# CZ4042 NEURAL NETWORK & DEEP LEARNING

# ASSIGNMENT 2

# ASSIGNMENT REPORT

Shearman Chua Wei Jie (U1820058D)

LAB GROUP: CS4

## Contents

# Part A: Object Recognition

## Question 1

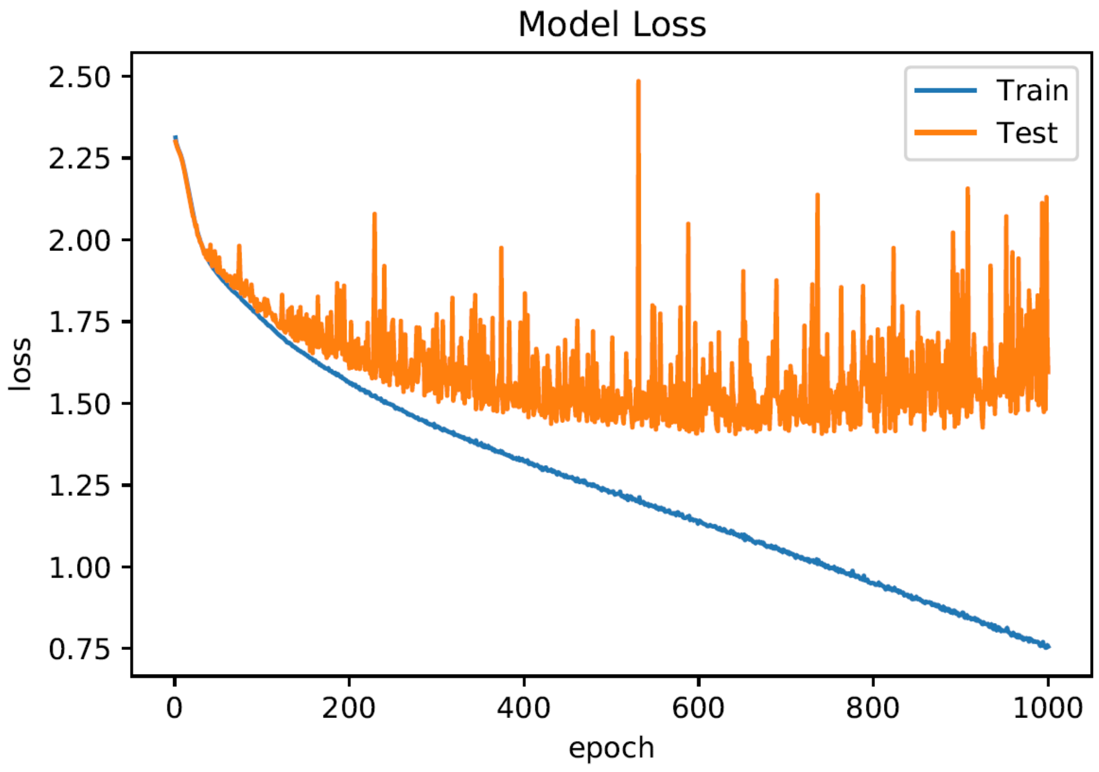
**Train the network using mini-batch gradient descent learning for 1000 epochs. Set the batch size to 128 and learning rate $\alpha$ = 0.001.**

For this question, we first create a sequential Convolutional Neural Network consisting of:

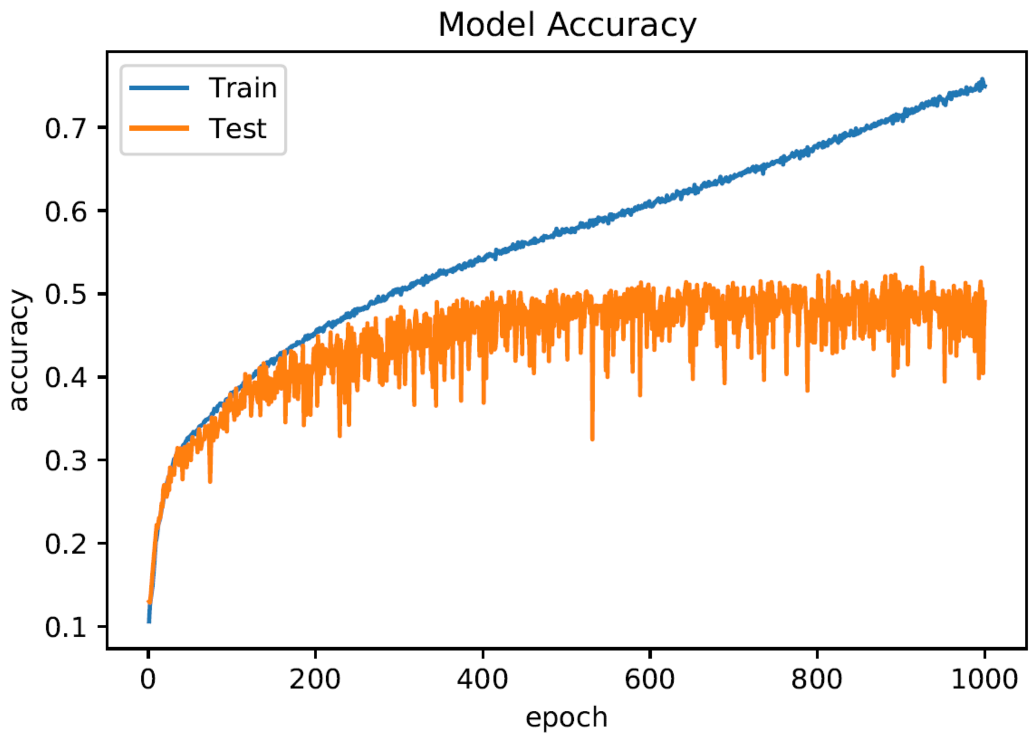
- An Input layer of 32x32x3 dimensions
- A convolution layer $C_1$ with 50 channels, window size 9x9, VALID padding, and ReLU
- activation
- A max pooling layer $S_1$ with a pooling window of size 2x2, stride = 2, and VALID padding
- A convolution layer $C_2$ with 60 channels, window size 5x5, VALID padding, and ReLU
- activation
- A max pooling layer $S_2$ with a pooling window of size 2x2, stride = 2, and VALID padding
- A fully connected layer $F_3$ of size 300 with no activation
- A fully connected layer $F_3$ of size 10

Instead of declaring the last fully connected layer with a "softmax" activation function, we declared **from_logits = True** in the loss function of the model which is essentially equivalent to having a "softmax" activation function but computationally more stable.

a. Plot the (1) training cost, (2) test cost, (3) training accuracy, and (4) test accuracy against learning epochs. One plot for the costs and one plot for the accuracies.



**<u>Training and Test Costs vs. Epochs</u>**



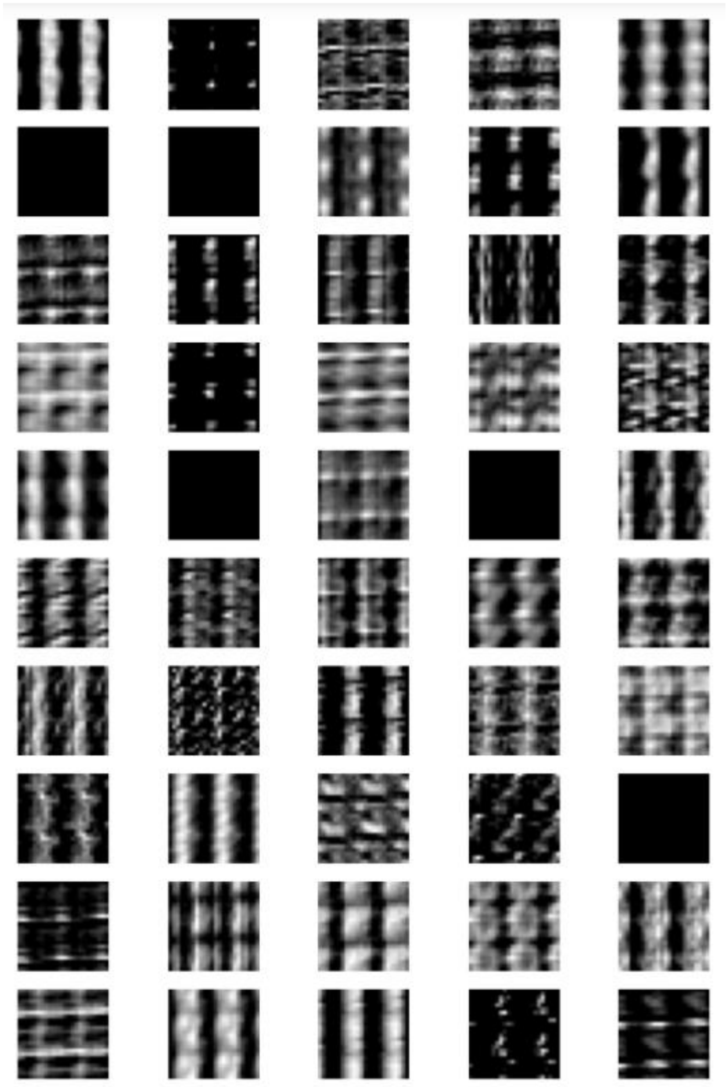
**Training and Test Accuracies vs. Epochs**

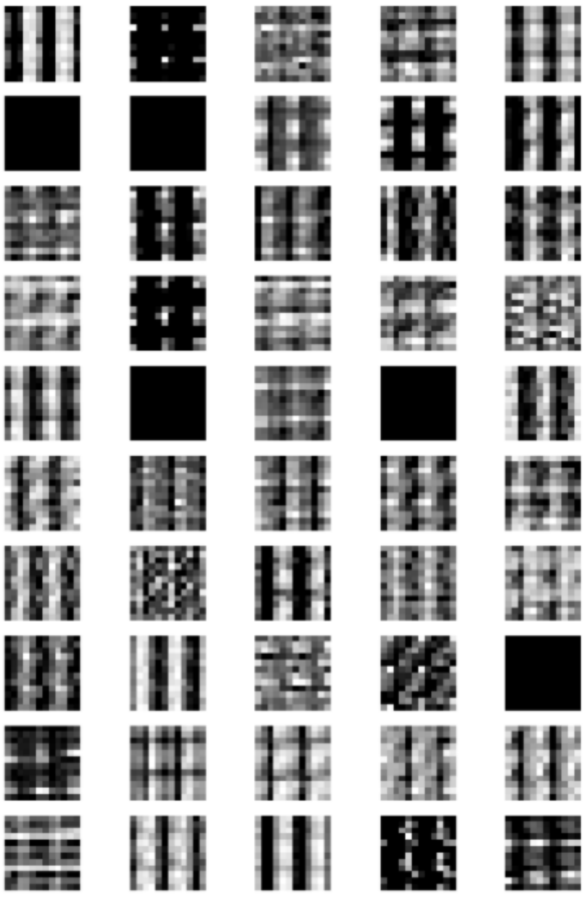
From the two graphs above, we can see that the test loss for the Convolutional Neural Network converges after 400 epochs at round 1.5 m.s.e and that the test accuracy converges roughly after 400 epochs as well around 0.49 to 0.50.
We can observe that the training loss continues to decrease even after the test loss converges as well as the training accuracy continuing to increase even after the test accuracy converges. This shows that for this particular model we have designed with the following parameters, the model may be overfitting on the training data.

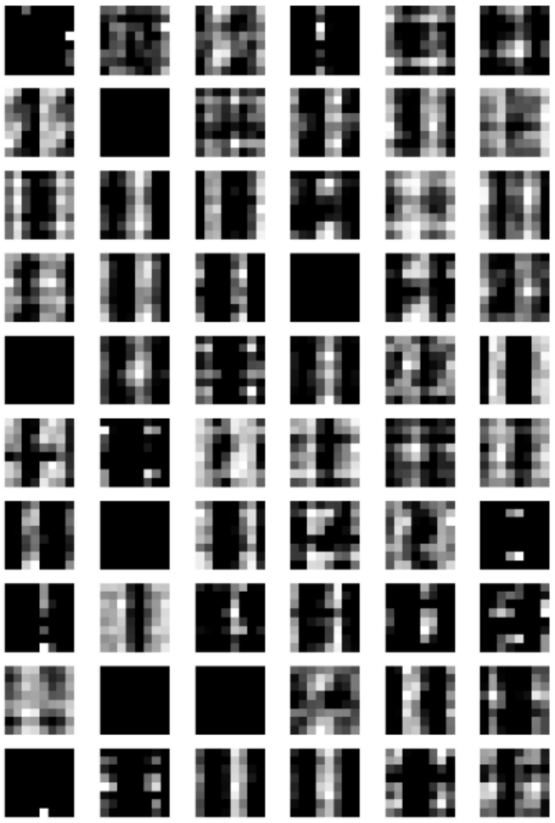
b. For the first two test images, plot the feature maps at both convolution layers ($C_1$ and $C_2$) and pooling layers ($S_1$ and $S_2$) along with the test images. (In total one image and four feature maps).

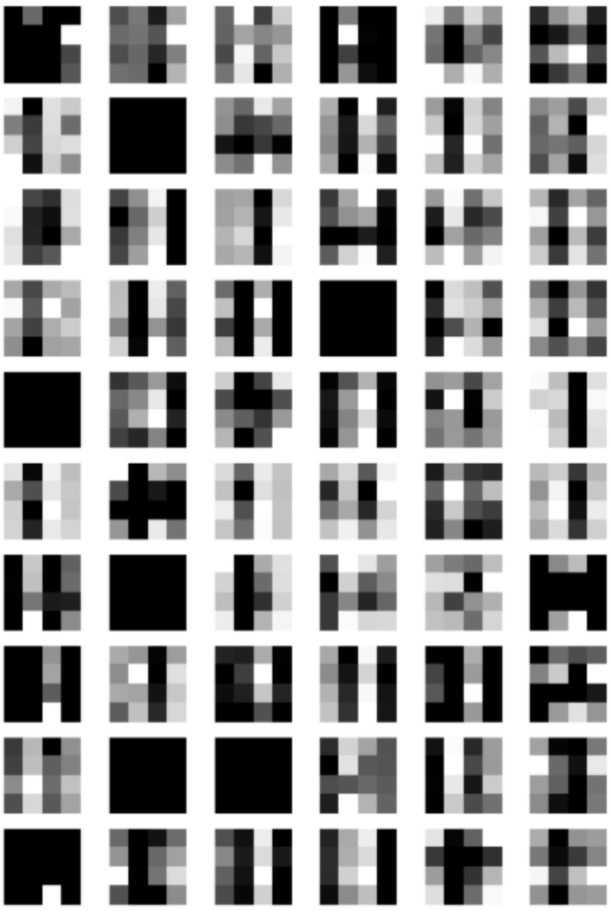
**First Test Image**

**Image**


**$C_1$ Feature Map**


**$S_1$ Feature Map**

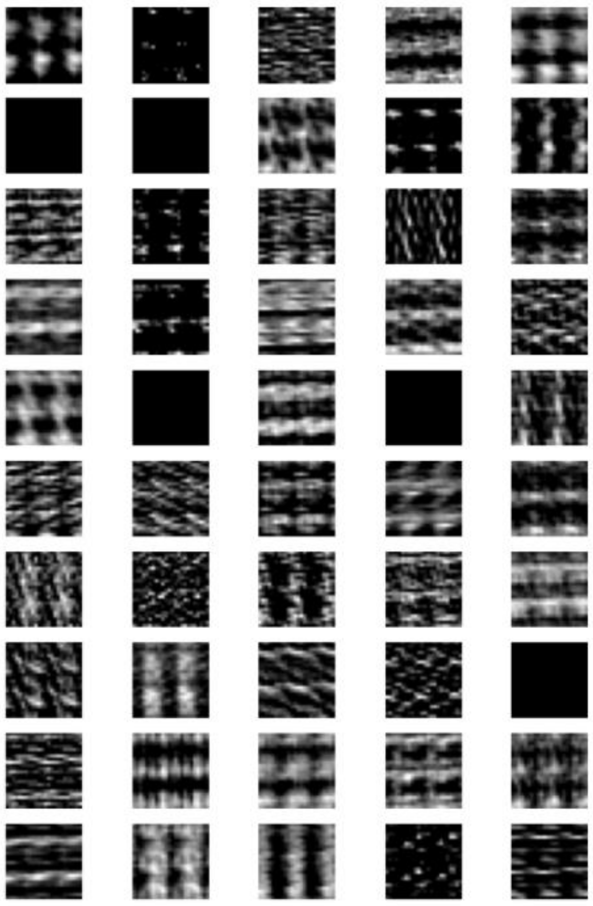

**$C_2$ Feature Map**



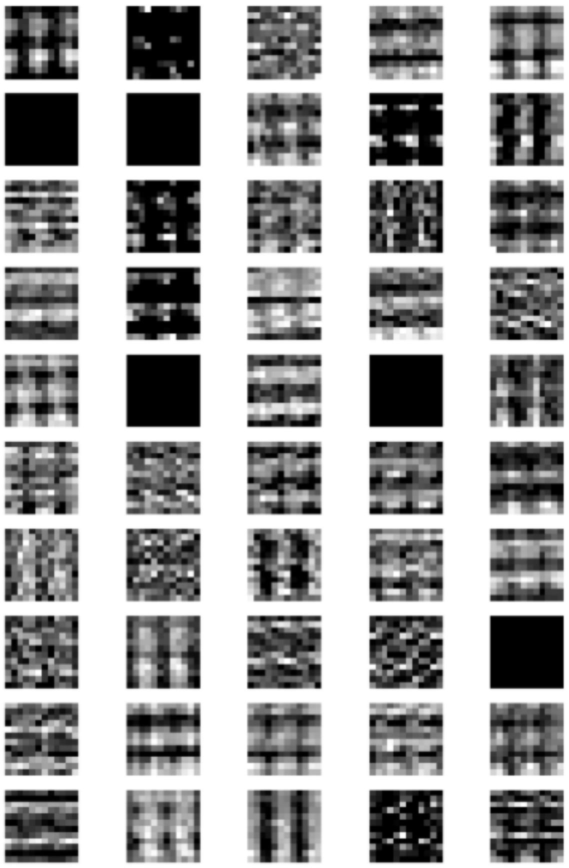
**$S_2$ Feature Map**

**Second Test Image**

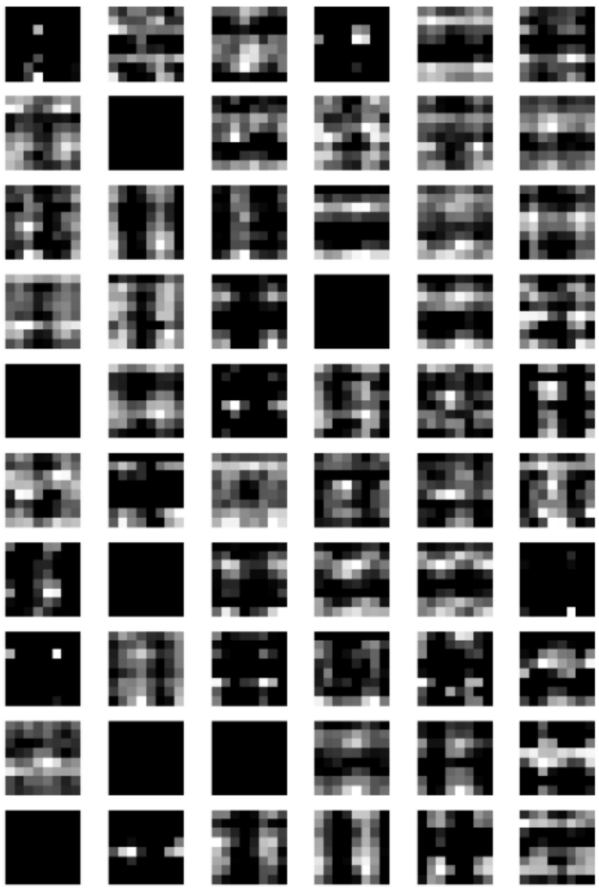
**Image**

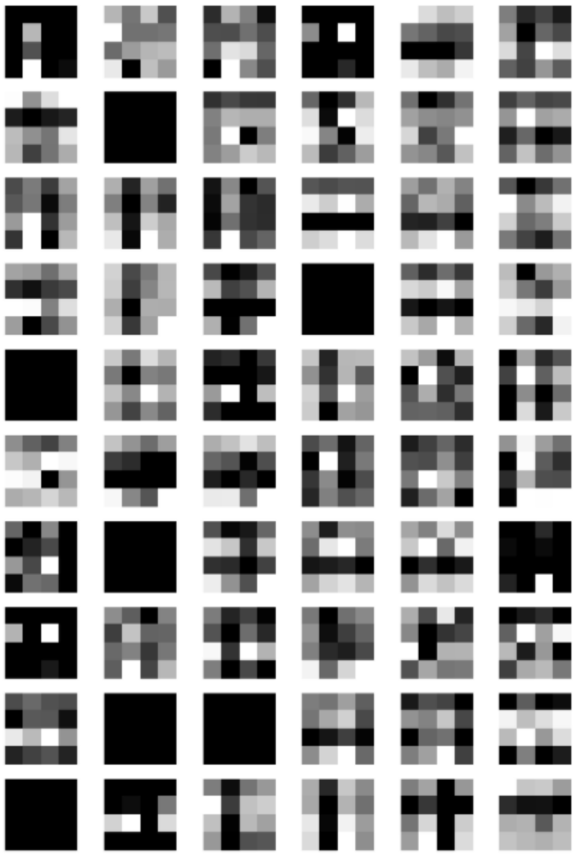


**$C_1$ Feature Map**



**$S_1$ Feature Map**


**$C_2$ Feature Map**

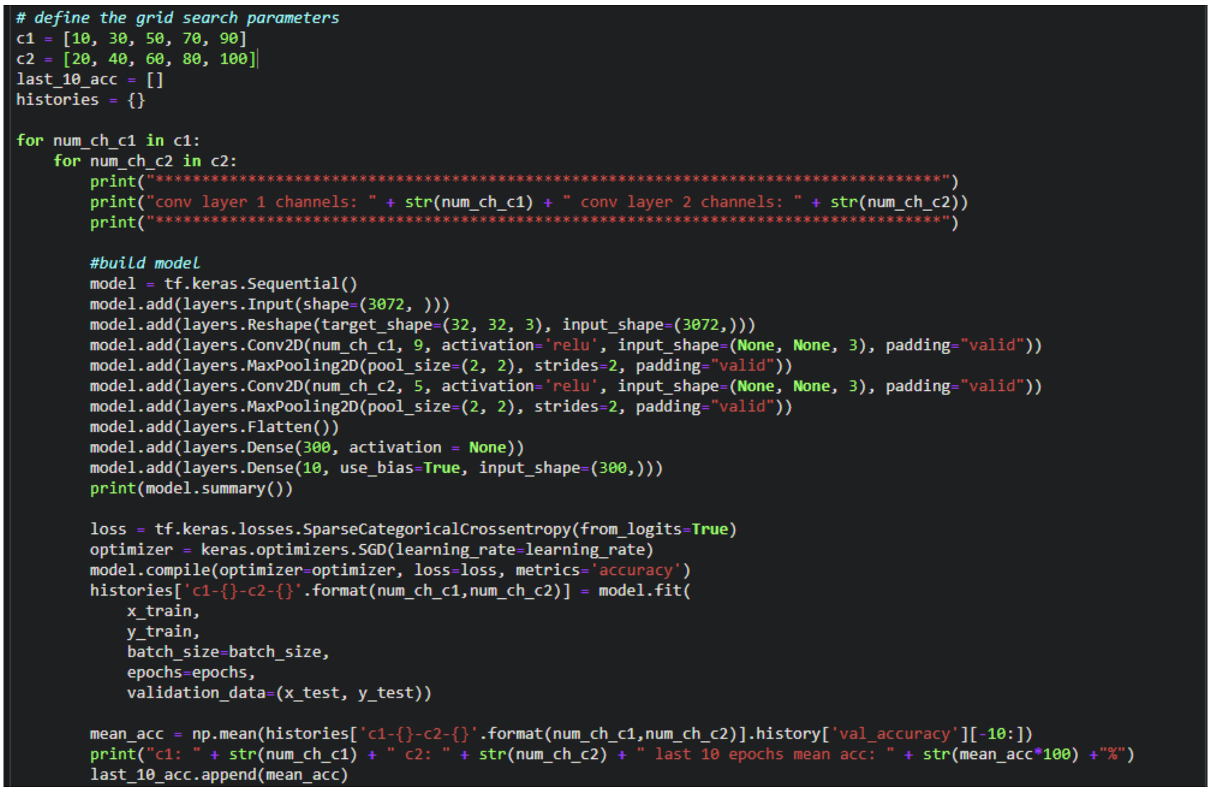

**$S_2$ Feature Map**

From the sets of feature maps for the first 2 images of the test set, we can see that not every channel of the convolution layers ($C_1$ and $C_2$) and pooling layers ($S_1$ and $S_2$) managed to map some significant feature or pattern from the images as we can see that they are fully blacked out, without any observable "feature" mapped to the channel.

## Question 2

**Use a grid search ( $C_1 \epsilon$ {10, 30, 50, 70, 90}, $C_2$ $\epsilon$ {20, 40, 60, 80, 100} , in total 25 combinations) to find the optimal combination of the numbers of channels at the convolution layers. Use the test accuracy to determine the optimal combination. Report all 25 accuracies.**

In order to tackle this question, create a double for-loop to loop through the 2 list of channels to be used for each of the convolutional layer, to conduct the grid search to find the optimal combination of the number of channels at the convolutional layers.



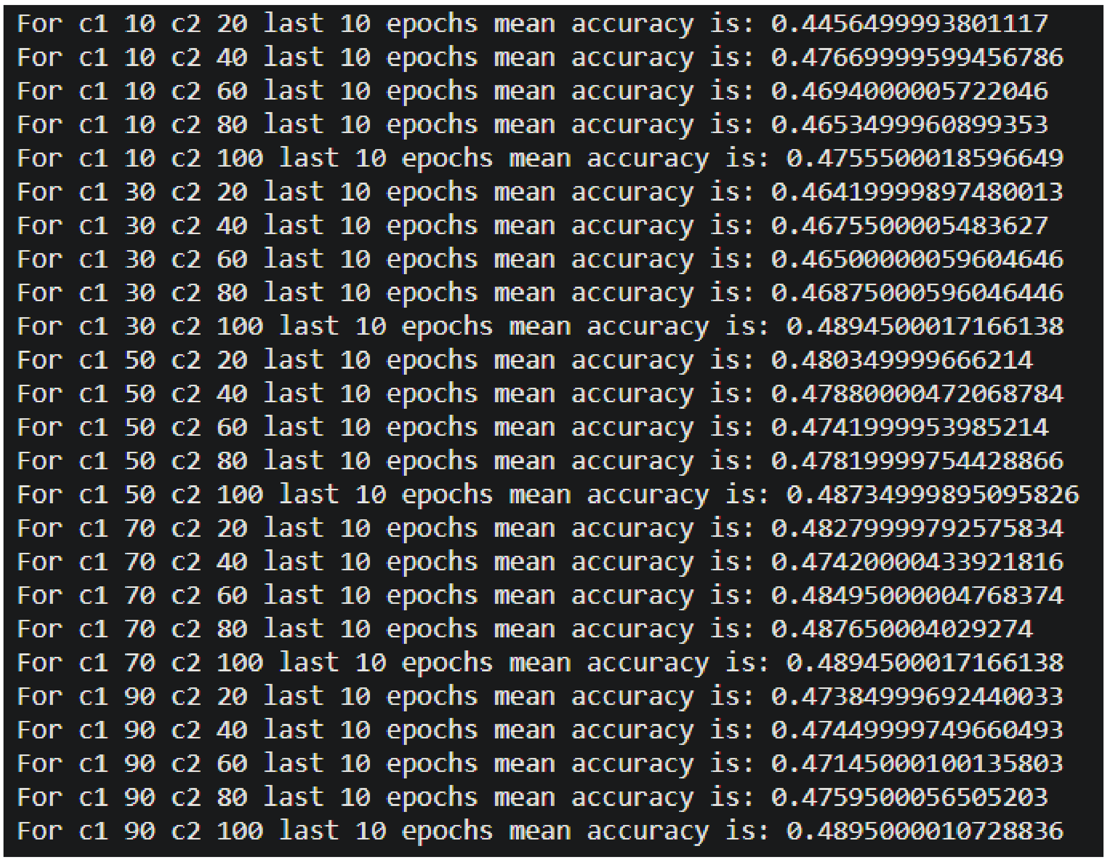
```python
# define the grid search parameters
c1 = [10, 30, 50, 70, 90]
c2 = [20, 40, 60, 80, 100]
last_10_acc = []
histories = {}

for num_ch_c1 in c1:
    for num_ch_c2 in c2:
        print("**************************************************************************************")
        print("conv layer 1 channels: " + str(num_ch_c1) + " conv layer 2 channels: " + str(num_ch_c2))
        print("**************************************************************************************")

        #build model
        model = tf.keras.Sequential()
        model.add(layers.Input(shape=(3072, )))
        model.add(layers.Reshape(target_shape=(32, 32, 3), input_shape=(3072,)))
        model.add(layers.Conv2D(num_ch_c1, 9, activation='relu', input_shape=(None, None, 3), padding="valid"))
        model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding="valid"))
        model.add(layers.Conv2D(num_ch_c2, 5, activation='relu', input_shape=(None, None, 3), padding="valid"))
        model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding="valid"))
        model.add(layers.Flatten())
        model.add(layers.Dense(300, activation = None))
        model.add(layers.Dense(10, use_bias=True, input_shape=(300,)))
        print(model.summary())

        loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
        optimizer = keras.optimizers.SGD(learning_rate=learning_rate)
        model.compile(optimizer=optimizer, loss=loss, metrics='accuracy')
        histories['c1-{}-c2-{}'.format(num_ch_c1,num_ch_c2)] = model.fit(
            x_train,
            y_train,
            batch_size=batch_size,
            epochs=epochs,
            validation_data=(x_test, y_test))

        mean_acc = np.mean(histories['c1-{}-c2-{}'.format(num_ch_c1,num_ch_c2)].history['val_accuracy'][-10:])
        print("c1: " + str(num_ch_c1) + " c2: " + str(num_ch_c2) + " last 10 epochs mean acc: " + str(mean_acc*100) +"%")
        last_10_acc.append(mean_acc)
```

As can be seen from the code above, we first define 2 list of parameters, 1 for each convolutional layer, that we wan to run the grid search on. The parameters are the number of channels to be used for the convolutional layers.

We then declare a double for-loop in the code to let the code run through all possible combinations of channels for the grid search. In the for-loop, every iteration of the for-loop represents one combination of channels, we the create the model for that combination of channels. We then train the model using the training dataset and validate the data on the test dataset.

For each of the model created for each of the combination, to determine the optimal combination of the numbers of channels at the convolution layers, we take the mean test accuracy of the last 10 epochs of training for each of the combination and store them in a list called **last_10_acc**.

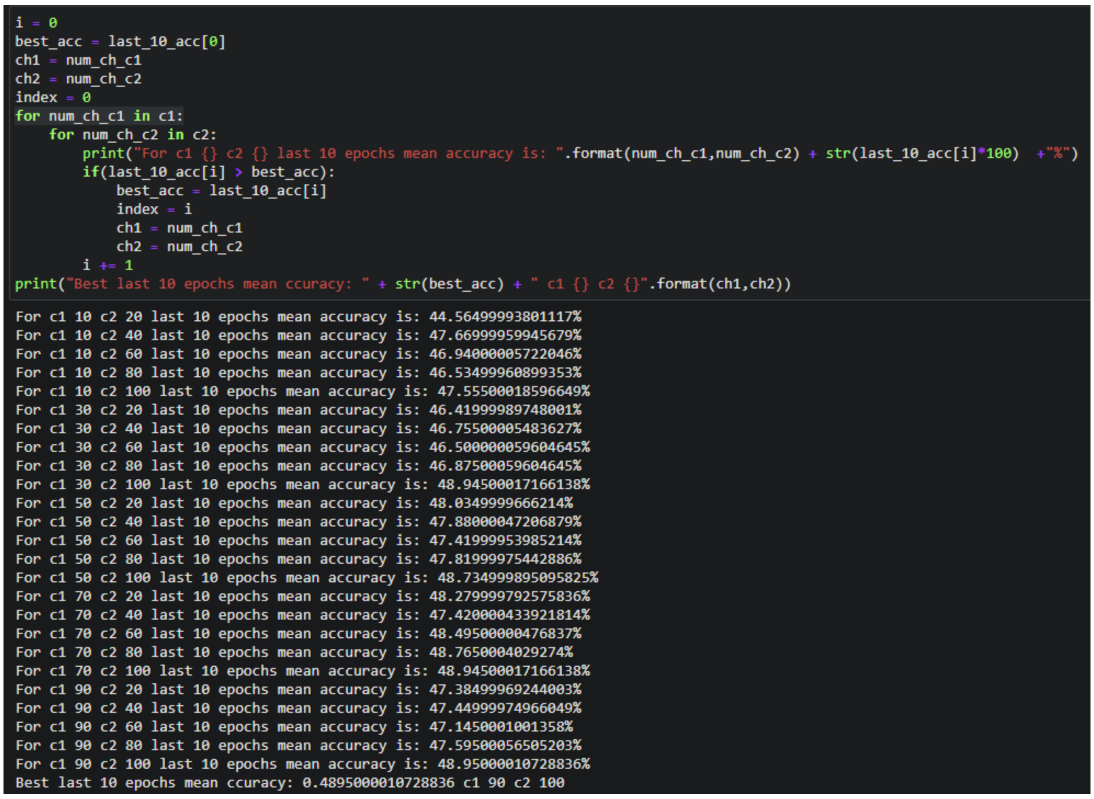
We then run the grid search and the accuracies for the different combinations are listed below:



```
For c1 10 c2 20 last 10 epochs mean accuracy is: 0.4456499993801117
For c1 10 c2 40 last 10 epochs mean accuracy is: 0.47669999599456786
For c1 10 c2 60 last 10 epochs mean accuracy is: 0.4694000005722046
For c1 10 c2 80 last 10 epochs mean accuracy is: 0.4653499960899353
For c1 10 c2 100 last 10 epochs mean accuracy is: 0.4755500018596649
For c1 30 c2 20 last 10 epochs mean accuracy is: 0.46419999897480013
For c1 30 c2 40 last 10 epochs mean accuracy is: 0.4675500005483627
For c1 30 c2 60 last 10 epochs mean accuracy is: 0.46500000059604646
For c1 30 c2 80 last 10 epochs mean accuracy is: 0.46875000596046446
For c1 30 c2 100 last 10 epochs mean accuracy is: 0.4894500017166138
For c1 50 c2 20 last 10 epochs mean accuracy is: 0.480349999666214
For c1 50 c2 40 last 10 epochs mean accuracy is: 0.47880000472068784
For c1 50 c2 60 last 10 epochs mean accuracy is: 0.4741999953985214
For c1 50 c2 80 last 10 epochs mean accuracy is: 0.47819999754428866
For c1 50 c2 100 last 10 epochs mean accuracy is: 0.48734999895095826
For c1 70 c2 20 last 10 epochs mean accuracy is: 0.48279999792575834
For c1 70 c2 40 last 10 epochs mean accuracy is: 0.47420000433921816
For c1 70 c2 60 last 10 epochs mean accuracy is: 0.48495000004768374
For c1 70 c2 80 last 10 epochs mean accuracy is: 0.487650004029274
For c1 70 c2 100 last 10 epochs mean accuracy is: 0.4894500017166138
For c1 90 c2 20 last 10 epochs mean accuracy is: 0.47384999692440033
For c1 90 c2 40 last 10 epochs mean accuracy is: 0.47449999749660493
For c1 90 c2 60 last 10 epochs mean accuracy is: 0.47145000100135803
For c1 90 c2 80 last 10 epochs mean accuracy is: 0.4759500056505203
For c1 90 c2 100 last 10 epochs mean accuracy is: 0.4895000010728836
```

**<u>Grid Search accuracies for the 25 combinations</u>**

We then run through the list of the accuracies of the various models and determine which is the combination of channels that provides us with the best mean test accuracy of the last 10 epochs of training.



```
i = 0
best_acc = last_10_acc[0]
ch1 = num_ch_c1
ch2 = num_ch_c2
index = 0
for num_ch_c1 in c1:
    for num_ch_c2 in c2:
        print("For c1 {} c2 {} last 10 epochs mean accuracy is: ".format(num_ch_c1,num_ch_c2) + str(last_10_acc[i]*100)  +"%")
        if(last_10_acc[i] > best_acc):
            best_acc = last_10_acc[i]
            index = i
            ch1 = num_ch_c1
            ch2 = num_ch_c2
        i += 1
print("Best last 10 epochs mean ccuracy: " + str(best_acc) + " c1 {} c2 {}".format(ch1,ch2))
```

```
For c1 10 c2 20 last 10 epochs mean accuracy is: 44.56499993801117%
For c1 10 c2 40 last 10 epochs mean accuracy is: 47.66999959945679%
For c1 10 c2 60 last 10 epochs mean accuracy is: 46.94000005722046%
For c1 10 c2 80 last 10 epochs mean accuracy is: 46.53499960899353%
For c1 10 c2 100 last 10 epochs mean accuracy is: 47.55500018596649%
For c1 30 c2 20 last 10 epochs mean accuracy is: 46.41999989748001%
For c1 30 c2 40 last 10 epochs mean accuracy is: 46.75500005483627%
For c1 30 c2 60 last 10 epochs mean accuracy is: 46.500000059604645%
For c1 30 c2 80 last 10 epochs mean accuracy is: 46.87500059604645%
For c1 30 c2 100 last 10 epochs mean accuracy is: 48.94500017166138%
For c1 50 c2 20 last 10 epochs mean accuracy is: 48.0349999666214%
For c1 50 c2 40 last 10 epochs mean accuracy is: 47.88000047206879%
For c1 50 c2 60 last 10 epochs mean accuracy is: 47.41999953985214%
For c1 50 c2 80 last 10 epochs mean accuracy is: 47.81999975442886%
For c1 50 c2 100 last 10 epochs mean accuracy is: 48.734999895095825%
For c1 70 c2 20 last 10 epochs mean accuracy is: 48.279999792575836%
For c1 70 c2 40 last 10 epochs mean accuracy is: 47.420000433921814%
For c1 70 c2 60 last 10 epochs mean accuracy is: 48.49500000476837%
For c1 70 c2 80 last 10 epochs mean accuracy is: 48.7650004029274%
For c1 70 c2 100 last 10 epochs mean accuracy is: 48.94500017166138%
For c1 90 c2 20 last 10 epochs mean accuracy is: 47.38499969244003%
For c1 90 c2 40 last 10 epochs mean accuracy is: 47.44999974966049%
For c1 90 c2 60 last 10 epochs mean accuracy is: 47.1450001001358%
For c1 90 c2 80 last 10 epochs mean accuracy is: 47.59500056505203%
For c1 90 c2 100 last 10 epochs mean accuracy is: 48.95000010728836%
Best last 10 epochs mean ccuracy: 0.4895000010728836 c1 90 c2 100
```

From the results above, we can see that the combination of the numbers of channels at the convolution layers of $C_1$ having 90 channels, $C_2$ having 100 channels gives us the best test accuracy amongst all the other combinations of channels.

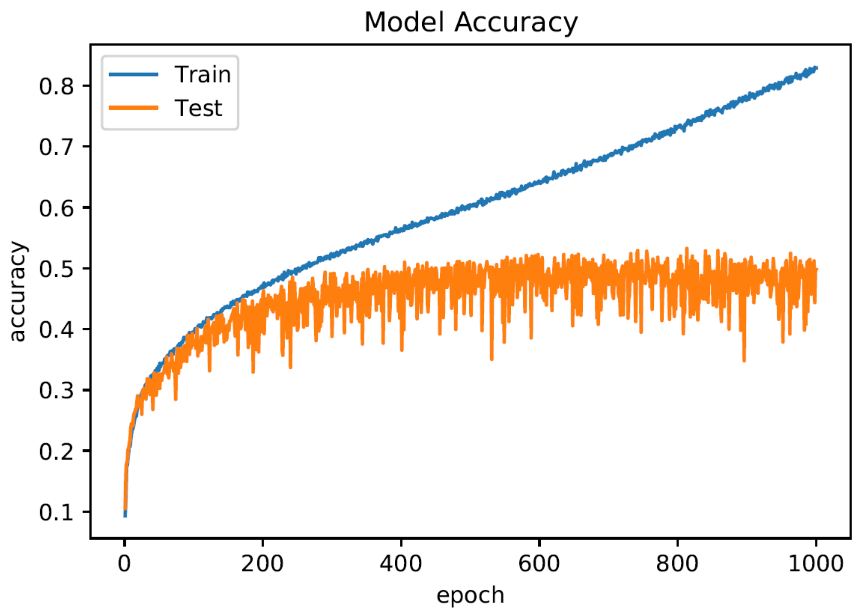
## Question 3

Using the optimal combination found in part (2), train the network by:

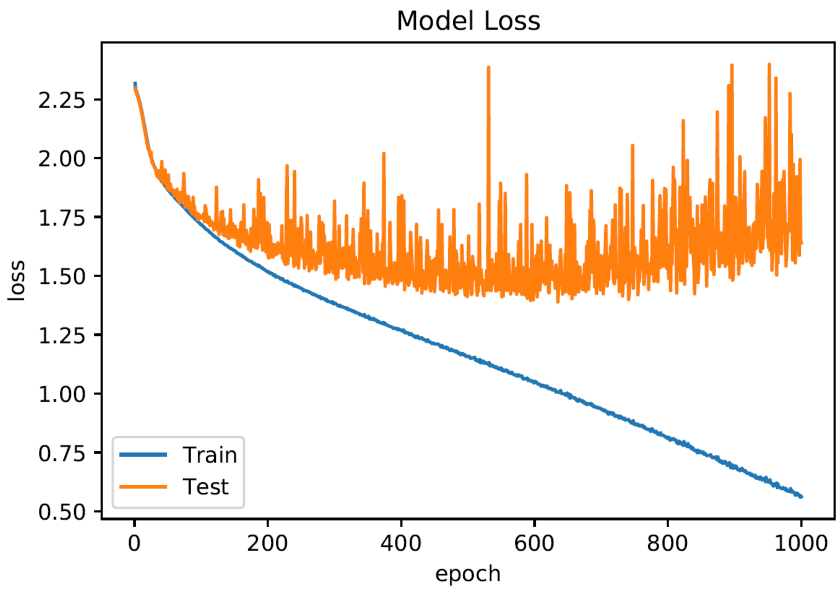
a. adding the momentum term with momentum $\gamma = 0.1$,
b. using RMSProp algorithm for learning,
c. using Adam optimizer for learning,
d. adding dropout (probability=0.5) to the two fully connected layers.

Plot the costs and accuracies against epochs (as in question 1(a)) for each case. Note that the sub-questions are independent. For instance, in (d), you do not need to modify the optimizer.

a. momentum $\gamma = 0.1$



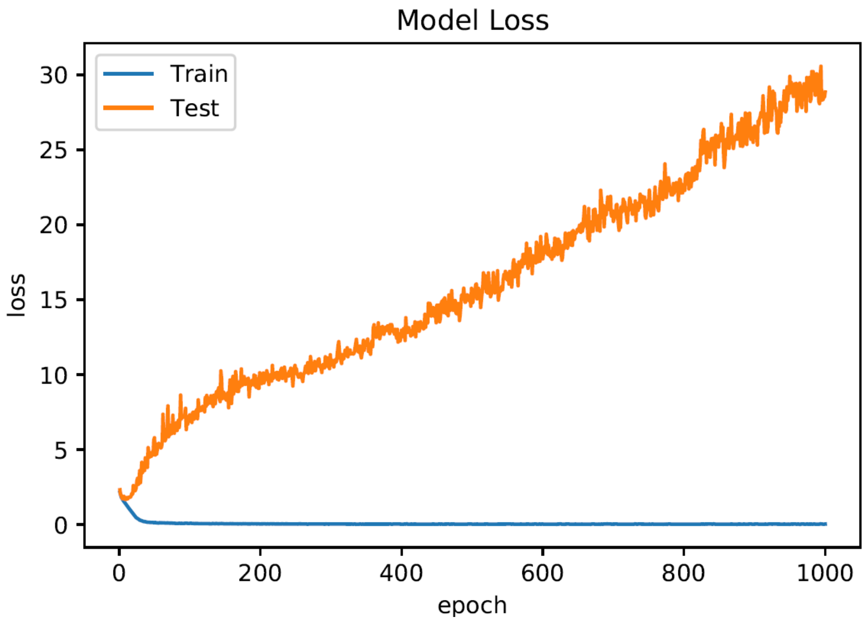
**Training and Test Accuracies vs. Epochs**



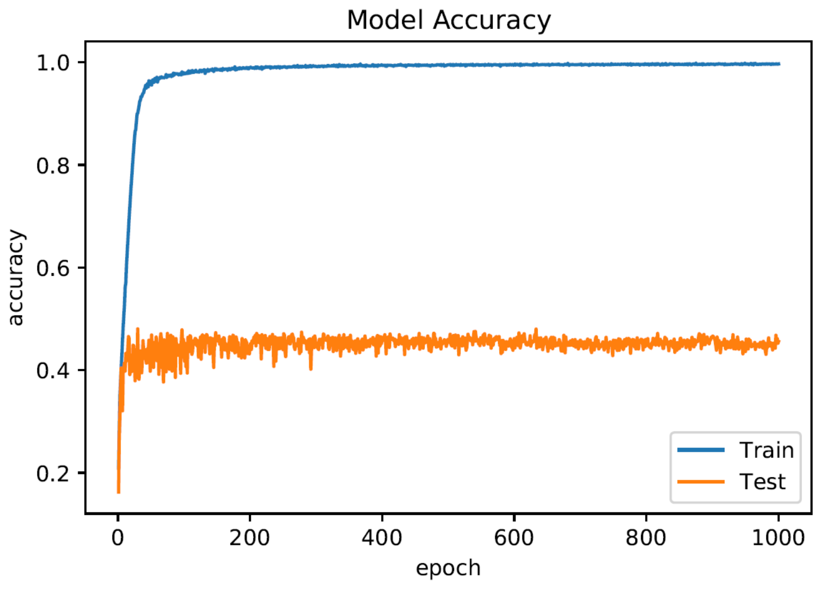
**Training and Test Loss vs. Epochs**

As can be seen from the graphs above, the test loss and test accuracies for the network with the same SGD loss function but with an added momentum factor behaves almost the same as the network without the momentum factor.
From the two graphs above, we can see that the test loss for the Convolutional Neural Network converges after 400 epochs at round 1.4 to 1.5 m.s.e and that the test accuracy converges roughly after 400 epochs around 0.49 to 0.52 test accuracy.
We can observe that the training loss continues to decrease even after the test loss converges as well as the training accuracy continuing to increase even after the test accuracy converges. This shows that even with the momentum factor in place, the network is still overfitting on the training data.

b. using RMSProp algorithm for learning
For this sub-question we use the same network, the only change is that instead of having a SGD loss function, we now use a RMSProp loss function for the network.



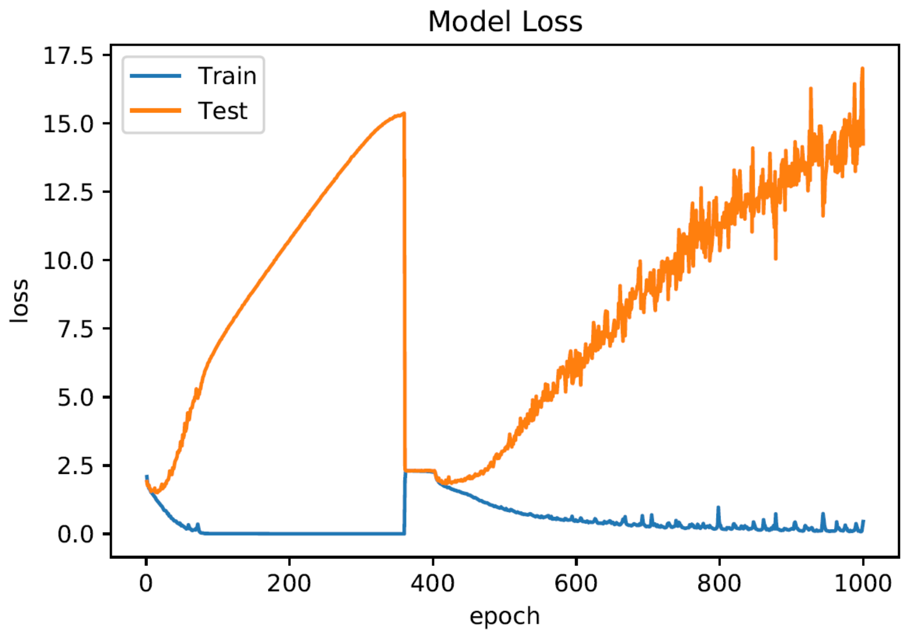
**Training and Test Loss vs. Epochs**



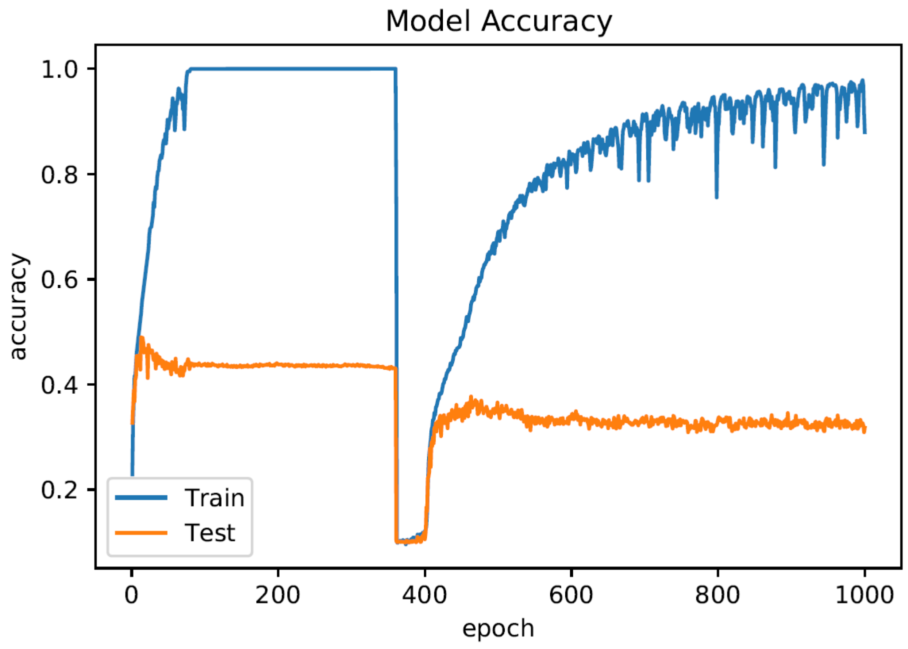
**Training and Test Accuracies vs. Epochs**

From the graphs above, we can see that with the other given parameters given for the network, RMSProp loss is not suitable for the network. With RMSProp loss, we can see that the test loss for the model is constantly increasing linearly and that the test accuracy remains stagnant and converges after the first few epochs at around 0.45 test accuracy.
The training loss, however, is very low as compared to the test loss for all the epochs and the training accuracies were at 0.99 accuracy for the model after only the first few epochs. This goes to show that the RMSProp loss made the network quickly learn the training data and the model is overfitted on the training data, and hence, the network is not able to capture a general pattern and this causes the network to not be able to generalize and give a poor test accuracy.

c. using Adam optimizer for learning



**<u>Training and Test Loss vs. Epochs</u>**

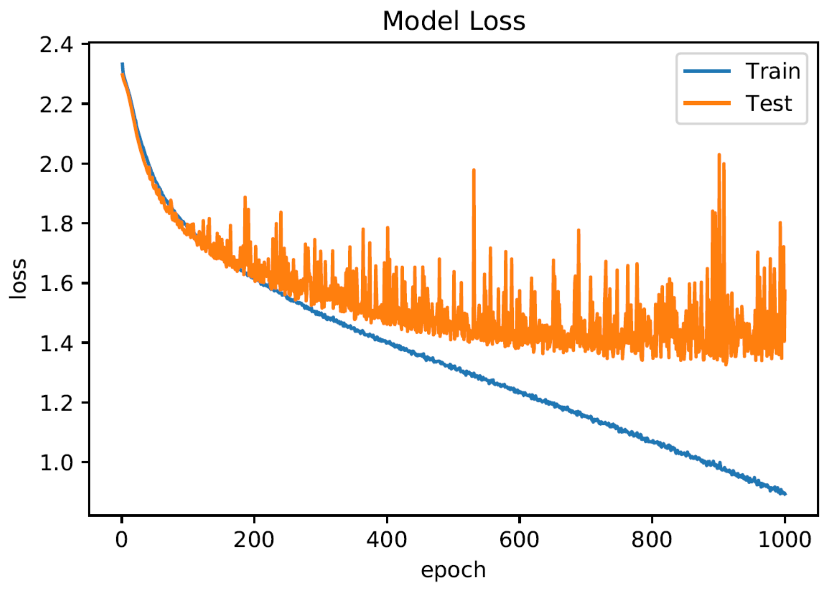


**<u>Training and Test Accuracies vs. Epochs</u>**

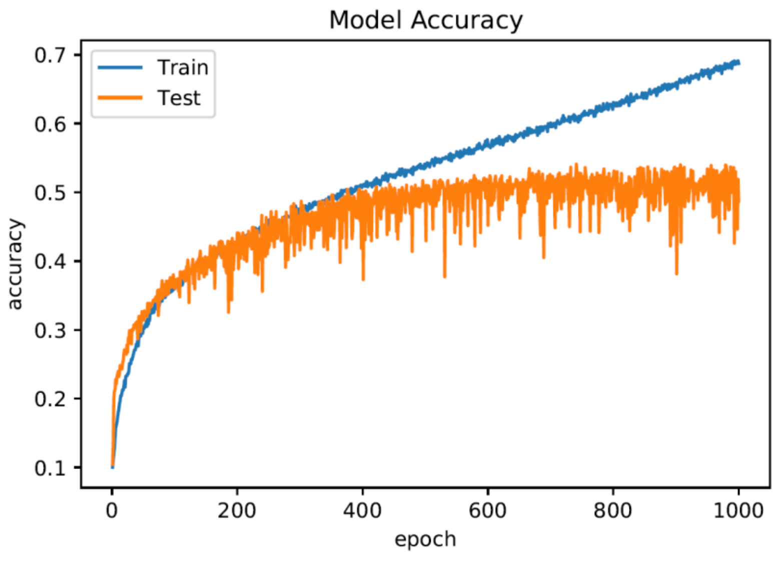
From the graphs above, we can see that with the other given parameters given for the network, the Adam loss function is not suitable for the network. From the plot of the test and training loss against epochs, we can see that the test loss is unstable, where the loss increases rapidly from the first epoch till about close to 400 epochs and then has a sharp decrease, followed by a rapidly increasing loss. The test accuracy plot is also unstable, with the test accuracies staying stagnant around 0.42 to 0.43 test accuracy for most epochs but having a sharp decrease at near 400 epochs and then increases and stagnates at around 0.31 to 0.33 accuracy rate.

We can also observe from the accuracy graph that the training accuracy for this network is mostly at 1.0 accuracy for the first 400 epochs, then after a drop in accuracy near 400 epoch, rises steadily to about 0.99 accuracy, showing that the Adam loss function made the network quickly learn the training data and the model is overfitted on the training data, and hence, the network is not able to capture a general pattern and this causes the network to not be able to generalize and give a poor test accuracy.

d. adding dropout (probability=0.5) to the two fully connected layers.



**<u>Training and Test Loss vs. Epochs</u>**



**<u>Training and Test Accuracies vs. Epochs</u>**

For the network with the added dropout layer with a probability of 0.5, to the two fully connected layers, we have the test accuracy against epochs and test loss against epochs graphs as shown above. As compared to the network with no dropout layer, the test loss for the network converges at a lower m.s.e between 1.3 to 1.4 m.s.e.
The test accuracy were also slightly better for the last 100 epochs, with the test accuracies for each epoch fluctuating around 0.53 to 0.54 test accuracy rate. Therefore, we can say that when we add a dropout layer to the Convolutional Neural Network, the accuracy of the model slightly improves.

## Question 4

**Compare the accuracies of all the models from parts (1) - (3) and discuss their performances.**

For this question, using the last epoch's test accuracy may not be a fair comparison as the test accuracies of the models fluctuate, to give us a better gauge of the accuracies of the models, we take the mean test accuracy of the each model for the last 10 epochs of training the model. The results are recorded below:

| Model | Mean Test Accuracy (last 10 epochs) |
|---|---|
| Base Model (Question 1) | 47.60999947786331% |
| Model with Optimal Combination | 49.63499993085861% |
| Model with Optimal Combination + momentum of 0.1 | 50.175000727176666% |
| Model with Optimal Combination + RMSProp algorithm | 46.4900004863739% |
| Model with Optimal Combination + Adam optimizer | 32.264999747276306% |
| Model with Optimal Combination + dropout (probability=0.5) | 52.11500018835068 % |

From the above mean test accuracies, we can observe that the model with the optimal combination of the numbers of channels at the convolution layers and a dropout layer of dropout probability of 0.5 has the best mean test accuracy at 52.12%.

We can also see that the model with the optimal combination of the numbers of channels at the convolution layers and an Adam optimizer has the worst mean test accuracy at 32.26% with the model with the optimal combination of the numbers of channels at the convolution layers and a RMSProp optimizer coming in with the second worst mean test accuracy at 46.49%.

The model with the optimal combination of the numbers of channels at the convolution layers, model with the optimal combination of the numbers of channels at the convolution layers and a momentum of 0.1 have fairly similar mean test accuracies.

### Performance of Models

From the various plots of  the costs and accuracies against epochs for the models that were being experimented on, there are a few observations that were made.

Firstly, for the baseline model, the model with the optimal combination of the numbers of channels at the convolution layers, the model with the optimal combination of the numbers of channels at the convolution layers and a momentum of 0.1, and the model with the optimal combination of the numbers of channels at the convolution layers and a dropout layer of dropout probability of 0.5, their performances are fairly similar, with the test accuracies and loss of the models converging at around 400 epochs for all 4 models.

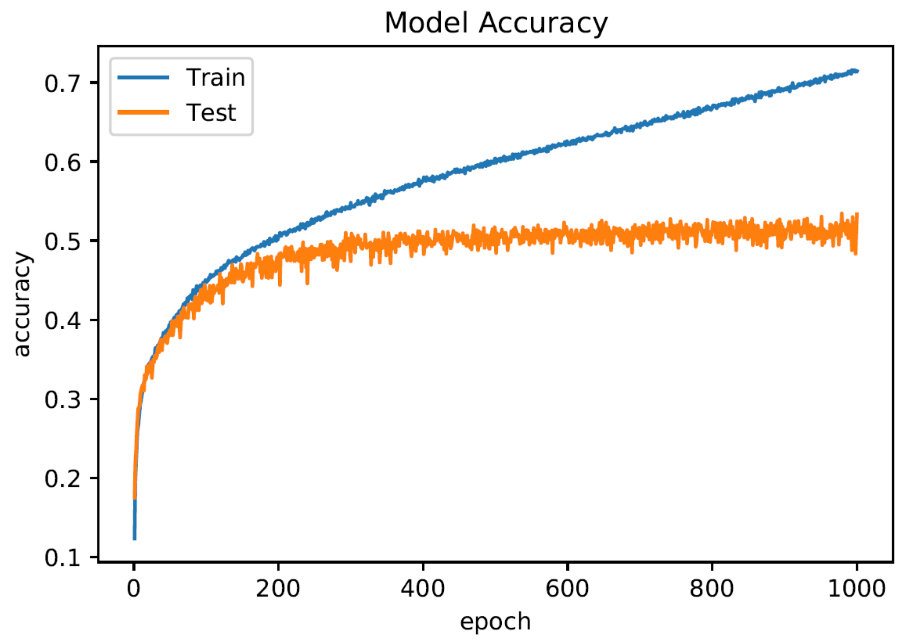
We can also observe that for these 4 model, after the 200 epoch, the models start to overfit on the training data where we can observe that the test accuracy and loss remains stagnant and converges while the training accuracy keeps increasing and test loss keeps decreasing for all the epochs that were run.

Next, we can observe that for the model with the RMSProp algorithm as the optimizer, the performance of the model was very bad as the test accuracy of the model converges just after the first few epochs and the test loss increases linearly with each epoch. We then observe an abnormally high training accuracy after the first few epochs at 0.99 accuracy, showing that the model has overfitted on the training data, using the RMSProp algorithm as the optimizer. From this we can conclude that for the other parameters that were fixed for this network, using a RMSProp algorithm as the optimizer is not the right choice.

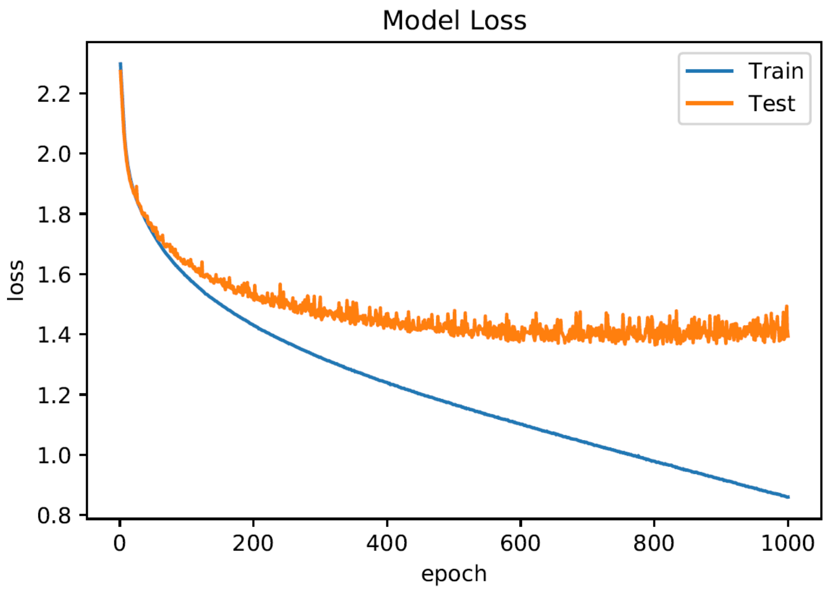
Finally, for the model with the Adam optimizer, the mean test accuracy for the last 10 epochs was the worst of all the networks that were experimented on. From the plots of  the costs and accuracies against epochs for the model, we can also observe that the performance of the model is very unstable, with the test and training accuracies having sudden drops in the middle of the training epochs and the test and training loss having sudden increases in the middle of the training epochs. The training accuracies was very high as compared to the test accuracies for most of the epochs, with the training accuracies being at 1.00 for most of the first 400 epochs and accuracy higher than that of test accuracy for the last 400 epochs, showing that the model has overfitted on the training data, using the Adam optimizer. From this we can conclude that for the other parameters that were fixed for this network, using an Adam optimizer is not the right choice.

## Improve Performance of Models using Adam and RMSProp Optimizers

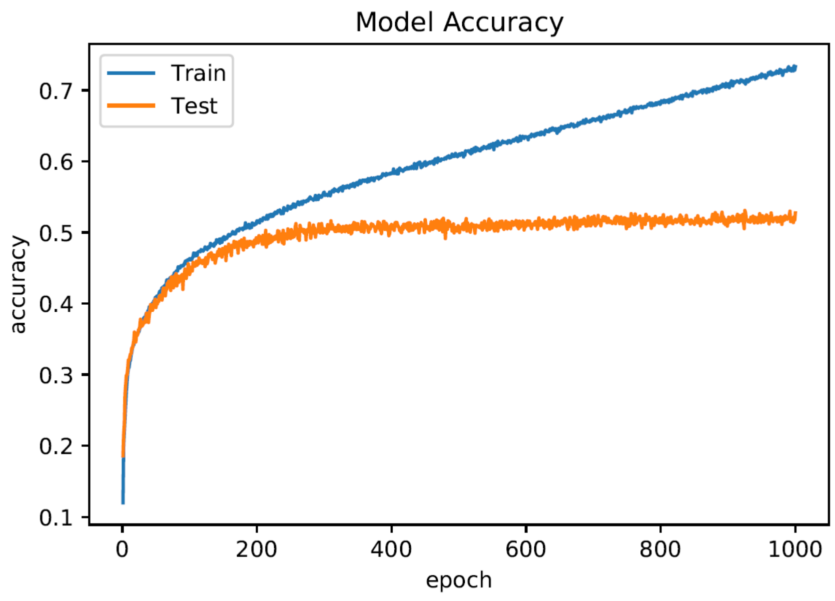
After conducting the experiments above, 2 more experiments were conducted to see if we can improve the test accuracies of the models that used the Adam and RMSProp optimizers, or at least make the test accuracies and test loss more stable. As I think that the learning rate for the models were 2 high for the 2 optimizers, seeing that they converge so fast in the previous 2 experiments, I decided to reduce the learning rate to 1e-5. These are the plots that were obtained after training the models with the reduced learning rate.



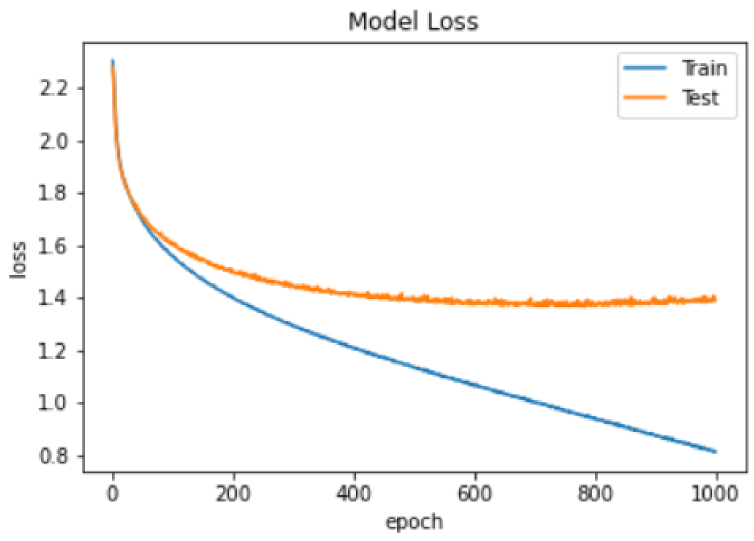
**<u>RMSProp Accuracy vs. Epochs</u>**



**<u>RMSProp Loss vs. Epochs</u>**



**Adam Accuracy vs. Epochs**



**Adam Loss vs. Epochs**

From the above plots, we can observe that for the model with the RMSProp optimizer, when the learning rate is decrease to 1e-5, we are able to achieve a higher test accuracy, converging at around 0.5 accuracy and that the test loss is not linear increasing with epochs as it was previously with a higher learning rate.

We can also observe that for the model with the Adam Optimizer, when the learning rate is decrease to 1e-5, we are able to achieve a higher test accuracy, converging at around 0.5 accuracy. Also, we can see that the accuracies and loss for the model are more stable with no sudden drops or increases in the middle of the training epochs.

| Model | Mean Test Accuracy (last 10 epochs) |
|---|---|
| Model with Optimal Combination + RMSProp algorithm | 50.99000096321106% |
| Model with Optimal Combination + Adam optimizer | 52.90499925613403% |

We can also observe that with a lower learning rate, the Mean Test Accuracy for the last 10 epochs of the models with the RMSProp and Adam optimizers also improved significantly.

## Conclusions

From the respective experiments we have conducted, we can observe that the test accuracies for the models at their best converges at around 53% test accuracy for the classification of the CIFAR-10 dataset.

We can also tell from the accuracy and loss plots that the models are overfitting on the training dataset as the training accuracy continues to increase with each epoch even after the test accuracy converges around 400 epochs.

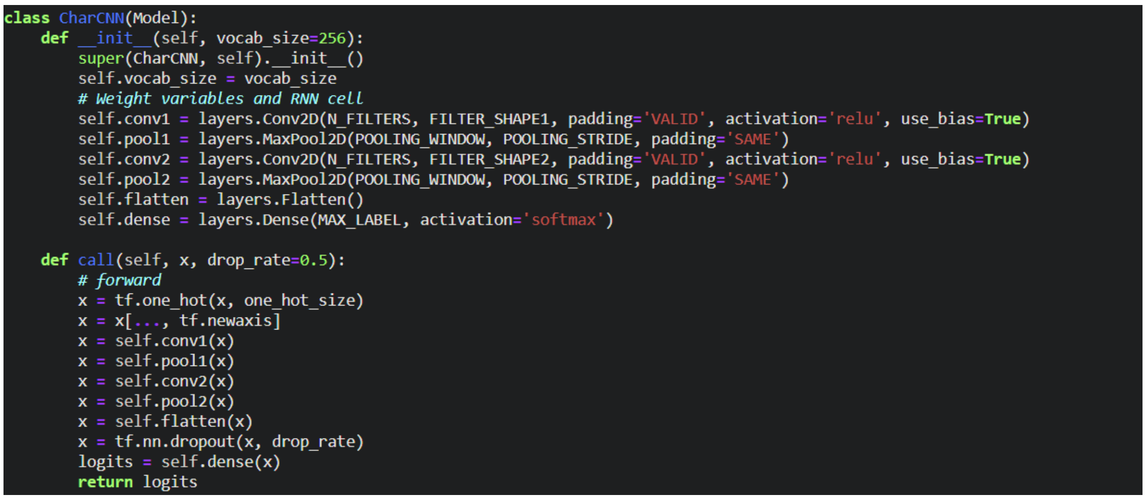
Therefore, we can conclude that for the follow parameters given for the experiment, the model is not able to efficient learn the CIFAR-10 dataset and the parameters for the models can be tuned to improve the test accuracies, such as reducing the window size of the Convolutional layers etc.

# Part B: Text classification

## Question 1

1. Design a Character CNN Classifier that receives character ids and classifies the input. The CNN has two convolution and pooling layers:

- A convolution layer $C_1$ of 10 filters of window size 20x256, VALID padding, and ReLU neurons. A max pooling layer $S_1$ with a pooling window of size 4x4, with stride = 2, and padding = 'SAME'.
- A convolution layer $C_2$ of 10 filters of window size 20x1, VALID padding, and ReLU neurons. A max pooling layer $S_2$ with a pooling window of size 4x4, with stride = 2 and padding = 'SAME'.



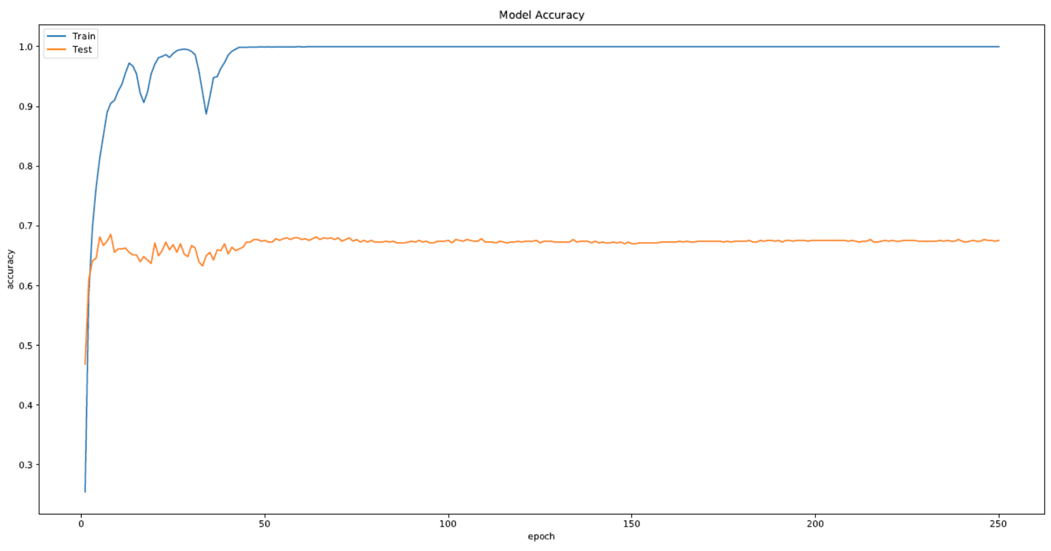
```python
class CharCNN(Model):
    def __init__(self, vocab_size=256):
        super(CharCNN, self).__init__()
        self.vocab_size = vocab_size
        # Weight variables and RNN cell
        self.conv1 = layers.Conv2D(N_FILTERS, FILTER_SHAPE1, padding='VALID', activation='relu', use_bias=True)
        self.pool1 = layers.MaxPool2D(POOLING_WINDOW, POOLING_STRIDE, padding='SAME')
        self.conv2 = layers.Conv2D(N_FILTERS, FILTER_SHAPE2, padding='VALID', activation='relu', use_bias=True)
        self.pool2 = layers.MaxPool2D(POOLING_WINDOW, POOLING_STRIDE, padding='SAME')
        self.flatten = layers.Flatten()
        self.dense = layers.Dense(MAX_LABEL, activation='softmax')

    def call(self, x, drop_rate=0.5):
        # forward
        x = tf.one_hot(x, one_hot_size)
        x = x[..., tf.newaxis]
        x = self.conv1(x)
        x = self.pool1(x)
        x = self.conv2(x)
        x = self.pool2(x)
        x = self.flatten(x)
        x = tf.nn.dropout(x, drop_rate)
        logits = self.dense(x)
        return logits
```

The figure above shows the Character CNN Classifier that we will be using, with a convolution layer $C_1$ of 10 filters of window size 20x256, VALID padding, and ReLU neurons, a max pooling layer $S_1$ with a pooling window of size 4x4, with stride = 2, and padding = 'SAME', a convolution layer $C_2$ of 10 filters of window size 20x1, VALID padding, and ReLU neurons, a max pooling layer $S_2$ with a pooling window of size 4x4, with stride = 2 and padding = 'SAME'.

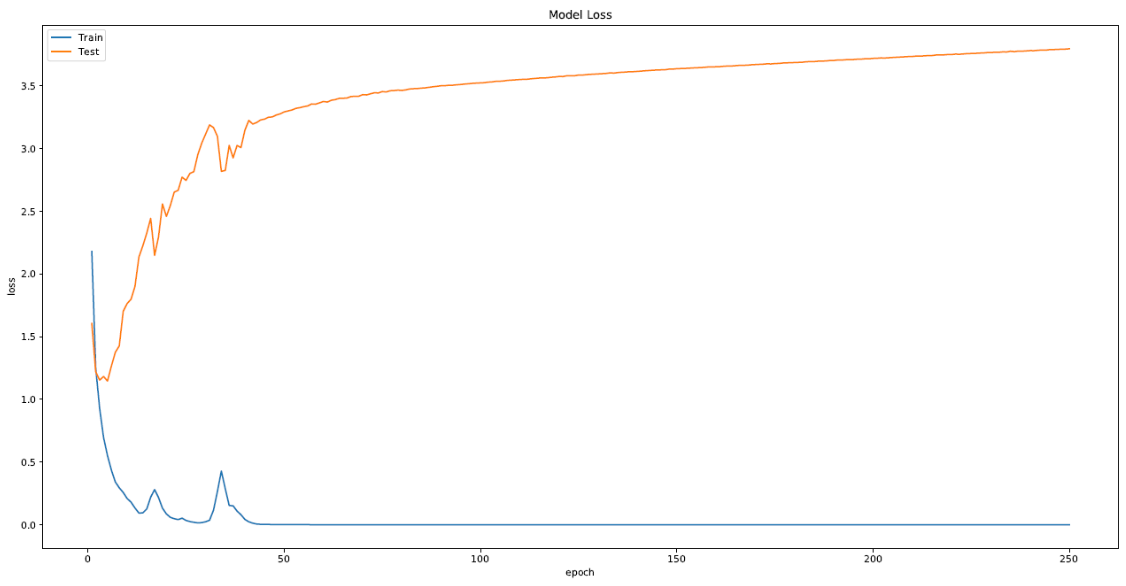
For the last layer, we will be using a dense layer with ‘softmax’ activation and 15 neurons for the number of classes for the classification. For Question 1, the model will have a dropout rate of 0.0. The model will be trained for 250 epochs.

We will use both SGD and Adam optimizers see which is a better model for this classification model. Both will have a learning rate of 0.01.

**Plot the entropy cost on the training data and the accuracy on the testing data against training epochs.**



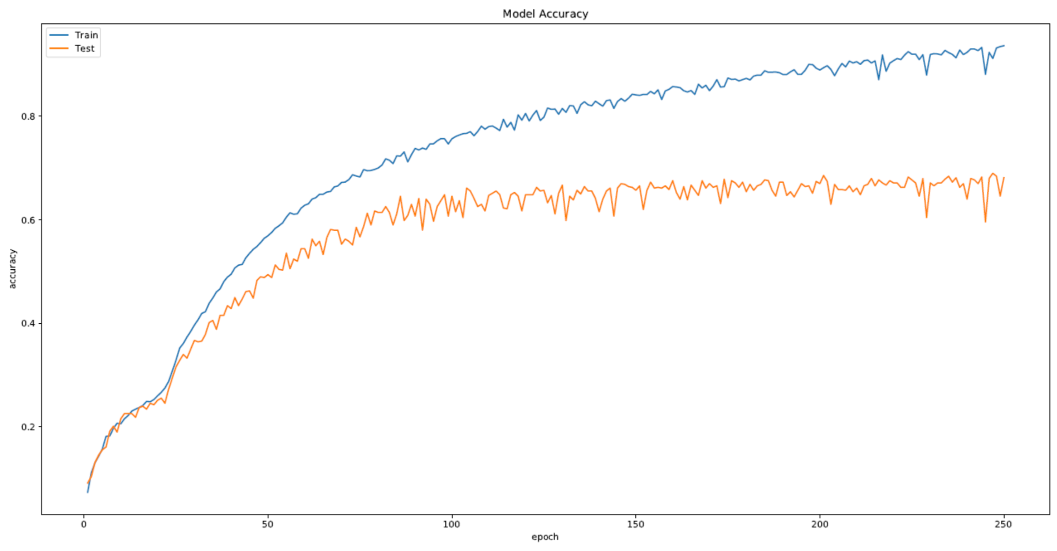
<u>**Training and Test Accuracies vs. Epochs (Adam)**</u>



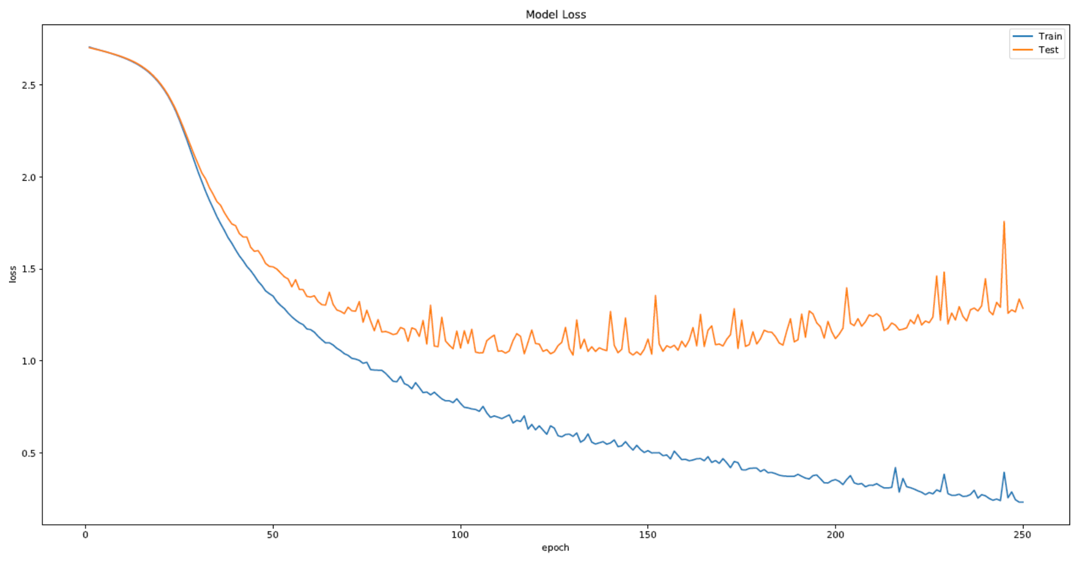
**Training and Test Loss vs. Epochs (Adam)**

From the Training and Test Accuracies vs. Epochs graph for the Adam optimizer, we can observe that the Character CNN Classifier converges very quickly at around the first few epochs. From there on, the test accuracy converges at around 0.67 accuracy and the training accuracy converges at 1.0 accuracy. Also, we can observe that the test loss is much higher than the training loss and that the test loss is slowly increasing with each epoch.

Therefore, from these results we can observe that the model is likely overfitting on the training data and not really learning a general pattern for classification of the 15 categories.



**Training and Test Accuracies vs. Epochs (SGD)**



**Training and Test Loss vs. Epochs (SGD)**

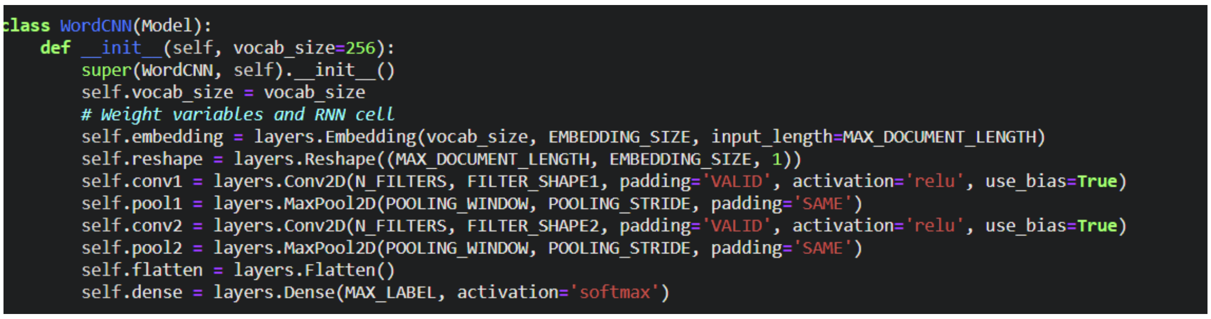
From the Training and Test Accuracies vs. Epochs graph for the SGD optimizer, we can see that the model converges slower as compared to using an Adam optimizer. However, we were able to achieve a higher test accuracy at convergence at around 0.68 accuracy. We can also observe that the test loss is lowest at around 120 epochs and after which it slowly increases every epoch.

Similar to the model with the Adam optimizer, we can observe that the model is likely overfitting on the training data and not really learning a general pattern for classification of the 15 categories.

## Question 2

Design a Word CNN Classifier that receives word ids and classifies the input. Pass the inputs through an embedding layer of size 20 before feeding to the CNN. The CNN has two convolution and pooling layers with the following characteristics:

- A convolution layer $C_1$ of 10 filters of window size 20x20, VALID padding, and ReLU neurons. A max pooling layer $S_1$ with a pooling window of size 4x4, with stride = 2 and padding = 'SAME'.
- A convolution layer $C_2$ of 10 filters of window size 20x1, , VALID padding, and ReLU neurons. A max pooling layer $S_2$ with a pooling window of size 4x4, with stride = 2 and padding = 'SAME'.



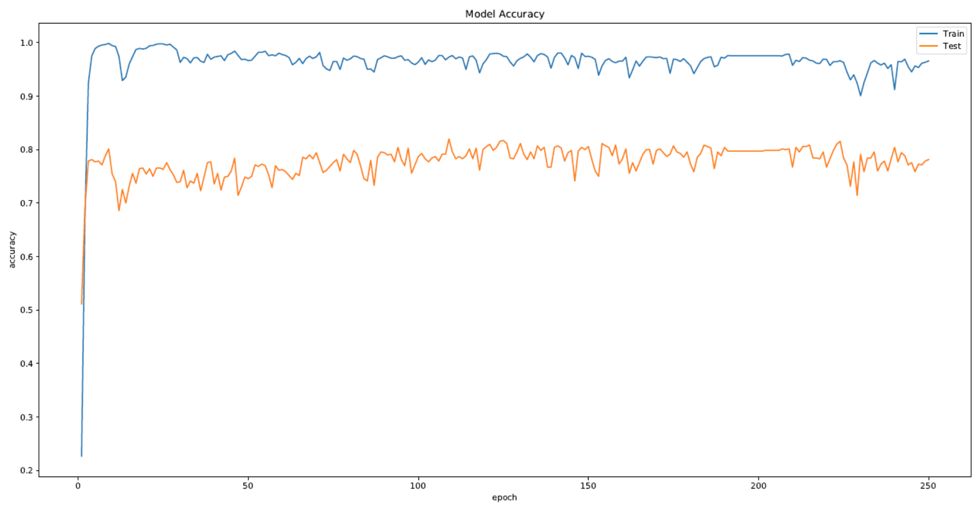
```
class WordCNN(Model):
    def __init__(self, vocab_size=256):
        super(WordCNN, self).__init__()
        self.vocab_size = vocab_size
        # Weight variables and RNN cell
        self.embedding = layers.Embedding(vocab_size, EMBEDDING_SIZE, input_length=MAX_DOCUMENT_LENGTH)
        self.reshape = layers.Reshape((MAX_DOCUMENT_LENGTH, EMBEDDING_SIZE, 1))
        self.conv1 = layers.Conv2D(N_FILTERS, FILTER_SHAPE1, padding='VALID', activation='relu', use_bias=True)
        self.pool1 = layers.MaxPool2D(POOLING_WINDOW, POOLING_STRIDE, padding='SAME')
        self.conv2 = layers.Conv2D(N_FILTERS, FILTER_SHAPE2, padding='VALID', activation='relu', use_bias=True)
        self.pool2 = layers.MaxPool2D(POOLING_WINDOW, POOLING_STRIDE, padding='SAME')
        self.flatten = layers.Flatten()
        self.dense = layers.Dense(MAX_LABEL, activation='softmax')
```

The figure above shows the Word CNN Classifier that we will be using, with an embedding layer of size 20, with a convolution layer $C_1$ of 10 filters of window size 20x20, VALID padding, and ReLU neurons, a max pooling layer $S_1$ with a pooling window of size 4x4, with stride = 2, and padding = 'SAME', a convolution layer $C_2$ of 10 filters of window size 20x1, VALID padding, and ReLU neurons, a max pooling layer $S_2$ with a pooling window of size 4x4, with stride = 2 and padding = 'SAME'.

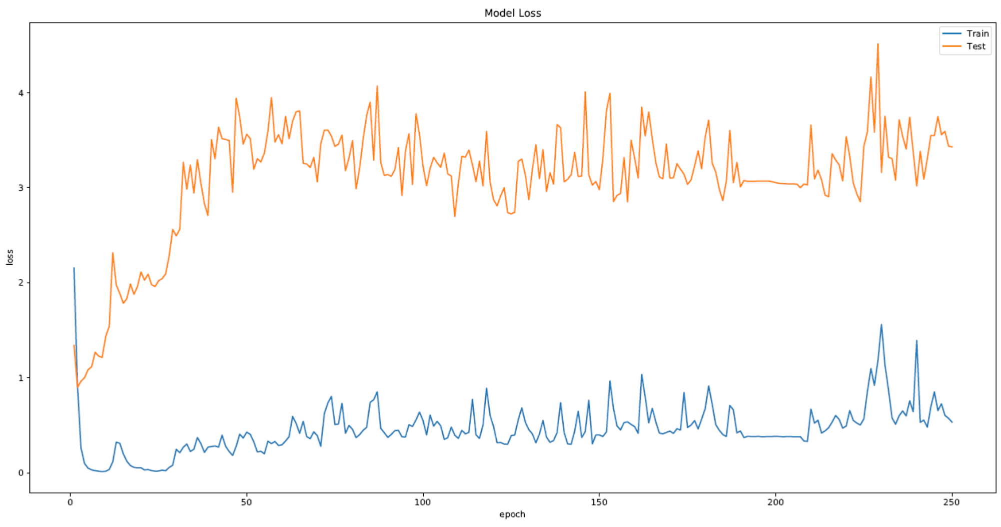
For the last layer, we will be using a dense layer with ‘softmax’ activation and 15 neurons for the number of classes for the classification. For Question 2, the model will have a dropout rate of 0.0. The model will be trained for 250 epochs.

We will use both SGD and Adam optimizers see which is a better model for this classification model. Both will have a learning rate of 0.01.

**Plot the entropy cost on training data and the accuracy on testing data against training epochs.**



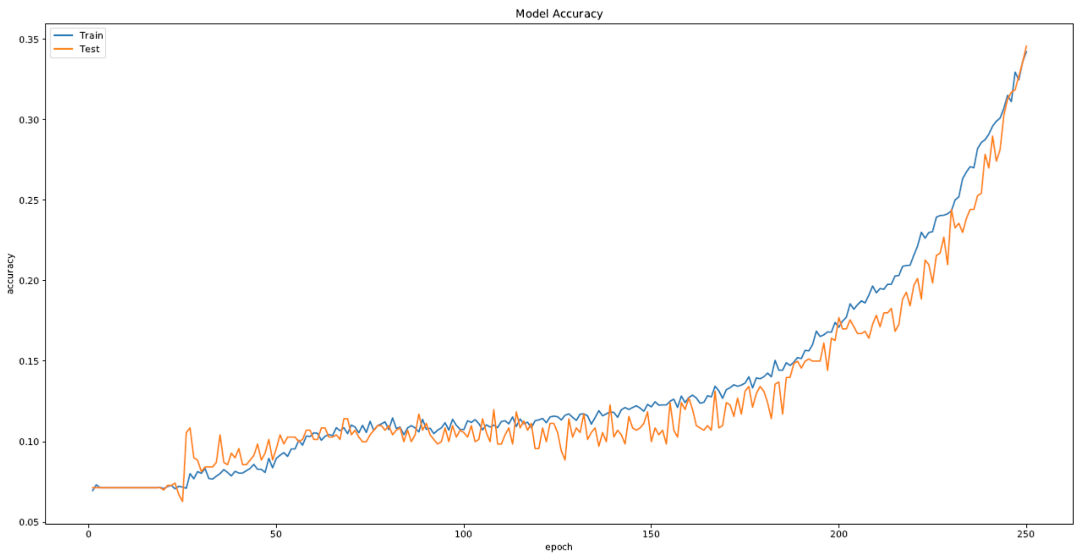
**<u>Training and Test Accuracies vs. Epochs (Adam)</u>**



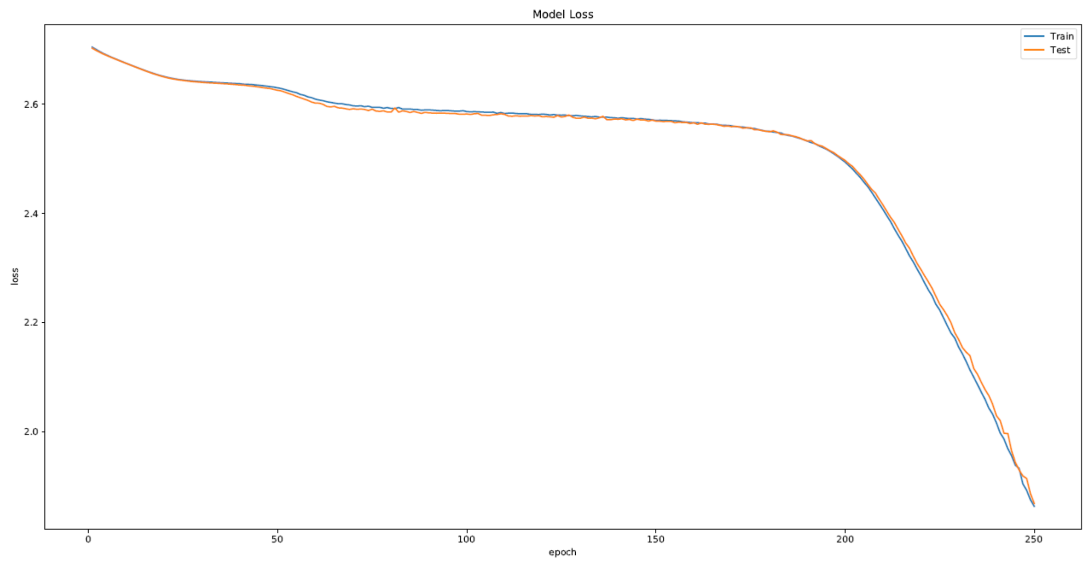
**Training and Test Loss vs. Epochs (Adam)**

From the Training and Test Accuracies vs. Epochs graph for the Adam optimizer, we can observe that the Word CNN Classifier converges very quickly at around the first few epochs. From there on, the test accuracy converges at around 0.78 to 0.80 accuracy and the training accuracy converges at 0.95 accuracy. Also, we can observe that the test loss is much higher than the training loss and that the test loss is slowly increasing with each epoch.

Therefore, from these results we can observe that the model is likely overfitting on the training data and not really learning a general pattern for classification of the 15 categories.



**Training and Test Accuracies vs. Epochs (SGD)**



**Training and Test Loss vs. Epochs (SGD)**

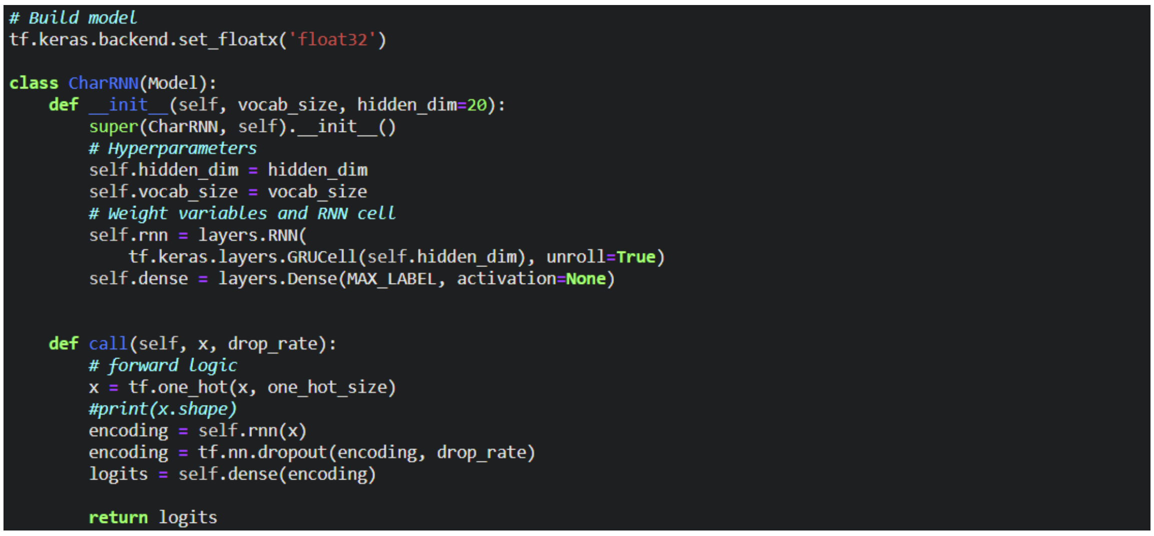
From the Training and Test Accuracies vs. Epochs graph for the SGD optimizer, we can observe that the Word CNN Classifier is training very slowly when using the SGD optimizer. It only achieved a test accuracy of 0.34 by 250 epochs. However, the good thing is that the model is not overfitting when using the SGD optimizer. To achieve a higher accuracy, the model can be trained for more epochs. From the Training and Test Loss vs. Epochs graph, we can see that both the test and training loss is decreasing with each epoch, showing that the model is still learning and reducing the m.s.e for the model.

From this, we can see that for this Word CNN Classifier, using an SGD optimizer allows the model to learn a more general pattern as compared to the Adam optimizer.

## Question 3

Design a Character RNN Classifier that receives character ids and classify the input. The RNN is GRU layer and has a hidden-layer size of 20.

**Plot the entropy cost on training data and the accuracy on testing data against training epochs.**

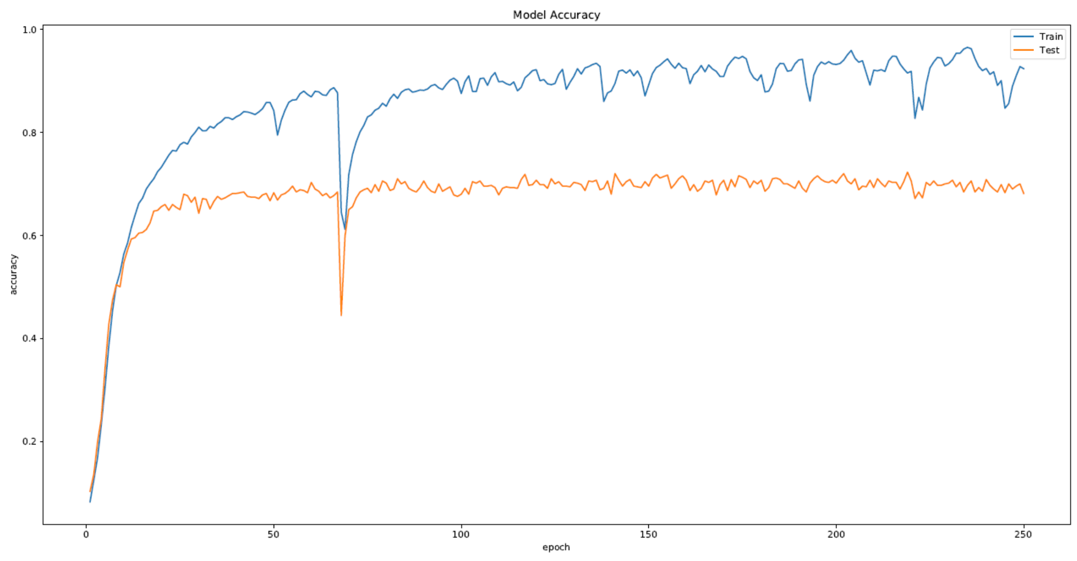


```
# Build model
tf.keras.backend.set_floatx('float32')

class CharRNN(Model):
    def __init__(self, vocab_size, hidden_dim=20):
        super(CharRNN, self).__init__()
        # Hyperparameters
        self.hidden_dim = hidden_dim
        self.vocab_size = vocab_size
        # Weight variables and RNN cell
        self.rnn = layers.RNN(
            tf.keras.layers.GRUCell(self.hidden_dim), unroll=True)
        self.dense = layers.Dense(MAX_LABEL, activation=None)


    def call(self, x, drop_rate):
        # forward logic
        x = tf.one_hot(x, one_hot_size)
        #print(x.shape)
        encoding = self.rnn(x)
        encoding = tf.nn.dropout(encoding, drop_rate)
        logits = self.dense(encoding)

        return logits
```

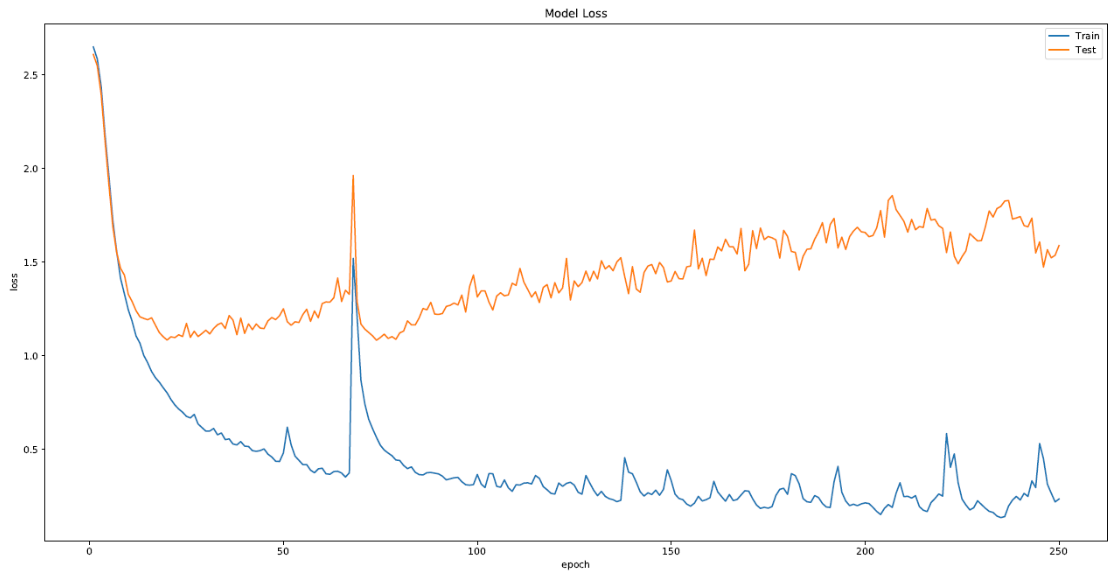
We first designed a Character RNN Classifier with an RNN layer, with a GRUCell contained within the RNN layer. The output layer is a dense layer with 15 neurons for the 15 class labels that we have.

There is no activation function for the last dense layer. Instead of declaring the last dense layer with a "softmax" activation function, we declared **from_logits = True** in the loss function of the model which is essentially equivalent to having a "softmax" activation function but computationally more stable. For Question 3, the model will have a dropout rate of 0.0. The model will be trained for 250 epochs.

We will use both SGD and Adam optimizers see which is a better model for this classification model. Both will have a learning rate of 0.01.



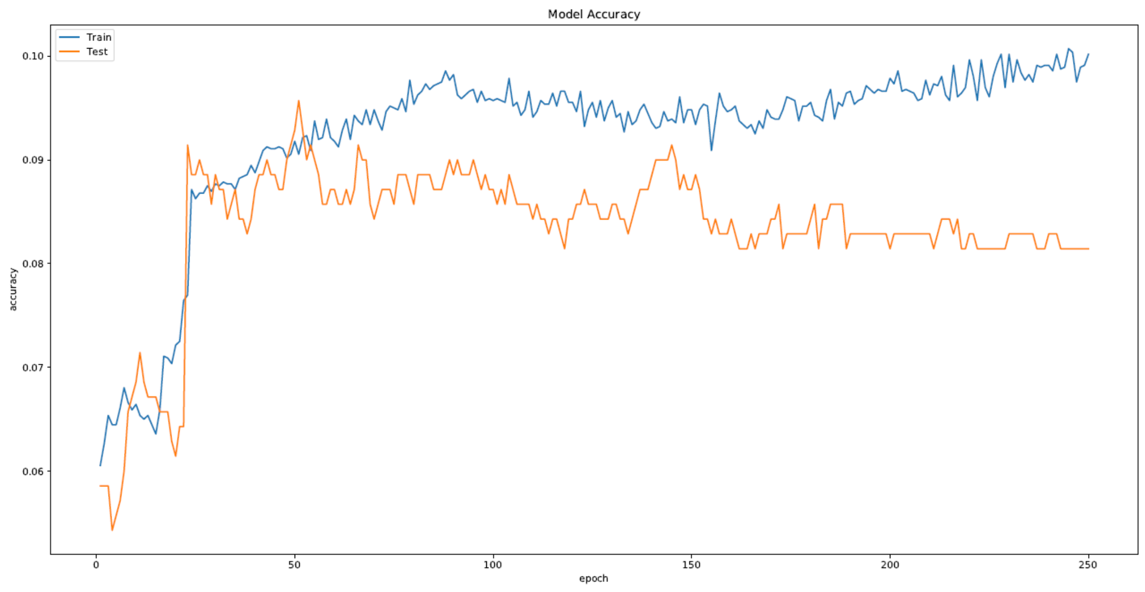
**Training and Test Accuracies vs. Epochs (Adam)**



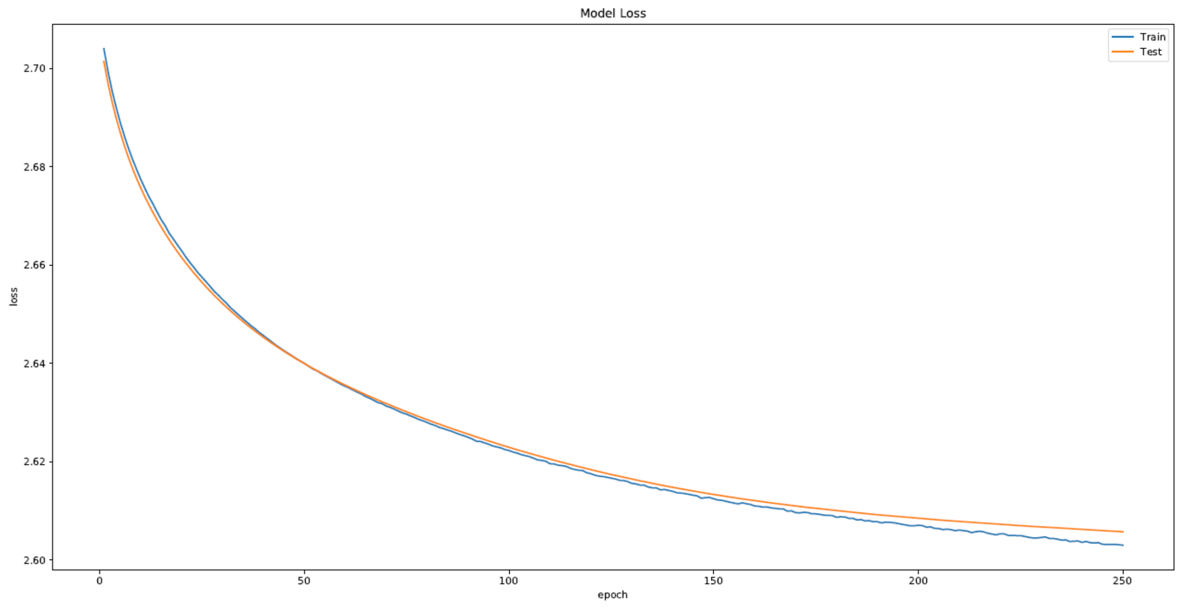
**Training and Test Loss vs. Epochs (Adam)**

From the Training and Test Accuracies vs. Epochs graph for the Adam optimizer, we can observe that the Character RNN Classifier converges very quickly at around the first few epochs. From there on, the test accuracy converges at around 0.69 to 0.70 accuracy and the training accuracy converges at around 0.85 to 0.96 accuracy with some slight fluctuations. Also, we can observe that the test loss is higher than the training loss and that the test loss is slowly increasing with each epoch.

Therefore, from these results we can observe that the model is likely overfitting on the training data and not really learning a general pattern for classification of the 15 categories.



**<u>Training and Test Accuracies vs. Epochs (SGD)</u>**



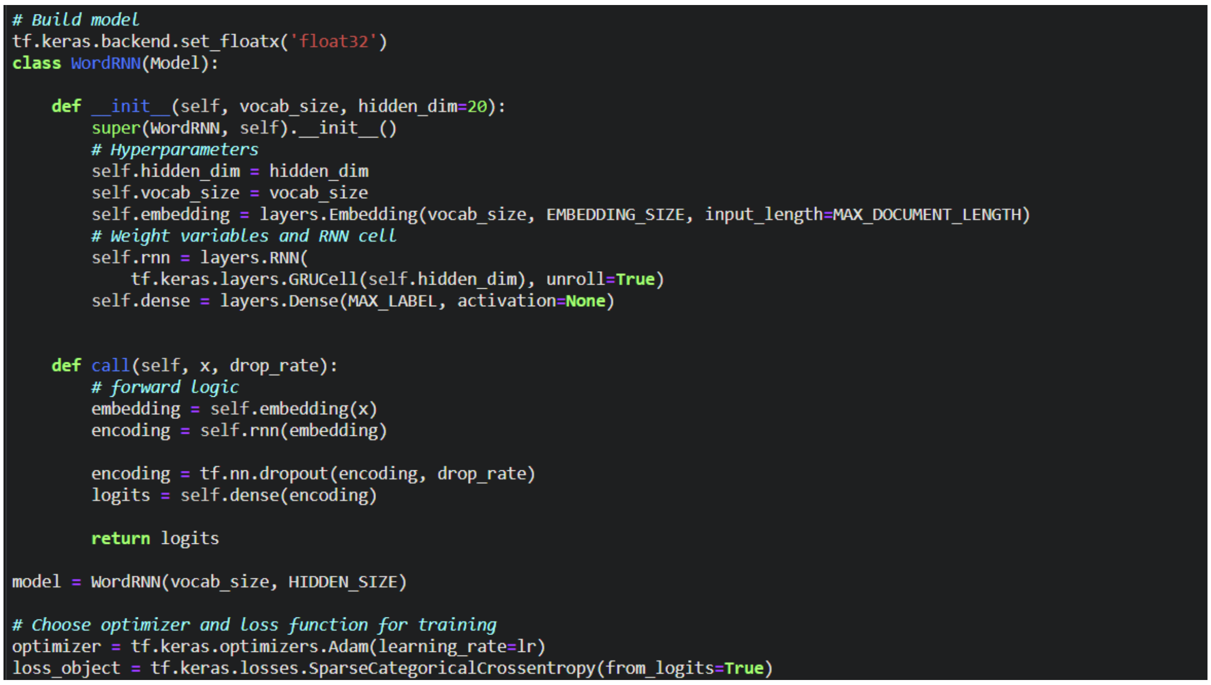
**<u>Training and Test Loss vs. Epochs (SGD)</u>**

From the From the Training and Test Accuracies vs. Epochs graph for the SGD optimizer, we can observe that the Character RNN Classifier should not utilize an SGD optimizer. As can be seen from the Training and Test Accuracies vs. Epochs graph, the test accuracy converges at around 0.08 accuracy and the training accuracy is also stagnant at around 0.10 accuracy. Although the test and training loss for the model is decreasing with every epoch, we should not use the SGD optimizer for the Character RNN Classifier as it produces poor test accuracies.

## Question 4

Design a word RNN classifier that receives word ids and classify the input. The RNN is GRU layer and has a hidden-layer size of 20. Pass the inputs through an embedding layer of size 20 before feeding to the RNN.

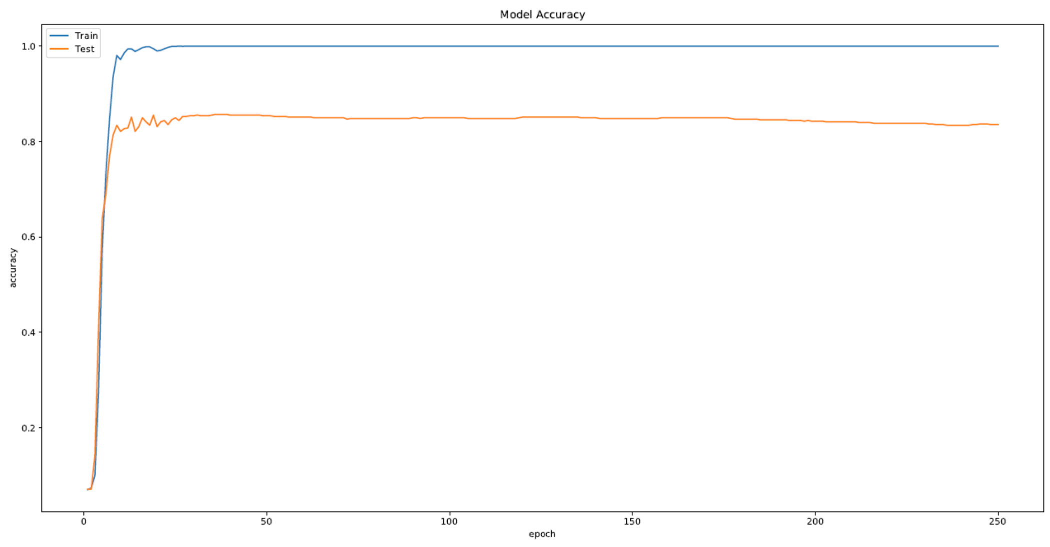
**Plot the entropy on training data and the accuracy on testing data versus training epochs.**



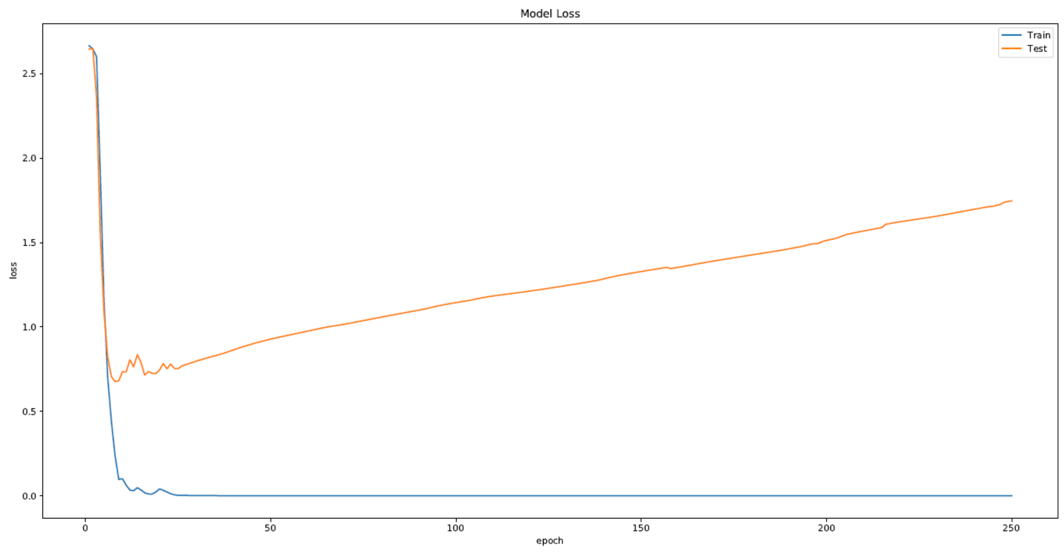
```python
# Build model
tf.keras.backend.set_floatx('float32')
class WordRNN(Model):

    def __init__(self, vocab_size, hidden_dim=20):
        super(WordRNN, self).__init__()
        # Hyperparameters
        self.hidden_dim = hidden_dim
        self.vocab_size = vocab_size
        self.embedding = layers.Embedding(vocab_size, EMBEDDING_SIZE, input_length=MAX_DOCUMENT_LENGTH)
        # Weight variables and RNN cell
        self.rnn = layers.RNN(
            tf.keras.layers.GRUCell(self.hidden_dim), unroll=True)
        self.dense = layers.Dense(MAX_LABEL, activation=None)


    def call(self, x, drop_rate):
        # forward logic
        embedding = self.embedding(x)
        encoding = self.rnn(embedding)

        encoding = tf.nn.dropout(encoding, drop_rate)
        logits = self.dense(encoding)

        return logits

model = WordRNN(vocab_size, HIDDEN_SIZE)

# Choose optimizer and loss function for training
optimizer = tf.keras.optimizers.Adam(learning_rate=lr)
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

We first designed a word RNN classifier with an embedding layer of size 20, and a RNN layer, with a GRUCell contained within the RNN layer. The output layer is a dense layer with 15 neurons for the 15 class labels that we have. There is no activation function for the last dense layer.  Instead of declaring the last dense layer with a “softmax” activation function, we declared **from_logits = True** in the loss function of the model which is essentially equivalent to having a “softmax” activation function but computationally more stable. For Question 4, the model will have a dropout rate of 0.0. The model will be trained for 250 epochs.

We will use both SGD and Adam optimizers see which is a better model for this classification model. Both will have a learning rate of 0.01.

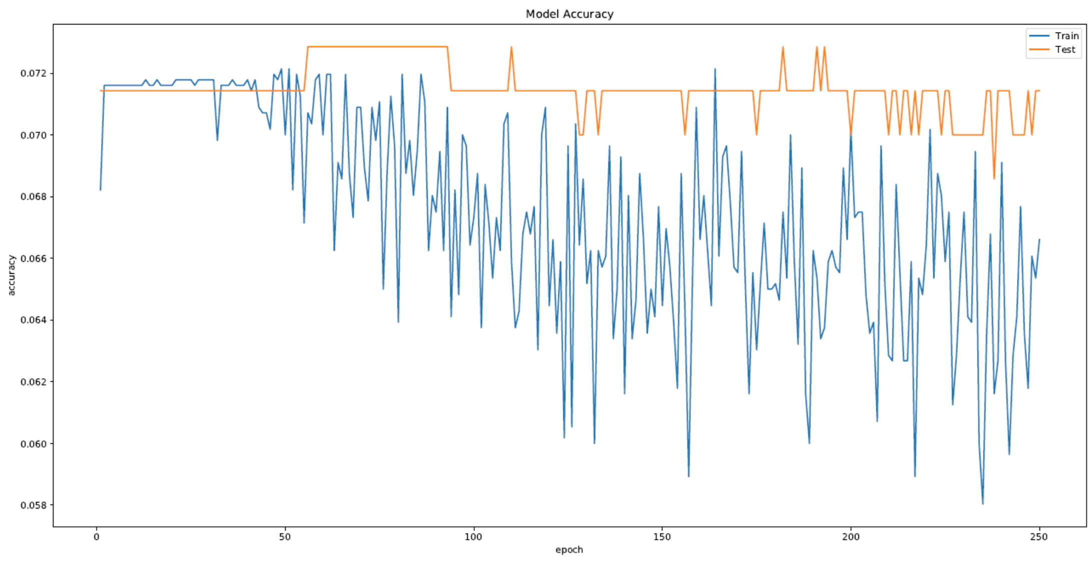


**Training and Test Accuracies vs. Epochs (Adam)**

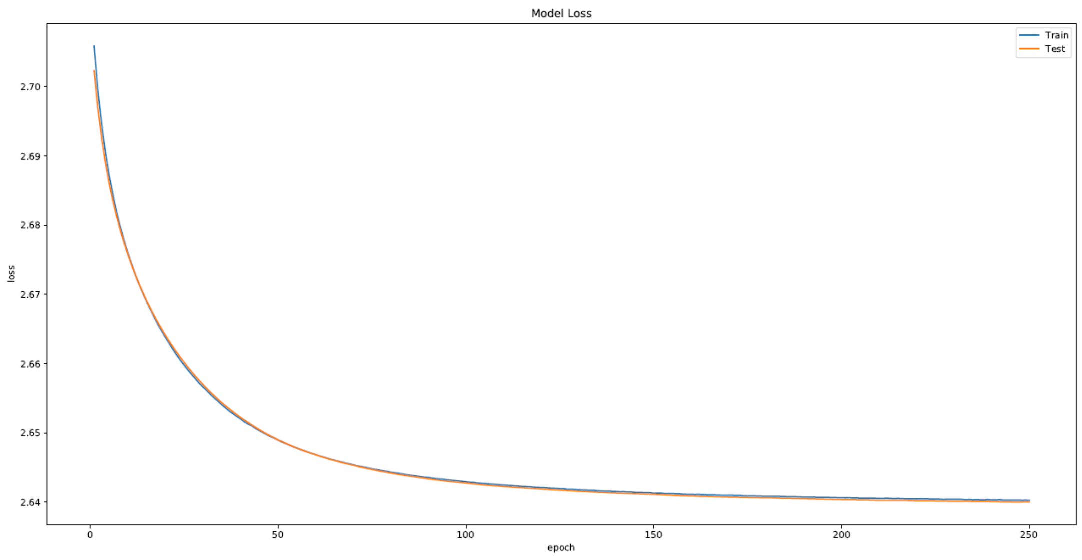


**Training and Test Loss vs. Epochs (Adam)**

From the Training and Test Accuracies vs. Epochs graph for the Adam optimizer, we can observe that the word RNN Classifier converges very quickly at around the first few epochs. The word RNN Classifier manages to achieve quite a high test accuracy after it converges at around 0.83 accuracy. However, after the initial dip in test loss for the first few epoch, the test lost increases linearly as the number of epochs increases.



**Training and Test Accuracies vs. Epochs (SGD)**



**Training and Test Loss vs. Epochs (SGD)**

From the From the Training and Test Accuracies vs. Epochs graph for the SGD optimizer, we can observe that similar to the Character RNN Classifier, the word RNN Classifier should not utilize an SGD optimizer. As can be seen from the Training and Test Accuracies vs. Epochs graph, the test accuracy is stagnant at around 0.07 accuracy sine the first epoch and the training accuracy is decreasing and fluctuating rapidly. Also, although the loss for both test and training are decreasing over the epoch, the m.s.e. values of the losses are still very high.

From the results, we can observe that when we use an SGD optimizer, the word RNN Classifier is not able to learn the data that was passed to it, with the training accuracy decreasing through the epochs and the test accuracy remaining stagnant at a very low accuracy for all the training epochs. Therefore, for a word RNN Classifier, we should not utilize an SGD optimizer.

## Question 5

**Compare the test accuracies and the running times of the networks implemented in parts (1) – (4).**

## Test Accuracies and performance

**Character CNN Classifier**

**Adam Optimizer:**

From the Training and Test Accuracies vs. Epochs graph, we can observe that the Character CNN Classifier with the Adam optimizer converges very quickly at around the first few epochs. From there on, the test accuracy converges at around 0.67 accuracy and the training accuracy converges at 1.0 accuracy. Also, we can observe that the test loss is much higher than the training loss and that the test loss is slowly increasing with each epoch.

Therefore, from these results we can observe that the model is likely overfitting on the training data and not really learning a general pattern for classification of the 15 categories.

**SGD optimizer:**

From the Training and Test Accuracies vs. Epochs graph for the SGD optimizer, we can see that the model converges slower as compared to using an Adam optimizer. However, we were able to achieve a higher test accuracy at convergence at around 0.68 accuracy. We can also observe that the test loss is lowest at around 120 epochs and after which it slowly increases every epoch.

Similar to the model with the Adam optimizer, we can observe that the model is likely overfitting on the training data and not really learning a general pattern for classification of the 15 categories.

**Word CNN Classifier**

**Adam Optimizer:**

From the Training and Test Accuracies vs. Epochs graph, we can observe that the Word CNN Classifier with the Adam optimizer converges very quickly at around the first few epochs. From there on, the test accuracy converges at around 0.78 to 0.80 accuracy and the training accuracy converges at 0.95 accuracy. Also, we can observe that the test loss is much higher than the training loss and that the test loss is slowly increasing with each epoch.

Therefore, from these results we can observe that the model is likely overfitting on the training data and not really learning a general pattern for classification of the 15 categories.

**SGD optimizer:**

From the Training and Test Accuracies vs. Epochs graph for the SGD optimizer, we can observe that the Word CNN Classifier is training very slowly when using the SGD optimizer. It only achieved a test accuracy of 0.34 by 250 epochs. However, the good thing is that the model is not overfitting when using the SGD optimizer. To achieve a higher accuracy, the model can be trained for more epochs. From the Training and Test Loss vs. Epochs graph, we can see that both the test and training loss is decreasing with each epoch, showing that the model is still learning and reducing the m.s.e for the model.

From this, we can see that for this Word CNN Classifier, using an SGD optimizer allows the model to learn a more general pattern as compared to the Adam optimizer.

**Character RNN Classifier**

**Adam Optimizer:**

From the Training and Test Accuracies vs. Epochs graph, we can observe that the Character RNN Classifier with the Adam optimizer converges very quickly at around the first few epochs. From there on, the test accuracy converges at around 0.69 to 0.70 accuracy and the training accuracy converges at around 0.85 to 0.96 accuracy with some slight fluctuations. Also, we can observe that the test loss is higher than the training loss and that the test loss is slowly increasing with each epoch.

**SGD optimizer:**

From the From the Training and Test Accuracies vs. Epochs graph for the SGD optimizer, we can observe that the Character RNN Classifier should not utilize an SGD optimizer. As can be seen from the Training and Test Accuracies vs. Epochs graph, the test accuracy converges at around 0.08 accuracy and the training accuracy is also stagnant at around 0.10 accuracy. Although the test and training loss for the model is decreasing with every epoch, we should not use the SGD optimizer for the Character RNN Classifier as it produces poor test accuracies.

## Word RNN Classifier

**Adam Optimizer:**

From the Training and Test Accuracies vs. Epochs graph, we can observe that the word RNN Classifier with the Adam optimizer converges very quickly at around the first few epochs. The word RNN Classifier manages to achieve quite a high test accuracy after it converges at around 0.83 accuracy. However, after the initial dip in test loss for the first few epoch, the test lost increases linearly as the number of epochs increases.

**SGD optimizer:**

From the From the Training and Test Accuracies vs. Epochs graph for the SGD optimizer, we can observe that similar to the Character RNN Classifier, the word RNN Classifier should not utilize an SGD optimizer. As can be seen from the Training and Test Accuracies vs. Epochs graph, the test accuracy is stagnant at around 0.07 accuracy sine the first epoch and the training accuracy is decreasing and fluctuating rapidly. Also, although the loss for both test and training are decreasing over the epoch, the m.s.e. values of the losses are still very high.

From the results, we can observe that when we use an SGD optimizer, the word RNN Classifier is not able to learn the data that was passed to it, with the training accuracy decreasing through the epochs and the test accuracy remaining stagnant at a very low accuracy for all the training epochs. Therefore, for a word RNN Classifier, we should not utilize an SGD optimizer.

## Various Classifiers Last Epoch Accuracy

| Classifier | Last Epoch Accuracy |
|---|---|
| Character CNN Classifier (Adam Optimizer) | 0.67571 |
| Character CNN Classifier (SGD Optimizer) | 0.68143 |
| Word CNN Classifier (Adam Optimizer) | 0.78143 |
| Word CNN Classifier (SGD Optimizer) | 0.34571 |
| Character RNN Classifier (Adam Optimizer) | 0.68143 |
| Character RNN Classifier (SGD Optimizer) | 0.08143 |
| Word RNN Classifier (Adam Optimizer) | 0.83571 |
| Word RNN Classifier (SGD Optimizer) | 0.07143 |

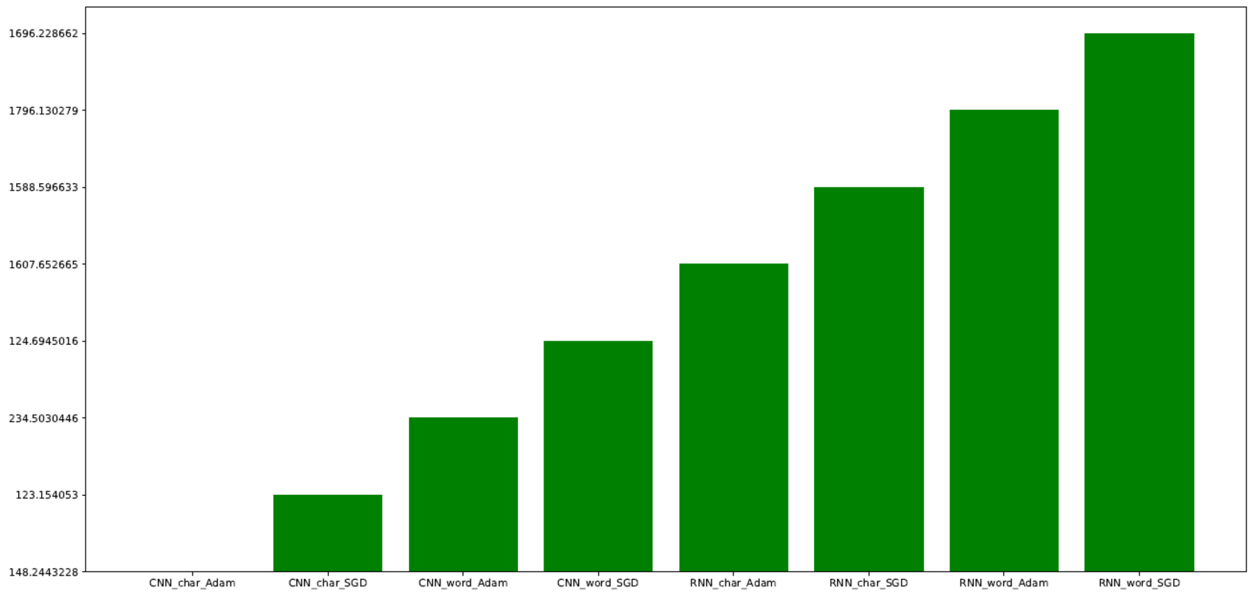
From the results above, we can see that the Word RNN Classifier with an Adam optimizer has the best test accuracy amongst all the text classifiers. We can see that for most of the classifiers, using an Adam optimizer produces better accuracy as compared to using the SGD optimizer.

## Running Times

In addition to the test accuracy, we also took down the time taken to run each of the networks in parts (1) – (4) in seconds. These are the times taken for each network to run 250 epochs of training:

| Classifier | Run Times (in seconds) |
|---|---|
| Character CNN Classifier (Adam Optimizer) | 148.2443 |
| Character CNN Classifier (SGD Optimizer) | 123.1541 |
| Word CNN Classifier (Adam Optimizer) | 234.503 |
| Word CNN Classifier (SGD Optimizer) | 124.6945 |
| Character RNN Classifier (Adam Optimizer) | 1607.653 |
| Character RNN Classifier (SGD Optimizer) | 1588.597 |
| Word RNN Classifier (Adam Optimizer) | 1796.13 |
| Word RNN Classifier (SGD Optimizer) | 1696.229 |

With the run times for each network obtained, we then plot the times for each network as a histogram as shown below:



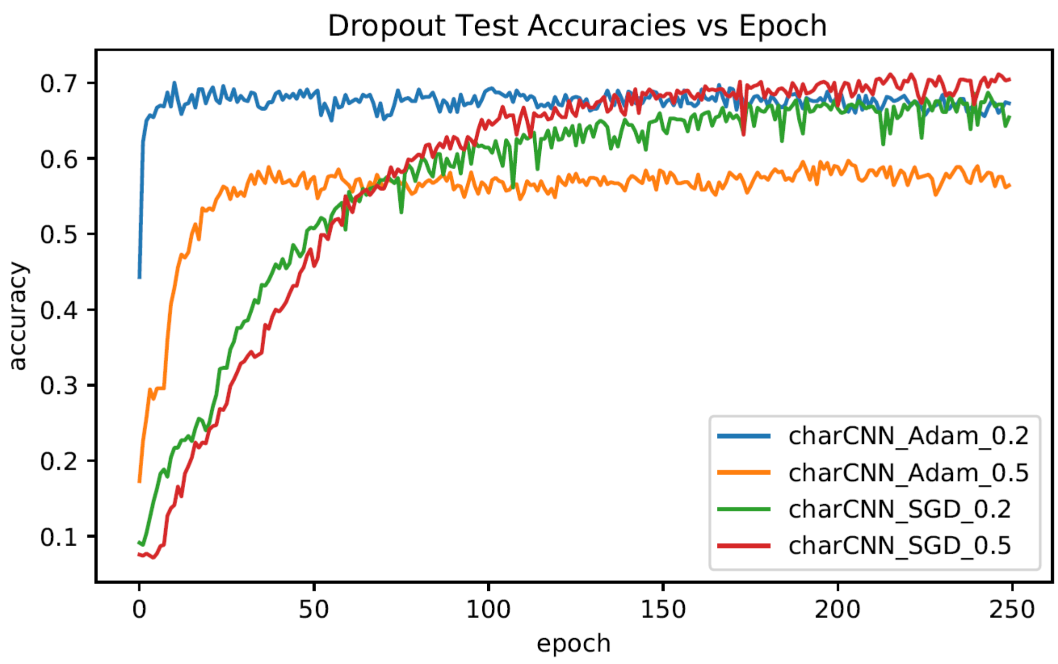
A can be seen from the graph as well as the table, we notice that for each network, using an Adam optimizer causes the network to run faster as compared to using an SGD optimizer. Next, we can observe that for both CNN and RNN networks, the Character Classifier runs faster than the Word Classifier. We can also observe that as compared to the CNN classifiers, the RNN Classifiers take a fairly long amount of time to run for 250 epochs. However, we can see that for RNN Classifiers using the Adam optimizers, they produce better results for both Character and Word Classification as compared to CNN Classifiers using the Adam optimizer, sacrificing execution speed for better accuracy.

## Test Accuracies with dropout

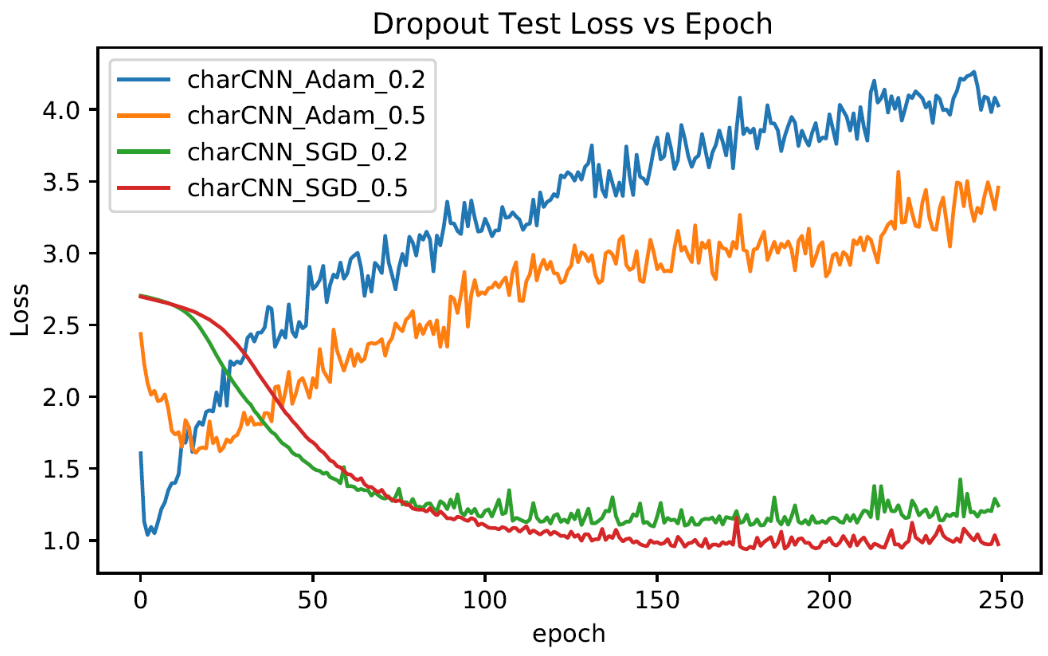
**Experiment with adding dropout to the layers of networks in parts (1) – (4) and report the test accuracies. Compare and comment on the accuracies of the networks with/without dropout.**

For each of the networks in parts (1) – (4), we experimented with adding a dropout of 0.2 and 0.5, with both Adam and SGD optimizers to see which provides a better performance.

**Character CNN Classifier**



**Character CNN Classifier Test Accuracies vs. Epochs**



**Character CNN Classifier Test Loss vs. Epochs**

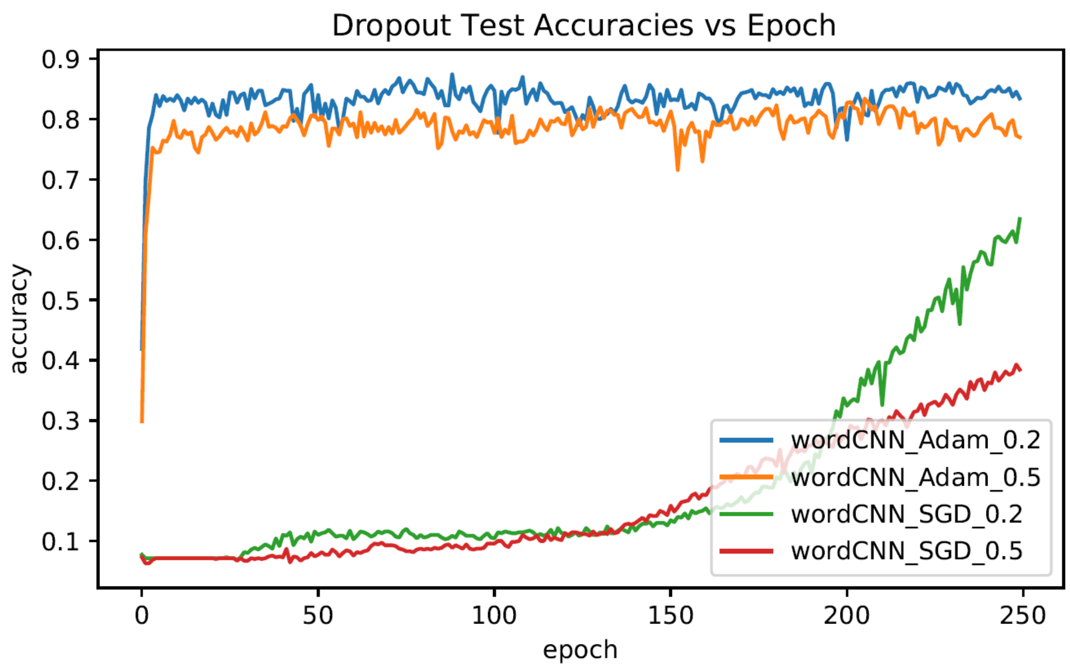
From the graphs above, we can see that for the Character CNN Classifier, when using an Adam optimizer, using a dropout rate of 0.2 produces better test accuracies as compared to a dropout rate of

0.5 and for SGD optimizer, using a dropout rate of 0.5 produces better test accuracies as compared to a dropout rate of 0.2. From this we can also see that using an SGD optimizer with a dropout rate 0.5 produces better test accuracies as compared to the other Character CNN Classifiers with dropouts.

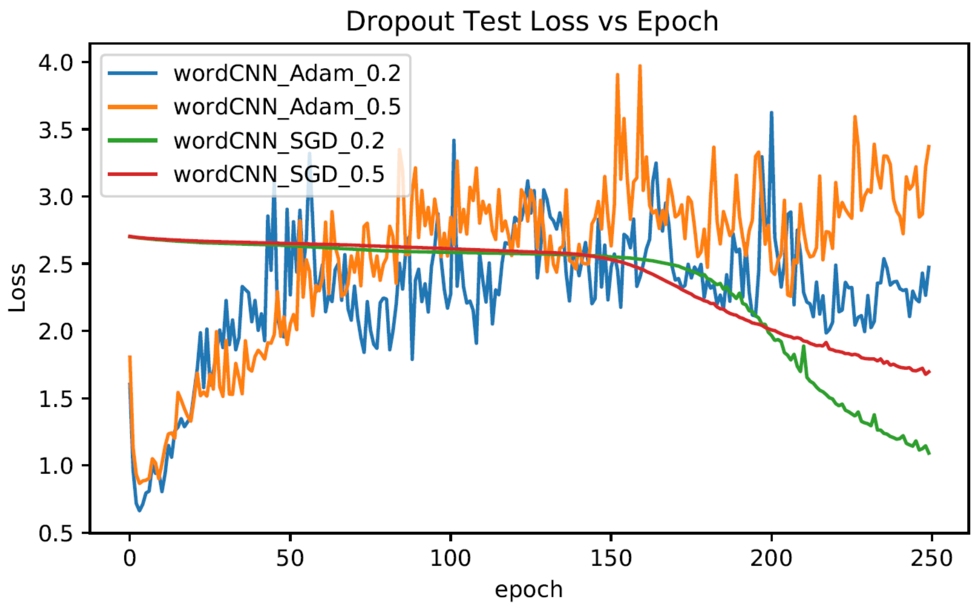
| Character CNN Classifier | **Last Epoch Test Accuracy** |
|---|---|
| Adam Optimizer + no dropout | 0.67571 |
| Adam Optimizer + 0.2 dropout | 0.67286 |
| Adam Optimizer + 0.5 dropout | 0.56429 |
| SGD Optimizer + no dropout | 0.68143 |
| SGD Optimizer + 0.2 dropout | 0.65429 |
| SGD Optimizer + 0.5 dropout | 0.70429 |

We then plot a table of the last epoch test accuracy for each of the Character CNN Classifiers. From the table, we can see that when using the Adam optimizer, the Character CNN Classifier performs better in terms of test accuracy with no dropout, and when using the SGD optimizer, the Character CNN Classifier performs better in terms of test accuracy with 0.5 dropout. Overall, the Character CNN Classifier with an SGD optimizer, a dropout layer with a 0.5 dropout rate produces the best last epoch test accuracy.

**Word CNN Classifier**



**Word CNN Classifier Test Accuracies vs. Epochs**



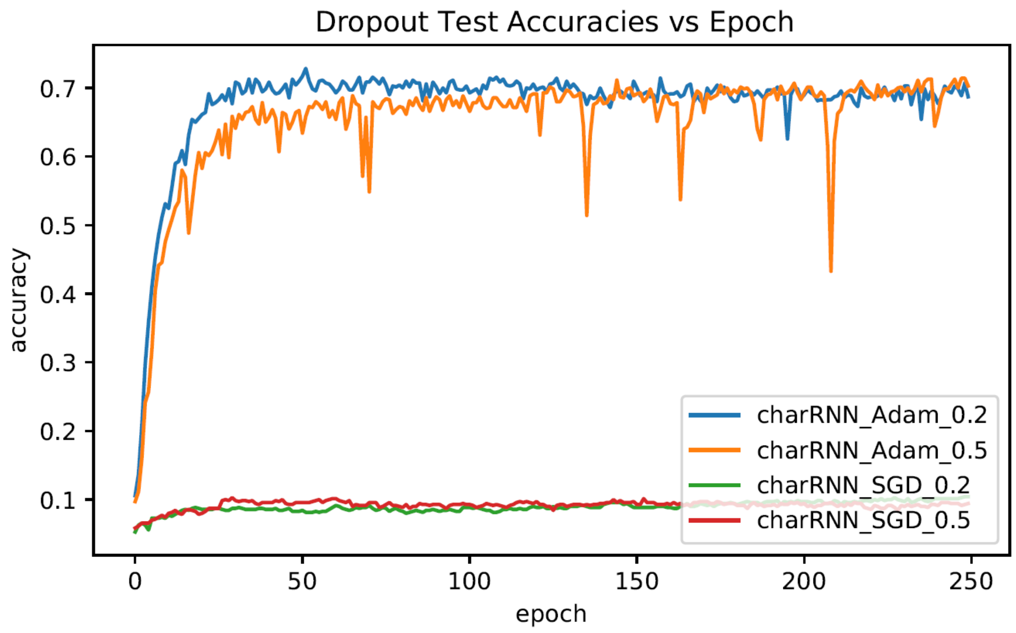
**<u>Word CNN Classifier Test Loss vs. Epochs</u>**

From the graphs above, we can see that for the Word CNN Classifier, when using an Adam optimizer, using a dropout rate of 0.2 produces better test accuracies as compared to a dropout rate of  0.5 and for SGD optimizer, using a dropout rate of 0.2 produces better test accuracies as compared to a dropout rate of  0.5. From this we can also see that using an Adam optimizer with a dropout rate 0.2 produces better test accuracies as compared to the other Word CNN Classifiers with dropouts  when running only 250 training epoch. We do not know the convergence test accuracy of the Word CNN Classifiers using SGD optimizers as the test accuracies are still increasing at 250 epochs.

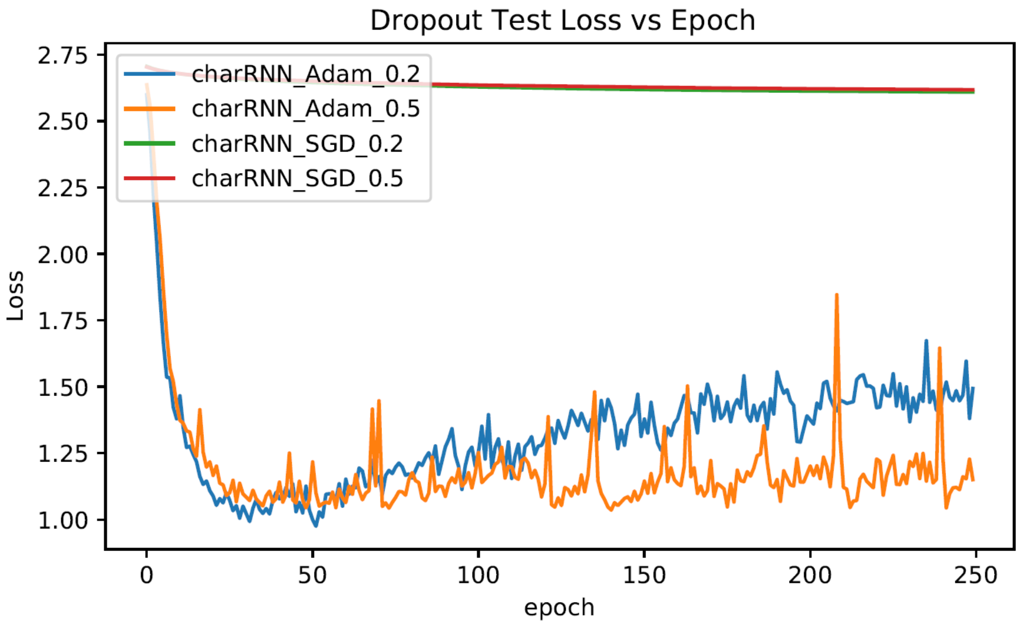
| Word CNN Classifier | **Last Epoch Test Accuracy** |
|---|---|
| Adam Optimizer + no dropout | 0.78143 |
| Adam Optimizer + 0.2 dropout | 0.83429 |
| Adam Optimizer + 0.5 dropout | 0.77000 |
| SGD Optimizer + no dropout | 0.34571 |
| SGD Optimizer + 0.2 dropout | 0.63429 |
| SGD Optimizer + 0.5 dropout | 0.38429 |

We then plot a table of the last epoch test accuracy for each of the Word CNN Classifiers. From the table, we can see that when using the Adam optimizer, the Word CNN Classifier performs better in terms of test accuracy with a 0.2 dropout rate, and when using the SGD optimizer, the Word CNN Classifier performs better in terms of test accuracy with 0.2 dropout. Overall, the Word CNN Classifier with an Adam optimizer, a dropout layer with a 0.2 dropout rate produces the best last epoch test accuracy.

**Character RNN Classifier**



**Character RNN Classifier Test Accuracies vs. Epochs**



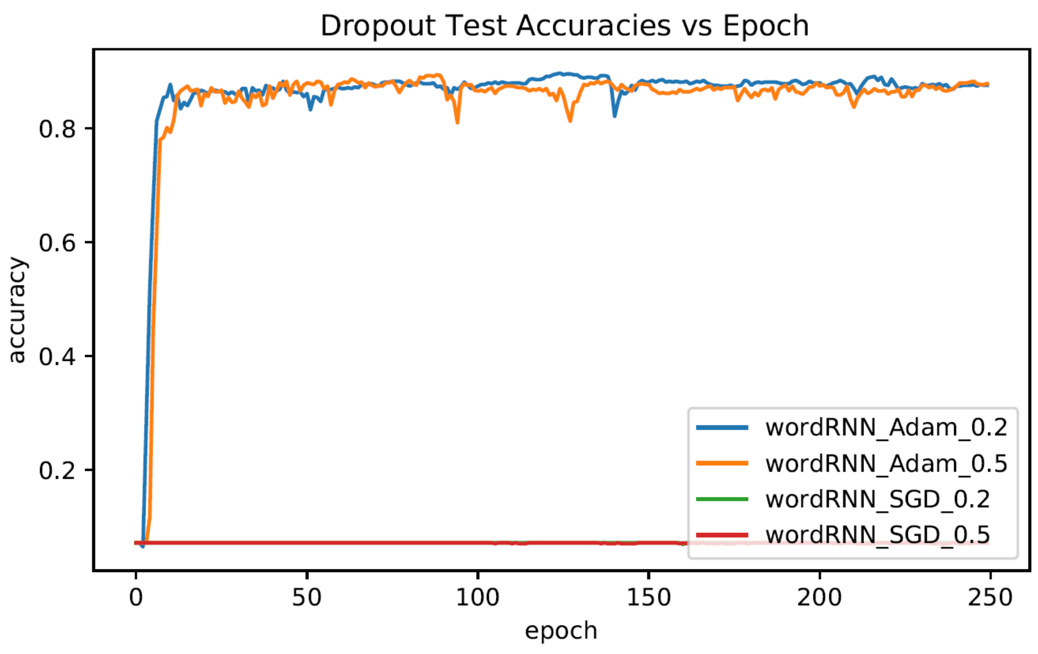
**Character RNN Classifier Test Loss vs. Epochs**

From the graphs above, we can see that for the Character RNN Classifier, when using an Adam optimizer, using a dropout rate of 0.2 or a dropout rate of  0.5 produced very similar test accuracies with both test accuracies converging at about the same accuracy rate, and for SGD optimizer, both dropout rates produces poor test accuracy results, with both the Classifiers having a stagnant test accuracy at 0.1. From this we can also see that using an Adam optimizer with a dropout layer produces better test accuracies as compared to the Character RNN Classifiers using SGD optimizers with dropout layers when running only 250 training epoch.

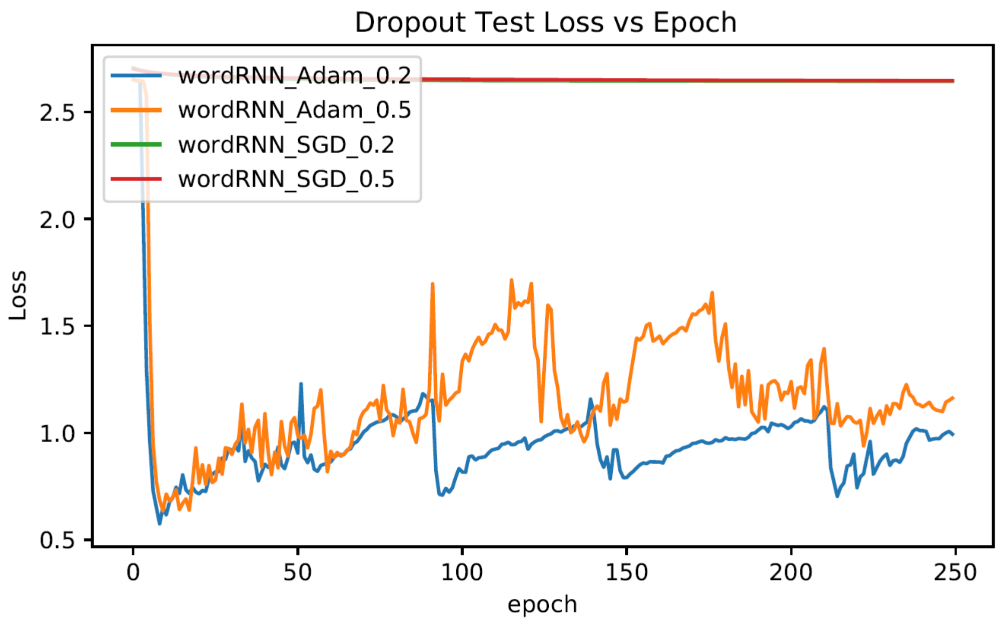
| Character RNN Classifier | **Last Epoch Test Accuracy** |
|---|---|
| Adam Optimizer + no dropout | 0.68143 |
| Adam Optimizer + 0.2 dropout | 0.68714 |
| Adam Optimizer + 0.5 dropout | 0.70286 |
| SGD Optimizer + no dropout | 0.08143 |
| SGD Optimizer + 0.2 dropout | 0.10429 |
| SGD Optimizer + 0.5 dropout | 0.09429 |

We then plot a table of the last epoch test accuracy for each of the Character RNN Classifiers. From the table, we can see that when using the Adam optimizer, the Character RNN Classifier performs slightly better in terms of last epoch test accuracy with a 0.5 dropout rate, and when using the SGD optimizer, the Character RNN Classifier performs badly even with a dropout layer added to the model, with the test accuracy not improving. Overall, the Character RNN Classifier with an Adam optimizer, a dropout layer with a 0.5 dropout rate produces the best last epoch test accuracy.

**Word RNN Classifier**



**Word RNN Classifier Test Accuracies vs. Epochs**



**Word RNN Classifier Test Loss vs. Epochs**

From the graphs above, we can see that for the Word RNN Classifier, when using an Adam optimizer, using a dropout rate of 0.2 or a dropout rate of  0.5 produced very similar test accuracies with both test accuracies converging at about the same accuracy rate, and for SGD optimizer, both dropout rates produces poor test accuracy results, with both the Classifiers having a stagnant test accuracy at 0.07 accuracy rate. From this we can also see that using an Adam optimizer, with a dropout layer, the Word RNN Classifier produces better test accuracies as compared to the Word RNN Classifiers using SGD optimizers with dropout layers when running only 250 training epoch.

| Word RNN Classifier | **Last Epoch Test Accuracy** |
|---|---|
| Adam Optimizer + no dropout | 0.83571 |
| Adam Optimizer + 0.2 dropout | 0.87571 |
| Adam Optimizer + 0.5 dropout | 0.87857 |
| SGD Optimizer + no dropout | 0.07143 |
| SGD Optimizer + 0.2 dropout | 0.07143 |
| SGD Optimizer + 0.5 dropout | 0.07143 |

We then plot a table of the last epoch test accuracy for each of the Word RNN Classifiers. From the table, we can see that when using the Adam optimizer, the Word RNN Classifier performs better in terms of last epoch test accuracy with a dropout layer added to the model, and when using the SGD optimizer, the Word RNN Classifier performs badly even with a dropout layer added to the model, with the test accuracy not improving and staying at 0.07 accuracy. Overall, the Word RNN Classifier with an Adam optimizer, and a dropout layer of 0.5 dropout rate added produces the best last epoch test accuracy.

## Conclusions

From the results that we have obtained in Question 5, adding a Dropout Layer to the model does improve the test accuracy obtained by the model. We also found out that for most of the networks, except for the Character CNN Classifier network, using an Adam optimizer produces significantly better results as compared to using an SGD optimizer.

## Question 6

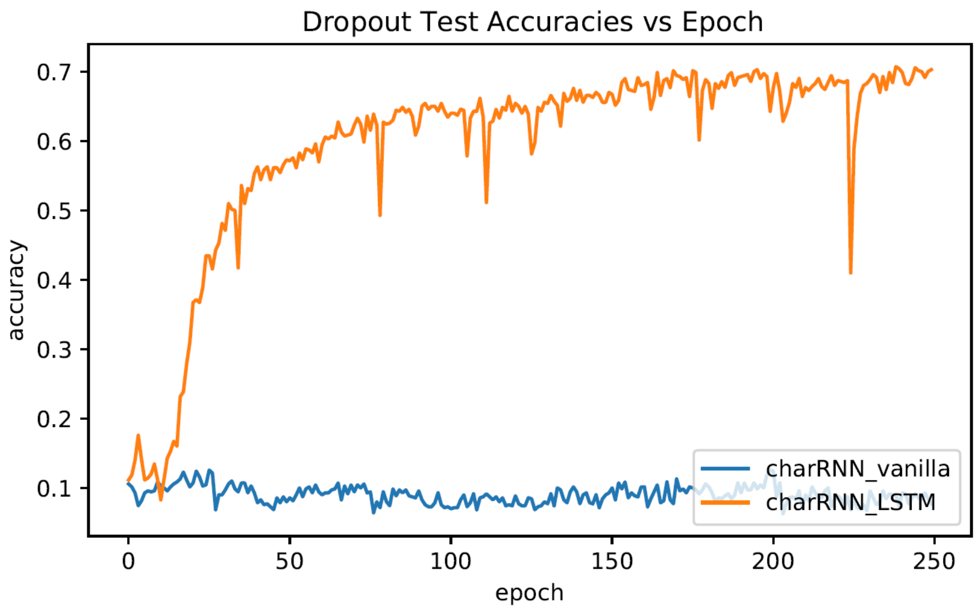
For RNN networks implemented in (3) and (4), perform the following experiments with the aim of improving performances, compare the accuracies and report your findings:

a. Replace the GRU layer with (i) a vanilla RNN layer and (ii) a LSTM layer
b. Increase the number of RNN layers to 2 layers
c. Add gradient clipping to RNN training with clipping threshold = 2.

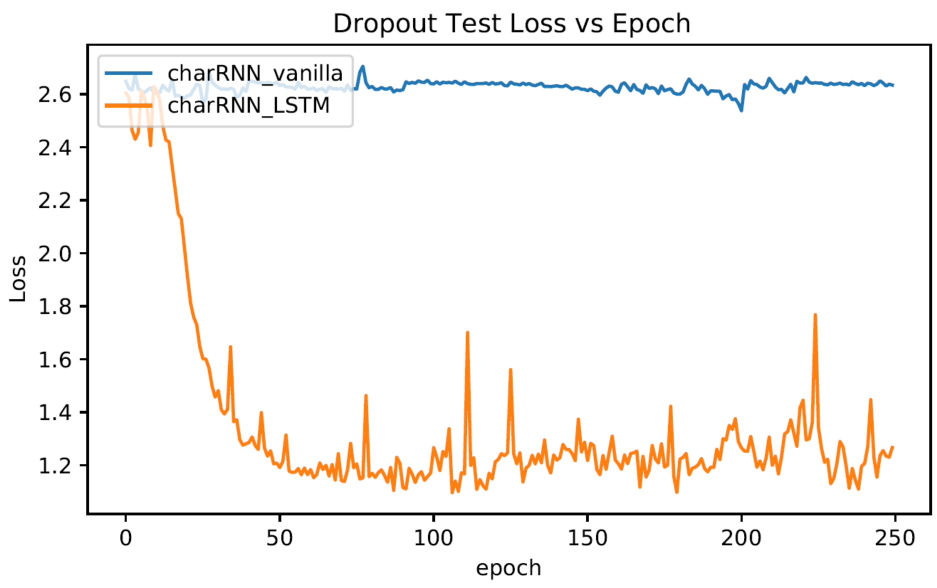
## Part A (a. Replace the GRU layer with (i) a vanilla RNN layer and (ii) a LSTM layer)

### Character RNN Classifier

For this experiment, for the Character RNN Classifiers, we used an Adam optimizer for the models, with a dropout layer of dropout rate 0.5. The RNN layer will have the GRU cell replaced by the vanilla RNN cell, as well as the LSTM cell and run for 250 epochs.



**Character RNN Classifier Test Accuracies vs. Epochs**



**Character RNN Classifier Test Loss vs. Epochs**

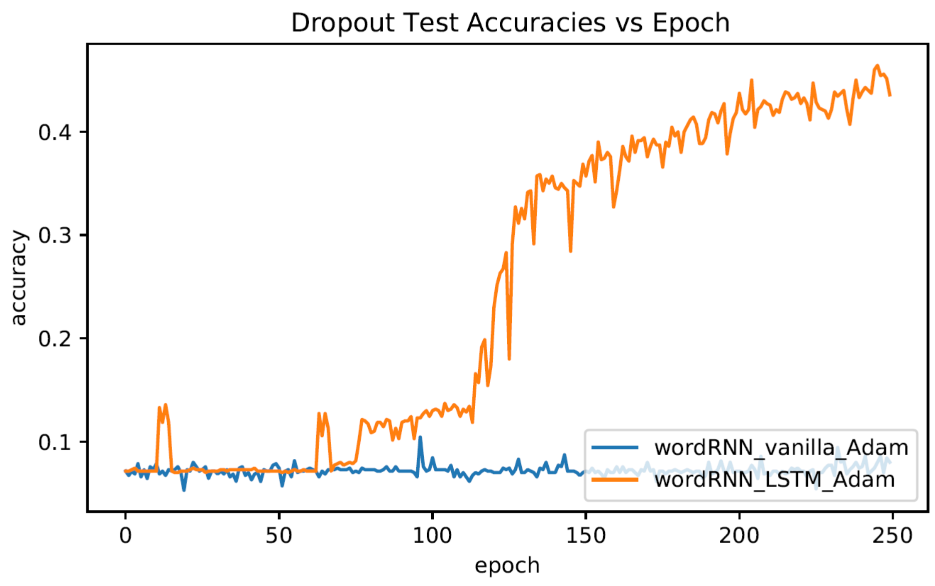
From the Test Accuracies vs. epochs graph, we can see that the Character RNN Classifier with the LSTM cell in the RNN layer has better test accuracy as compared to the  Character RNN Classifier with the vanilla RNN cell in the RNN layer. The Character RNN Classifier with the LSTM cell in the RNN layer converges at 0.7 test accuracy while the Character RNN Classifier with the vanilla RNN cell in the RNN layer has a stagnant test accuracy of around 0.1 for all 250 training epochs. From the Test Loss vs. Epochs graph, we can also see that the test loss for the vanilla RNN Classifier does not drop for all the 250 epochs.

| Character RNN Classifier | **Last Epoch Test Accuracy** |
|---|---|
| GRU Layer | 0.70286 |
| Vanilla RNN Layer | 0.07714 |
| LSTM Layer | 0.70286 |

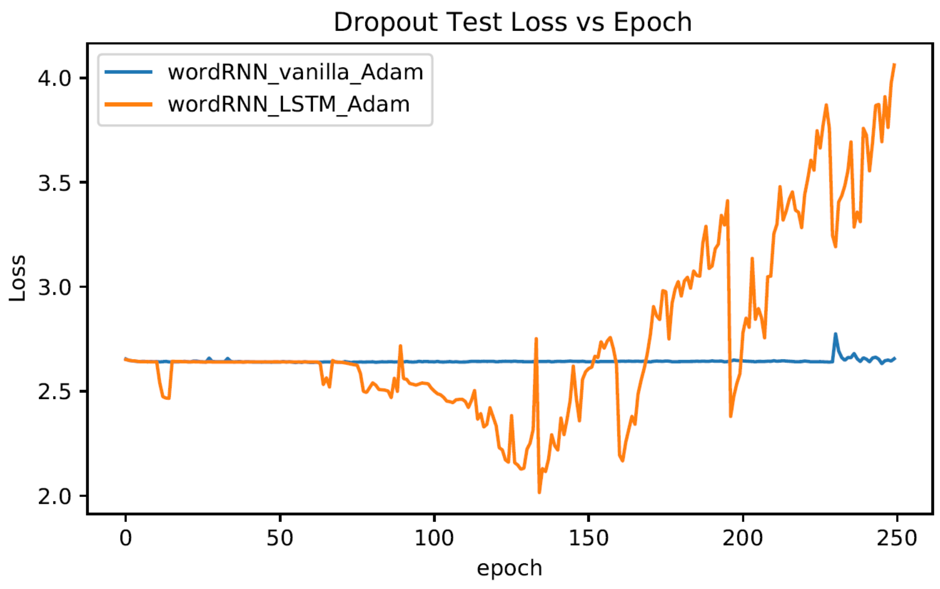
After training the Character RNN Classifier with the LSTM cell and Vanilla RNN cell, we plot the last epoch test accuracy in the table above. As can be seen from the table, the Character RNN Classifier with the Vanilla RNN cell produces poor test accuracy, while the Character RNN Classifier with the LSTM cell produces similar test accuracy as the Character RNN Classifier with the GRU cell at about 0.7 test accuracy.

## Word RNN Classifier

For this experiment, for the Word RNN Classifiers, we used an Adam optimizer for the models, with a dropout layer of dropout rate 0.5. The RNN layer will have the GRU cell replaced by the vanilla RNN cell, as well as the LSTM cell and run for 250 epochs.



**Word RNN Classifier Test Accuracies vs. Epochs**



**Word RNN Classifier Test Loss vs. Epochs**

From the Test Accuracies vs. epochs graph, we can see that the Word RNN Classifier with the LSTM cell in the RNN layer has better test accuracy as compared to the  Word RNN Classifier with the vanilla RNN cell in the RNN layer. The Word RNN Classifier vanilla RNN cell in the RNN layer has a stagnant test accuracy of around 0.07 for all 250 training epochs. However, we can observe that the Word RNN Classifier with the LSTM cell in the RNN layer may not be fitting well as the test loss is increasing with every epoch.

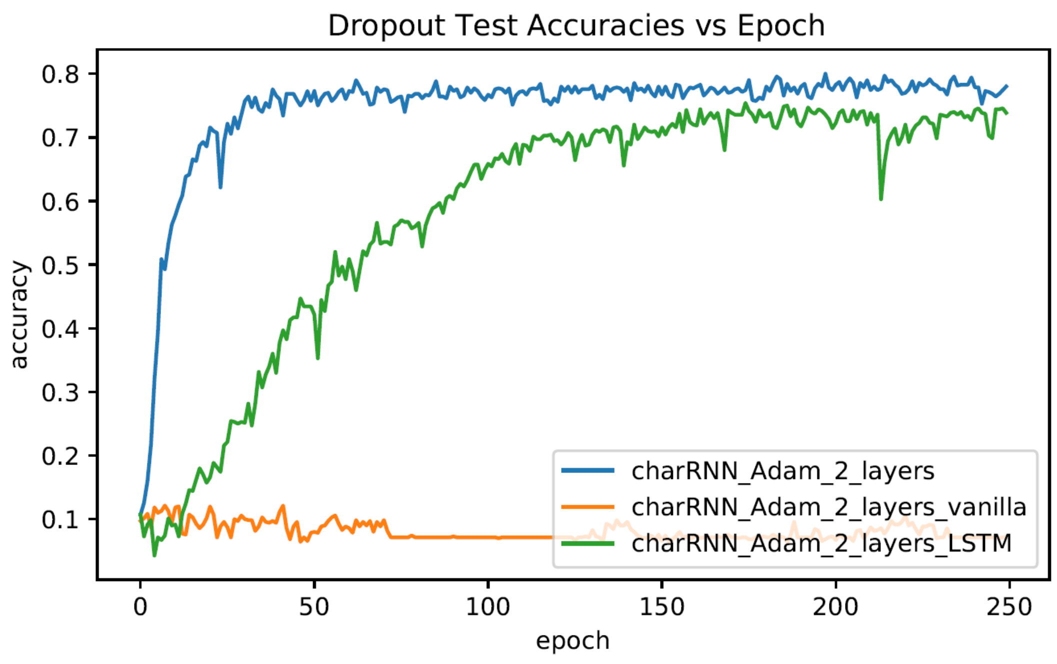
| Word RNN Classifier | **Last Epoch Test Accuracy** |
|---|---|
| GRU Layer | 0.87857 |
| Vanilla RNN Layer | 0.08000 |
| LSTM Layer | 0.43571 |

After training the Word RNN Classifier with the LSTM cell and Vanilla RNN cell, we plot the last epoch test accuracy in the table above. As can be seen from the table, the Word RNN Classifier with the Vanilla RNN cell produces poor test accuracy, while the Word RNN Classifier with the LSTM cell only manages to achieve a test accuracy of 0.43571 after 250 epochs which is quite a low test accuracy as compared to the test accuracy obtained by the Word RNN Classifier with the GRU cell with a test accuracy of 0.87857 after 250 epochs.

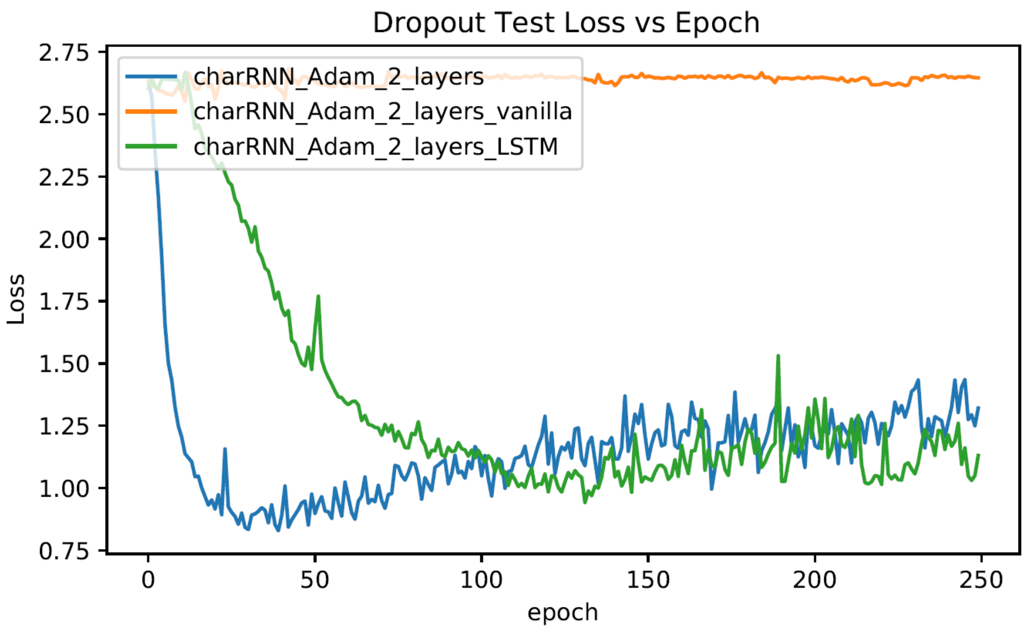
## Part B (b. Increase the number of RNN layers to 2 layers)

### Character RNN Classifier

For this experiment, for the Character RNN Classifiers, we used an Adam optimizer for the models, with a dropout layer of dropout rate 0.5. The 2 RNN layers will be implemented with the GRU cell, the vanilla RNN cell, as well as the LSTM cell and run for 250 epochs.



**Character RNN Classifier Test Accuracies vs. Epochs**



**Character RNN Classifier Test Loss vs. Epochs**

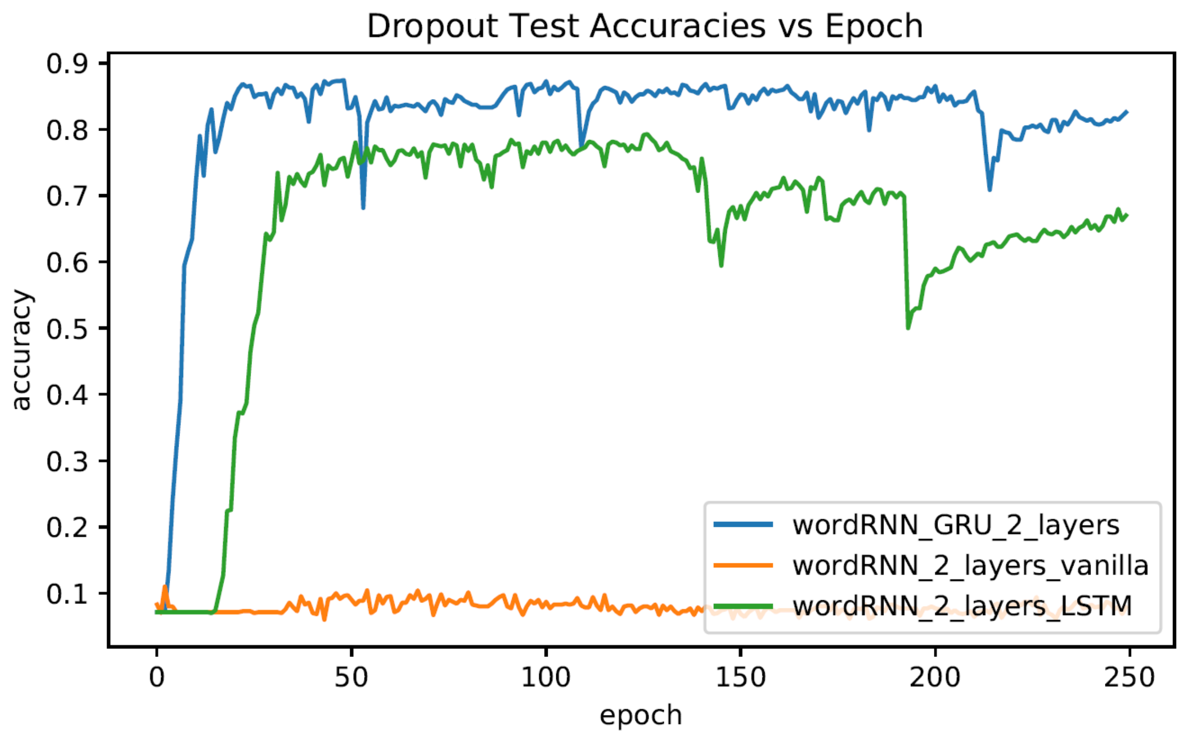
From the Test Accuracies vs. Epochs graph above, we can observe that for a Character RNN Classifier with 2 Layers, using GRU cells produces better test accuracies as compared to using Vanilla RNN cells as well as LSTM cells. The test accuracy for the Character RNN Classifier with 2 Layers, using Vanilla RNN cells has a stagnant test accuracy of around 0.07 for most of the training epochs, while the Character RNN Classifier with 2 Layers, using LSTM cells converges with a test accuracy of around 0.73 and the Character RNN Classifier with 2 Layers, using GRU cells converges with a test accuracy of around 0.77.

| Character RNN Classifier | **Last Epoch Test Accuracy** |
|---|---|
| GRU 2 Layers | 0.78000 |
| Vanilla RNN 2 Layers | 0.07143 |
| LSTM 2 Layers | 0.73857 |

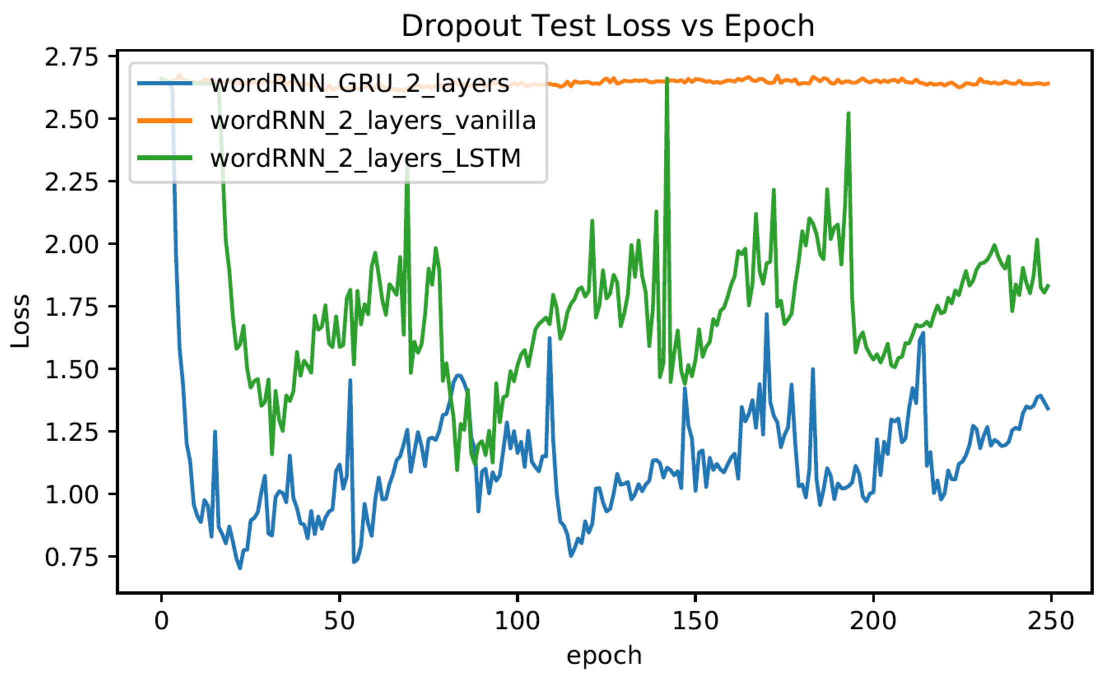
After training the Character RNN Classifier with the GRU cell, LSTM cell and Vanilla RNN cell, we plot the last epoch test accuracy in the table above. As can be seen from the table above, using 2 layers, the Character RNN Classifier with the GRU cell performs the best out of the 3 types of RNN cells that were used for the experiment, with the highest last epoch test accuracy of 0.78000.

## Word RNN Classifier

For this experiment, for the Word RNN Classifiers, we used an Adam optimizer for the models, with a dropout layer of dropout rate 0.5. The 2 RNN layers will be implemented with the GRU cell, the vanilla RNN cell, as well as the LSTM cell and run for 250 epochs.



**Word RNN Classifier Test Accuracies vs. Epochs**



**Word RNN Classifier Test Loss vs. Epochs**

From the Test Accuracies vs. Epochs graph above, we can observe that for a Word RNN Classifier with 2 Layers, using GRU cells produces better test accuracies as compared to using Vanilla RNN cells as well as LSTM cells. The test accuracy for the Word RNN Classifier with 2 Layers, using Vanilla RNN cells has a stagnant test accuracy of around 0.07 for most of the training epochs, while the Word RNN Classifier with 2 Layers, using LSTM cells converges with a test accuracy of around 0.67 and the Word RNN Classifier with 2 Layers, using GRU cells converges with a test accuracy of around 0.82.

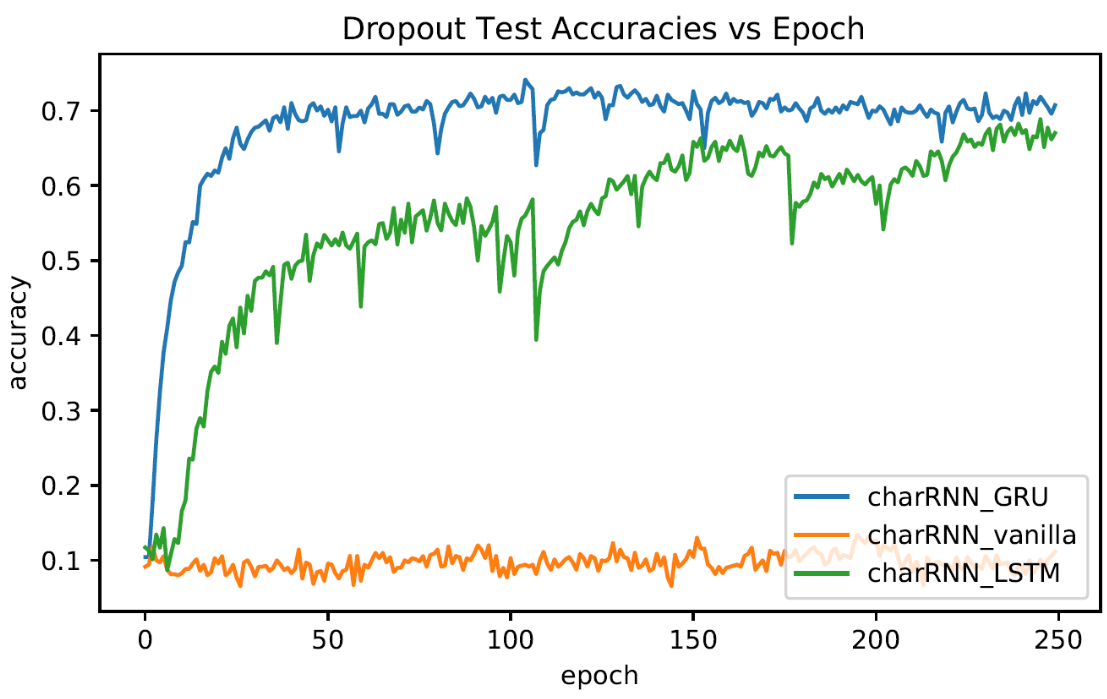
| Word RNN Classifier | **Last Epoch Test Accuracy** |
|---|---|
| GRU 2 Layers | 0.82571 |
| Vanilla RNN 2 Layers | 0.07857 |
| LSTM 2 Layers | 0.67000 |

After training the Word RNN Classifier with the GRU cell, LSTM cell and Vanilla RNN cell, we plot the last epoch test accuracy in the table above. As can be seen from the table above, using 2 layers, the Word RNN Classifier with the GRU cell performs the best out of the 3 types of RNN cells that were used for the experiment, with the highest last epoch test accuracy of 0.82571.

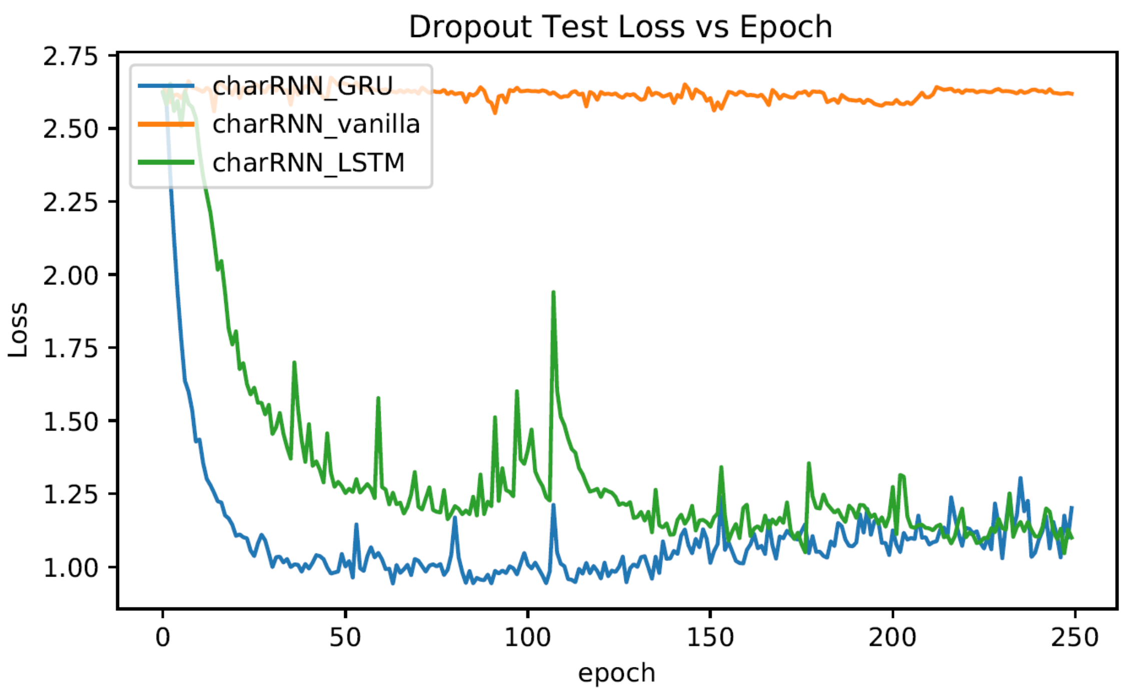
## Part C (c. Add gradient clipping to RNN training with clipping threshold = 2)

### Character RNN Classifier

For this experiment, for the Character RNN Classifiers, we used an Adam optimizer for the models, with a dropout layer of dropout rate 0.5. The RNN layer will be implemented with the GRU cell, the vanilla RNN cell, as well as the LSTM cell and run for 250 epochs. We will use a gradient clipping of threshold = 2 for all the models.



**Character RNN Classifier Test Accuracies vs. Epochs**



**Character RNN Classifier Test Loss vs. Epochs**

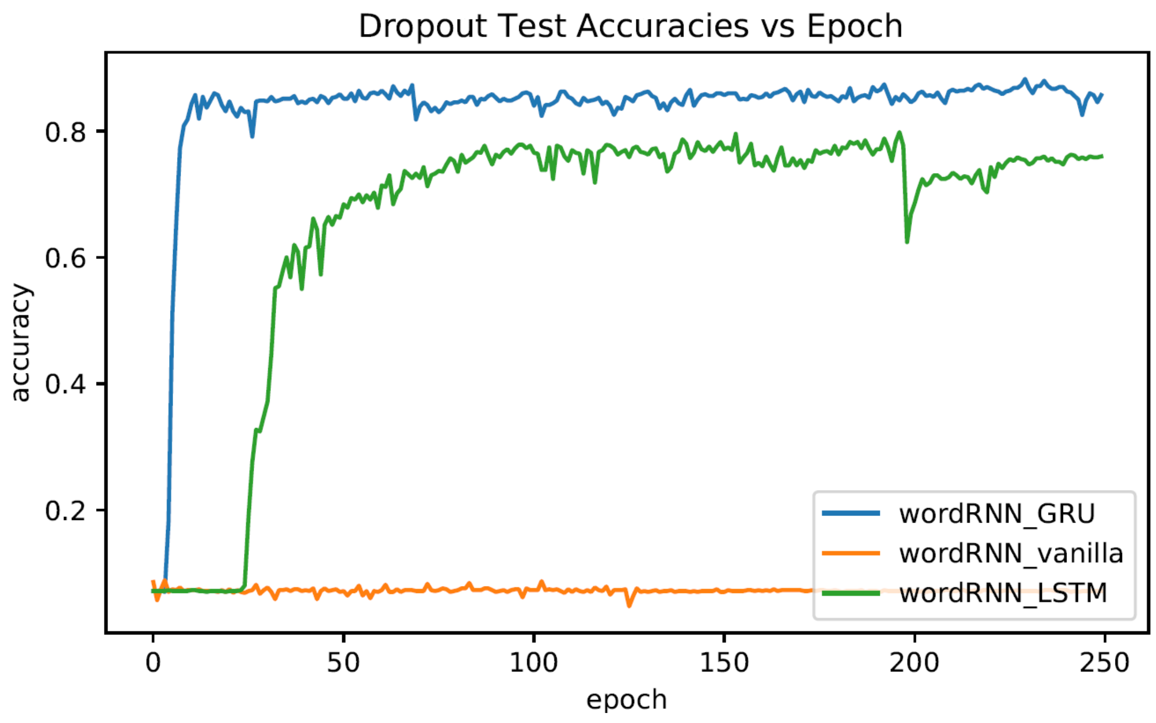
From the Test Accuracies vs. Epochs graph above, we can observe that for a Character RNN Classifier with gradient clipping, using a GRU cell produces better test accuracies as compared to using a Vanilla RNN cell as well as LSTM cell. The test accuracy for the Character RNN Classifier with gradient clipping, using Vanilla RNN cell has a stagnant test accuracy of around 0.10 for most of the training epochs, while the Character RNN Classifier with gradient clipping, using LSTM cell converges with a test accuracy of around 0.67 and the Character RNN Classifier with gradient clipping, using GRU cells converges with a test accuracy of around 0.70.

| Character RNN Classifier | **Last Epoch Test Accuracy** |
|---|---|
| GRU Layer | 0.70714 |
| Vanilla RNN Layer | 0.11143 |
| LSTM Layer | 0.67000 |

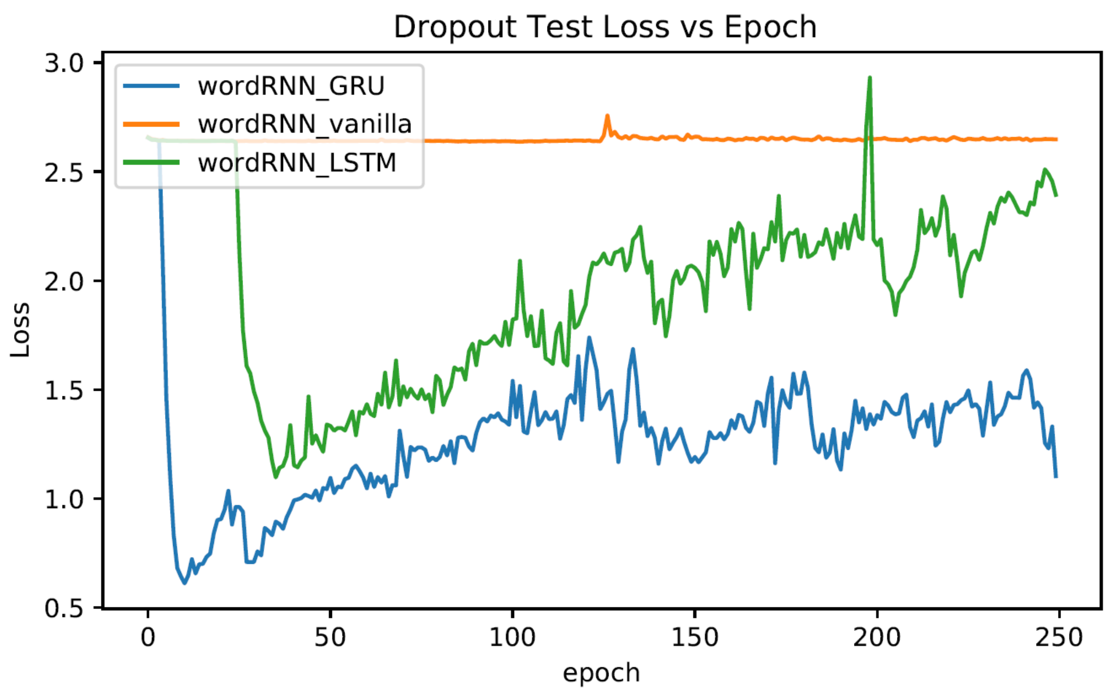
After training the Character RNN Classifier with the GRU cell, LSTM cell and Vanilla RNN cell, we plot the last epoch test accuracy in the table above. As can be seen from the table above, using gradient clipping, the Character RNN Classifier with the GRU cell performs the best out of the 3 types of RNN cells that were used for the experiment, with the highest last epoch test accuracy of 0.70714.

## Word RNN Classifier

For this experiment, for the Word RNN Classifiers, we used an Adam optimizer for the models, with a dropout layer of dropout rate 0.5. The RNN layer will be implemented with the GRU cell, the vanilla RNN cell, as well as the LSTM cell and run for 250 epochs. We will use a gradient clipping of threshold = 2 for all the models.



**Word RNN Classifier Test Accuracies vs. Epochs**



**Word RNN Classifier Test Loss vs. Epochs**

From the Test Accuracies vs. Epochs graph above, we can observe that for a Word RNN Classifier with gradient clipping, using a GRU cell produces better test accuracies as compared to using a Vanilla RNN cell as well as LSTM cell. The test accuracy for the Word RNN Classifier with gradient clipping, using Vanilla RNN cell has a stagnant test accuracy of around 0.07 for most of the training epochs, while the Word RNN Classifier with gradient clipping, using LSTM cell converges with a test accuracy of around 0.76 and the Word RNN Classifier with gradient clipping, using GRU cells converges with a test accuracy of around 0.85.

| Word RNN Classifier | **Last Epoch Test Accuracy** |
|---|---|
| GRU Layer | 0.85714 |
| Vanilla RNN Layer | 0.07143 |
| LSTM Layer | 0.76000 |

After training the Word RNN Classifier with the GRU cell, LSTM cell and Vanilla RNN cell, we plot the last epoch test accuracy in the table above. As can be seen from the table above, using gradient clipping, the Word RNN Classifier with the GRU cell performs the best out of the 3 types of RNN cells that were used for the experiment, with the highest last epoch test accuracy of 0.85714.

## Conclusions

From the results obtained in Question 6, we can observe that if we were to use RNN Classifiers for Text Classification, using a GRU RNN layer or a LSTM Layer may be more suitable for the task as compared to using a Vanilla RNN Layer. For Both Character and Word RNN Classification, the RNN Classifiers that use a Vanilla RNN Layer perform quite badly in terms of test accuracy. The Vanilla RNN network seems to not be able to capture the features and patterns for text classification, with the test accuracy stagnant at a low accuracy rate for all of the 250 training epochs.

Next, we can also observe from the experiments that the best RNN type to use for Text Classification is the GRU RNN. It performs better than both the LSTM RNN and Vanilla RNN for both Character and Word RNN Classification.

Also, we can observe that using a GRU RNN implementation, for Character RNN Classifiers, using a 2-layer RNN network improves the test accuracy to 0.78000 from 0.70286 while using gradient clipping produces similar test accuracy as that of the baseline model with a test accuracy of 0.70714. Using a GRU RNN implementation, for Word RNN Classifiers, the baseline model produces the best test accuracy at 0.87857 and the model with 2 RNN layers producing a test accuracy of 0.82571 and the model with gradient clipping producing a test accuracy of 0.85714.