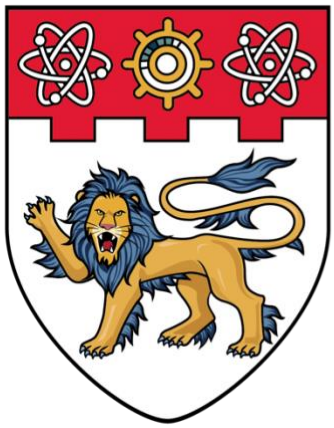


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CE/CZ4042: Neural Networks and Deep Learning

Project 2

Name – Genevieve Lam

Matric No. – U1521863H

Part 1: Convolutional Neural Network on images

The goal of this part of the project is to use deep Convolutional Neural Network(CNN) for object recognition in images. We experimented with different CNN Architectures, to find the best performing model (in terms of accuracy). Using training error and test accuracy as measures of model performance. Training errors are computed using the entropy cost of the training data.

1.1. CNN Architecture

The architecture consist of an input layer of 3x32x32 dimensions, a convolution layer $C1$ of 50 filters of window size 9x9, VALID padding, and ReLU neurons, a max pooling layer $S1$ with a pooling window of size 2x2, with stride = 2 and padding = 'VALID', a convolution layer $C2$ of 60 filters of window size 5x5, VALID padding, and ReLU neurons. A max pooling layer $S2$with a pooling window of size 2x2, with stride = 2 and padding = 'VALID', a fully connected layer $F3$ of size 300, a softmax layer $F4$ of size 10. The architecture is trained using mini-batch gradient descent learning of batch size, 128, and learning rate α = 0.001 with 500 epoch. We did data normalized, because the input values are well understood, we can easily normalize to the range 0 to 1 by dividing each value by the maximum observation which is 255. As shown the model did not perform very well in both aspects. Hence hyperparameter tuning is needed to optimise the results.

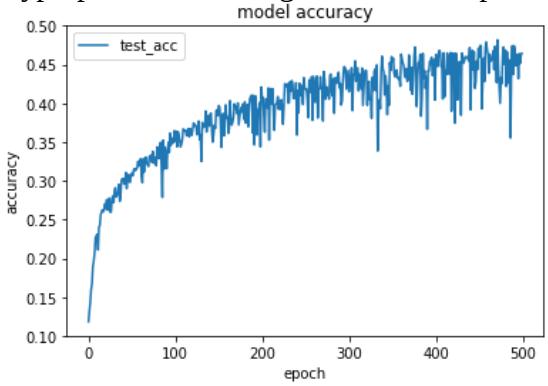


Fig 1: Test accuracy

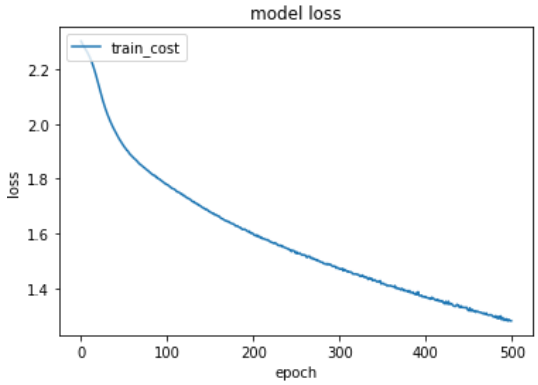
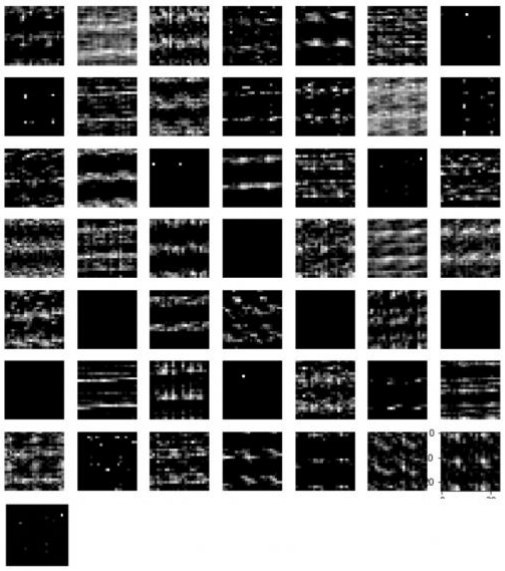


Fig 2: Training cost

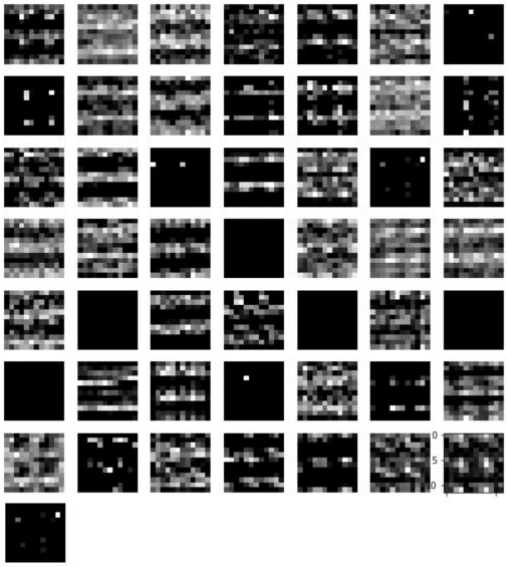
As shown in the figure 4, $C1$ and $S1$ have a feature map of 50, and both $C2$ and $S2$ have a feature map of 60.

Fig 3: Test patterns 1

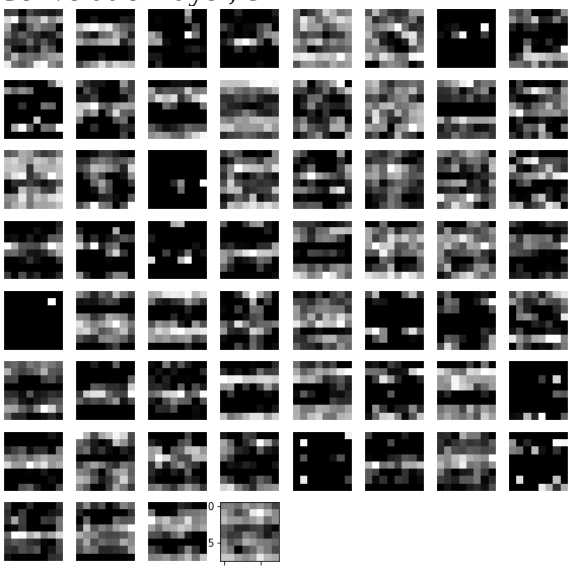
Convolution layer, $C1$



Pooling layer, $S1$



Convolution layer, $C2$



Pooling layer, $S2$

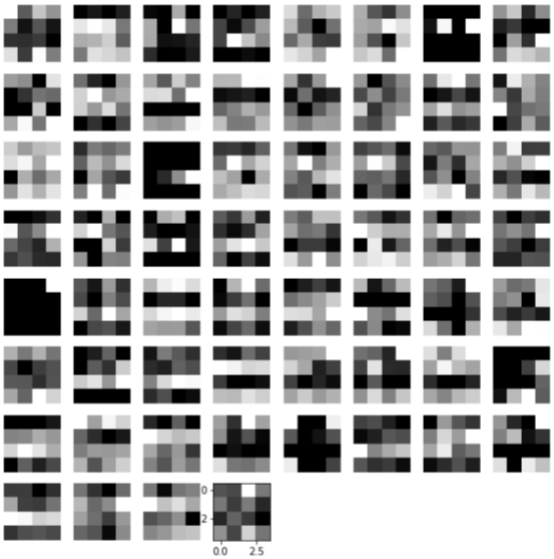
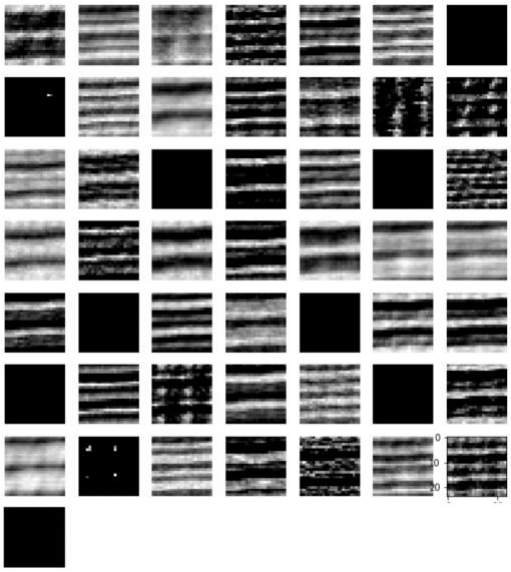


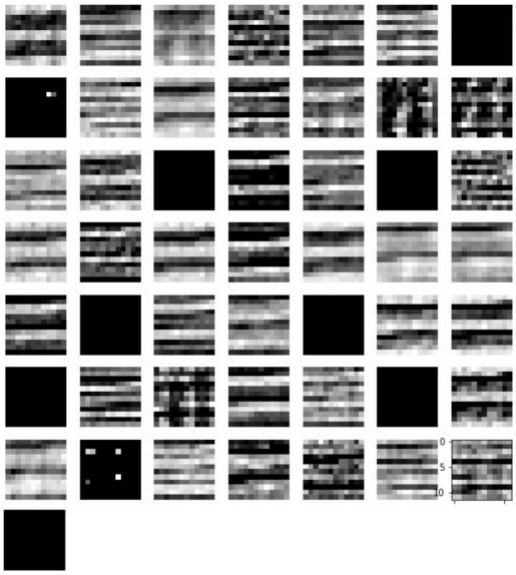
Fig 4: Test patterns 1's feature maps

Fig 5: Test patterns 2

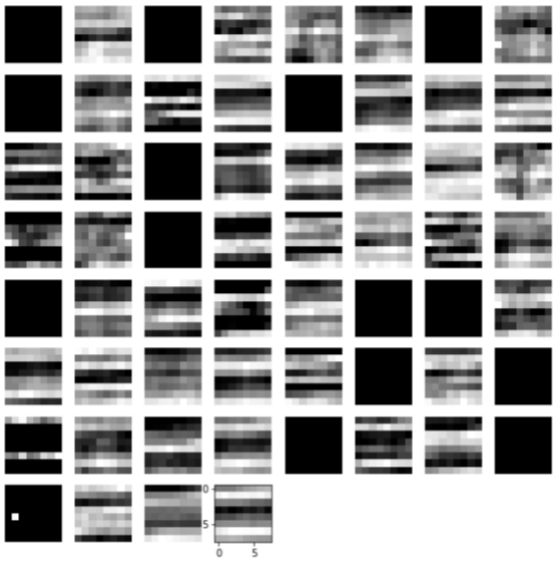
Convolution layer, $C1$



Pooling layer, $S1$



Convolution layer, $C2$



Pooling layer, $S2$

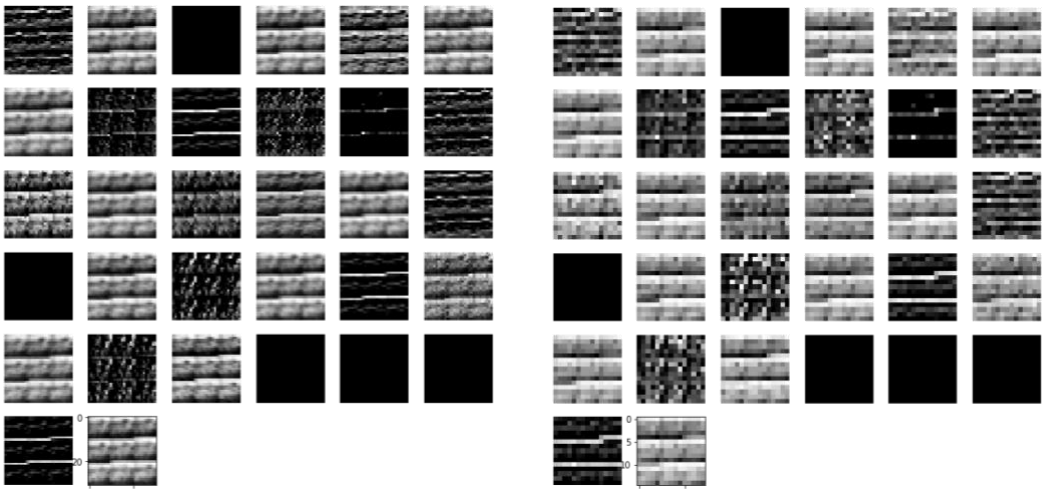
Fig 6: Test patterns 2's feature maps

1.2. Hyperparameters and Feature maps

Grid search was ran with the following hyperparameters: number of convolution layer filters {32,64,128}, window sizes {2,3,5}, number of epochs {100,200,500}.The best performing (purely on accuracy) model consist of an input layer of 3x32x32 dimensions with , a convolution layer $C1$ of 32 filters of window size 2x2, VALID padding, and ReLU neurons, a max pooling layer $S1$ with a pooling window of size 2x2, with stride of 2 and padding of 'VALID', a convolution layer $C2$ of 128 filters of window size 3x3, VALID padding, and ReLU neurons. A max pooling layer $S2$with a pooling window of size 2x2, with stride of 2 and padding of 'VALID', a fully connected layer $F3$ of size 300 and a softmax layer $F4$ of size 10. It is trained using mini-batch gradient descent learning of batch size, 128, and learning rate α = 0.001 and ran with 500 epochs.

The optimal number of feature map for $C1$ and $S1$ are 32 and $C2$ and $S2$ are 128. This model Obtained a test accuracy of 0.5032. Fig 7 feature maps uses fig 5 test patterns.

Convolution layer, $C1$ Pooling layer, $S1$



Convolution layer, $C2$ Pooling layer, $S2$

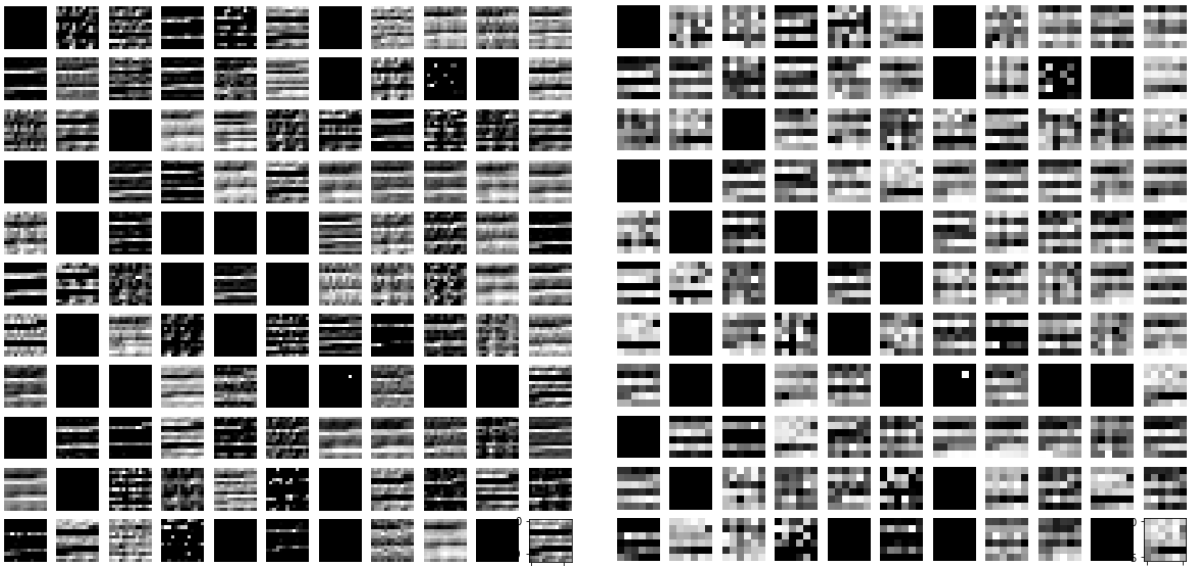


Fig 7: optimal number of feature maps

1.3.Experiments

1.3.1. Momentum

The model used to obtain Fig 8 , 9 are produced by using the same model as Section 1.2, adding the momentum term with $\gamma = 0.1$. Adding momentum to the gradient decent learning is used to obtain faster convergence (fig 9). momentum method helps the parameter vector to build up velocity in any direction with constant gradient descent so as to prevent oscillations. As shown below the model's loss is a lot lower than the initial model(0.6 , 1.8 respectively). And the accuracy was slightly better than the optimised model in section 1.2. As shown in the Table 1.

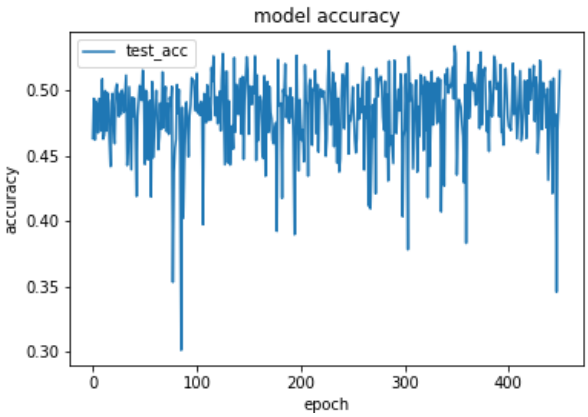


Fig 8: Test accuracy (momentum)

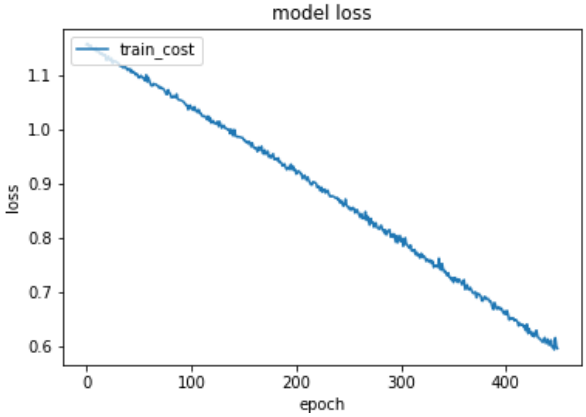


Fig 9: Training cost (momentum)

1.3.2. RMSProp algorithm

The model used to obtain Fig 10 , 11 are produced by using the same model (Section 1.2) and learning rate, differences are using RMSProp algorithm for learning instead of gradient descent learning and running for 90 epoch. Due to the sparse data an adaptive gradient descent algorithms would be preferred over classical gradient descent learning. As the per-parameter learning rate method provide heuristic approach without requiring expensive work in tuning hyperparameters for the learning rate schedule manually. As seen in our experiment. RMSProp algorithm is type of adaptive gradient descent algorithms. RMSProp algorithm outperforms the other model in terms of training loss as and outperforms the classical shown gradient descent learning shown in Table 1.

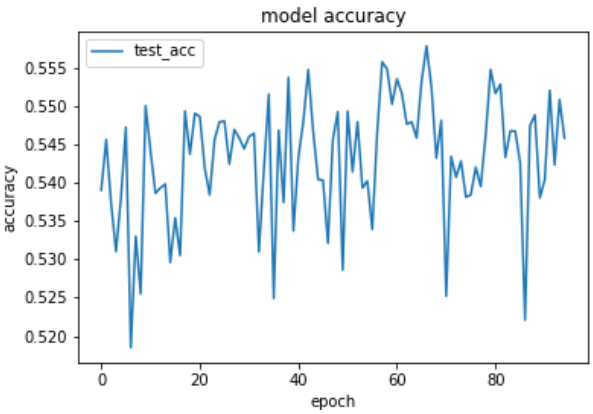


Fig 10: Test accuracy (RMSProp)

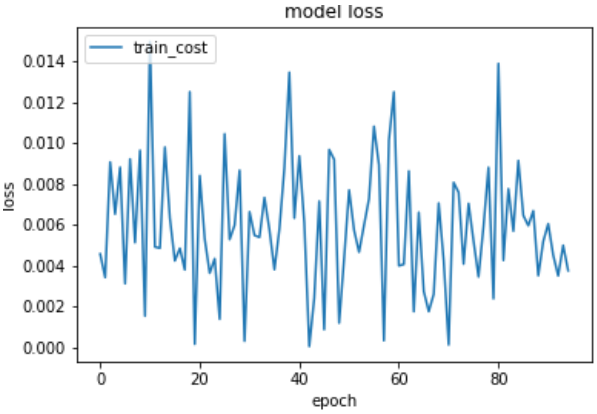


Fig 11: Training cost (RMSProp)

1.3.3. Adam optimizer

The model used to obtain Fig 12 , 13 is produced by using the model (Section 1.2), and learning rate , the differences are using Adam optimizer for learning instead of gradient descent learning and running for 40 epochs. Adam optimizer is also a type of adaptive gradient descent algorithms. Hence, it also outperforms the classical model. Adam is an update to the RMSProp optimizer which is like RMSprop with momentum. As seen in the previous experiment (section 1.3.1), by adding momentum to the learning, it obtain faster convergence. In the graph below, we can see that within the first few epochs Adam has performed better than RMSprop. This also explains why Adam optimizer out performs the other optimisers.

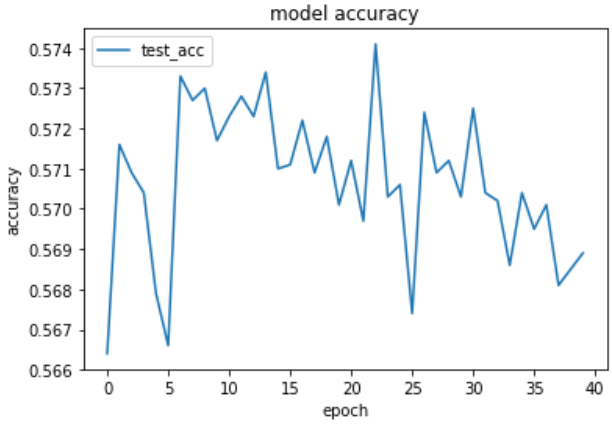


Fig 12: Test accuracy (Adam)

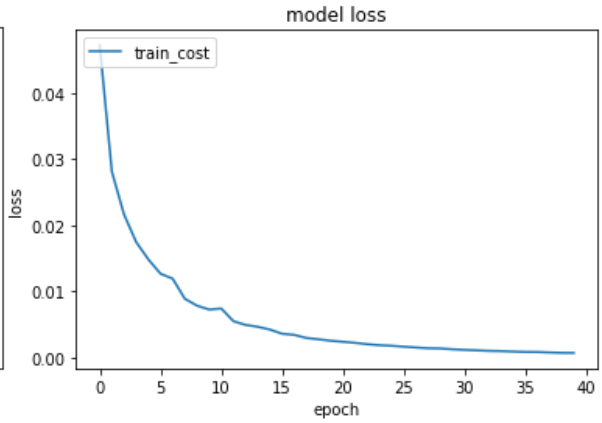


Fig 13: Training cost (Adam)

1.3.4. Dropouts

The model used to obtain Fig 14,15 is produced by using the model(Section 1.2) and learning rate, with dropouts of 0.3 in between convolutional layers, Adam optimizer for learning instead of gradient descent learning and run with 40 epoch. Dropout technique is essentially a regularization method used to prevent over-fitting while training neural nets. The role of hidden units in neural networks is to approximate a 'function' efficiently from the available data-samples which can be generalized to unseen data. Hence allowing the model to learn better and also making it the best performing model. It is the best performing model with the highest accuracy and the lowest loss.

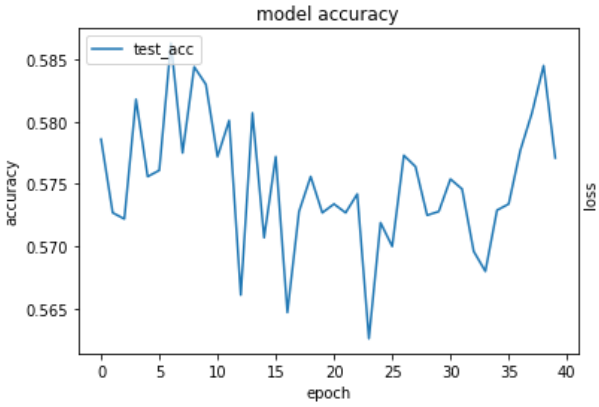


Fig 14: Test accuracy (Dropout)

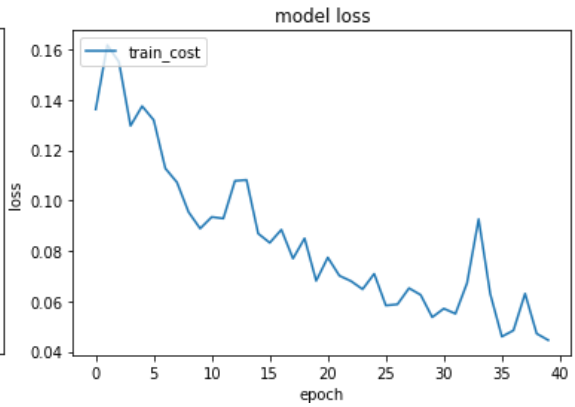


Fig 15: Training cost (Dropout)

1.4. CNN results

RMSProp has the lowest training error (0.004), this may be a sign of overfitting where validation error is high, training error low. The baseline model has the highest training error 1.8. This maybe a sign of underfitting, where validation and training error high. The dropout model did the best (0.58) and also training error (0.04). As mention above, dropout is a regularization method used to prevent over-fitting. Add dropouts helps reduce the error rate and tremendously and improves accuracy performance.

Adam optimiser, was the best performing gradient descent algorithms (in terms of accuracy). With the least amount of epochs required to obtain the highest test accuracy. As it is a type of adaptive gradient descent algorithms.

Table 1: CNN results

Model	Test Accuracy
Q1 CNN (baseline)	0.46
Q2 CNN(optimal numbers of feature maps)	0.50
Q3a CNN (momentum)	0.51
Q3b CNN (RMSProp)	0.55
Q3c CNN (Adam)	0.57
Q3d CNN (Dropouts)	**0.58**

1.5. Conclusion

In conclusion, the best performing model would be a model with Adam optimiser with dropouts. Another, consideration to improve performance is by performing data augmentation. This way it increases the amount of data, we can experiment it with larger models, and also improves performance as whole.

Part 2: Text classification

The goal of this part of the project is to experiment with CNN and Recurrent Neural Networks (RNN) models and with either word or character input. In order to determine the best performing model, test accuracy of each model are taken to obtain the best model. The training errors are computed for all the model using the entropy cost of the training data.

2.1. Character CNN Classifier

The character CNN classifier comprises of the following: a convolution layer $C1$ of 10 filters of window size 20x256, VALID padding, and ReLU neurons. A max pooling layer $S1$ with a pooling window of size 4x4, with stride = 2, and padding = 'SAME' and a convolution layer $C2$ of 10 filters of window size 20x1, VALID padding, and ReLU neurons. A max pooling layer $S2$ with a pooling window of size 4x4, with stride = 2 and padding = 'SAME'. It uses the Adam optimizer for training with a batch size = 128 and learning rate = 0.01.

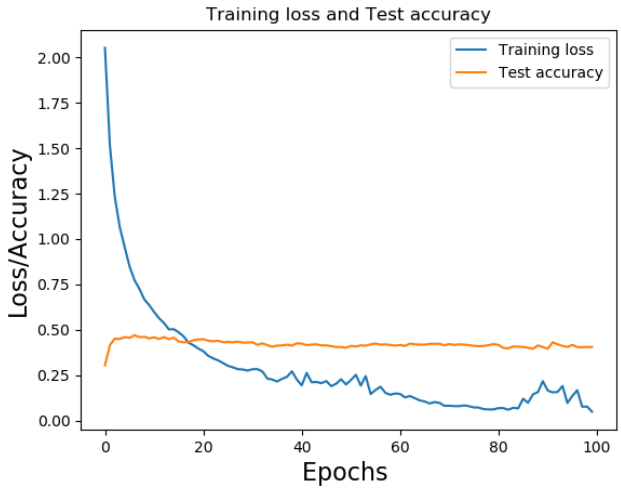


Fig 16: CNN (Character)

2.2. Word CNN Classifier

The Word CNN Classifier consist of the following, a convolution layer $C1$ of 10 filters of window size 20x20, VALID padding, and ReLU neurons. A max pooling layer $S1$ with a pooling window of size 4x4, with stride = 2 and padding = 'SAME', a convolution layer $C2$ of 10 filters of window size 20x1, VALID padding, and ReLU neurons. A max pooling layer $S2$ with a pooling window of size 4x4, with stride = 2 and padding = 'SAME'. The inputs are pass through an embedding layer of size 20 to the model for training and uses the Adam optimizer for training with a batch size = 128 and learning rate = 0.01 and early stopping was done manually to prevent overfitting.

In NLP filters are slide over full rows of the matrix (words). ConvNets do not require the knowledge about the syntactic or semantic structure of a language. This simplify the engineering. This helps it to find strong local clues regarding class (word) membership, as these clues can appear in different places in the input. It also eliminates the vanishing gradients which is problematic in RNNs. Hence helping it to obtain high model performance.

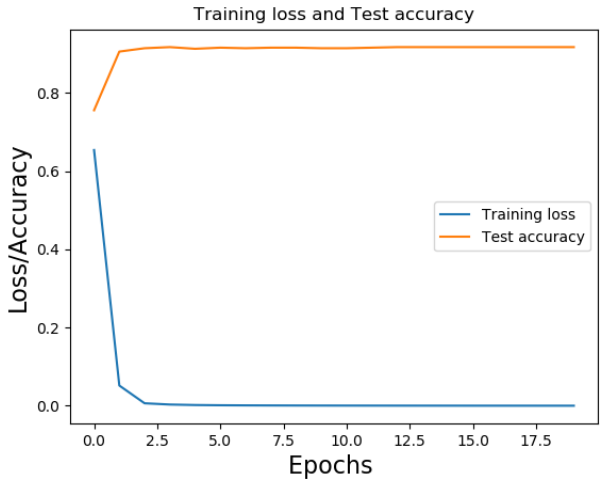


Fig 17: CNN (Word)

2.3. Character RNN Classifier

The RNN model comprises of a Gated Recurrent Unit (GRU) layer with a hidden-layer size of 20. It uses the Adam optimizer for training with a batch size = 128 and learning rate = 0.01 and run for 100 epochs.

Fig 18: RNN (Character)

2.4. Word RNN classifier

The RNN model comprises of GRU layer and has a hidden-layer size of 20. The inputs are passed through an embedding layer of size 20 before feeding to the RNN. It uses the Adam optimizer for training with a batch size = 128 and learning rate = 0.01, ran for 20 epochs and early stopping was done manually to prevent overfitting.

GRU aims to solve the vanishing gradient problem which comes with a standard recurrent neural network (Vanilla). It is able to encapsulate middle and long term sequential dependencies, unlike traditional neural networks. And allowing previous information to be preserved. As we know that current word/character are related to the previous and next word/character.

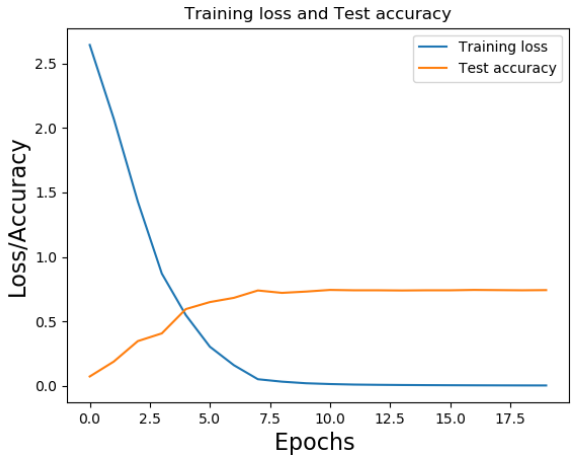


Fig 19: RNN (Word)

2.5. Model run time & test accuracy

As shown in table 3, out of the 4 models the word CNN Classifier performs the best. Convolutional Filters learn good representations automatically, without needing to represent the whole vocabulary. And character RNN model outperformed character CNN Classifier for classification. As it is able to encapsulate middle and long term sequential dependencies.

It is safe to say that word-level inputs generally outperforms character-level. This shows that both models underperforms classification of text to categorises. Due to the following reasons: simpler models and a small datasets (hundreds of thousands of examples).

In terms of run time CNN has proven to be the faster than RNN (Table 2). It is known that RNN is not hardware friendly and takes a lot of resources. That is the opposite of CNN. CNNs are also efficient in terms of representation. Furthermore, convolutional Filters learn good representations automatically, without needing to represent the whole vocabulary.

Table 2: Running time results

Model	Epochs	Time(s)
CNN (Character)	100	885 (177)
CNN (Word)	20	26
RNN (Character)	100	908 (181.6)
RNN (Word)	20	39

2.6. Classifier with dropouts

As mention in section 1.3.4. dropouts is a regularization method used to prevent over-fitting. This allows the model to learn more meaningful insights. As shown in table 3, most of the model's accuracy improved. Especially for RNN with word embedding. Furthermore, CNN (Word) and RNN (Word) from Fig 16,19 the training error was 0 after a few epoch. This implies the likelihood of the model to overfit. Hence, adding dropouts has solve this problem for overfitting, allowing it to obtain higher accuracy as shown in table 3.

2.7. Experimental models

2.7.1 Vanilla RNN model

The character RNN classifier model comprises of a vanilla RNN layer with a hidden-layer size of 20. And the word RNN classifier model comprises of a Vanilla RNN layer and has a hidden-layer size of 20. It uses the Adam optimizer for training with a batch size = 128 and learning rate = 0.0, ran for 20 epochs and early stopping was done manually. The inputs are passed through an embedding layer of size 20 before feeding to the RNN.

Vanilla RNN models fails to encapsulate middle and long term sequential dependencies. It has very high training error that will result in exponential gradients to flow through the network (Fig 20, 21). This shows that it has vanishing gradient issues. Resulting to the lowest performing models (Table 3).

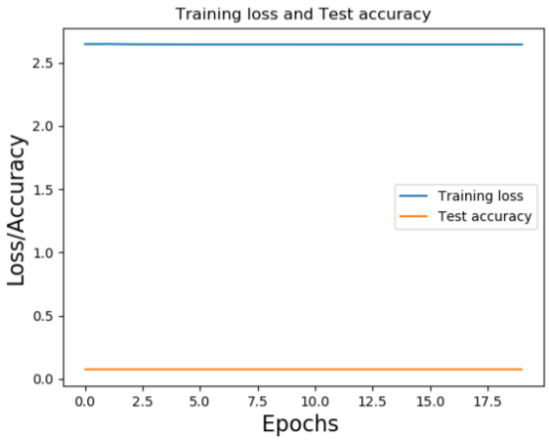


Fig 20: Vanilla RNN (Character)

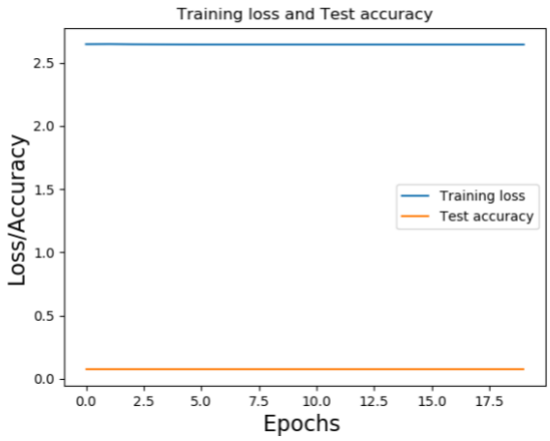


Fig 21: Vanilla RNN (Word)

2.7.2. LSTM model

The character RNN classifier model comprises of a LSTM layer with a hidden-layer size of 20. And the word RNN classifier model comprises of a LSTM layer and has a hidden-layer size of 20. It uses the Adam optimizer for training with a batch size = 128 and learning rate = 0.01, and ran for 20 epochs for the charater classifier and 90 for word classifier models. The inputs of the word classifier model are passed through an embedding layer of size 20 before feeding to the LSTM model.

Long-term information has to sequentially travel through all cells before getting to the present processing cell. This means it can be easily corrupted by being multiplied many time by small numbers < 0. This is the cause of vanishing gradients. LSTM only solves some of those problems by using multiple switch gates, and thus remember for longer time steps. However, there is still has tendency in vanishing gradients. As shown in Fig 22, with the character RNN classifier model, however not with the word RNN classifier model of Fig 23.

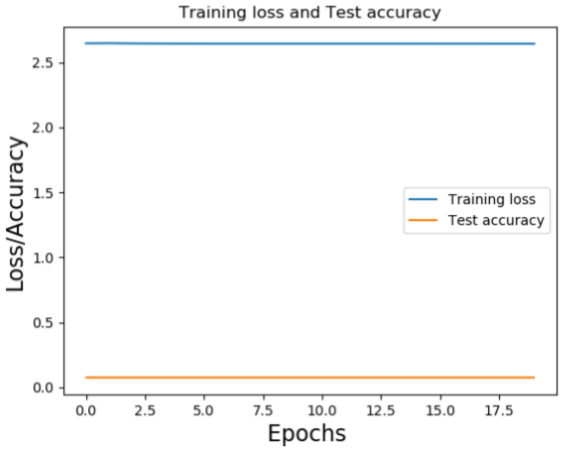


Fig 22: LSTM (Character)

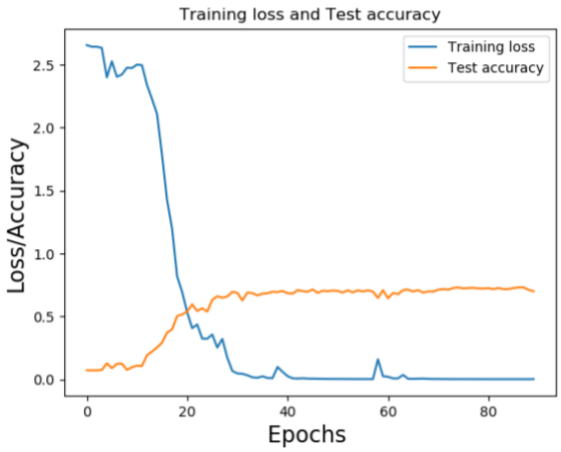


Fig 23: LSTM (Word)

2.7.3. Two GRU model

The character RNN classifier model comprises of 2 GRU layers with hidden-layer size of 20. It uses the Adam optimizer for training with a batch size = 128 and learning rate = 0.01 and ran for 100 epochs. And the word RNN classifier model comprises of 2 GRU layers with hidden-layer size of 20. The inputs are passed through an embedding layer of size 20 before feeding to the RNN. It uses the Adam optimizer for training with a batch size = 128 and learning rate = 0.01 and ran for 20 epochs.

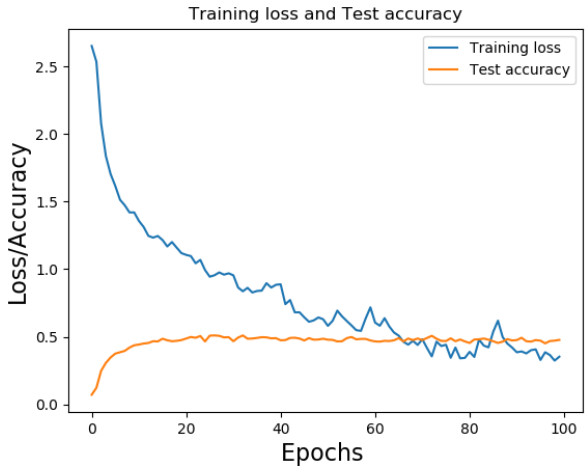


Fig 22: Two GRU (Character)

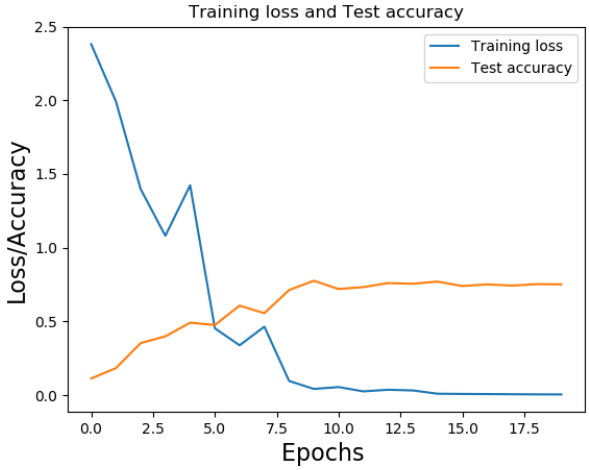


Fig 23: Two GRU (Word)

2.7.4. Gradient Clipping

The character RNN classifier model uses model from section 2.3 with an addition of gradient clipping. And the word RNN classifier model uses the model from section 2.4 with an addition of gradient clipping. The gradient clip ranges from -2 to 2 is applied to both models.

Gradient clipping basically helps in case of exploding or vanishing gradients. As seen in Fig 24, 25, it has reduce training error for both models section 2.3 and 2.4. This model is one of the best performing model among the RNN models (excluding models with dropout).

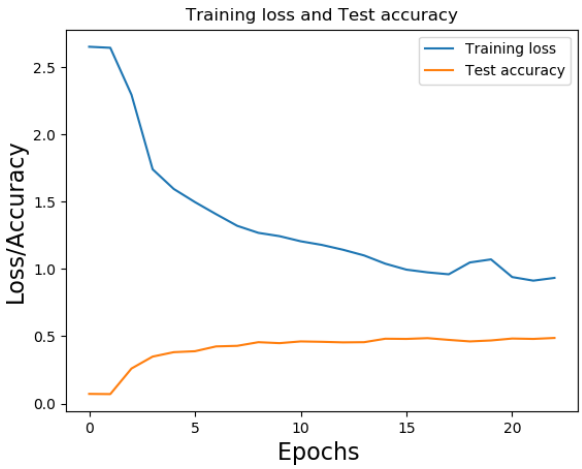


Fig 24: Gradient Clipping (Character)

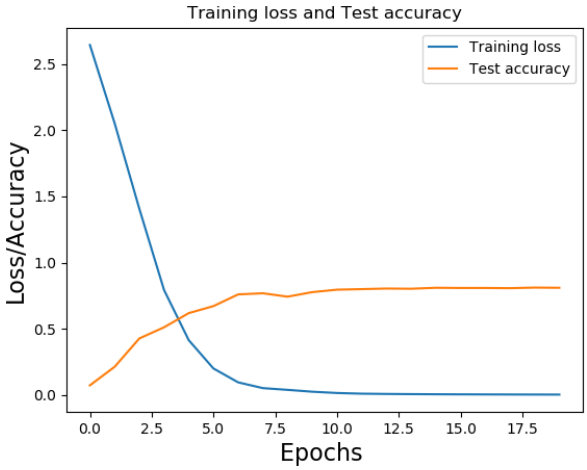


Fig 25: Gradient Clipping (Word)

2.7.5. Results

As seen in the table, word-level inputs generally outperforms character-level. In other research shows that CNN and RNN character-level input works very well on large datasets. However, this comes at the cost of requiring larger models that are slower to train.

The best performing RNN model is the GRU RNN models based on test accuracy. As it was able to overcome the vanishing gradients issue, where the other RNN models encounters. And out of all classifiers the word CNN classifier with performs the best (Table 3).

Table 3: Text classification results

Model	Test accuracy
CNN (Character)	0.41
CNN (Word)	**0.92**
GRU RNN (Character)	0.44
GRU RNN (Word)	0.74
CNN +Dropout (Character)	0.41
CNN + Dropout (Word)	**0.93**
GRU RNN + Dropout (Character)	0.46
GRU RNN + Dropout (Word)	0.89
Vanilla RNN(Character)	0.07
Vanilla RNN(Word)	0.08
LSTM(Character)	0.07
LSTM (Word)	0.73
RNN 2 (Character)	0.48
RNN 2 (Word)	0.75
Gradient Clipping(Character)	**0.49**
Gradient Clipping(Word)	**0.81**

2.8. Conclusion

In conclusion, the CNN word-level input with dropouts outperforms all the other models. As seen in our experiment (section 2.6, 2.7.4.), it is good practice to implement dropouts to prevent over-fitting and gradient clipping to prevent vanishing gradients. As seen in the table

3 it improves test accuracy too. In our models we use integer to encode, however it lose the ‘meaning’ of the word. Hence, Word2Vec is a more optimal way of encoding symbols to vector. This will result in an improve of test accuracy.