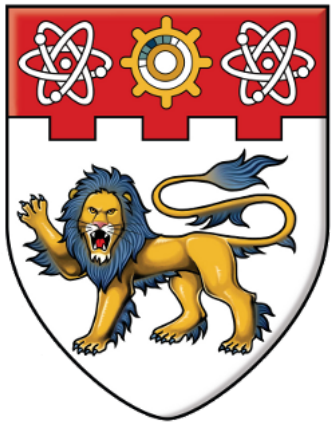


NANYANG TECHNOLOGICAL UNIVERSITY

NEURAL NETWORKS & DEEP LEARNING

ASSIGNMENT 2

Team Members / Emails:

Ang Yang / U1522821C / yang025@e.ntu.edu.sg

Ng Tze Yang/ U2520987B / tng016@e.ntu.edu.sg

1. Introduction

The goal of project 2 is to help us better understand the structure of Convolution Neural Networks and Recurrent Neural Networks and apply them to varying datasets.

Part A of project 2 deals with tuning the parameters of a CNN to recognize objects from the given dataset.

Part B of project 2 deals with experimenting with the usage of RNNs or CNNs for text classification based of the given dataset. We also experiment with dropouts and note down the elapsed time to run the different neural networks to compare performance and comment on effectiveness.

2. Methods

The various methods adopted in this project as we understand are detailed below.

2.1 Convolutional and Pooling Layers

At convolutional layers, the learning algorithm performs a convolution function on the input and the feature map with a predetermined stride and padding. Convolutional layers are great at identifying features in images such as edges and patterns. The output of the convolutional layer are then fed into a max pooling layer where it decreases the dimensions of the output by choosing the maximum element within the stride and outputs it.

2.2 SAME Padding vs VALID Padding

In Part B, the padding values are manipulated according to instructions. There is a fundamental difference in SAME padding as compared to VALID padding. Essentially, padding is an operation to increase the size of the input data. VALID padding leaves input data as is, with no padding. SAME padding pads the input data such that the produced output is the same size as the input.

2.3 Embedding layer

For RNNs in Part B, embedding is applied to the input before feeding it into the RNN. Embedding layers provide a dense representation of words and their relative meanings. Compared to the traditional bag-of-word encoding scheme, in an embedding, words are represented by dense vectors where a vector represents the projection of words into a continuous vector space. Position of the word in the learned vector space is thus referred to as the embedding.

2.4 Gradient Clipping

Gradient clipping is used to prevent 'exploding' gradients. It clips the gradient between two numbers to prevent it from getting too large. It is a method to regularize the weights. Exploding gradients can lead to an unstable network and result in models that are unable to learn from training data.

2.5 Dropouts

The term 'dropout' refers to a probability that the model negate the outputs of a particular neuron while undergoing training. Using dropouts prevents the network from overfitting on the training data and have too high weights on a few particular neurons.

3. Experiments and results

The following is a simple presentation of our experiment and results for each question and its parts.

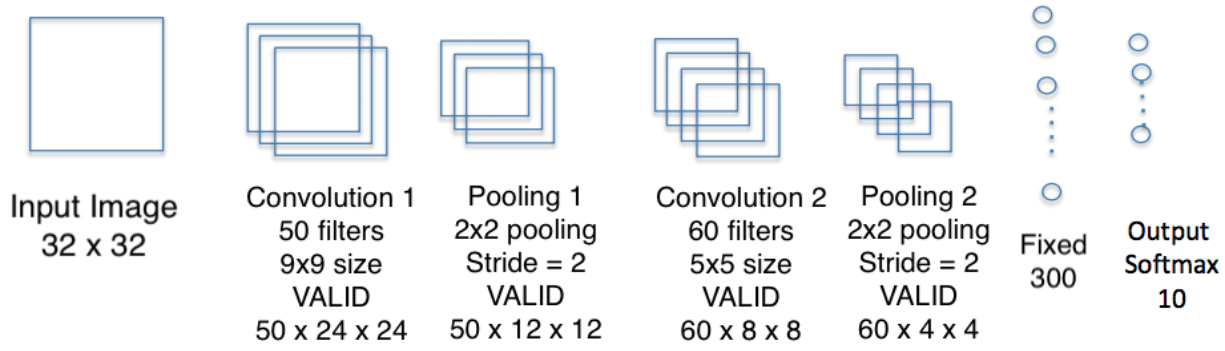
3.1 Part A Question 1

In this problem, we are given a dataset containing 10,000 RBG images of size 32x32 and are tasked to train a neural net to classify these images into 10 different classes. Testing of the classifier is done on 2,000 similar images.

The first step of the program deals with preprocessing the data which includes extracting the data from a text file and modifying the classification column of the data into a one-hot matrix to be used in the model.

Next, we perform a max-min scaling on both the training and test pixel data. It is important to note that for both sets of data, minimum and maximum values were obtained from the pre-scaled training set data.

We are now going to build the computational graph of the Neural Network. The architecture of the CNN is shown in figure below:

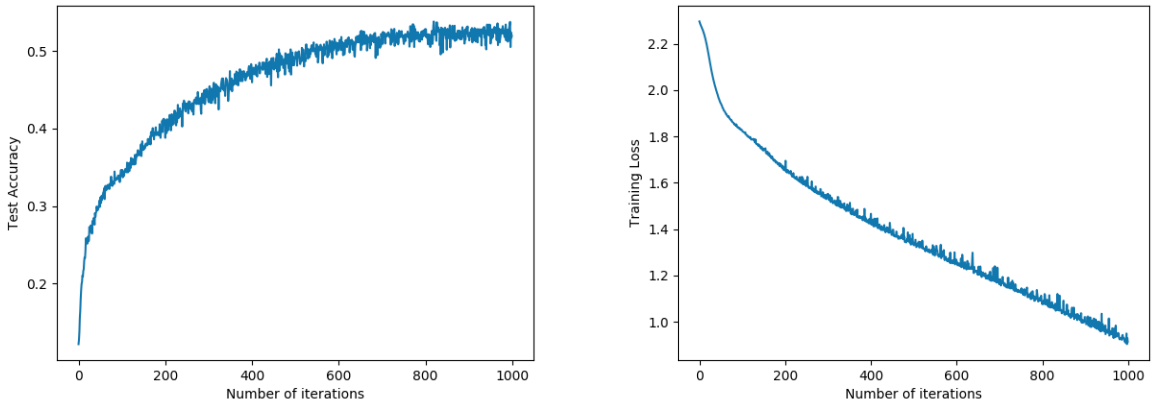


Architecture of CNN for Question 1a

The weights of a CNN are a four-dimensional tensor where each dimension corresponds with the length and width of the convolutional kernel and the number of input and output channels. These weights, along with the two dimensional weights of the fully connected layer, were initialized using the truncated normal method as explained previously and set the biases to zero.

For the activation function, we used the relu function and calculated the loss of the model based on its softmax cross entropy.

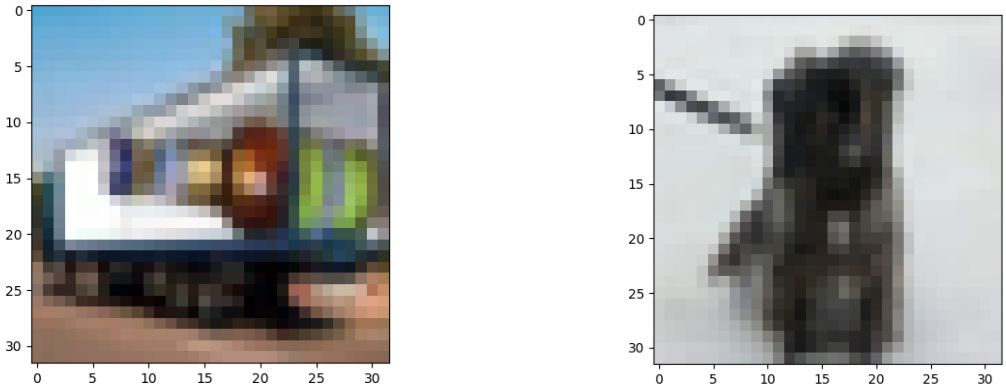
Finally, we trained the model with the full training data set with a mini-batch size of 128 using the gradient descent optimiser. The model was trained using 1000 epochs. With it, we achieved a training accuracy rate of **51.7%.**



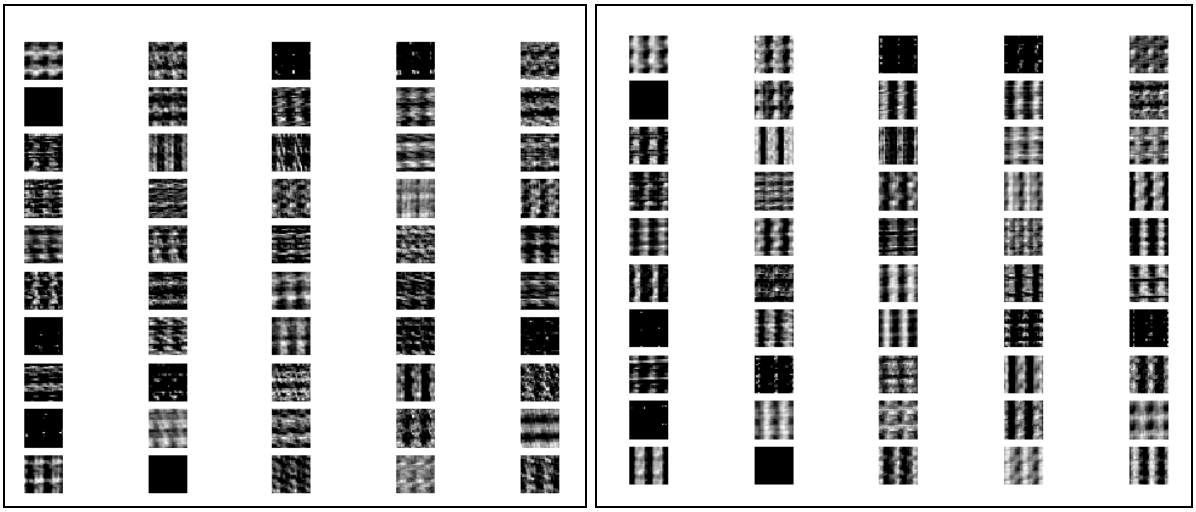
Test accuracy and Training loss for part A question 1

Looking at the graphs above, we noticed that for epochs 600-1000, training loss continued to decrease at a constant rate while improvements to test accuracy stagnated. This suggests that the model is overfitting its parameters to the training data and not yielding improvements to the test accuracy.

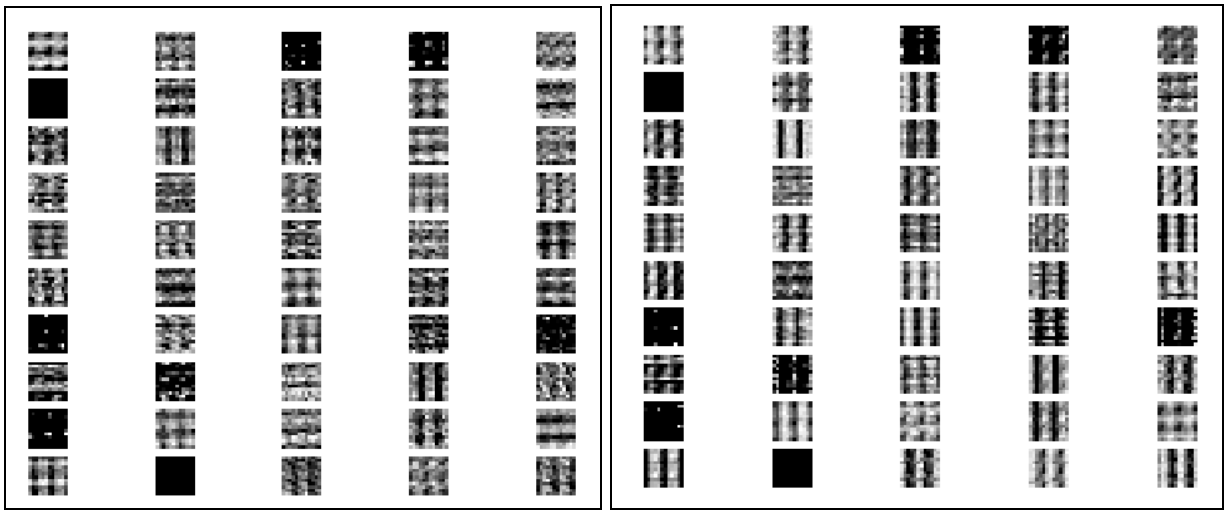
Below are the test patterns as well as the feature maps at each convolutional and pooling layer.



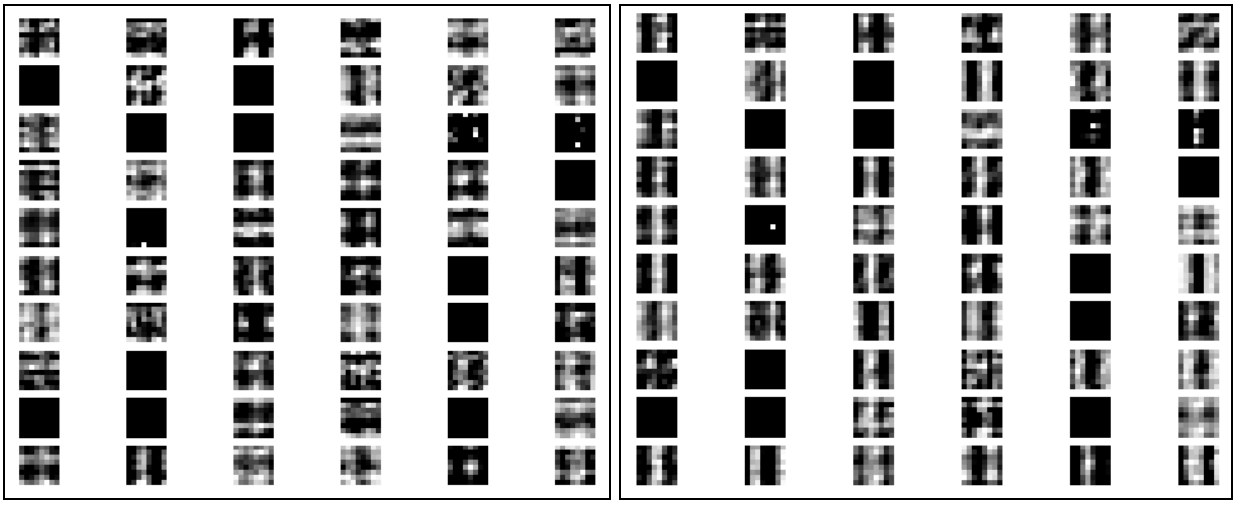
2 Test Patterns used for part A question 1



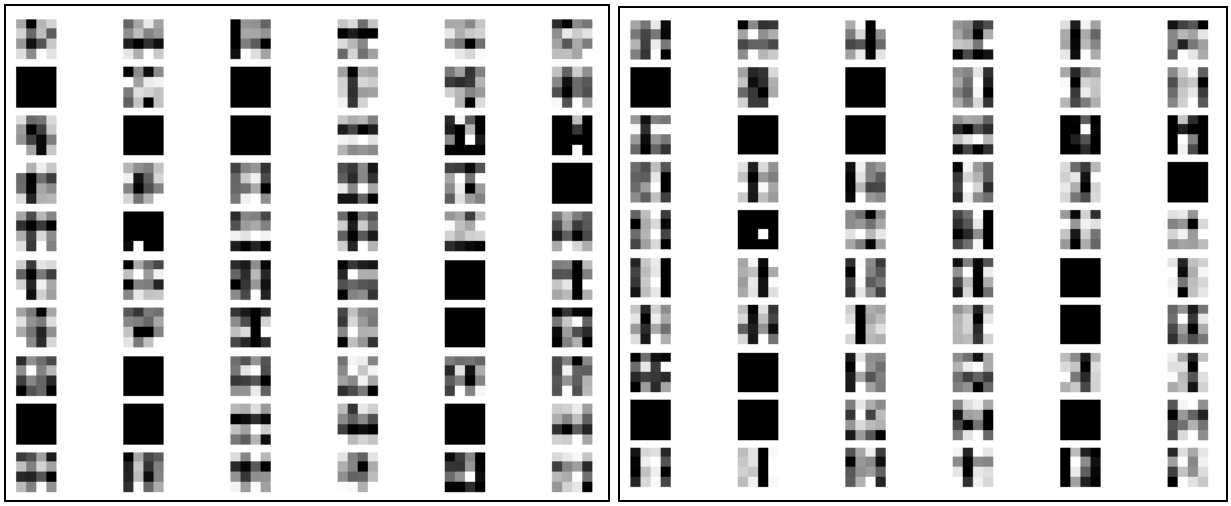
Feature Maps at convolutional layer 1



Feature Maps at pooling layer 1



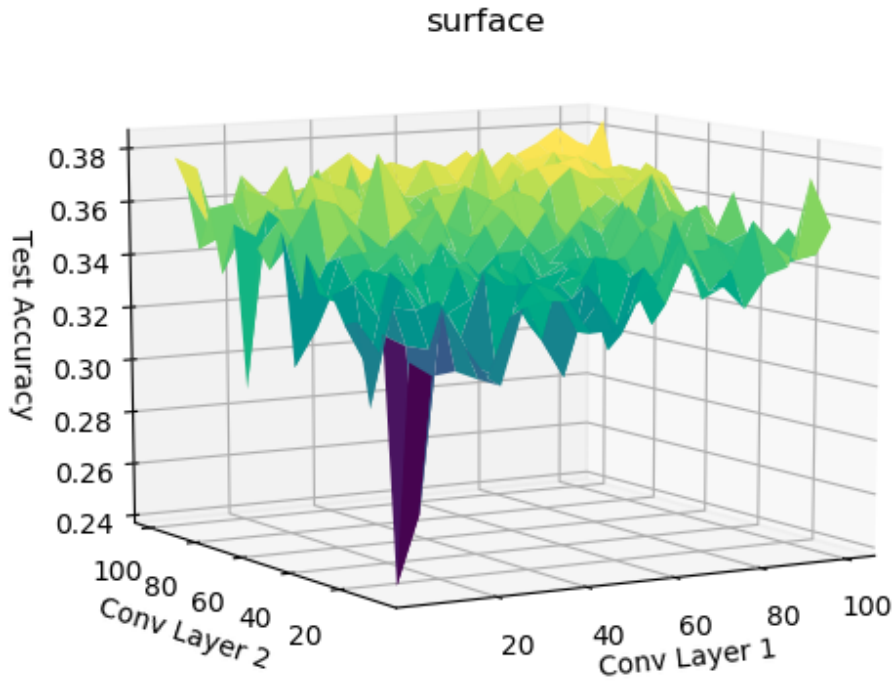
Feature Maps at convolutional layer 2



Feature Maps at pooling layer 2

3.2 Part A Question 2

Next, we used a grid search to find the optimal number of feature maps at the convolutional layers. We performed the search from 5 to 100 feature maps at each convolutional layer at intervals of 5. To save time, we trained the model on 100 epochs per feature map combination. While most searches yielded similar results, varying only between 30 - 40% accuracy, the most accurate feature map number achieved was with 90 features in the first layer and 100 in the second. The graph below plots the accuracies achieved for each combination searched.



Plot of Test Accuracies against the number of feature maps at each layer

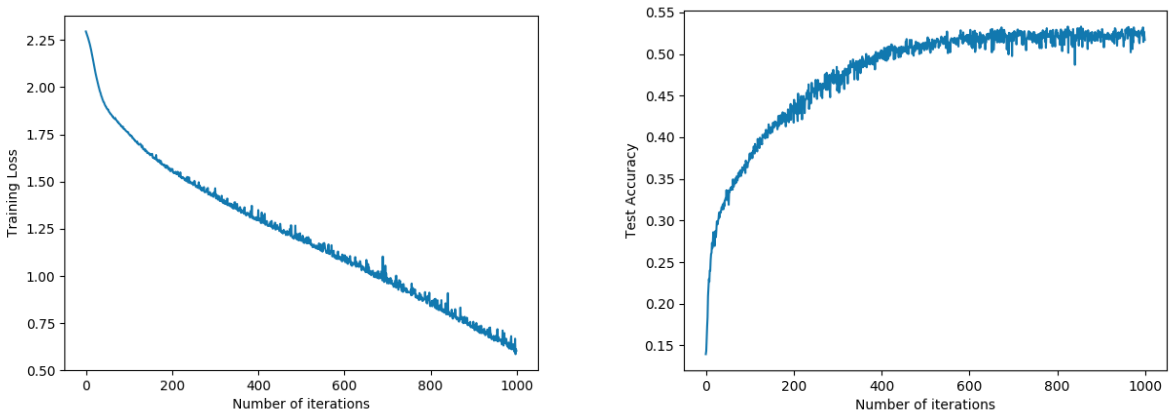
Optimal feature map numbers = 90 & 100

3.3 Part A Question 3

Setting the number of feature maps of the first convolutional layer to 90 and the second layer to 100, we next experimented with the different optimisers available in the tensorflow library.

Question 3a

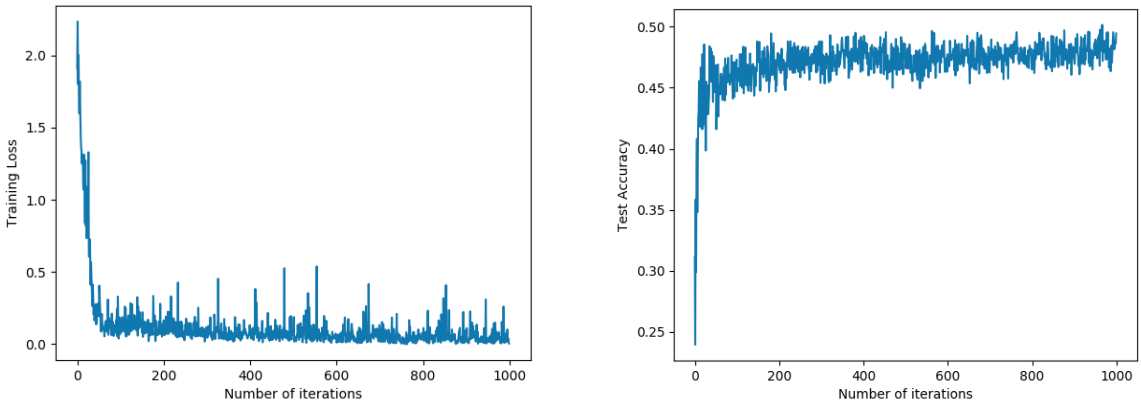
Using the new hyperparameters, we first trained the model using the Momentum Optimiser with momentum parameter = 0.1. The use of the momentum parameter incorporates past learning gradients into the current gradient. With it we obtained a final training error of 0.531 and a testing accuracy of 53.3%.



Test accuracy and Training loss for part A question 3a

Question 3b

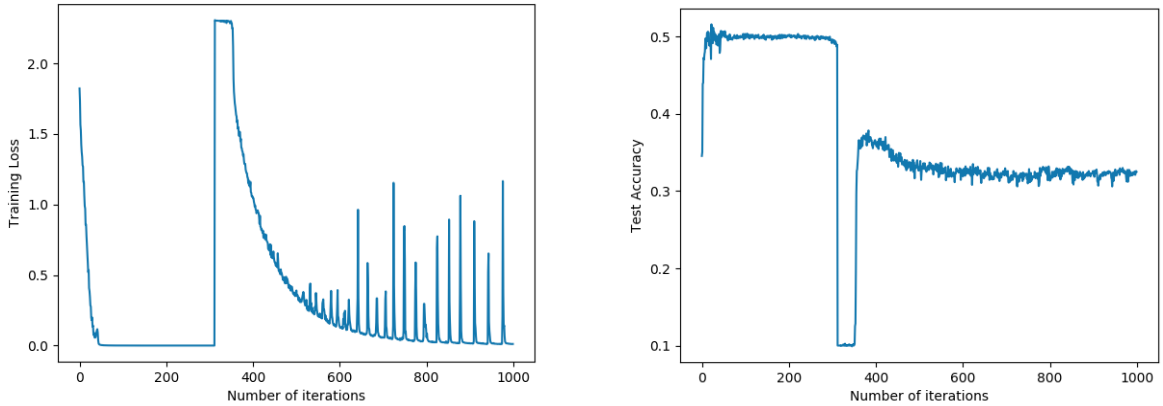
Next, we trained the model using the RMSProp Optimiser with its default parameters. RMSProp uses an exponentially decaying average in the calculation of the new weights and biases. With it we obtained a final training error of 1.67E-2 and a testing accuracy of 49.8%.



Test accuracy and Training loss for part A question 3b

Question 3c

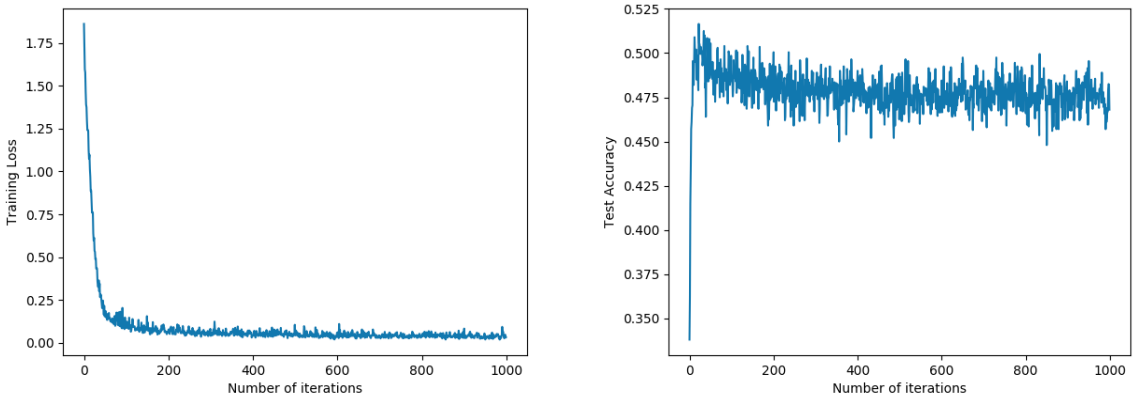
The last optimiser we experimented with was the Adam Optimiser with its default parameters. The Adam Optimiser is a combination of both techniques used by the momentum and RMSProp Optimiser. At around epoch 300, the training loss variable encountered an error and caused it to spike upwards. Upon further investigation, we noticed that loss was in the order or 1E-8 just before the overflow and concluded that an overflow of the variable had occurred and therefore we will ignore all results after that epoch. The test accuracy just before the error was 50.4%.



Test accuracy and Training loss for part A question 3c

Question 3d

Lastly, we added dropouts at the convolutional layers to the Optimiser. The dropout rate used was 10%. With it we obtained a final training error of 3.43E-2 and a testing accuracy of 48.5%. However, the maximum accuracy achieved by the model was 51.9%.



Test accuracy and Training loss for part A question 3d

3.4 Part A Question 4

From our experiments, we can see that the accuracies across all experiments have very similar accuracies, with the difference between the highest and lowest around 5%. All accuracies achieved are around 50% accuracy, which is not very impressive for implementation. Greater improvements in the accuracy may be achieved by increasing the size of the training and testing datasets as well as improving the resolution of the pictures used.

In terms of accuracy, the Momentum Optimiser yielded the best accuracy at 53.3%. This is followed by the Gradient Descent Optimiser with 50&60 feature maps in question 1. While these two experiments yielded the best accuracies, it is important to note that they also took the most number of epochs to converge. Looking at their test accuracy graphs, it took them about 400 epochs for the accuracy to converge whereas the other experiments took only about 100 epochs.

It is also premature to conclude that the Momentum Optimiser is the best optimiser. The Adam optimiser only had 400 epochs before it faced the error and when we added dropouts to it, it achieved a comparable accuracy at 51.9%. In terms of training error, the RMSProp and Adam Optimiser also consistently achieved lower values than the Momentum Optimiser.

These discrepancies may be attributed to the overfitting of the model to the training data. It is likely that if we repeated the experiment with cross-validation techniques we will be able to achieve better results for the RMSProp and Adam Optimiser.

Model	Accuracy
Gradient Descent with 50&60 Feature Maps	51.7%
Momentum Optimiser with 90&100 Feature Maps	53.3%
RMSProp Optimiser with 90&100 Feature Maps	49.8%
Adam Optimiser with 90&100 Feature Maps	50.4%
Adam Optimiser with 90&100 Feature Maps and Dropouts	48.5%

3.5 Part B Question 1

In this problem, we design a Character CNN using two convolution and pooling layers. With the required parameters and an epoch value of 1000, we achieved the following results:

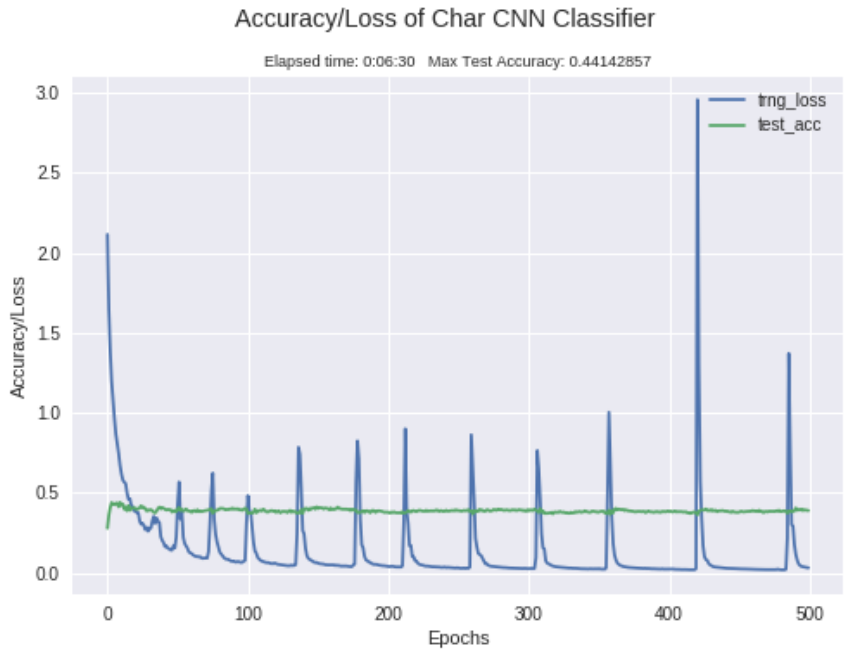


Figure 1: Graph of Training loss and Test Accuracy on Char CNN Classifier

As training loss approaches zero, test accuracy tapers off at around 43%.

Explanation of spikes in training loss vs iterations

In the above and following graphs we notice that there are spikes in training loss as the number of iterations increase. We conducted some research and discovered that these spikes are a consequence of mini-batch gradient descent for Adam Optimizers. These spikes come about due to the chance of bad data within some mini-batches.

3.6 Part B Question 2

For question 2, we designed a word convolutional neural network for our input data. Below you will find the results of our CNN model.

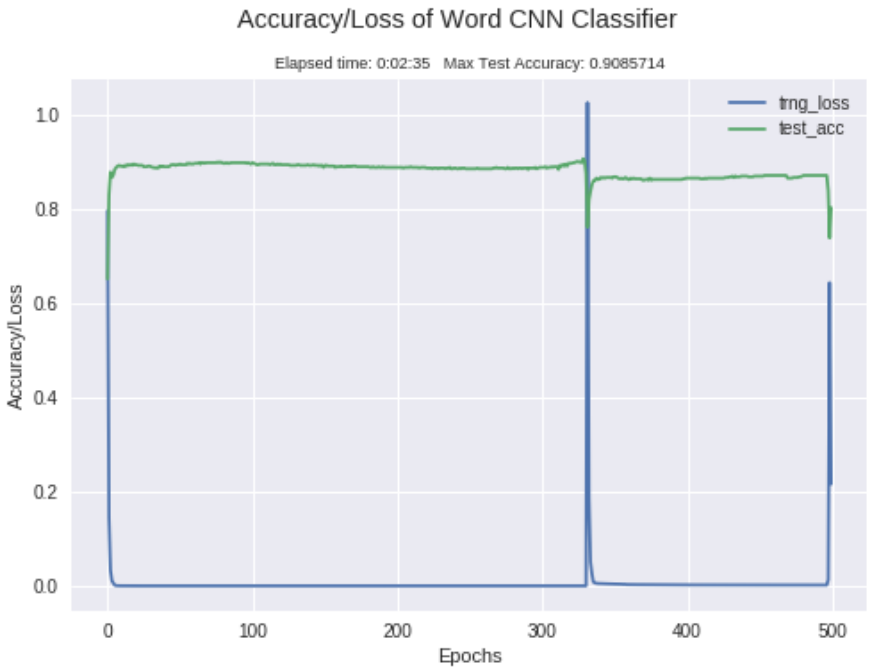


Figure 2: Graph of Trng Loss vs Test accuracy on Word CNN Classifier

As seen above, as training loss approaches 0, test accuracy approaches 90%, showing that using words ids produces a model with higher accuracy.

3.7 Part B Question 3

For this part, we designed a Character Recurrent Neural Network as required by the question that receives character ids. The RNN has a Gated Recurrent Unit layer and a hidden layer of size 20. GRUs are similar to LSTM but have fewer parameters (it does not have an output gate like an LSTM) and has been shown to perform better on smaller datasets. Below you will find the graph for out implemented model.

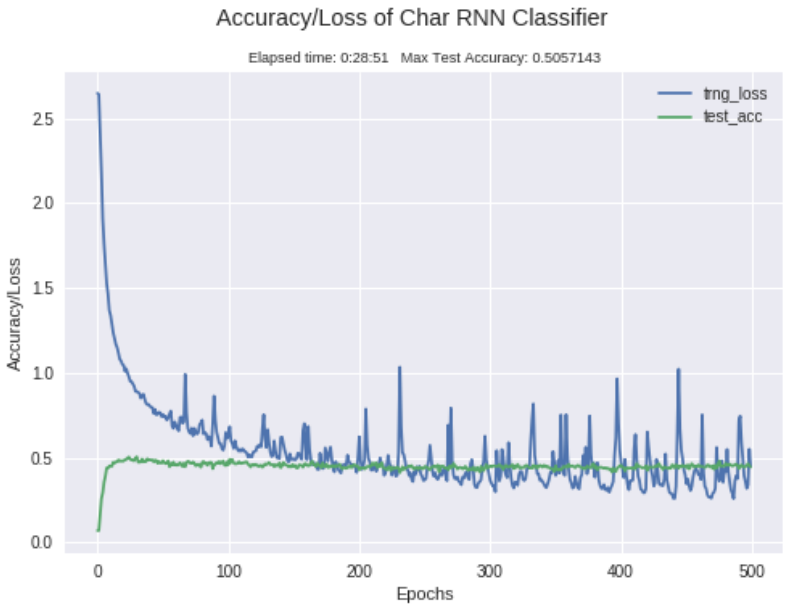


Figure 3: Graph of Trng Loss vs Test Accuracy for Char RNN Classifier

As seen above, test accuracy approaches 50% as training loss approaches 0. The spikes have in training loss have been explained in the latter part of Part B Question 1. Comparing our Accuracy vs Loss graph results of a RNN classifier for Characters against the results of the CNN classifier, we notice that the RNN classifier achieves slightly better accuracy results.

3.8 Part B Question 4

We were required to design a Recurrent Neural Network that receives word ids for classification. The RNN is GRU layer and a hidden layer of size 20. An embedding layer of size 20 was also implemented before feeding into the RNN.

The graph produced is shown below.

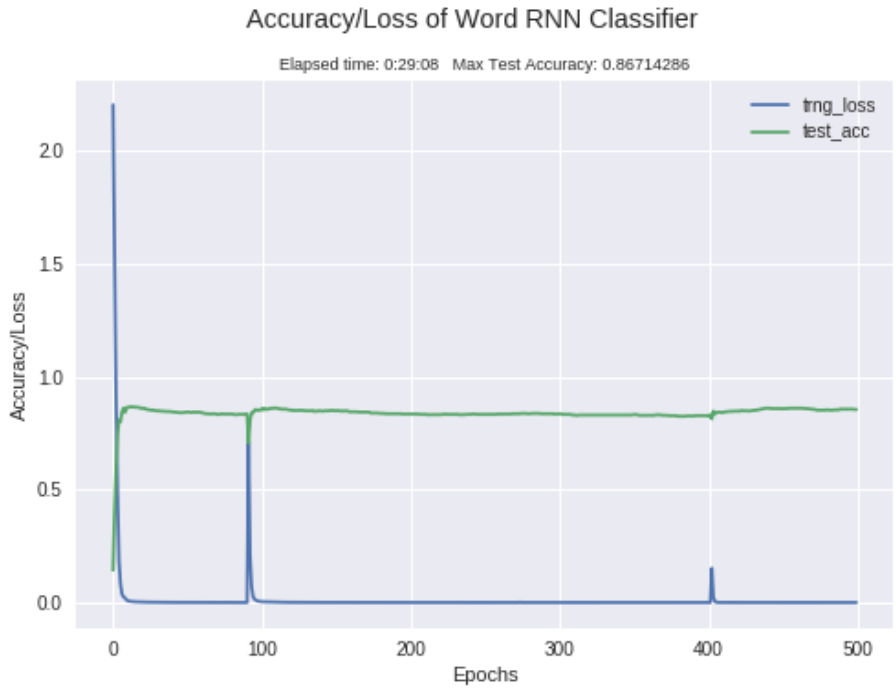


Figure 4: Graph of Trng Loss vs Test Accuracy for Word RNN Classifier

As shown above, the test accuracy of this model is relatively high, approaching 87% as training loss approaches 0.

3.9 Part B Question 5

Comparison of CNN vs RNN

We notice a difference in test accuracies and running times when we compare the 4 graphs generated from the different models in the previous 4 questions.

Figure 5: Comparison of CNN against RNN for Char Classifiers

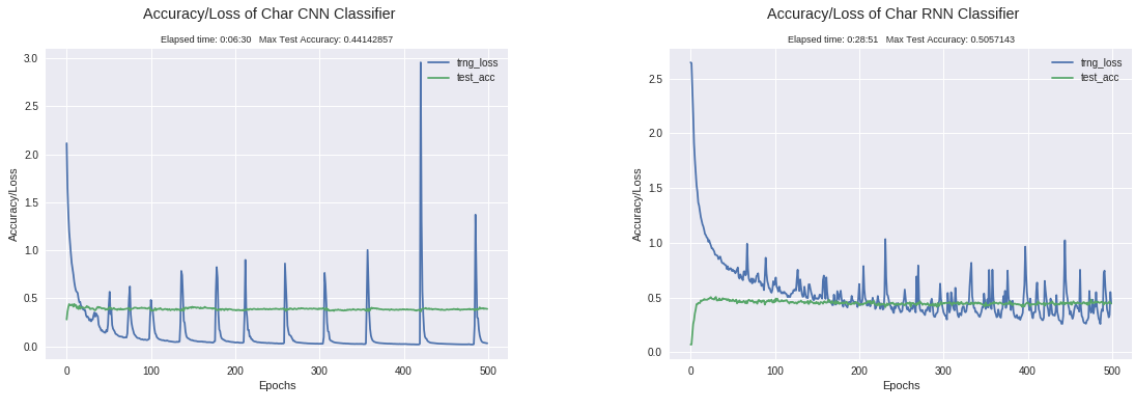
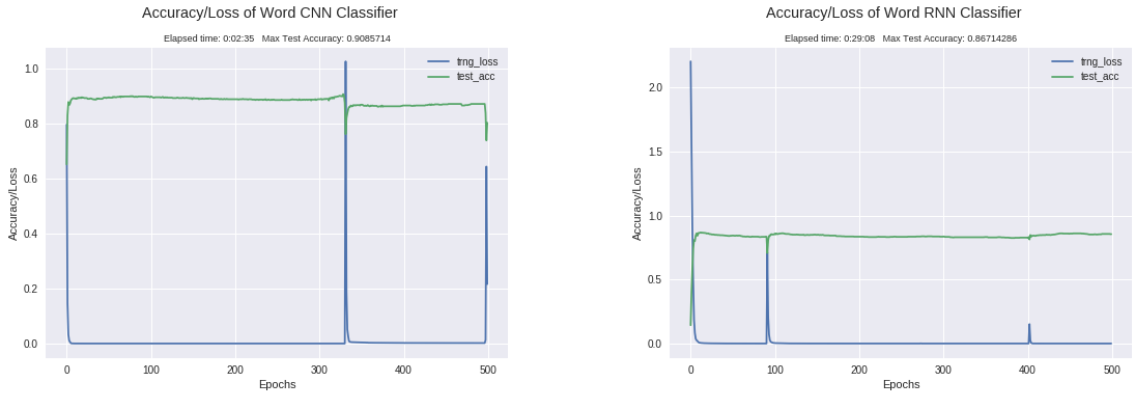


Figure 6: Comparison of CNN against RNN for Word Classifiers



As seen from the above graphs, using words as input as compared to characters produces a higher test accuracy. RNNs also take a significantly longer time as compared to CNNs. Although RNNs are a more common sense approach to classifying text due to the sequential nature of sentences, they take a long time to train as shown above.

Below you will find a summary of the running times and test accuracies of the 4 models.

Model	Running Time	Test Accuracy
Char CNN Classifier	6:30min	44%
Char RNN Classifier	28:51min	50%
Word CNN Classifier	2:35min	90%
Word RNN Classifier	29:06min	86%

We observe that CNNs take significantly lesser time than RNNs to train while not compromising too much on accuracy.

RNNs might provide higher accuracy but at the expense of running time. In this case, applications which require high accuracy such as language translation should use RNNs. However applications which require speed such as classification problems should use CNNs. As shown from our results CNNs assigns the labels within a shorter period of time while preserving the accuracy.

Comparison of using dropouts vs not using dropouts

Dropouts are an approach to regularization for neural networks to prevent overfitting. The graphs below show our results using dropout and not using dropout for the different questions.

Figure 7: Comparison of CNN against RNN for Char Classifiers using Dropouts

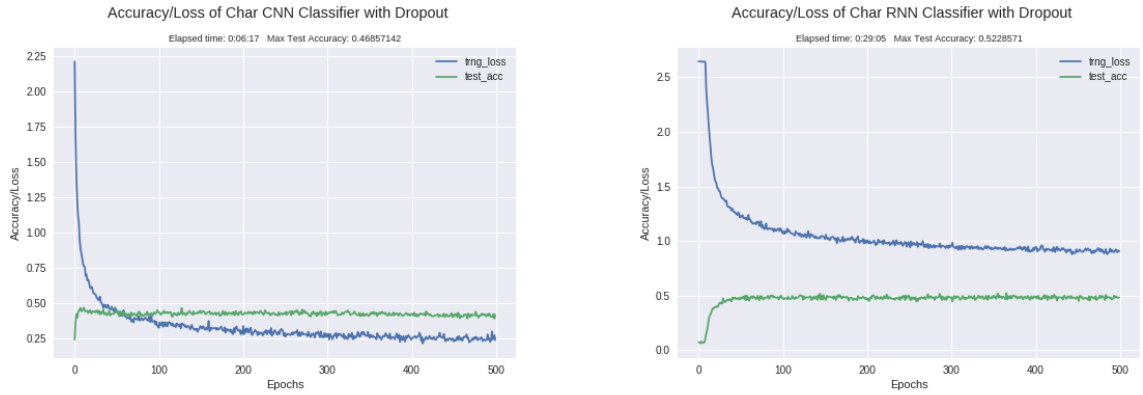
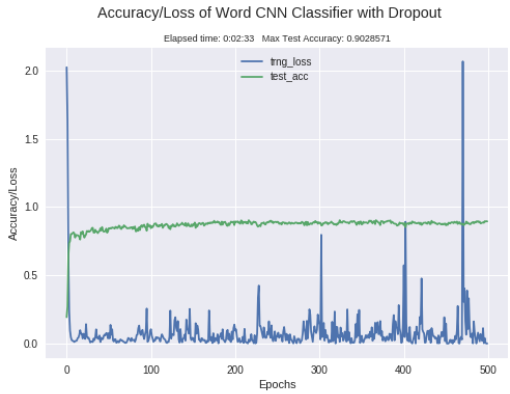
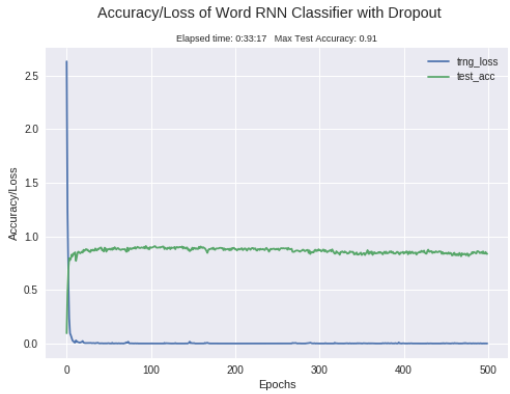


Figure 8: Comparison of CNN against RNN for Word Classifiers using Dropouts





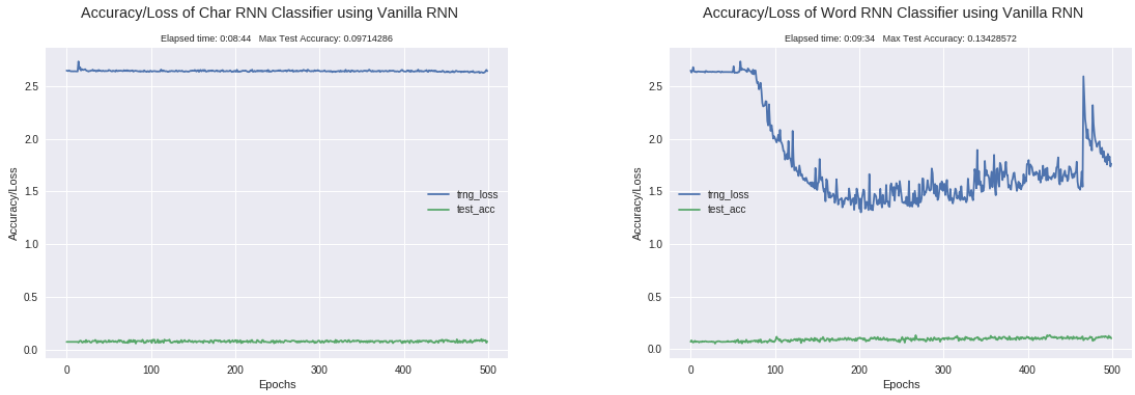
In the above side by side comparisons, we notice that the graphs without dropout do show signs of overfitting. We observe that the plot of training loss consistently decreases while test accuracy does not and remains unstable. This shows that while the model improves on training data, when presented with new data from the test set, the error remains large. This means that the model fits the training data, but has not learned to generalize to new situations.

Our observation is validated in the graphs which applied dropouts. Overfitting is reduced as seen from the tandem increase / decrease in test accuracy / training loss, especially in our char CNN/RNN Classifiers.

3.11 Part B Question 6

Vanilla RNN Layer Results

Figure 9: Char and Word RNN with Vanilla RNN Layer

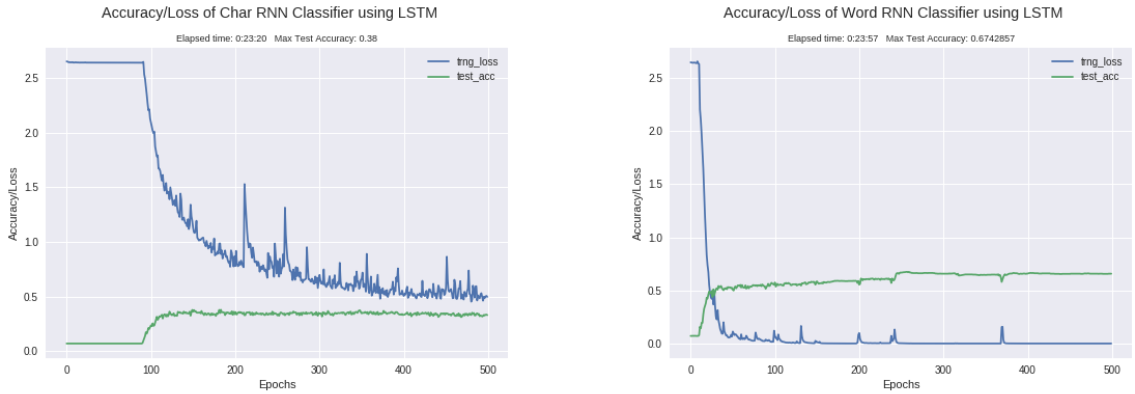


As seen from the above graphs, the results with Vanilla RNN layers are relatively disastrous and do not produce any tangible results. This may be the result of exploding gradients as shown from how the model is unable to gain traction on the training data. This is seen from the non-existent training loss in the Char RNN model and the unstable plot of training loss in the Word RNN model.

The exploding gradients problem arises during the backpropagation to the initial layer. The gradients coming from deeper layers have to go through continuous multiplications due to the chain rule and as they approach earlier layers they get too large if the layers have large values (>1) and eventually crash the model.

LSTM Layer Results

Figure 10: Char and Word RNN with LSTM Layer

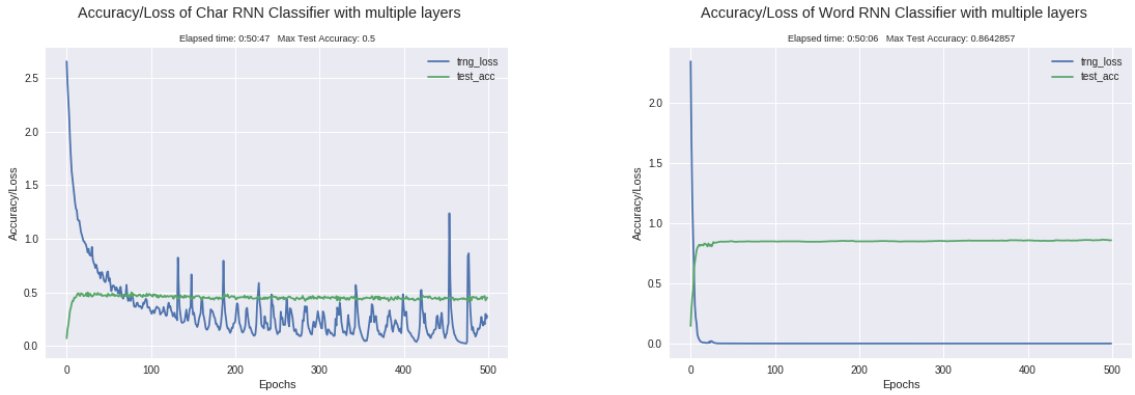


Gradient explosion can happen due to the inherent instability of the model, thus we apply LSTM, which helps curb gradient explosion. This can be seen from how the graph plots start to show better results as compared to a Vanilla RNN layer. Run times still remain relatively long and the test accuracy starts to improve and training loss drops.

However overfitting seems to occur for the char RNN model as seen from how the training loss drops while test accuracy remains constant, indicating that the model may just be fitting the model to the training data.

RNN 2 Layer Results

Figure 11: Char and Word RNN with Multiple Layers (2)

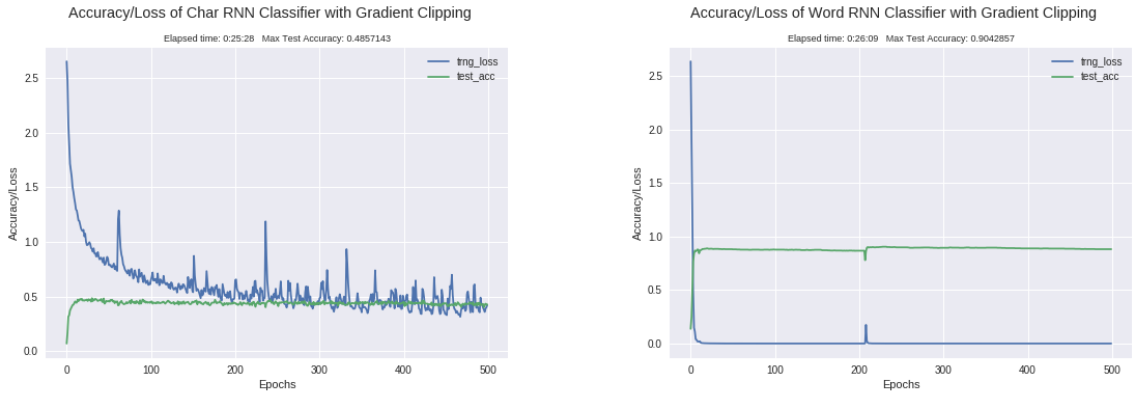


Using multiple layers, 2 in this case, seems to produce more accurate results. Using more hidden layers increases the depth of the neural network and allows the problem to be better represented. An additional layer in this case allows the model to better represent the problem, in which each layer solves a part of the problem before passing it to the next layer, similar to a processing pipeline.

As shown in the results, this produces better training/testing results.

Gradient Clipping to RNN training Results

Figure 12: Char and Word RNN with Gradient Clipping



Gradient clipping clips the gradient between 2 numbers to prevent them from getting too large. A threshold of 2 is set here and this seems to be effective in preventing exploding/vanishing gradients. This can be seen in how different the results are in the above 2 graphs compared to the results from RNNs with Vanilla RNN layer. Gradient clipping also seems to perform slightly better than multilayer RNNs. Below you see a table with a summary of the RNNs with different parameters applied and their corresponding results.

Model	Parameter Applied	Running Time	Test Accuracy
Char RNN	Vanilla RNN Layer	8:44min	0.07
Word RNN	Vanilla RNN Layer	9:34min	0.13
Char RNN	LSTM Layer	23:20min	0.38
Word RNN	LSTM Layer	23:57min	0.67
Char RNN	Multiple Layers	50:47min	0.5
Word RNN	Multiple Layers	50:06min	0.86
Char RNN	Gradient Clipping	25:28min	0.48
Word RNN	Gradient Clipping	26:09min	0.90

4. Conclusions

4.1 Part A conclusion

Part A uses a multilayer convolutional neural network to perform object recognition on a series of pictures. In these experiments, we alternated the number of feature maps to identify the combination that achieved the best accuracy. We also experimented with the Momentum, RMSProp and Adam Optimisers to evaluate their performances on the dataset.

We conclude that while the Momentum Optimiser achieved the best accuracy for this dataset, multiple problems with the Adam Optimiser impacted their performance and if we were to perform the experiment again with cross validation techniques it is possible that the Adam Optimiser will outperform the Momentum Optimiser.

4.2 Part B conclusion

Part B examines the usage of different datasets, either char or word, for classifying texts in paragraphs. We also experiment with using either CNNs or RNNs and running them with different parameters for text classification.

The conclusion is that CNNs run significantly faster than RNNs and do not compromise too much on test accuracy. In fact the accuracy for word CNN seems to be better than word RNN model. **This could be attributed to how words have more features** and thus a CNN model is suited for this classification problem.

Also we notice that both models have a tendency to overfit, hence running with dropouts to reduce overfitting is a good measure. We then compared the results of different parameters for the RNN models. Our observation shows that a Vanilla RNN layer succumbs to gradient explosion and thus additional parameters such as LSTM cell units, multiple layers and gradient clipping are required to keep the gradients in check and prevent the model from crashing. Based on our graphs, gradient clipping produces the best results and is thus the most effective parameter for this text classification problem.