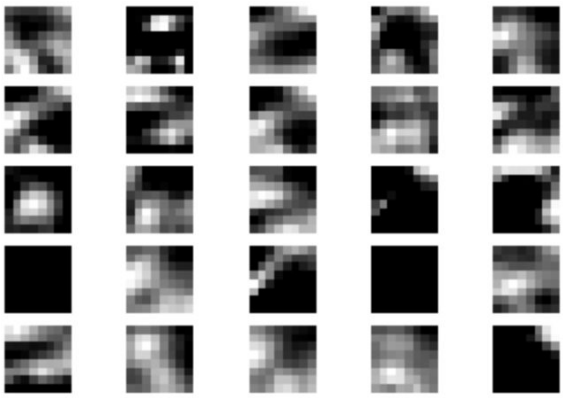
C_1 Feature map



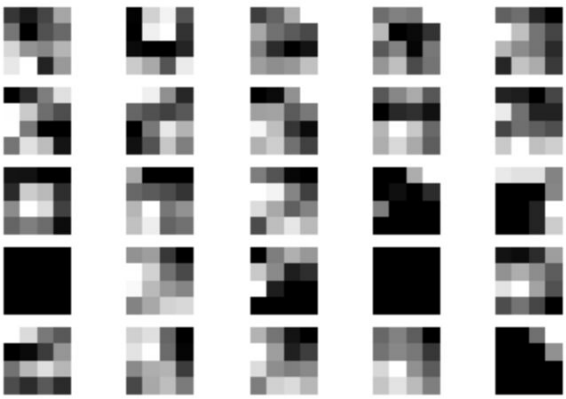
S_1 Feature map



C_2 Feature map

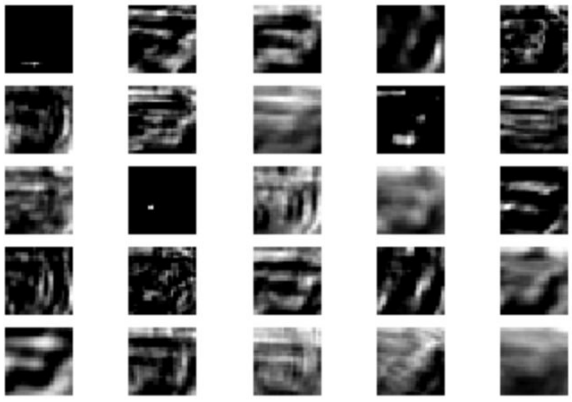


S_2 Feature map

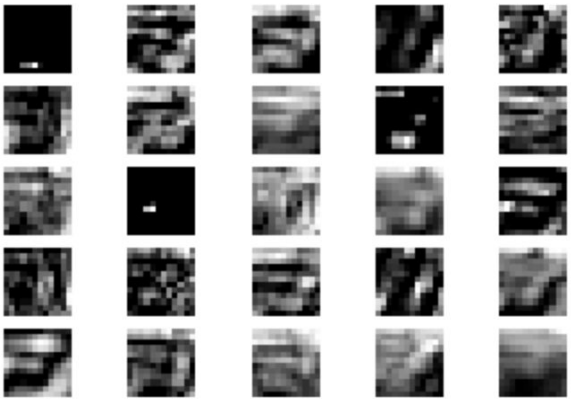


Test Example 2

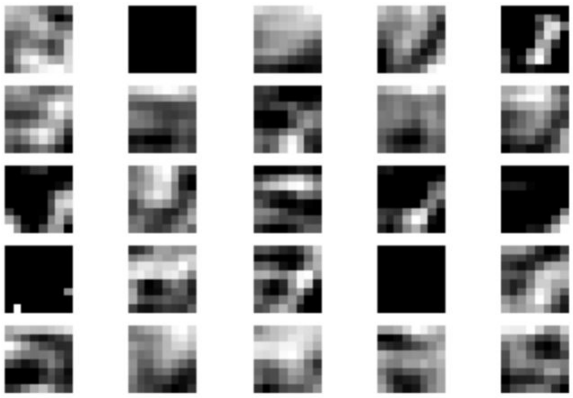
C_1 Feature map



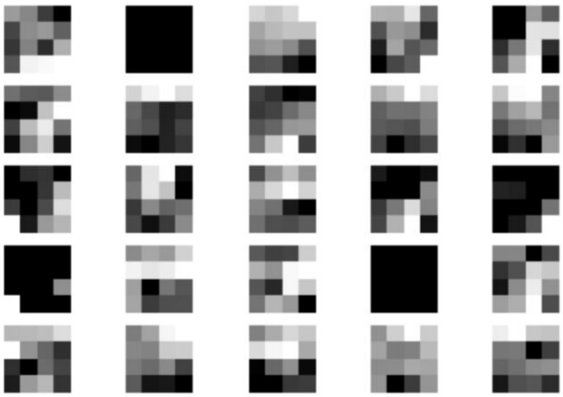
S_1 Feature map



C_2 Feature map



S_2 Feature map



2. **Grid Search**

Based on our research, we found that most of the current popular CNN architecture like VGG[1], Alexnet[2], they tend to increase the number of feature maps as the images are being convolved at the deeper layers.
This is because at the first layer, where the inputs are raw images, it is necessary to first extract out the low-level features. Thus, we do not need that many feature maps initially. As more layers are added, the model will need to extract higher-level features from the low-level features, which was the output from previous layers.
Hence, more feature maps will be required at the deeper layers for the model to have the capacity to extract the highly abstract features of the image.

Inspired by these architectures mentioned, we decided to do a grid search with increasing number of feature maps from Conv1 to Conv2. We also use values that are multiples of 8, to increase efficiency of the computations by the GPUs.

Grids	**Number of feature maps**		**Test Accuracy**
	Convolution Layer 1	**Convolution Layer 2**	
1	10	10	38.6%
2	32	32	37.7%
3	32	64	40.1%
4	64	64	36.6%
5	64	128	43.0%
6	128	128	40.5%
7	128	256	44.9%
8	256	256	44.7%
9	256	512	49.1%

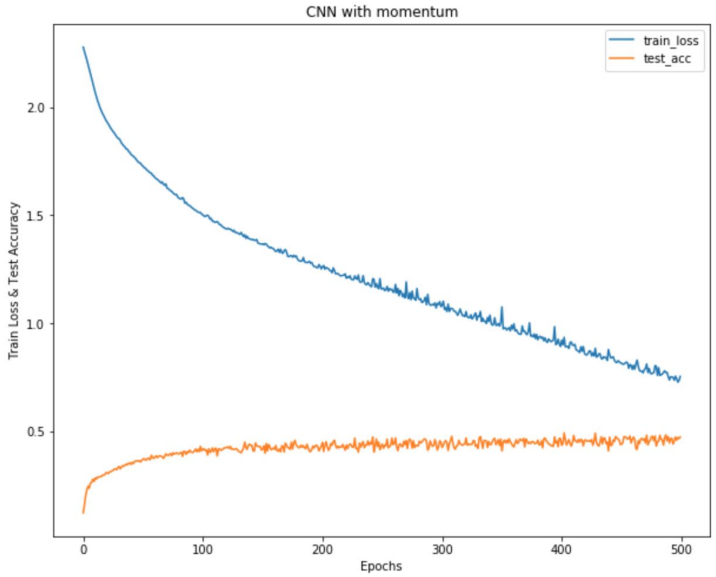
Table 1. Grid Search Results

Based on our grid search results in Table 1, we found that having 256 feature maps at C_1 layer and 512 feature maps in C_2 layer will give the highest test accuracy. This is consistent with our research and hypothesis of increasing number of feature maps at deeper convolution layer. With increasing feature maps, computation time will also be longer. However, because

[1] A convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition"
[2] Convolutional neural network, designed by Alex Krizhevsky, and published with Ilya Sutskever and Krizhevsky's PhD advisor Geoffrey Hinton

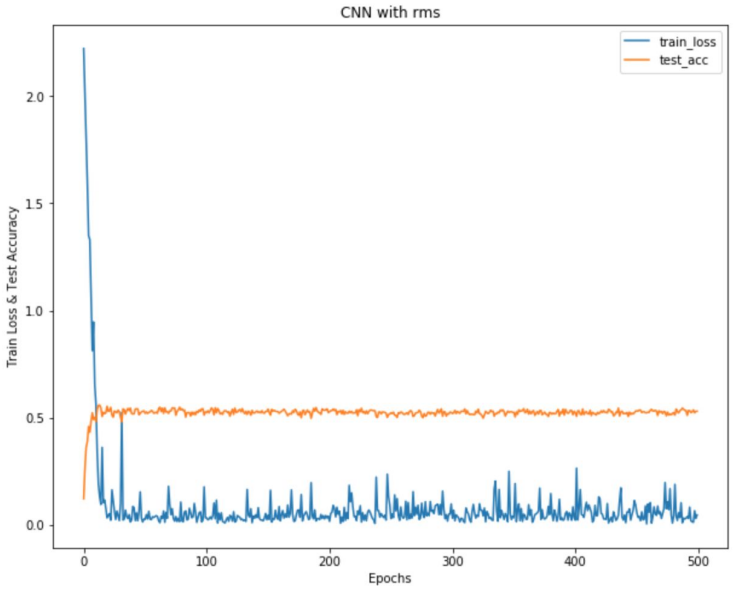
computation time is not part of the consideration in this question, we conclude that the optimum number of feature maps for both convolution layers are **(256,512)**.

3(a) **Momentum = 0.1**



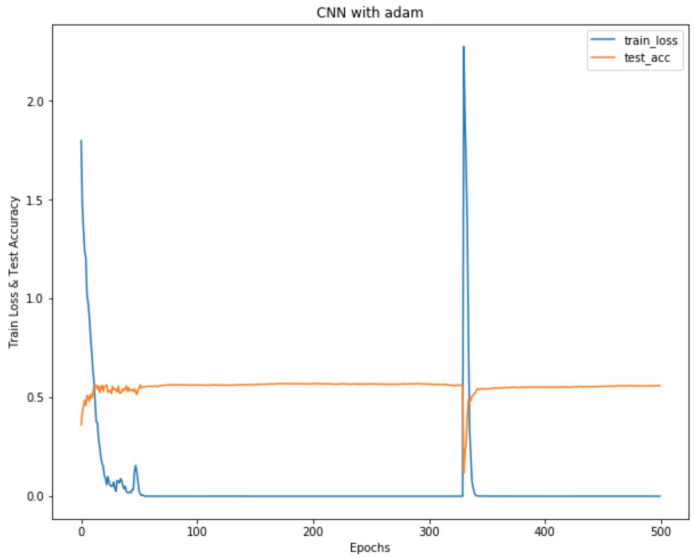
The training loss decreases at a similar rate as the model with Gradient Descent Optimizer. The final test accuracy after training stands at 47.5%.

RMSProp algorithm



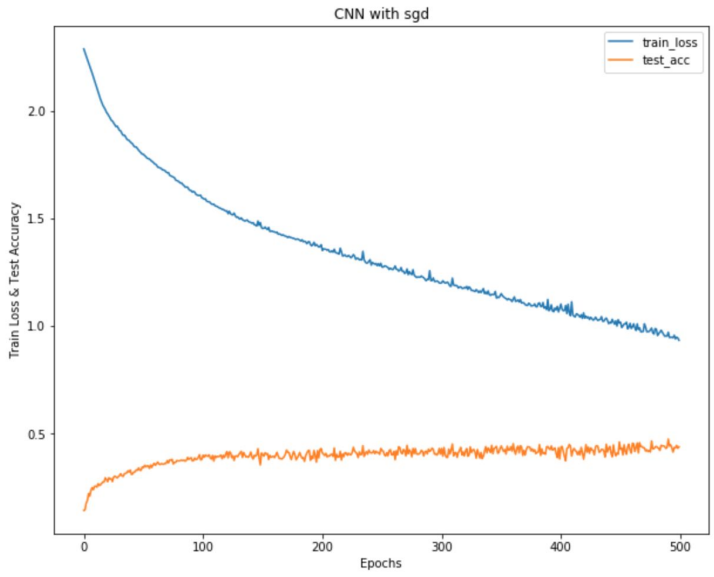
We observed that the training loss converges much faster with RMS optimizer, converging after ~20 epochs. Test accuracy also converges much faster at around the same number of epochs, with an accuracy of 53.0%.

Adam Optimizer



Training loss using Adam Optimizer also converges quickly at around 10 epochs, similar to RMS optimizer. However, we observe a spike in the training loss at ~350 epochs, before working its way back to convergence. One reason could be because gradient is very small at convergence, the denominator will be even smaller. Given a fixed learning rate, it will cause gradients explosion, causing the training to suddenly diverge. Test accuracy = 55.9%.

Dropouts



With the use of dropouts, the training loss decreases in a similar fashion as using normal gradient descent optimizer without dropout, since adding dropout does not affect training. Dropout only helps in reducing overfitting of the model, where the uncertainty in the neurons being used during training cause the model to not be too reliant and giving heavy weights to

the neurons in the fully connected layer. Overall accuracy is 45%, which is higher than the same model without any dropout.

4. Comparing accuracies for all models

	Original CNN	**GridSearch CNN**	**CNN with momentum**	**CNN with RMSProp**	**CNN with Adam Optimizer**	**CNN with Dropout**
Test Accuracy	35%	49.1%	47.5%	53.0%	55.9%	45%

From part(1) to part(2), we can see that by increasing the number of feature maps in each convolution layers, the model is able to extract more information about the image, increasing the test accuracy by almost 15%.

Comparing between all the optimizers used in part(3), we observed that using Adam Optimizer will produce a model that gives the highest test accuracy, followed by RMSProp. This is because it allows the training to converge much faster than gradient descent. Hence, at low epoch size of 500, model using RMSProp and Adam optimizer would have converge, giving a good prediction of the test set. On the other hand, model with gradient descent is still in the process of finding the global minimum at 500 epochs, not able to give a higher prediction accuracy on data it has not seen before.

Despite the improvements in the model through hyper tuning the number of feature maps used in convolution layer and exploring different types of optimizer, we see that the accuracy of the model still has high bias, peaking at 55.9% for CNN with Adam optimizer with 256 and 512 feature maps in the respective convolution layer. This is only slightly better than a random guess; 0.5.

Hence, further improvements that we can make to further improve performance will be to look into the actual CNN architecture. By taking reference from models like Alexnet, VGG, we can better build a model that is able to have high capacity and provide better prediction. This will mean adding more convolution layers to extract higher level features of the image, and more pooling layers as well. We can also consider increasing the number of fully connected layer to learn more information from the outputs of the pooling layer.

Part B Text Classification

1.1 Introduction

In this section, we are given paragraphs collected from Wikipedia entries. We will be required to design CNN and RNN model at both word and character levels for classification of text. The data is split into *train_medium.csv* and *test_medium.csv* containing 5600 and 700 entries respectively. The model will be able to categorise the entries into 1 of the 15 labeled categories in the dataset.

1.2 Methods

Since we are required to train models using data converted to both word/character IDs, there is a need to have a preprocessing function to convert the data into suitable form for training. Below are the model requirements:

1. Adam Optimizer
2. Batch size of 128
3. Learning rate = 0.01

1.2.1 Functions

- **read_data_chars():** Function that extracts the required input features and labels from the training and testing dataset for Character Classifier.
- **data_read_words():** Function that extracts the required input features and labels from the training and testing dataset for Word Classifier.
- **char_cnn_model(x, dropoutBoolean):** Function for designing the architecture for Character CNN Classifier. The *dropoutBoolean* parameter checks whether dropout should be implemented.
- **word_cnn_model(x, dropoutBoolean):** Function for designing the architecture for Word CNN Classifier. The *dropoutBoolean* parameter checks whether dropout should be implemented.
- **rnn_char_model():**Function for designing the architecture for Character RNN Classifier. The *dropoutBoolean* parameter checks whether dropout should be implemented.
- **rnn_word_model():**Function for designing the architecture for Word RNN Classifier. The *dropoutBoolean* parameter checks whether dropout should be implemented.
- **main(dropoutBoolean):** Main function for training and validation of model performance (used in CNN)
- **vanilla_rnn():** Designs the architecture of the RNN Classifier with a Vanilla layer
- **lstm_rnn():** Designs the architecture of the RNN Classifier with a LSTM layer
- **multi_lstm_rnn():** Designs the architecture of the RNN Classifier with 2 LSTM layers
- **multi_vanilla_rnn():** Designs the architecture of the RNN Classifier with 2 Vanilla layers

- **main(dropoutBoolean, gradientClipping):** Main function for training and validation of model performance (used in RNN). *GradientClipping* checks whether gradient clipping of threshold 2 should be implemented.

1.3 Experiments & Results

1. **Character Level CNN**

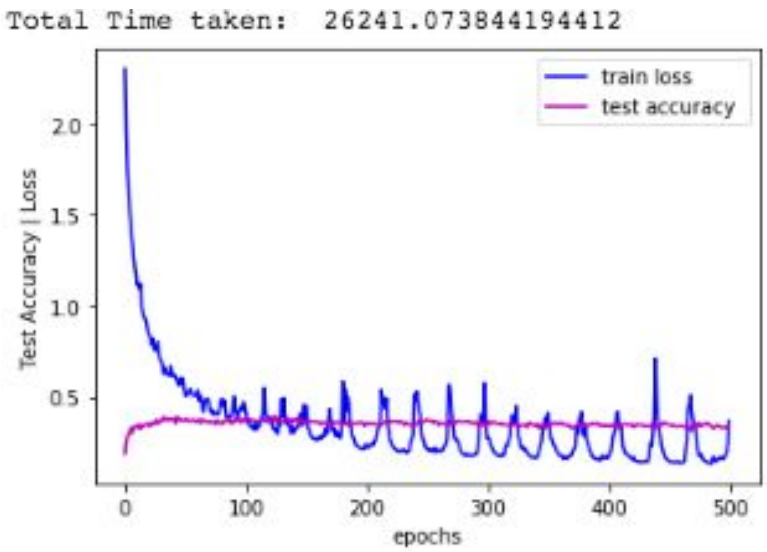


Figure 1: Training Loss and Test Accuracy of Character CNN Classifier

Figure 1 shows the entropy loss on the training data and the accuracy on the testing data against training epoch of 500 for the Character CNN Classifier.

The Character CNN Classifier receives characters ids for classification. It consists of 2 convolution and pooling layers.

As shown in the figure above, the training loss of the classifier follows a pattern of regular spikes through the training. This might be due to the usage of Mini-Batch Gradient Descent whereby for every different iteration, the model is training on a different dataset. This results in some mini-batches having a “easier” data to train and some having “harder” data to train. Training loss would still decrease as the training epoch increases, though noisier as compared to training with Batch Gradient Descent. [1]

Test Accuracy of the Character CNN Classifier converges around 40%.

2. **Word Level CNN**

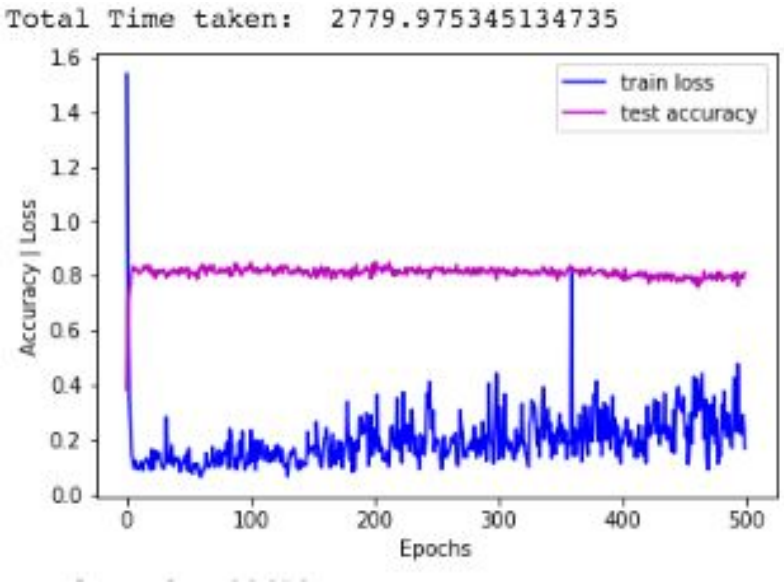


Figure 2: Training Loss and Test Accuracy of Word CNN Classifier

Figure 2 shows the entropy loss on the training data and the accuracy on the testing data against training epoch of 500 for the Word CNN Classifier.

Test accuracy of the Word CNN Classifier converges around 81%, which is a considerable improvement as compared to that of the Character CNN Classifier. This could be attributed to the fact that given the same input, character based classifier would view the input as containing more tokens as opposed to a word based classifier. This makes the task of the character based classifier harder as it must take into consideration the dependencies between more tokens.

3. **Character Level RNN**

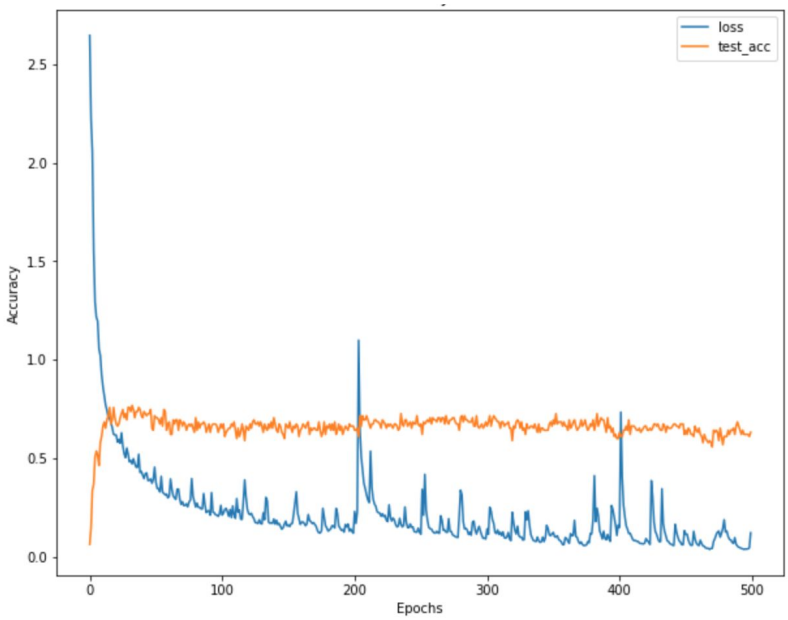


Figure 3: Training Loss and Test Accuracy of Character RNN Classifier

Figure 3 shows the entropy loss on the training data and the accuracy on the testing data against training epoch of 500 for the Character RNN Classifier.

The Character RNN Classifier has a Gated Recurrent Unit (GRU) layer and a hidden layer of size 20.

In comparison with the Character CNN Classifier, the Character RNN Classifier provides a higher test accuracy of 66.3% with 500 epochs. RNN is more suited on task that is dependent on long semantics and CNN is more suited for task that required feature detection. Furthermore, the usage of CNN is more suited for spatial data such as images and videos. On the other hand, RNN is more suitable sequential data and is ideal for text and speech analysis, explaining the higher test accuracy by the RNN classifier.

4. **Word Level RNN**

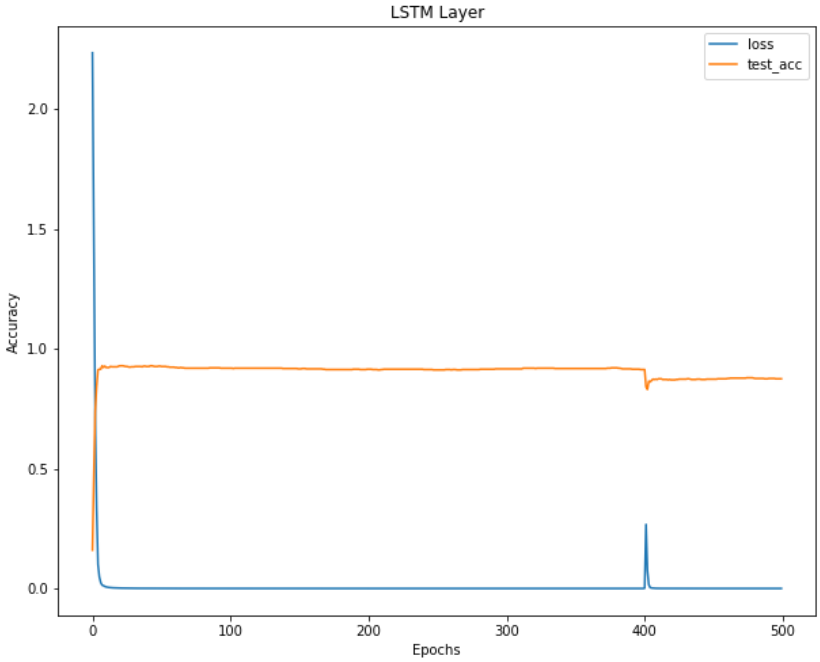


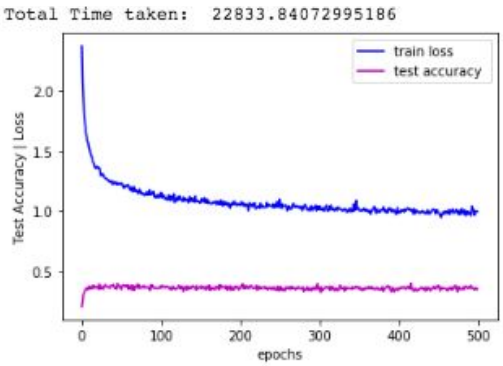
Figure 4: Training Loss and Test Accuracy of Word RNN Classifier

Figure 4 shows the entropy loss on the training data and the accuracy on the testing data against training epoch of 500 for the Word RNN Classifier.

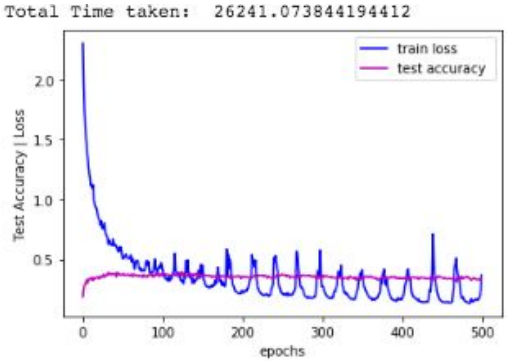
The Word RNN Classifier receives word ids for classification. It has a GRU layer and a hidden size of 20. An embedding layer of size 20 was implemented before feeding to the RNN.

The Word RNN Classifier achieved an 88% test accuracy and a 1.89e-05 training loss with 500 epochs.

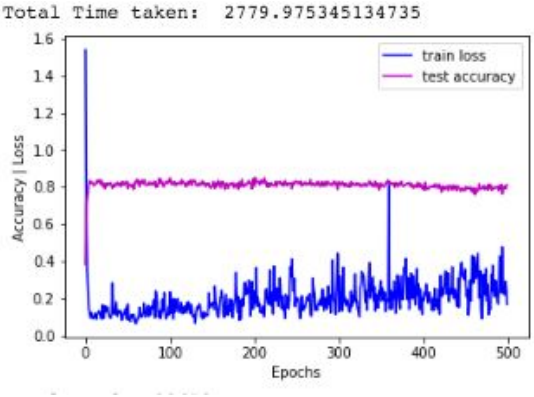
5. **Comparison between (1) to (4)**



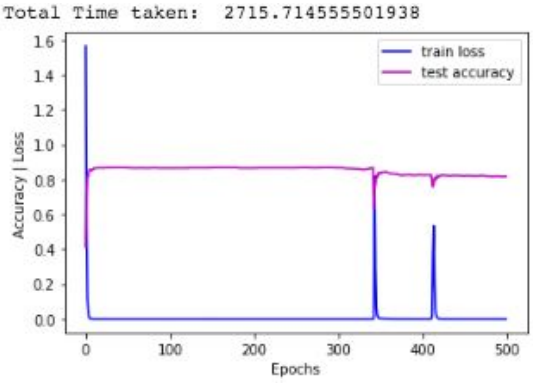
Character CNN Classifier with Dropouts



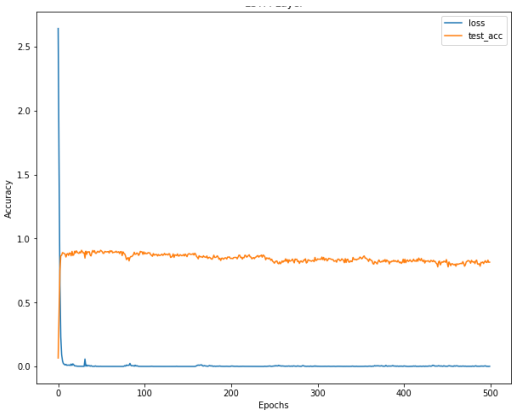
Character CNN Classifier without Dropouts



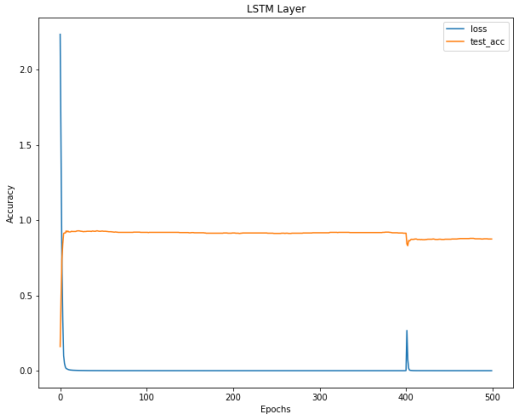
Word CNN Classifier with Dropouts



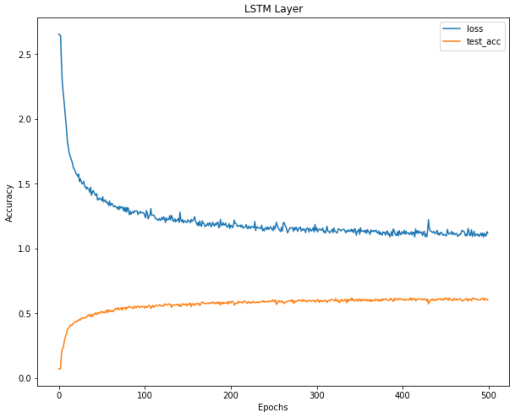
Word CNN Classifier without Dropouts



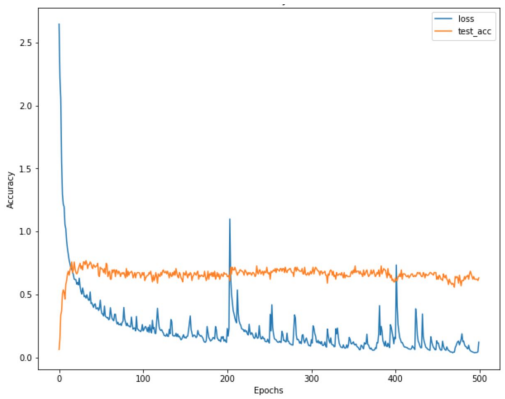
Word RNN Classifier with Dropouts



Word CNN Classifier without Dropouts



Character RNN Classifier with Dropouts



Character CNN Classifier without Dropouts

Models	CNN				RNN			
	Character CNN Classifier		Word CNN Classifier		Character RNN Classifier		Word RNN Classifier	
	Dropout	Without Dropout	Dropout	Without Dropout	Dropout	Without Dropout	Dropout	Without Dropout
Time Elapsed (s)	22834	26241	2780	2716	6590	4576	1840	1397
Test Accuracy	40%	40%	81%	82%	61%	66.3%	81%	88%
Training Loss	1	0.4	0.17	1.86e-5	1.1	0.1	0.00249	1.9e-05

Table 1: Comparison of the Time taken, Test Accuracy and Training loss across various Models implemented

As shown in Table 1, The time taken to train a Word classifier is considerably lesser than to train a Character classifier. This is due to the fact that given the same input, the Character classifier would view the input as containing more tokens as opposed to a Word classifier. This makes the task of the Character CNN classifier harder as it must take into consideration the dependencies between more tokens.

It is also observed that the introduction of dropout as a regularization technique does not improve the test accuracy of our models. One explanation to the reduction of accuracy may to due to the small network implemented. When the network is small relative to the dataset used, regularization would further lower the model capacity, hurting performance.

6 (a) In this section, we will focus on the word level model and change the respective cells based on the requirements

i) **Vanilla RNN Layer**

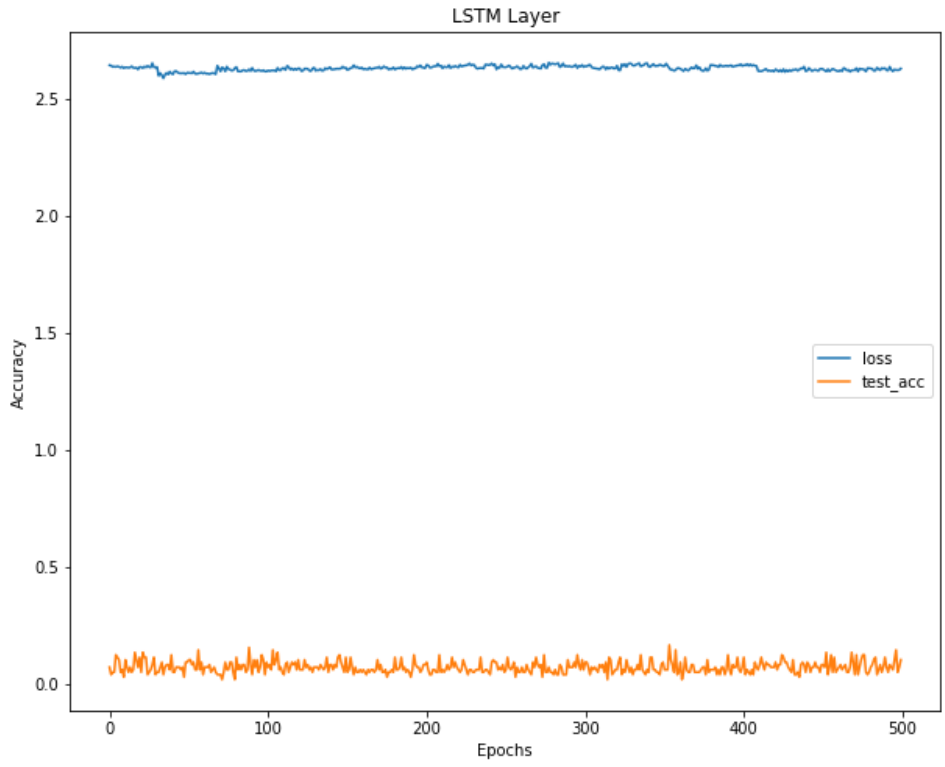


Figure 5: Vanilla RNN Layer

Figure 5 depicts the RNN network with a Vanilla RNN layer, instead of the GRU layer implemented in the previous questions. It is observed that the test accuracy of the RNN network is considerably lower, at 7.37%, than the RNN network trained with a GRU layer. This can be attributed to the occurence of the vanishing gradient problem.

To reduce the loss function, gradients of the activation function are found using backpropagation. The derivatives of each layer is then multiplied down the network to compute the derivatives of the initial layers. When small derivatives are multiplied together, the gradient of the neural network would decrease exponentially as it propagates down to the initial layers of the network.

With a vanishing gradient, the network is unable to learn, hence explaining the low test accuracy noticed in the Figure above.

ii) **LSTM RNN Layer**

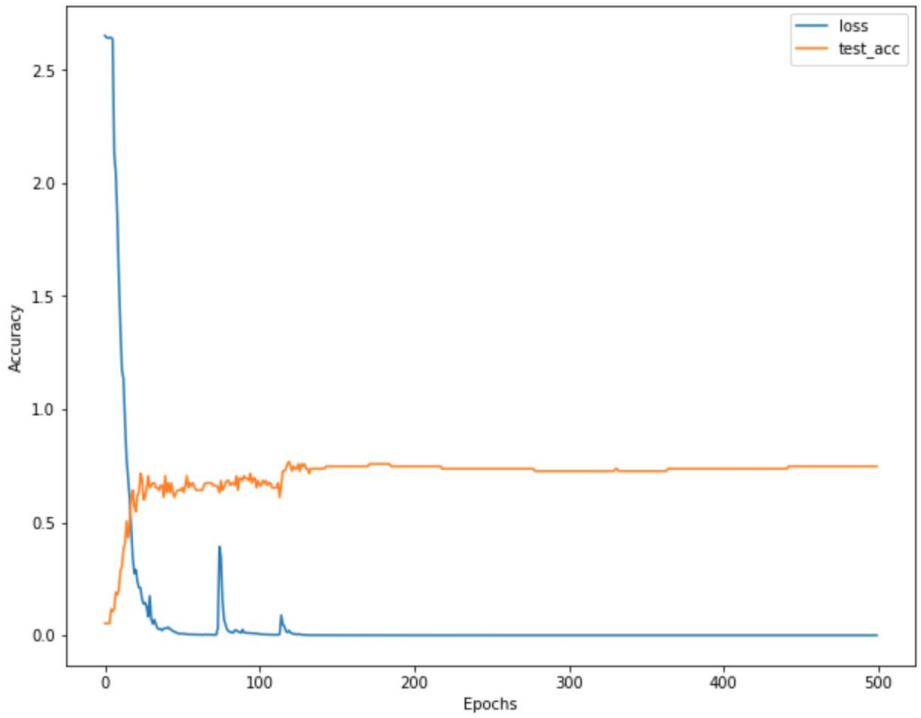


Figure 6: LSTM RNN Layer

Figure 6 depicts the RNN network with a LSTM RNN layer. It is observed that the test accuracy of the RNN network is relatively higher, at 74.7%, compared to the 7.37% shown in the Vanilla RNN Layer model (Figure 5)

In a nutshell, LSTM helps to prevent the occurence of the vanishing gradient problem by turning the multiplication of small derivatives into addition. The LSTM architecture makes it easier for the RNN to preserve information over multiple timesteps, providing an easier way for the network to learn long-distance dependencies, resulting in the higher test accuracy depicted in the figure above.

(ii) I**ncreasing RNN layer to 2 layers**

1. **Multi-Vanilla RNN**

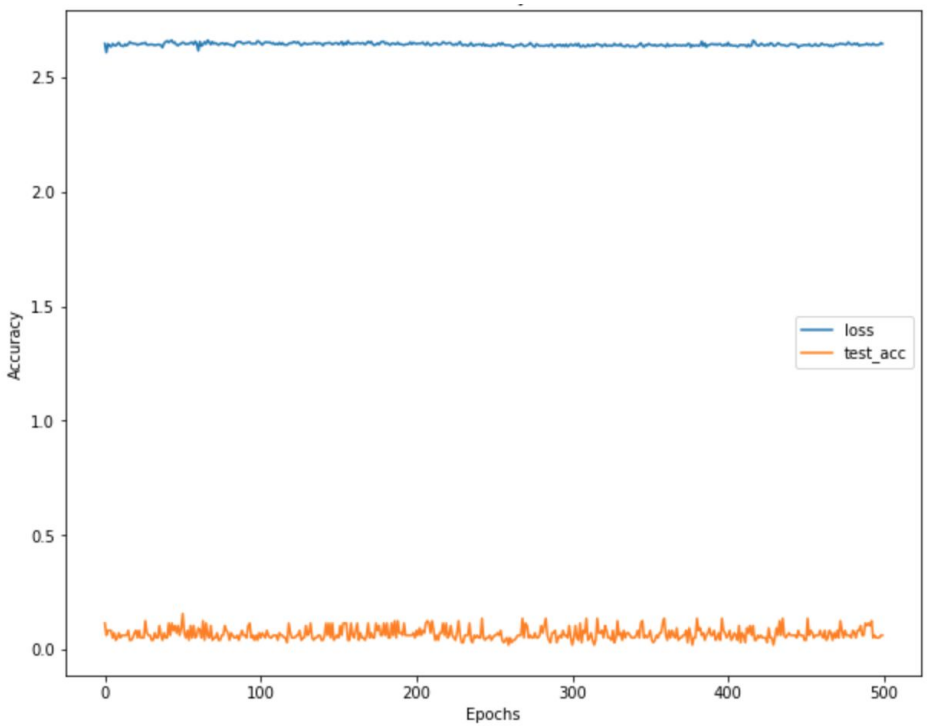


Figure 7: Multi-Vanilla RNN Layers

Figure 7 shows the RNN network with multi-vanilla RNN layers. The test accuracy of the model that implements multi-vanilla RNN layers performs better than the model implementing a single vanilla RNN layer.

The stacked vanilla layers allows the network to store more information throughout the hidden states between the input and output layers, resulting in better performance in terms of accuracy.

2. Multi-LSTM RNN

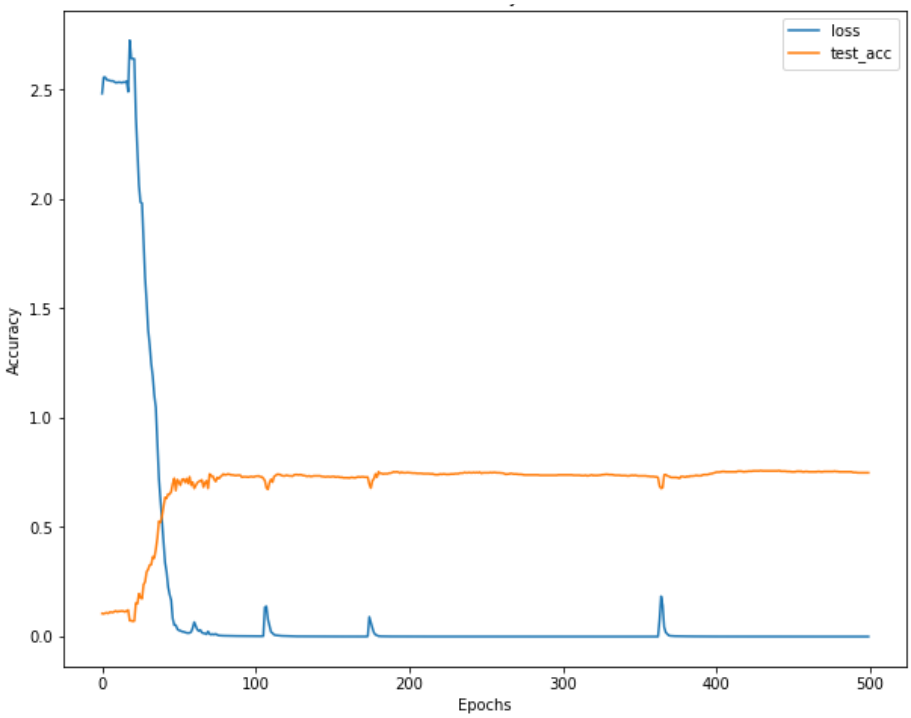


Figure 8: Multi-LSTM RNN Layers

Figure 8 shows the Word RNN model implemented with 2 layer of LSTM layers. The test accuracy of the multi-LSTM RNN layer model is an improvement from the single LSTM RNN layer model (Figure 6).

The stacking of an additional hidden layer makes the RNN model deeper, allowing it to recombine the learned representation from previous layers and create new representations at a higher level of abstraction. All of which results in a higher observed test accuracy of 75.6%.

(iv) **Add Gradient Clipping, threshold = 2**

1. **Vanilla Multi RNN**

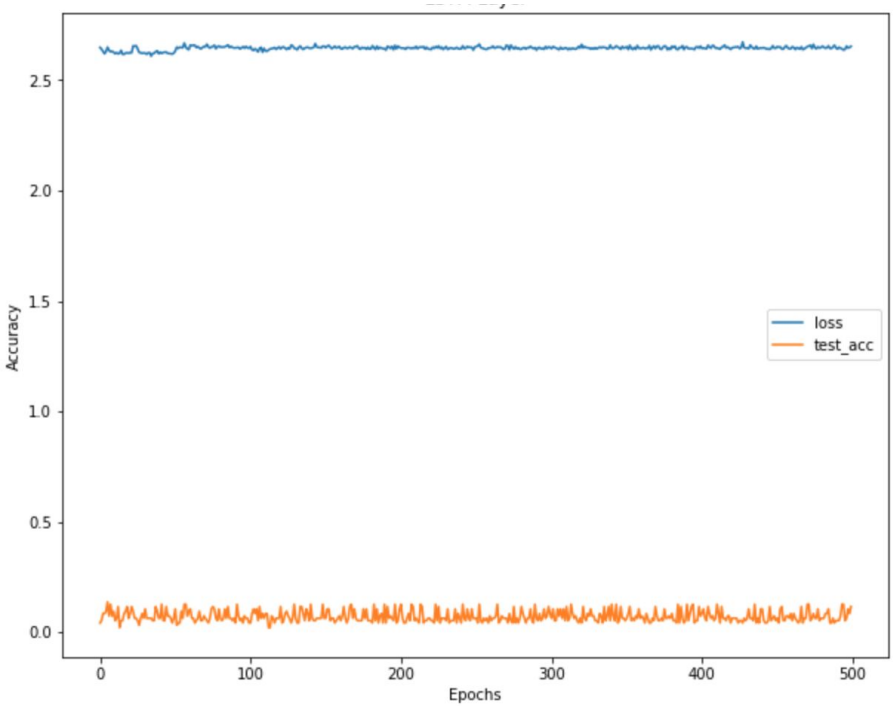


Figure 9: Multi-Vanilla RNN Layers with gradient clipping

Figure 9 shows the test accuracy and training loss of the RNN Word Classifier implemented using multi-vanilla layers and gradient clipping of threshold = 2. The test accuracy of the model is lower as compared to models implemented without the introduction of gradient clipping.

Gradient clipping is used to solve the issue of exploding gradients, however, the vanilla Word RNN Classifier suffers from vanishing gradient problem. As such. the implementation of gradient clipping would only aggravate the vanishing gradient issue faced by the classifier, resulting in an even lower test accuracy.

2. LSTM Multi RNN

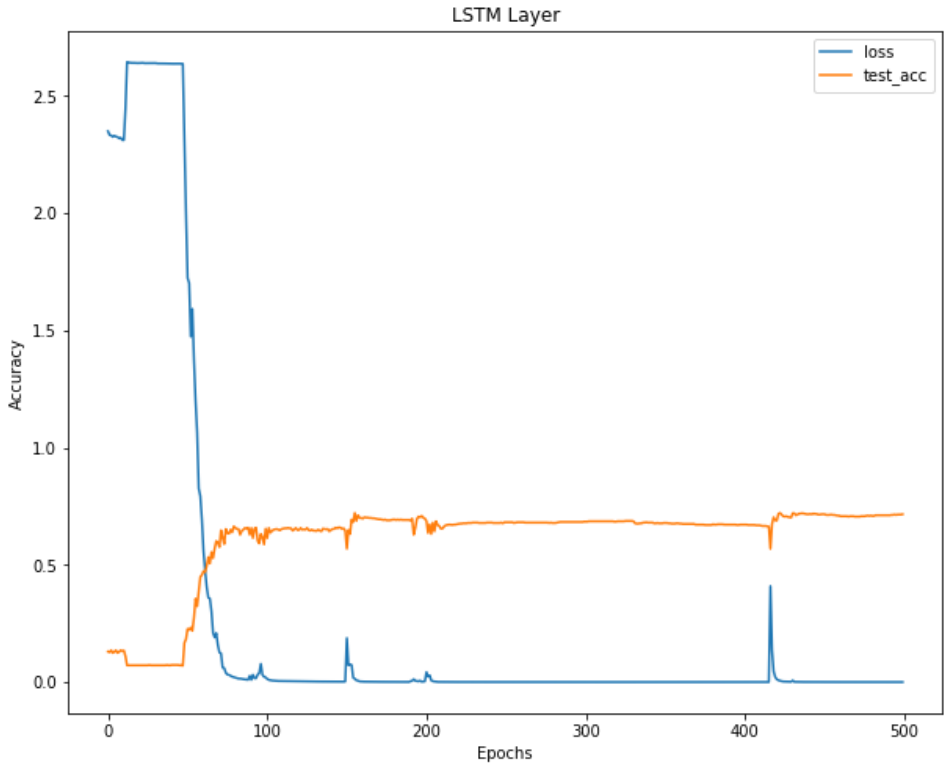


Figure 10: Multi-LSTM RNN Layers with gradient clipping

Figure 10 shows the test accuracy and training loss of the RNN Word Classifier implemented using multi-LSTM layers and gradient clipping of threshold = 2. The test accuracy of the model does not show any improvement with the implementation of a gradient clipping.

Gradient clipping is primarily used to solve the issue of exploding gradient, which the model in Figure 10 do not possess, hence rendering its implementation rather pointless.

Model	Parameters	Time Elapsed (s)	Test Accuracy
RNN Word Classifier	**Original**	1397	88%
	Vanilla	533	7.37%
	LSTM	1444	74.7%
	2 Layers (Vanilla)	580	11.6%
	2 Layers (LSTM)	1357s	75.6%
	Gradient Clipping with threshold 2 layers (Vanilla)	680	5.26%
	Gradient Clipping with threshold 2 layers (LSTM)	1842	71.1%

Table 2: Overall Comparison of the Time taken, Test Accuracy across various parameters implemented

Table 2 shows the overall comparison of the time taken and test accuracy of the RNN Word Classifier, implemented with various parameters.

Conclusion

Overall, it is noticed that Word Classifier results in better model performance than Character Classifier due to lesser token dependencies to take into consideration.

Furthermore, through the implementation of dropout on both CNN and RNN Classifier, we noticed that the usage of dropout does not always improve model performance. In our case study, we observed that the implementation of dropout ends up reducing the test accuracy of the models. It was found that when the network is small relative to the dataset, the usage of regularization techniques such as dropout would only seek to lower an already low model capacity, hurting model performance.

We also observed that the implementation of a stacked LSTM/Vanilla layer in the RNN Classifier improves the model performance. This is due to the fact that stacked layers allows for more information storage within the hidden states between the layers, hence, resulting in a higher test accuracy.

Through this project, we realised the importance of utilizing the correct technique on the correct context as the implementation of techniques on a wrong context would worsen model performance. One such case observed is the implementation of gradient clipping on the RNN Classifier with Vanilla layers. Upon the implementation of Gradient Clipping with a threshold of 2 on the RNN Classifier with Vanilla layers, the model performance dropped further. This is because gradient clipping is used primarily to solve the issue of exploding gradient, whereas the RNN Classifier with Vanilla layers exhibit the vanishing gradient problem. This renders the implementation of gradient clipping useless in improving model performance.

References:

[1] Andrew Ng, “Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization: Understanding mini-batch gradient descent: Understanding mini-batch gradient descent”, deeplearning.ai, Coursera