NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4042 Neural Network & Deep Learning

Assignment 1

Name: Loh Yi Xuan Renice

Matriculation No.: U1822247D

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

2021

Part A: Classification Problem

Introduction

For Part A of this assignment, we are required to build a 2-layer feedforward deep neural network (DNN) consisting of an input layer, a hidden layer of 16 neurons with ReLu activation function and an output softmax layer to classify the GTZAN dataset. The original dataset consisted of 1000 audio tracks, spanning 30 seconds each. There are 10 different genres in total. We will be using a pre-processed dataset where these audio tracks have been processed into features. The 30 seconds long audio files consist of 57 features. The aim is to predict the genre of the corresponding audio files in the test dataset after training the neural network on the training dataset. The genres are blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae and rock.

Model

The created model was of the architecture shown below.

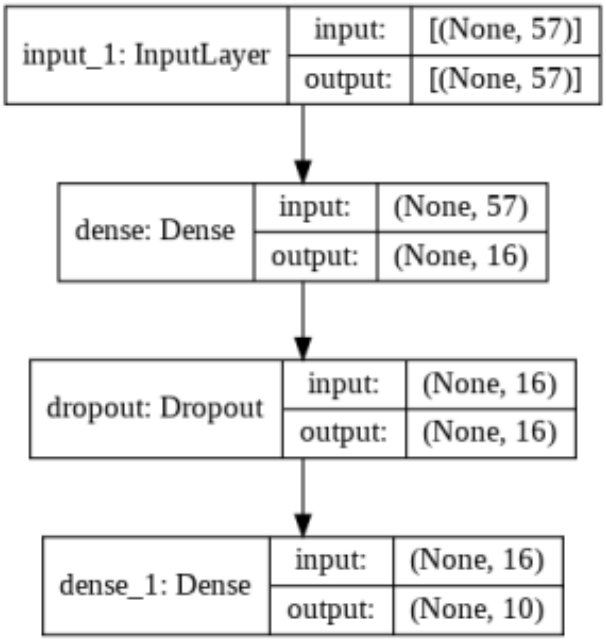

Figure 1

The dataset has 57 input features therefore the input layer consists of 57 neurons. Followed by a hidden layer with 16 hidden neurons with ReLU activation. Next, there is a dropout layer with dropout probability 0.3 and lastly, the output layer predicts the genre of the song.
Since, there are 10 genres in the dataset therefore the output layer consists of 10 neurons. Softmax activation function is used in the output layer because the model predicts the probability of each genre with highest probability considered as the actual predicted value for the song.

The predicted labels were in range (0-9) for genres which were converted in the data preprocessing stage to one hot label vector for increased accuracy. Mean Squared Error is used as a loss function in the model with Adam optimizer. Batch size used is 1 this means gradients are calculated for each song separately.

Question 1

a) Use the training dataset to train the model for 50 epochs. Note: Use 50 epochs for subsequent experiments.
b) Plot accuracies on training and test data against training epochs and comment on the plots

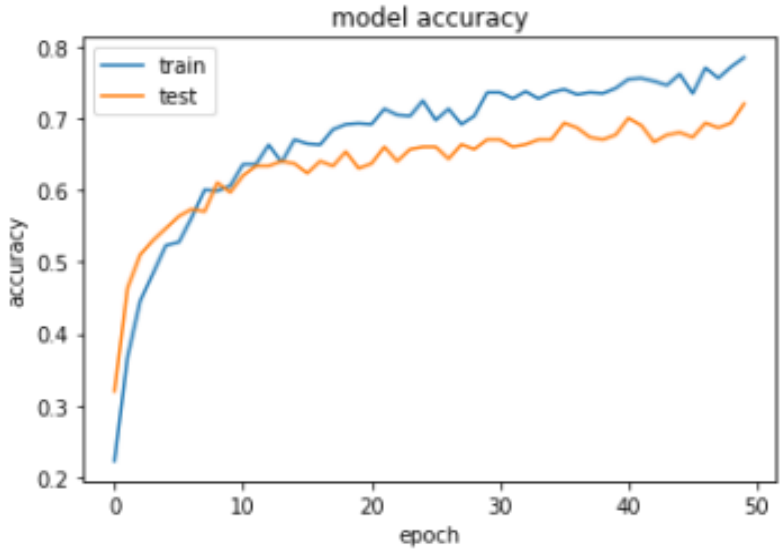

Figure 1

As seen in Figure 1, both test and train accuracy increases with each subsequent epoch. This indicates that the model is learning the features of the dataset and making accurate predictions. There is a sharp rise in accuracies in initial epochs because of high gradient

values. Initially, train error is less than test error but as the number of epochs increase, train accuracy is more than test accuracy. This can be attributed to the number of data - training data has more data than the test data which results in better learning.

c) **Plot the losses on training and test data against training epochs. State the approximate number of epochs where the test error begins to converge.**

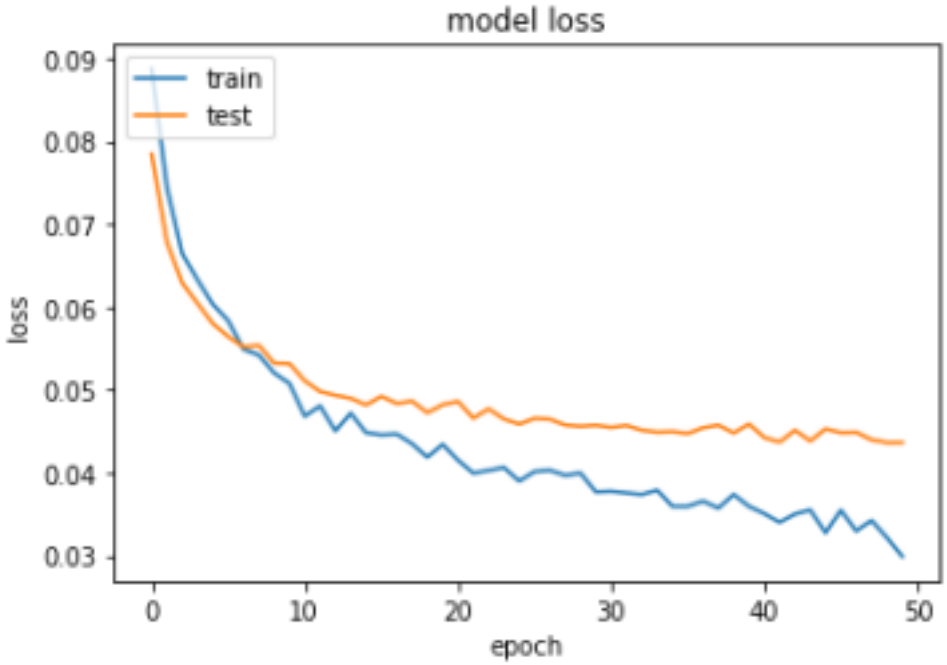

Figure 2

As seen in Figure 2, the test loss starts to converge after approximately 15-20 epochs.

Question 2

For K-Fold Cross Validation K = 3, the dataset was splitted into 3 equal parts. Two parts were used for training and the third part for testing as follows:

- Training (1,2) Testing (3)
- Training (1,3) Testing (2)
- Training (2,3) Testing (1)

The average test and train accuracies were then recorded after training for these 3 Splits.

a) **Plot mean cross-validation accuracies over the training epochs for different batch sizes. Limit search space to batch sizes {1,4,8,16,32, 64}.**

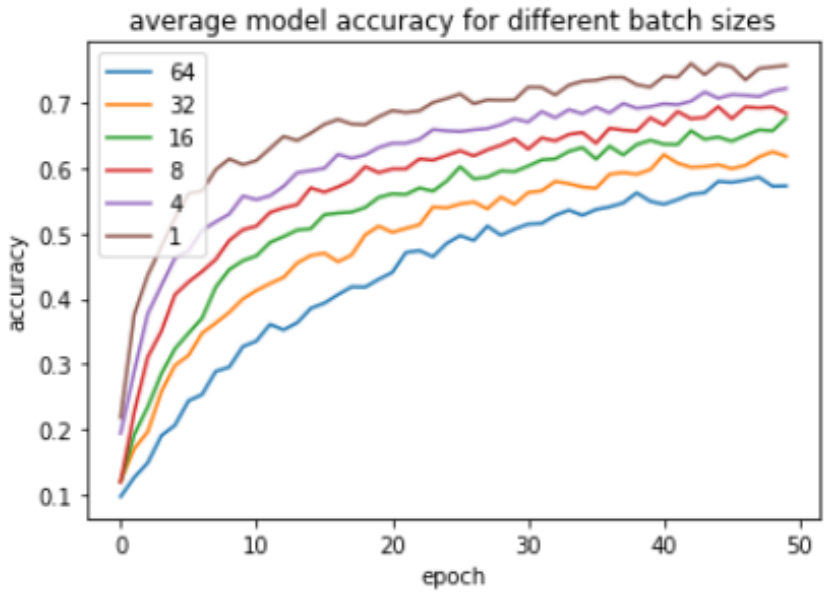

Figure 3

The plot of average model training accuracy for different numbers of batches can be seen in Figure 3. It can be seen that accuracy is highest for the smallest batch 1. As the batch size increases, the average accuracy decreases. This is because in batch gradient descent, average gradient for the whole batch is used to update the weights which essentially translates to moving in about the right direction of local minima. Batch size 1 means that you calculate exact gradient and exact direction of local minima and update weights accordingly hence high accuracy.

b) **Create a table of median time taken to train the network for one epoch against different batch sizes. (Hint: Introduce a callback)**

Below shows a table of mean time taken in seconds to train the network for one epoch against different batch sizes.

Batch Size	Average Time Taken for a single epoch
64	0.11035
32	0.10806
16	0.14147
8	0.20527
4	0.31444
1	0.59796

Below shows a plot of the average model training time for a single epoch for each batch size.

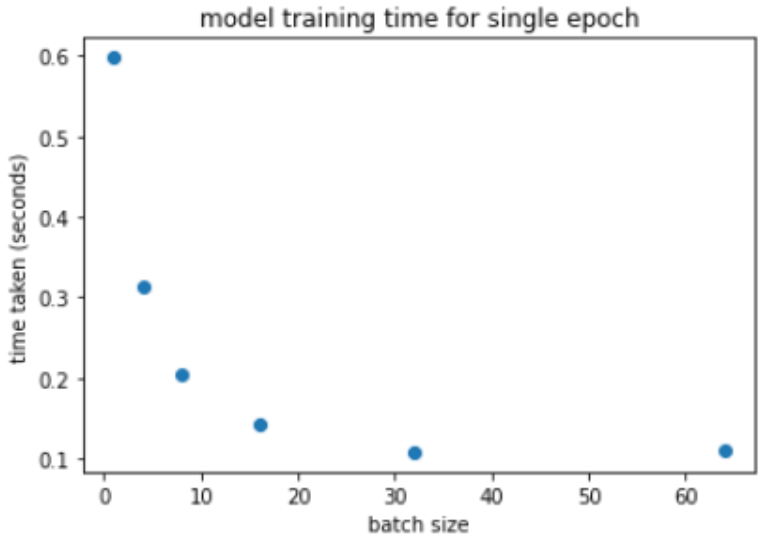

Figure 4

c) Select the optimal batch size and state reasons for your selection.

The choice of optimal batch size is made by considering both accuracy and time taken. The trade off is such that small batch size gives higher accuracy but higher training time per epoch compared to larger batch size which train quickly but are not as accurate.

By considering the tradeoffs, the ideal batch size that is chosen for the subsequent experiments is 16. This is because it took 0.14 secs per epoch which is less than half that of batch size 1. Furthermore, the time taken is relatively short and has a training accuracy that is similar to batch size 1. This makes batch size 16 ideal due to its efficiency and accuracy.

d) What is the difference between mini-batch gradient descent and stochastic gradient descent and what does this mean for model training?

When the batch size equals one, the learning algorithm is called stochastic gradient descent. When the batch size is more than one sample but less than the size of the training dataset, the learning algorithm is called mini-batch gradient descent.

For model training this means that the stochastic algorithm is iterative i.e. the search process occurs over multiple discrete steps, each step hopefully slightly improving the model parameters but in just the right direction. For batch gradient descent, average gradient for the whole batch is used to update the weights which essentially translates to moving in about the right direction of local minima but not exact. Stochastic gradient descent is far more accurate than mini-batch gradient descent but mini-batch gradient descent is far more efficient in terms of training time than stochastic gradient descent.

e) Plot the train and test accuracies against epochs for the optimal batch size.

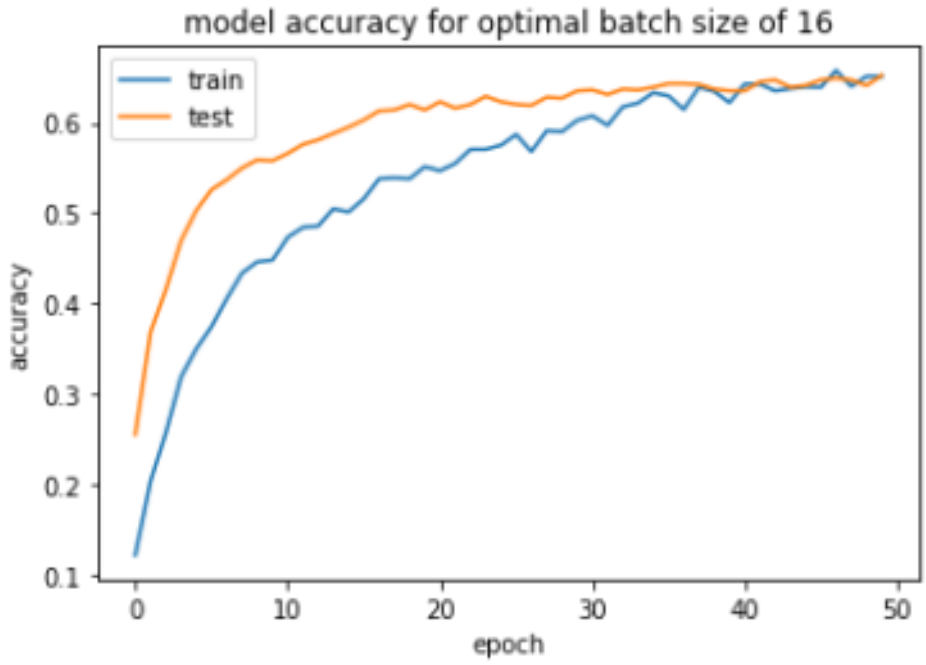

Figure 5

Question 3

New Model Architecture changes the number of neurons from 16 to {8, 16, 32, 64} to find the optimal number of neurons for the 2-layer network designed in Question 1 and 2.

a) **Plot the cross-validation accuracies against training epochs for different numbers of hidden-layer neurons. Limit the search space of the number of neurons to {8, 16, 32, 64}.**

   Continue using 3-fold cross validation on training dataset

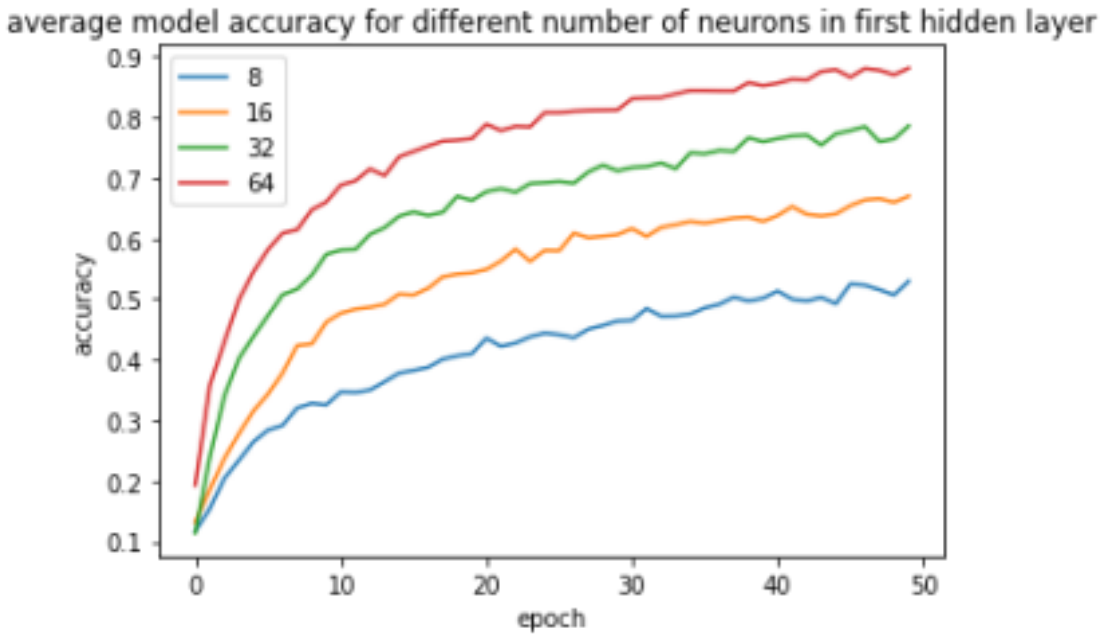

Figure 6

b) **Select the optimal number of neurons for the hidden layer. State the rationale for your selection.**

   The optimal number of neurons selected is 32 neurons for the hidden layer even though 64 neurons gives the maximum accuracy. This is because test and train accuracies for 64 neurons was indicating a slight over fitting which is not desirable in any model. Therefore, 32 neurons are selected to build the neural network.

c) Plot the train and test accuracies against training epochs with the optimal number of neurons.

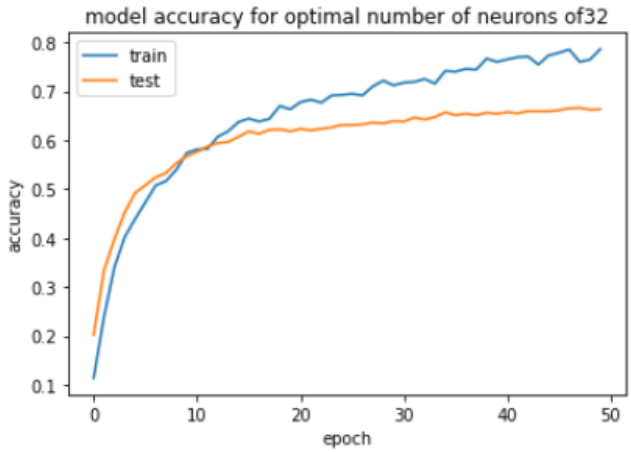

Figure 7

d) What other parameters could possibly be tuned?

Other parameters that can be possibly tuned are the activation functions, regression, drop out probability, learning rate, weight and bias initialization of the hidden layer and batch normalization.

Question 4

In this iteration, we changed the model architecture from 2 layer network to 3 layer network. Batch size 1 is used.

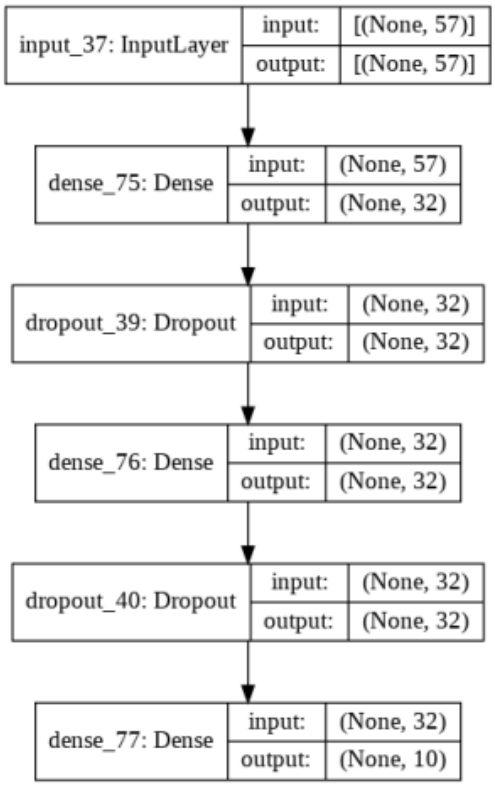

Figure 8

a) Plot the train and test accuracy of the 3-layer network against training epochs.

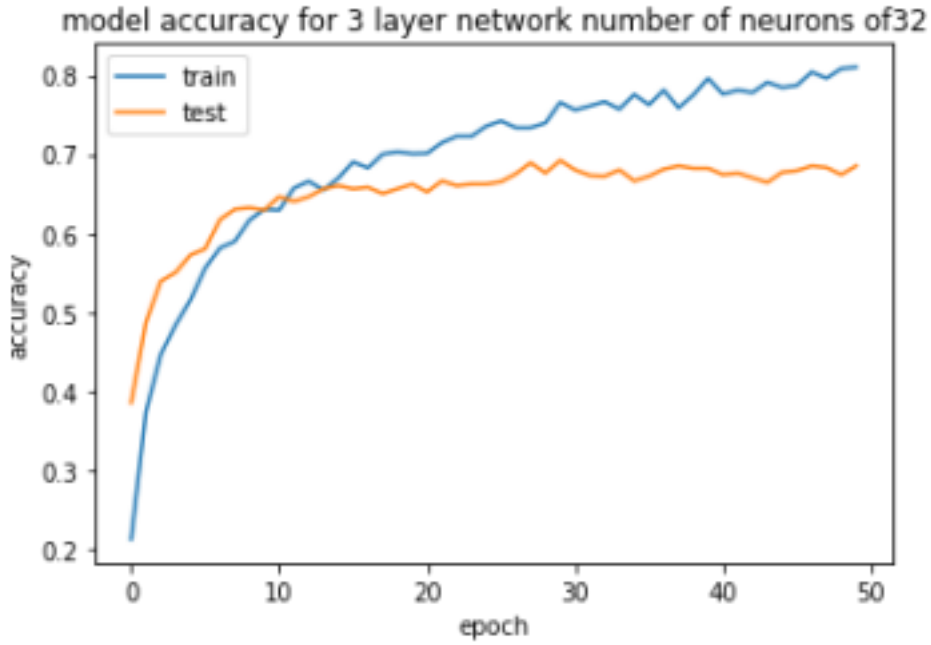

Figure 9

Below shows the model loss for optimal number of neurons of 3 layer network with 32 neurons in the hidden layer.

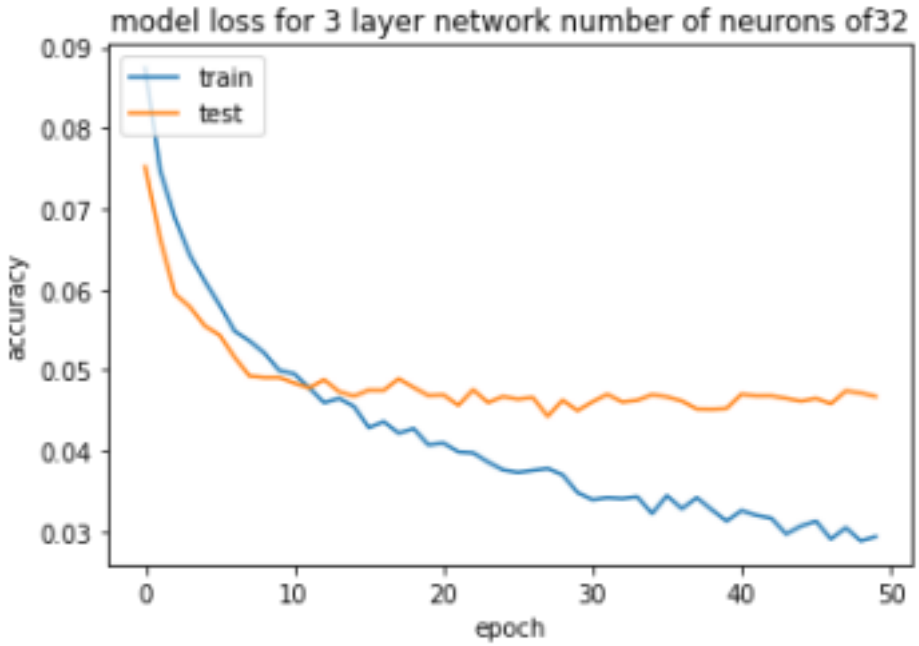

Figure 10

b) **Compare and comment on the performances of the optimal 2-layer network from your hyperparameter tuning in Question 2 and 3 and the 3-layer network.**

The training of the 3-layer network took 10 times longer than that of the optimal network created in Question 2 and 3. The training accuracy of our optimal neural network is higher compared to the 3-layer network which indicates slight overfitting in our model. Furthermore, although the test accuracy of the optimal network is only slightly lower than the 3-layer network, the training time was way faster. Therefore, the performance evens out.

Question 5

a) **Why do we add dropouts? Investigate the purpose of dropouts by removing dropouts from your original 2-layer network (before changing the batch size and number of neurons). Plot accuracies on training and test data with neural network without dropout. Plot as well the losses on training and test data with neural network without dropout.**

We add dropout to prevent model from overfitting on the dataset which may result in great training accuracies and poor test accuracies. Training and Test accuracies with and without dropout can be seen in Figure 11 below.

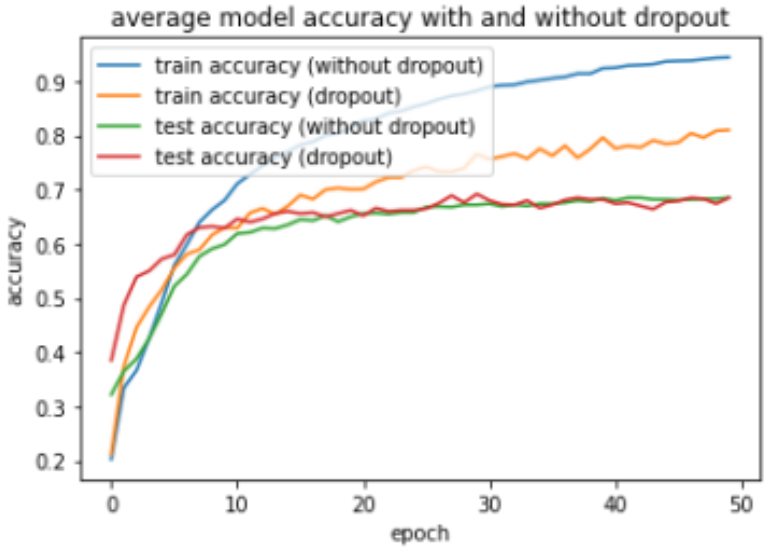

Figure 11

Training and test loss with and without dropout can be seen in Figure 12 below.

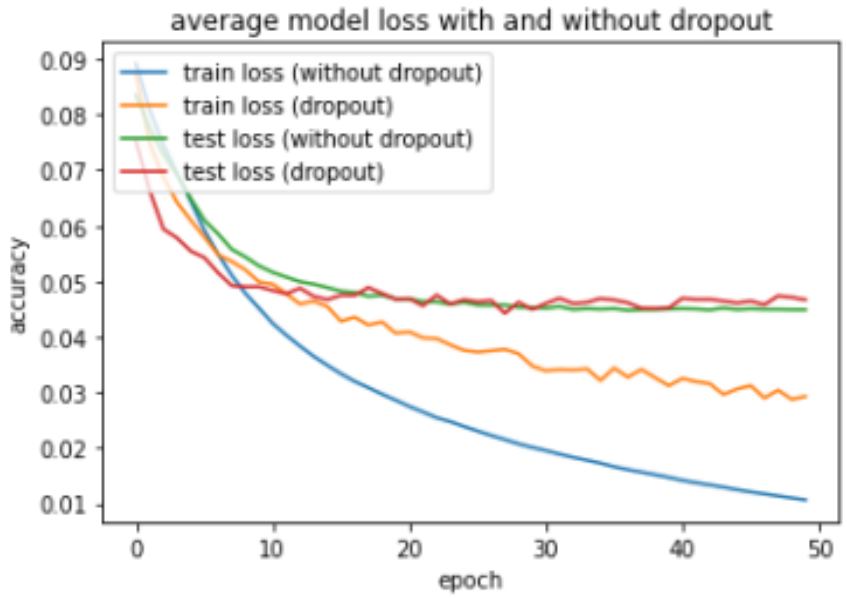

Figure 12

b) Explain the effect of removing dropouts.

The effects of removing the dropout can be easily seen in the above 2 plotted graphs. The above graph shows that the training accuracy is way greater if we do not use drop out layers which can be perfectly explained by the fact that the model is overfitting since we are not using dropout layer to regularize overfitting. If the model overfits, its train accuracy is way higher compared to its test accuracy which is what we can see in the graphs above. However, train and test accuracies are relatively close to each other in graphs where there is a dropout layer which perfectly explains why dropouts should be used to prevent our model from overfitting on the data. This results in a good generalized model that accurately models the underlying distribution.

c) What is another approach that you could take to address overfitting in the model?

Reduce the network's capacity by removing layers or reducing the number of neurons in the hidden layers and the number of hidden layers Apply regularization, which comes down to adding a cost to the loss function for large weights.

Conclusion

The limitations of the current approach is that the current model that we used is Fully Connected Neural Network. Since a fully connected neural network is unable to identify spatial patterns, this type of modelling cannot utilize spatial patterns in audio files to make accurate predictions. A convolutional neural network CNN, does exactly that. It finds patterns that are closely spaced in the dataset and exploits this pattern to make accurate models and accurate predictions for the underlying dataset.

Of all the parameters tuned, the one that had the most impact on the model accuracy was batch size. Larger batch sizes train faster but are less accurate because there is a number of samples processed before the model is updated, so larger batch size means larger samples to be processed at a time. The batch size of 1 had the greatest accuracy but it was also the slowest to train. It was the most accurate because the model weights were updated accurately after every iteration and every sample resulting in the best model.

Considering the audio files were originally in waveform in which the information relatively close to each other have meaning and is important. This kind of information is exploited and best utilized by Convolutional Neural Networks i.e. CNNs. So another architecture besides FNN that can be used in this kind of modelling is CNN.

The other dataset that can be used for modelling waveform data is feature integration where different features are combined to create a new feature that can be used in model prediction. Another thing that can be done is PCR or dimensionality reduction where the dataset is translated into another dimension and predictions are made in this lower dimensional space. Removing redundant or not so useful features is also a part of the process of feature engineering. All of this can be done by feature engineering and by using the Conv2D layer in the model architecture.

Part B: Regression Problem

Introduction

For Part B of this assignment, our aim is to perform retrospective prediction of HDB housing prices and identify the most important features that contribute to the prediction. The data used is the publicly available data on HDB flat prices in Singapore, obtained from data.gov.sg on 5th August 2021. The original set of features have been modified and combined with other datasets to include more informative features.

Question 1

a) Divide the dataset ('HDB_price_prediction.csv') into train and test sets by using entries from year 2020 and before as training data (with the remaining data from year 2021 used as test data). Why is this done instead of random train/test splits?

We use values at the rear of the dataset, which is the data from the year 2021, for testing. All the data before that is used for training which are the data entries from the year 2020 as this is a time series data set. In a time series data set, our goal is to predict values in the future. That is why we split our data such that the training data is that of the past and test data is in the future. This explains why we split our dataset like this instead of a random test/train split.

d) Plot the train and test root mean square errors (RMSE) against epochs

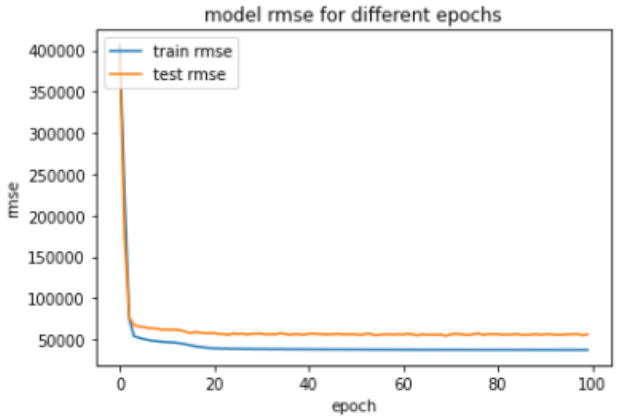

Figure 13

e) State the epoch with the lowest test error. State the test R2 value at that epoch.

Epoch 76 had the lowest test error of 4926007808. R2 Value at this epoch was 0.8081

f) Using the model from that best epoch, plot the predicted values and target values for a batch of 128 test samples.

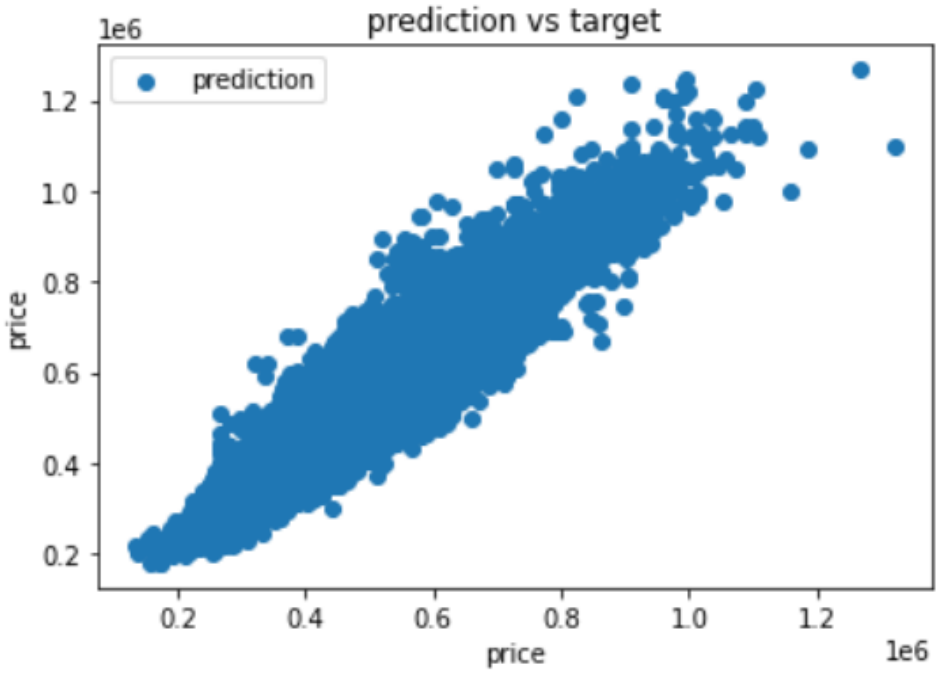

Figure 14

Question 2

c) Compare the current model performances in terms of both test RMSE and test R 2 with the model from Q1 (at their own best epochs) and suggest a possible reason for the difference in performance.

	Test Loss	**RMSE**	**R2**
Q1 Best Epoch = 76	4926007808	54586.6562	0.8081
Q2 Best Epoch = 77	3865624832	47246.6602	0.8487

As observed, an embedding layer improves the performance of our model. We use an embedding layer to compress the input feature space into a smaller one such that the features that closely resemble are close to each other in this smaller space. The vectors of each embedding get updated while training the neural network. This approach is way better than one-hot encoding because one-hot encoded vectors are high-dimensional and sparse, hence it is difficult to learn representation. This explains why embeddings in general perform better than one-hot encodings

Question 3

c) Compare the performances of the model with all 9 input features (from Q2) and the best model arrived at by RFE, in terms of both RMSE and R2.

	Test Loss	**RMSE**	**R2**
Q2 Best Epoch = 77	3865624832	47246.6602	0.8487
Q4 Best Epoch = 48	4107402240	4107402240	0.8398

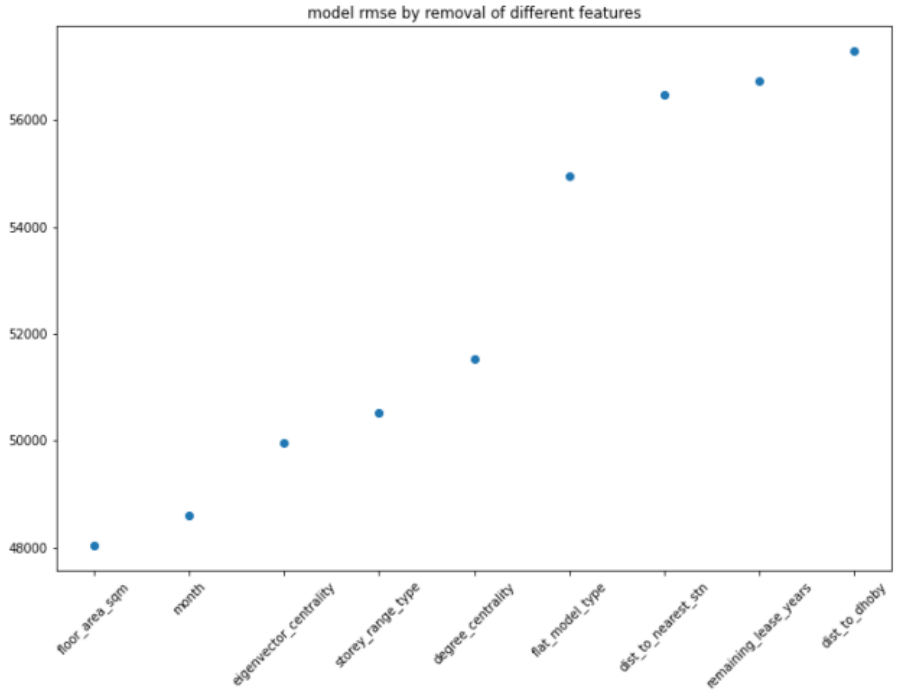

Figure 15

d) By examining the changes in model performance whenever a feature is removed, evaluate the usefulness of each feature for the task of HDB resale price prediction.

As seen in Figure 15, the rmse value increases if a feature is dropped. Removing the features floor_area_sqm and month feature has the least drop in performance therefore we can deduce that this feature is not very useful in correct model prediction.The usefulness of each feature can be seen by the increase in RMSE value when a feature is removed. Dst_to_dhoby has the largest RMSE value so this means that this is the most important feature in price prediction. Other important features include remaining lease years and dst_to_nearest_stn and flat type. On the contrary, the features month, eigenvector centrality and storey range type are not as important which can be observed by the drop in performance compared to other features.

Conclusion

As can be seen in the last graph, it is evident that from the listed features that we used for model prediction, the most important feature is dst_to_dhboy. When this feature was removed it resulted in the most drop in model performance. In other words, dropping these features resulted in the worst predictions, resulting in predictions that were way different than the actual values. We can conclude that the feature dst_to_dhboy is best to predict price fluctuation.

There are multiple ways to determine the factors that lead to price increase. Firstly, after a model is trained, the value of a feature is changed while the rest is kept constant in order to analyse the change in the predicted price value. The feature resulting in the highest predicted price value can be deduced to have the highest impact on price increase.

Another approach is to modify the loss function such that the decrease in predicted price results is penalized more than the increase. This way we can identify the feature that has the highest impact on price prediction resulting in its increase