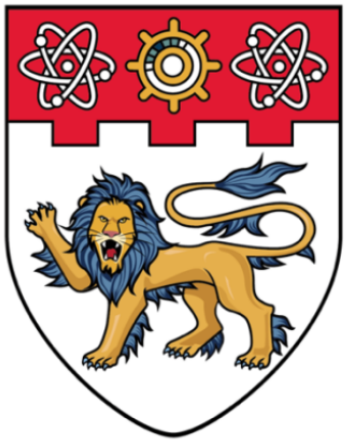

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4032: NEURAL NETWORKS AND DEEP LEARNING

NAME: VAIDYANATHAN ABHISHEK
MATRIC NO: U1923980D

ASSIGNMENT 1

INTRODUCTION

In this project we are required to solve two problems. The first problem is to classify the audio files into a given set of 10 categories. The second problem is a regression problem to predict HDB house values.

PART A : CLASSIFICATION PROBLEM

AIM

The first problem requires us to build a neural network model in order to perform the classification task on the GTZAN dataset.

DATASET

The dataset provided to us consists of 30 second long audio files each of which has 57 unique features. The response variable required to be classified is “label” that outputs the genre of the audio file.

ANALYSIS OF DATASET

[1]

```
df['label'].value_counts()
✓ 0.9s
blues       100
classical   100
country     100
disco       100
hiphop      100
jazz        100
metal       100
pop         100
reggae      100
rock        100
Name: label, dtype: int64

print("Number of categories: ", df['label'].nunique())
print("The categories are: ", df["label"].unique())
✓ 0.6s
Number of categories:  10
The categories are:  ['blues' 'classical' 'country' 'disco' 'hiphop' 'jazz' 'metal' 'pop'
 'reggae' 'rock']
```

[1] ***Figure 1:*** *number of unique categories of labels.*

On performing basic analysis on the dataset given, a brief overview of the information stored in the dataset can be obtained.

From the image shown we can conclude that the data that we have consists of 10 unique genre labels that are required to be classified. Further each genre label has 100 unique audio files that can be used to perform the classification task.

Preprocessing of Dataset

The given dataset consists of both numerical as well as categorical variables. Categorical variables can't be inputted directly into the model. Furthermore, even though numerical variables can be used directly, it is necessary to standardise the values of these numerical variables.

```
# Encode the labels from 0 to n_classes-1
label_encoder = preprocessing.LabelEncoder()
df['label'] = label_encoder.fit_transform(df['label'])
```

Figure 2: *label encoder*

The categorical variables are converted to numerical representation using the label encoder function. This function converts each of the categories into a set of numbers from one to the total number of unique categories for the particular variable. For example for a categorical variable which has values from January to December, each of these values are converted to a set of numbers from 1 to 12.

```
standard_scaler = preprocessing.StandardScaler()
x_train_scaled = standard_scaler.fit_transform(x_train)
```

Figure 3: *standardisation for numeric variables*

Numerical variables on the other hand have to be standardised to a particular range of values. This is done so as to reduce bias for values which may otherwise be too large without standardization. The standardisation technique applied here converts the values so that it follows a normal distribution. The formula that is implemented for this purpose is ((x-u)/s) where x is the individual data point for the data that has been inputted, u the mean of data and s is standard deviation.

```
# scale the training inputs
x_train = df_train.drop(columns_to_drop,axis=1)
y_train = df_train['label'].to_numpy()
```

Figure 4: *preprocessing of dataset*

Apart from applying transformation techniques on the input data, columns such as filename, label and length are removed since they aren't used as features for the model.

QUESTION 1

QUESTION 1a)

The first question requires the implementation of a two layer feed forward neural network. The neural network consists of a single hidden layer followed by an output layer. The hidden layer has 16 input neurons with relu activation function. The output layer consists of an output softmax layer with 10 output neurons. The model is implemented to use stochastic gradient descent with adam optimizer.

The model also makes use of dropout layers to the hidden layer with a probability of 0.3.

The model is trained for 50 epochs keeping the batch-size 1 which is using the dataset as a whole. For the purpose of training and validating the model on unseen data, the dataset is split into a ratio such that the training data is 70% of the entire data while 30% of the entire data is used for validation.

QUESTION 1b)

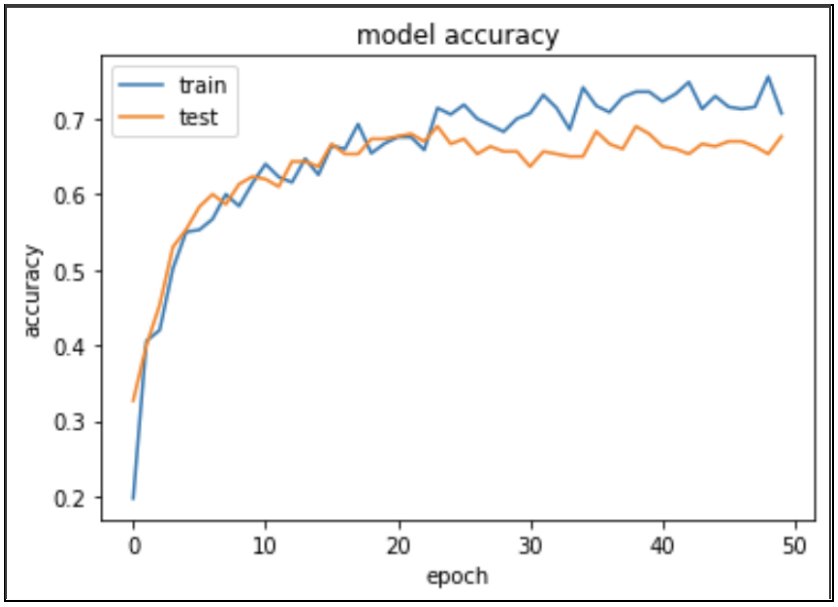

Figure 5: *Model accuracy plot.*

The plots shown in figure 5 have been plotted for a single run of the feed-forward model with two layers, trained for epochs.

Inference from the above plot:

1. The plots seem quite self explanatory since we can conclude that the train and test accuracy achieved by the model is around 76% and 68% respectively.
    a. Therefore, this means that for a given set of 100 unseed data samples, 68 of these can be correctly predicted.
2. The accuracy plots seem to slowly diverge with the increasing number of training epochs.
    a. This may also suggest that training with 50 epochs might be optimal, however this cannot be drawn as a concrete conclusion since there might be changes in model performance with increasing epochs.
3. One other noticeable detail is the fluctuation in training and validation accuracy.
    a. The model can be tested out using different sets of hyperparameters, change in the dropout probability, training on more data, applying regularisation techniques, etc.
4. The above plot clearly depicts the absence of overfitting since the gap between the training and validation accuracies is considerably less.

QUESTION 1c)

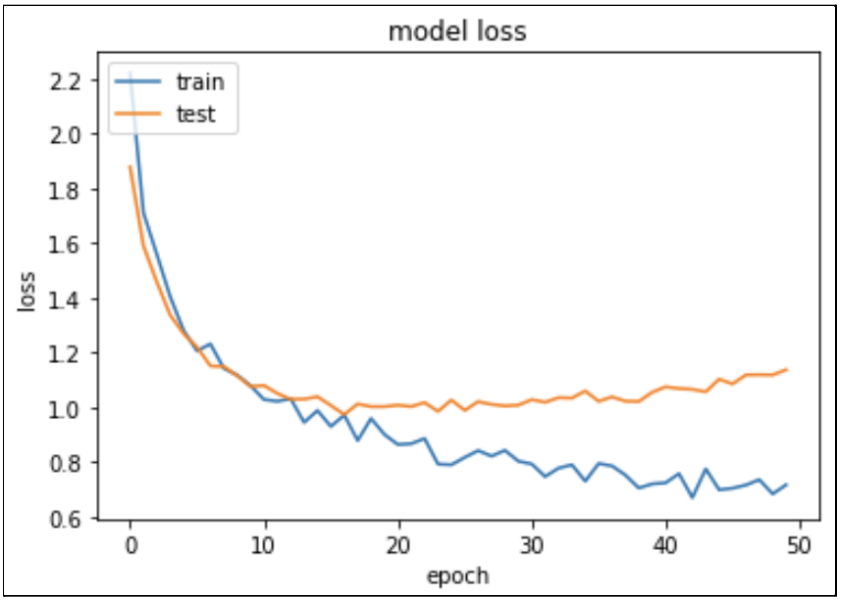

Figure 6: *model losses plot*

From the plot (figure 6), it can be inferred that from around **10 epochs the test error begins to converge**. However this converging pattern can be seen only till around 30 epochs beyond which the model losses begin to diverge.

Inference from the plot:

1. The plot above clearly shows the divergence in the training and validation losses.
    a. Techniques such as early stopping can help to reduce unnecessary model training.
2. The best possible training loss turns out to be around 0.6 while the validation loss comes out to be around 1.
3. From the plot we can also see that the fluctuation is quite low.
    a. Therefore the plot suggests that the parameters chosen are quite good for reducing model training.
4. Further improvements can be made to the model to get lower losses on validation data.
    a. Train on larger datasets
    b. Testing on different standardization techniques such as min-max scaler.
    c. Hyper-parameter tuning such as changing decay rate or loss functions, dropout probabilities, batch-size, number of neurons, number of hidden layers etc.

QUESTION 2

The aim of the second question is to determine the difference between stochastic gradient descent and mini batch gradient descent. This experiment makes use of training the model with different batch sizes, 1,4,8,16,32 and 64. The model is trained for each of the models and are analysed using appropriate plots so as to select the optimal batch size for obtaining best model performance.
For the purpose of this question 3-fold cross validation is used. This technique works with the idea that 2 folds of data are used for training purposes and the remaining one fold is used for testing. The advantage in using such a technique is that every data point given in the data is used for training and testing purposes.

```
def K_fold_cross_validation(no_folds,no_epochs,batch_size,X,Y,hidden_neurons,epoch_times_dict):
  print("Model training for:")
  print("Batch size",batch_size)
  print("Number of epcohs:" ,no_epochs)
  print("No folds used for k-fold cross validation:", no_folds)
  print("")

  kf = KFold(n_splits=no_folds,random_state=None, shuffle=False)
  KFold(n_splits=no_folds, random_state=None, shuffle=False)
  for train_index, test_index in kf.split(X_train):
      X_train_K, X_test_K = X[train_index], X[test_index]
      y_train_K, y_test_K = Y[train_index], Y[test_index]

      model = Sequential()
      model.add(Dense(hidden_neurons, activation='relu'))
      model.add(Dropout(0.3))
      model.add(Dense(10, activation='softmax'))

      model.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['accuracy'])

      tb = time_for_batch()
      te = time_for_epoch()

      history = model.fit(X_train_K, y_train_K,
                          batch_size=batch_size,
                          epochs=no_epochs,
                          verbose=2,
                          use_multiprocessing=False,
                          callbacks = [tb, te],
                          validation_data=(X_test_K, y_test_K))
      epoch_times_dict[batch_size] = te.times
      print("")
      return history
```

Figure 7: *K-Fold cross validation model*

Figure 7 depicts the usage of 3-fold cross validation where in KFold is the package imported from sklearn. The neural network model used for training remains the same.

QUESTION 2a)

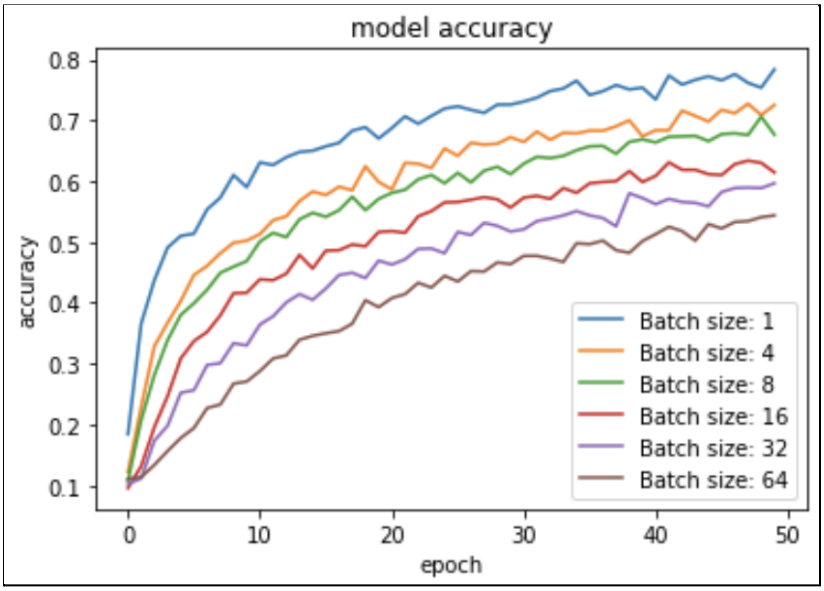

Figure 8: *Model accuracies for different batch-size*

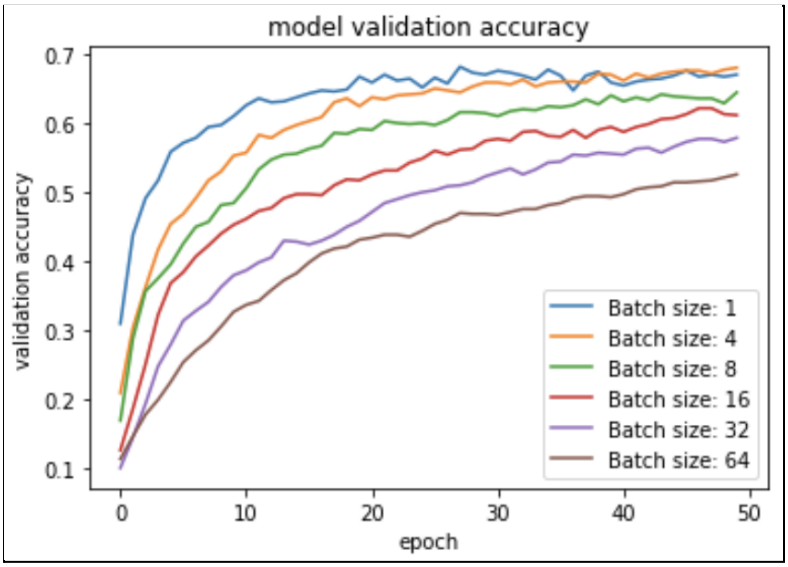

Figure 9: *Model validation accuracies for different batch-size*

Figures 8 and 9 depict the model accuracy and validation accuracy for different batch-sizes. It can be deduced that batch-size 1 has the best train accuracy along with high validation accuracy. On the other hand batch-sizes 4 and 8 have similar validation accuracies with considerably less over-fitting.

QUESTION 2b)

	Batch_sizes	median_epoch_times
0	1	0.874394
1	4	0.229945
2	8	0.153851
3	16	0.083358
4	32	0.061521
5	64	0.058546

Figure 10: *median time taken for each epochs*

The figure 10 depicts the table showing median epoch times. From the table above it can be inferred that as the batch size increases the training time for each increases.

QUESTION 2c)

In order to select the optimal batch size, the accuracy plots along with the median time for each epoch has to be taken into consideration. From figures 8 and 9 we can conclude that batch-size may not be the best model due to the difference in train and validation accuracies. Thus the model with batch size suffers the problem of overfitting on the train data. On the other hand the model with batch-size 4 and 8 perform better since they have lesser difference in their accuracies. The optimal batch size that can be considered here should be 8. This is because even though the validation accuracy is slightly less than the model with batch size 4, the training time is much lesser. Therefore with faster training as well as with comparable validation accuracies batch size 8 seems to be the most optimal.

This can also be seen in figures 11 and 12.

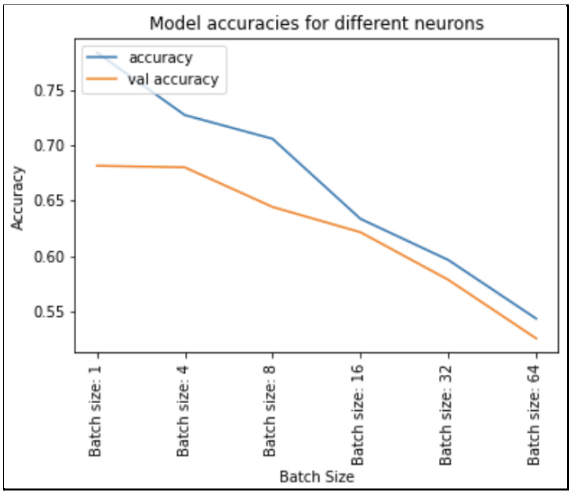

Figure 11: *best model accuracies for different batch-sizes*

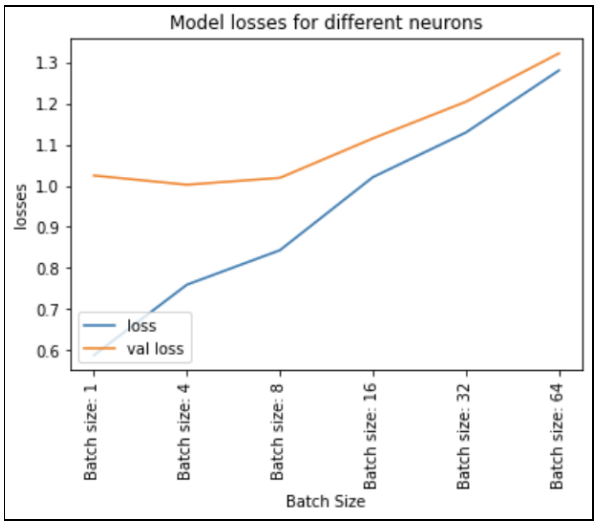

Figure 12: *lowest model losses for different batch-size*

QUESTION 2d)

The difference between mini-batch gradient descent and stochastic gradient descent is in the computation that takes place in the model. In stochastic gradient descent every data point is used to calculate the gradient. On the other hand batch gradient descent takes into consideration a small set of data points for the calculation of gradient descent. So in every iteration of the dataset, in stochastic gradient descent, the loss function is computed for the entire dataset, however for batch gradient descent the loss function is calculated for every batch at each iteration. This is later passed through the model for training.

QUESTION 2e)

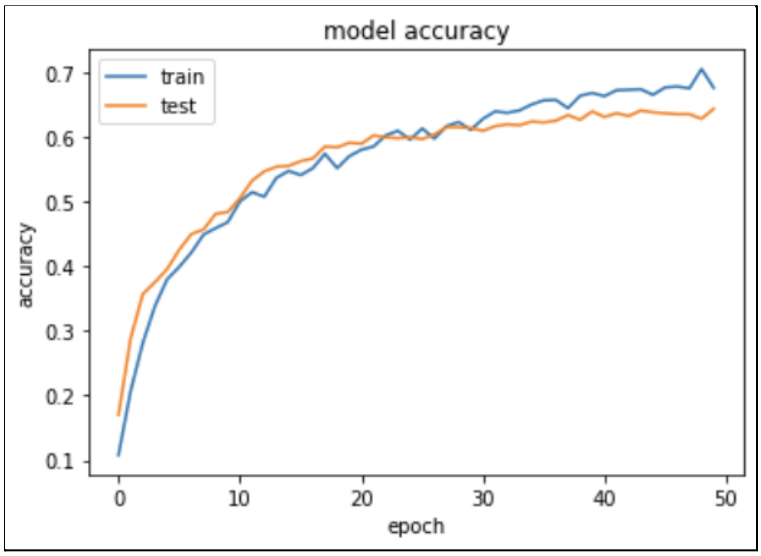

Figure 13: *Model accuracy with batch size 8*

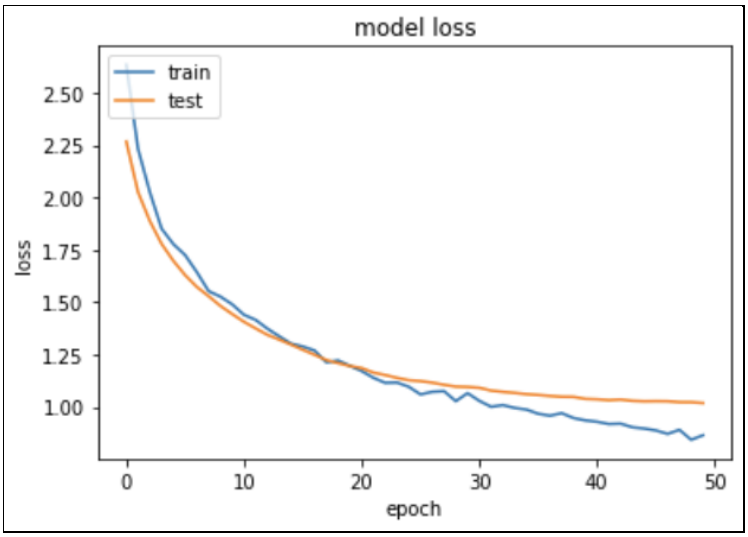

Figure 14: *Model losses with batch size 8*

QUESTION 3

The aim of this experiment is to find the optimal number of neurons that is used in each of the hidden layers. The model architecture remains the same with the use of 3-fold cross validation for training the model on the given dataset. Batch size 8 is used for this experiment as found in the previous experiment.

The set of neurons that is required to be tested out are as follows :- 8,16,32 and 64.

QUESTION 3a)

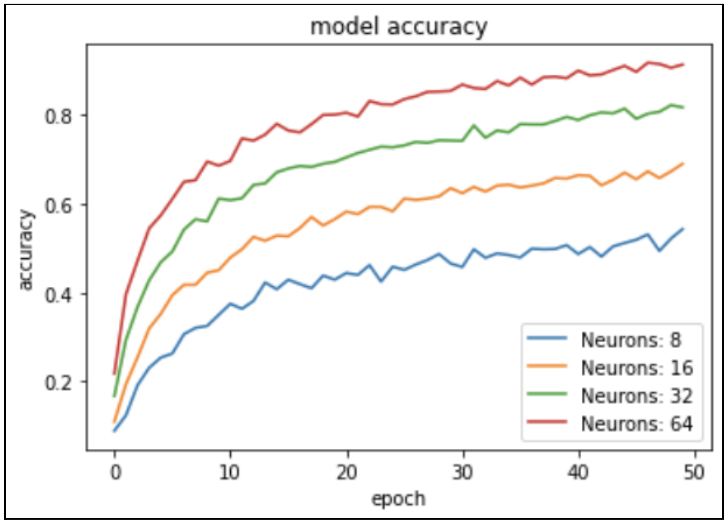

Figure 15: *Model accuracy for different number of neurons used in each hidden layer.*

QUESTION 3b)

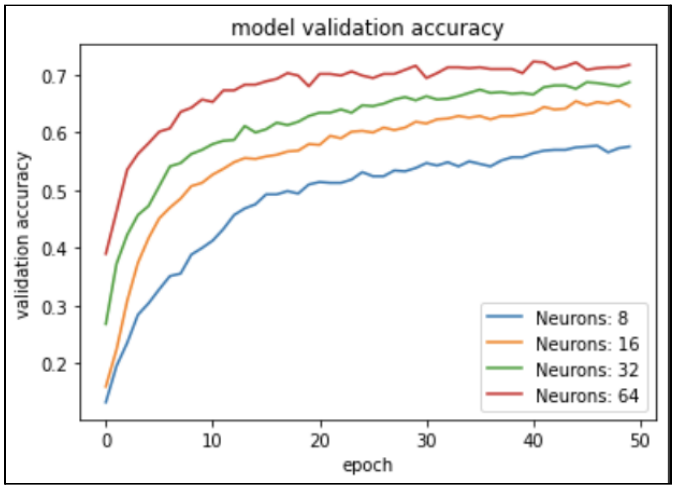

Figure 16: *Model accuracy for different neurons used in each hidden layer.*

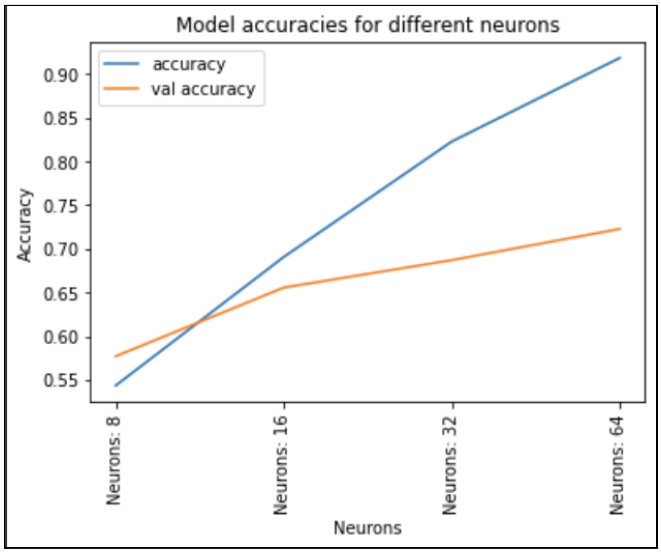

Figure 17: *Best model accuracy for different neurons.*

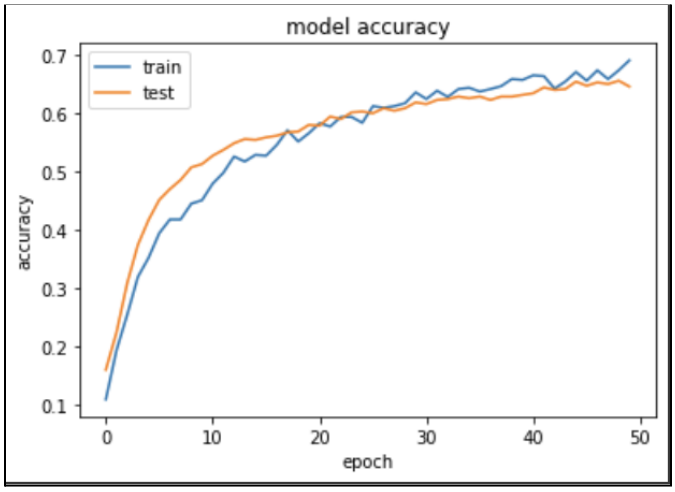

Figure 18: *lowest losses for different neurons.*

From figure 16,17 and 18 we can conclude that the optimal number of neurons that has to be used is 16. This is because the model with 64 neurons and 32 neurons suffer from overfitting when compared to the model with 16 neurons. Figure 18 also shows that the loss is considerably low when compared to the models with different number of neurons, The model with neurons 4 is rejected due to the fact that the loss is higher as well as with lower accuracy. Therefore the optimal number of neurons is 16.

QUESTION 3c)

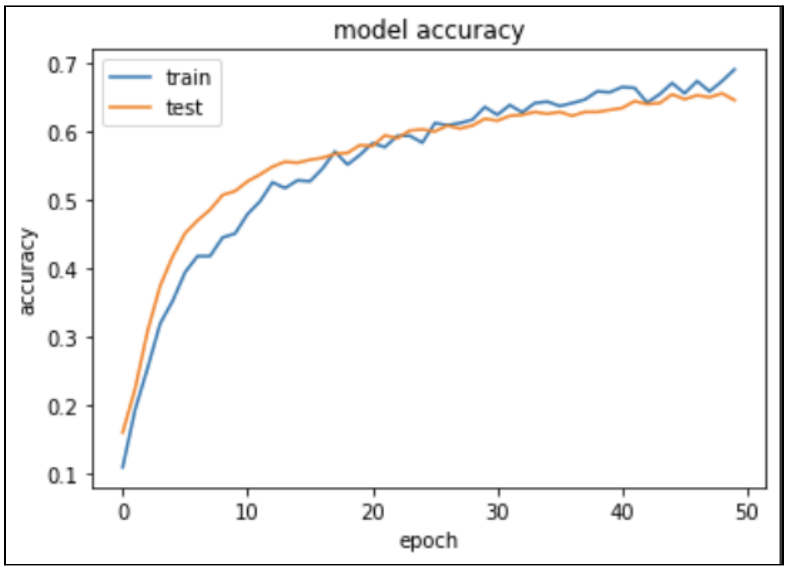

Figure 19: *best model accuracy for number of neurons 16*

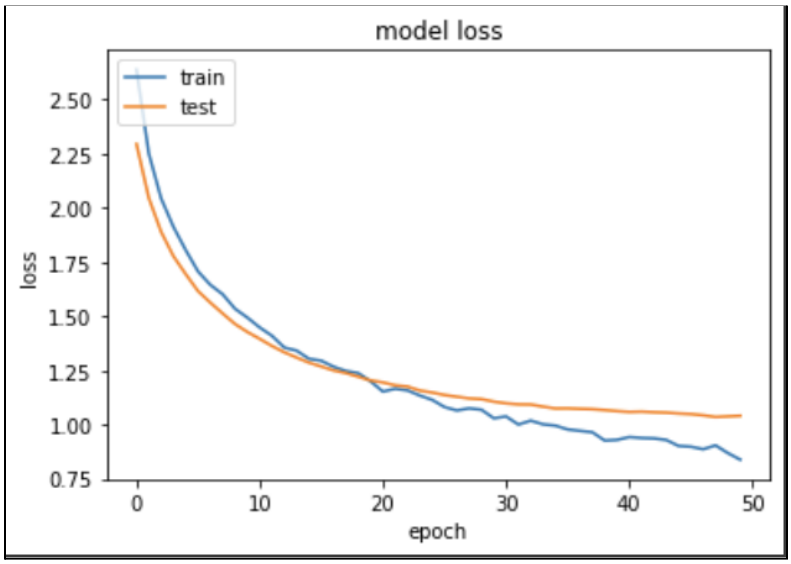

Figure 20: *lowest model loss for number of neurons 16*

QUESTION 3d)

A few other parameters that can be finetuned are :-

- Number of epochs
- Learning rate
- Dropout probability
- Optimizer being used, for example we can use other optimizers other than adam
- Different activation functions.
- Different train, test split for training the model we can also consider random vs non-random split.

The above mentioned are a few changes that can be considered while training the model since each such hyper-parameter can greatly impact model performance.

QUESTION 4

QUESTION 4a)

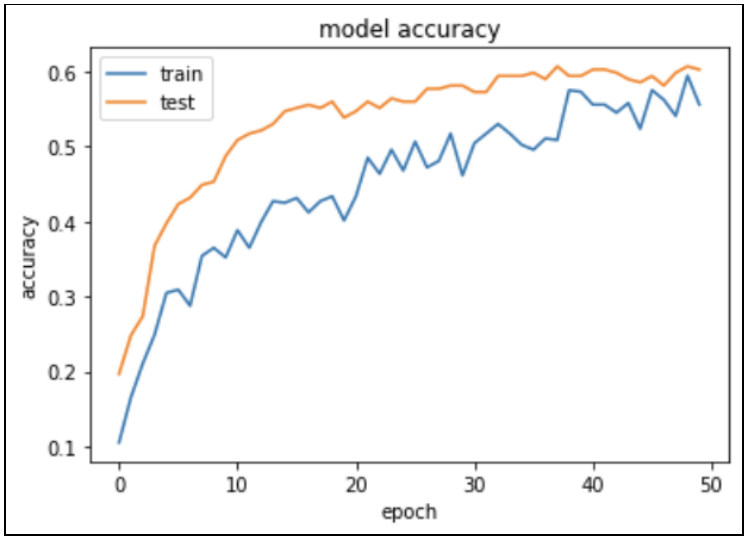

Figure 21: *model accuracy for three layer neural network*

QUESTION 4b)

From the figure 19 and 21 we can see that the model training improves since the gap between training and testing accuracy reduces. However it can be noticed that the validation accuracy is

greater than that of train accuracy. This means that even though the model performs well on new data, the model can perform even better if training accuracy increases. Thus the model performance can be improved by proper hyper parameter tuning for example the dropout probability might not be optimal for the three layer network. The model can also be tested out with different train and test splits of data. Therefore, to conclude the model performance can be improved greatly by getting the optimal set of hyperparameters.

The hyperparameter that can be tuned are as follows:

- Batch size
- Number of neurons
- Loss function
- Optimizer function
- Learning rate etc,.

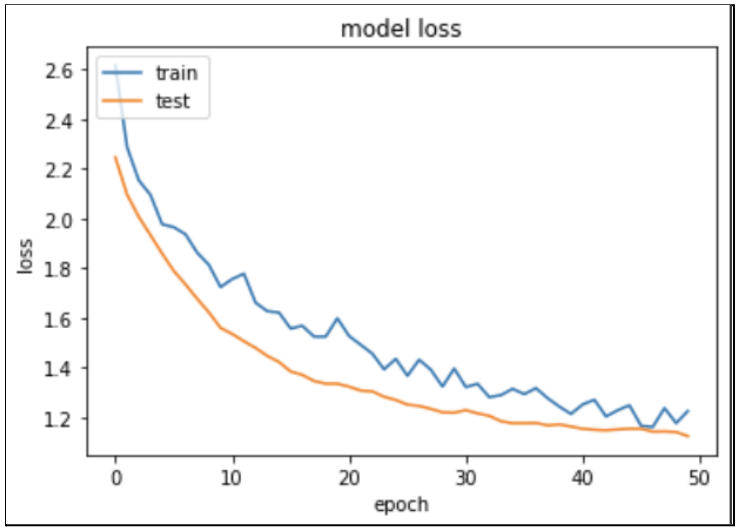

Figure 22: *model loss for three layer neural network.*

QUESTION 5

QUESTION 5a)

The reason to add dropouts is to reduce the possibility of overfitting while training the model. The reason why this works is because the dropout layer brings in a factor of removing neurons while training the model. So in the case of training our model we use a dropout layer with probability 0.3. This means that each neuron in the hidden layer has a 0.3 probability of being removed from the neural network.

From the plots we can notice the fact that the train accuracy keeps improving, achieving a score of almost 0.9 on the train data. On the other hand, the best possible test accuracy achieved is

around 0.68. This suggests that removing the dropouts after the hidden layers results in overfitting of the model. Therefore, the model performance is greatly impacted when testing on unseen data. Figure 23 and 24 show the model performance without dropouts.

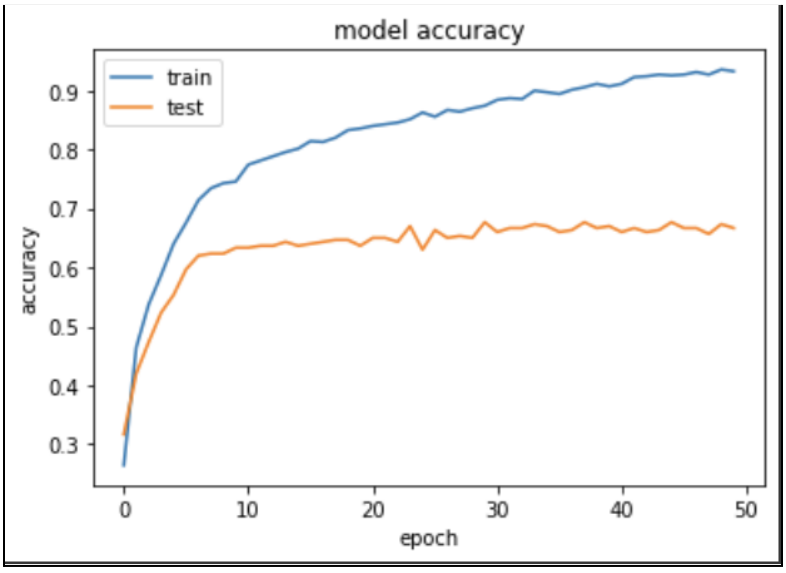

Figure 23: *Model accuracy without dropout layers.*

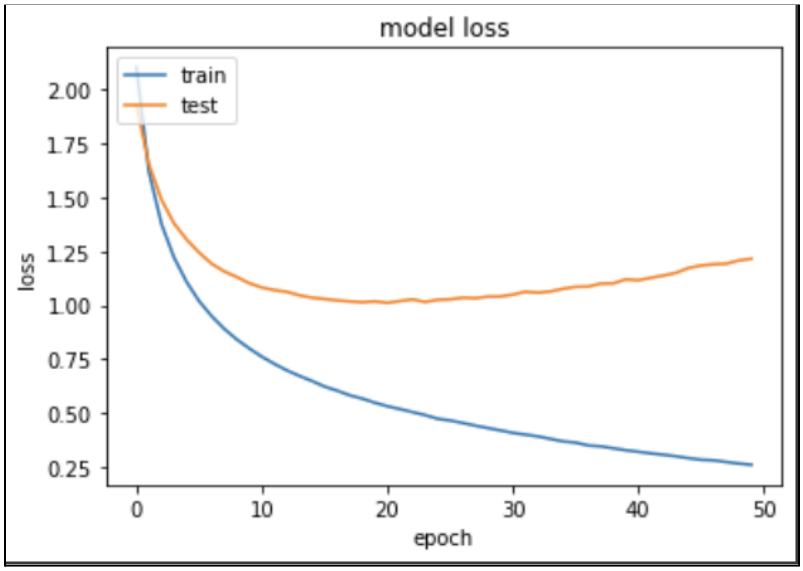

Figure 24: *Model loss without dropout layers.*

QUESTION 5b)

Other methods to avoid overfitting could be :

- Early stopping
- Data augmentation(possible applications for a CNN models)
- Regularization techniques such as using L1 or L2 regularisation

- Cross validation also helps in addressing the problem of overfitting
- Train the model on more data

DISCUSSION POINTERS

1. From the above network, the drawback can be attributed to the limited amount of data to train the model. The implementation can be considered to use multi-processing to improve training time as well as possibly improve model performance.
2. The audio files can also be considered as a whole such as to possibly convert it to text so as to extract more features from what is already available. This can help get more features, however training will have to be done according to increase in the size of data as well as the increase in features.
3. Possible improvements in the model is the use of RNN's or LSTM's so as to understand the context of the audio files. These implementations might work well given that the audio files are converted to textual data.
4. The alternative approach is as mentioned above, to convert it to textual data.
5. Other than the features that have already been used, further analysis of waveforms can be conducted to extract more data. For example features such as energy , entropy of energy can be considered to be used for this classification task.
6. Out of the parameters that were trained it can be inferred that the batch size has a huge impact with respect to reducing training time as well as the reducing overfitting on the model. Testing the model with a different number of neurons does help, but with respect to improving model performance batch size has the greatest impact.
7. Sentiment classification tasks can be performed using the pipeline. Datasets involving classification of signals can be used to classify using the model since it can be considered as another form of waveform.
8. The features can be compared against the r2 score to better understand the impact of each of the features. With this it can be possible to train the model to give more importance to features which have a high impact on the predictor variable.
9. Further it should be checked if the the features themselves have a high correlation with each other. In the case that they have a high correlation between each other it might result in poor model performance.
10. Other possible ways to improve the model is to perform rigorous hyperparameter tuning. For example the keras tuner can be used to get the optimal hyperparameters.
11. Hyper-parameters that can be tuned are optimizer used, learning rate, regularization technique etc.
12. Similar to the regression problem, rfe can be implemented to understand if it is required to train the model with all the features.

PART B : REGRESSION PROBLEM

AIM

To perform a regression task to predict the values of HDB prices in singapore.

DATASET

The dataset given contains data on the flat prices of HDB in singapore. The dataset consists of 12 features that can be used as predictor variables to perform the regression task.
However, all the features don't belong to the same category, that is can be either numeric or categorical features. Furthermore, all the features aren't used for this purpose. In line with the requirements of the task a few variables are excluded from the set of predictor variables. Further task dependent preprocessing is done for each of the categorical variables as well as for the numeric variables.

QUESTION 1

QUESTION 1a)

The dataset is first split into train and test sets based on the "year" feature. This is done by considering all data points with the values of year less than or equal to 2020 as the train dataset while all data points with the value of year greater than 2020 as the test dataset.

```
train_data = df[df['year']<=2020]
test_data = df[df['year']>2020]
```

Figure 25: *train test split of data*

Such a train test split is considered instead of random split because for the above mentioned problem we need to predict the values for the cost of the HDB. In such a case it would be best for the model to learn how the trends change over the years rather than to learn relationships based on random data points. For example it is better for the model to learn the trends followed in 2018 and how they change in 2019 rather than to learn trends in 2018 and then learn trends followed in 2020. In this case the model would end up predicting HDB costs in 2019 which may not be entirely useful in this case.

QUESTION 1b)

A two layer feedforward model is implemented to perform the regression task. The model comprises a single hidden layer with 10 hidden neurons followed by a linear output layer. The inputs to the hidden layer are first pre-processed before inputting it into the model.

The preprocessing here is based on encoding the string type categorical variables as well as normalising the numeric variables. One hot encoding of the categorical variables is done in order to get a numeric vector representation of these features. This helps the model learn better from the features rather than inputting strings as features.

```
def encode_categorical_feature(feature, name, dataset, is_string):
    lookup_class = StringLookup if is_string else IntegerLookup
    # Create a lookup layer which will turn strings into integer indices
    lookup = lookup_class(output_mode="binary")

    # Prepare a Dataset that only yields our feature
    feature_ds = dataset.map(lambda x, y: x[name])
    feature_ds = feature_ds.map(lambda x: tf.expand_dims(x, -1))

    # Learn the set of possible string values and assign them a fixed integer index
    lookup.adapt(feature_ds)

    # Turn the string input into integer indices
    encoded_feature = lookup(feature)
    return encoded_feature
```

Figure 26: *preprocessing for categorical variables*

The function shown in figure 2 is used to encode the categorical variables. This encoding method is used to encode categorical variables such as “month”, “storey_range” and “flat_model_type”.

```
def encode_numerical_feature(feature, name, dataset):
    # Create a Normalization layer for our feature
    normalizer = Normalization()

    # Prepare a Dataset that only yields our feature
    feature_ds = dataset.map(lambda x, y: x[name])
    feature_ds = feature_ds.map(lambda x: tf.expand_dims(x, -1))

    # Learn the statistics of the data
    normalizer.adapt(feature_ds)

    # Normalize the input feature
    encoded_feature = normalizer(feature)
    return encoded_feature
```

Figure 27: *preprocessing for numerical variables*

A similar function is defined in order to standardise the numeric features. This is done to ensure that the model doesn't get biased by very large values of certain features.

QUESTION 1c)

The model architecture as defined above is depicted in the images given below.

```
def regression_model():
    x = layers.Dense(10, activation="relu")(all_features)
    output = layers.Dense(1, activation="linear")(x)
    model = keras.Model(all_inputs, output)
    opt = tf.keras.optimizers.Adam(learning_rate=0.05)
    model.compile(opt,
            tf.keras.losses.MeanSquaredError(),
            metrics=[tf.keras.metrics.MeanSquaredError(),tf.keras.metrics.RootMeanSquaredError(),r2_score])

    keras.utils.plot_model(model, show_shapes=True, rankdir="LR")

    model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
        filepath='/Users/abhishekvaidyanathan/Desktop/NNDL/Neural-Networks-and-Deep-Learning/best_epoch_question1_checkpoint.tf',
        save_weights_only=True,
        monitor='val_loss',
        mode='min',
        save_best_only=True)
    history = model.fit(train_ds, epochs=100, validation_data=test_ds,callbacks=[model_checkpoint_callback])

    return [history,model]
```

Figure 28: *Model Architecture*

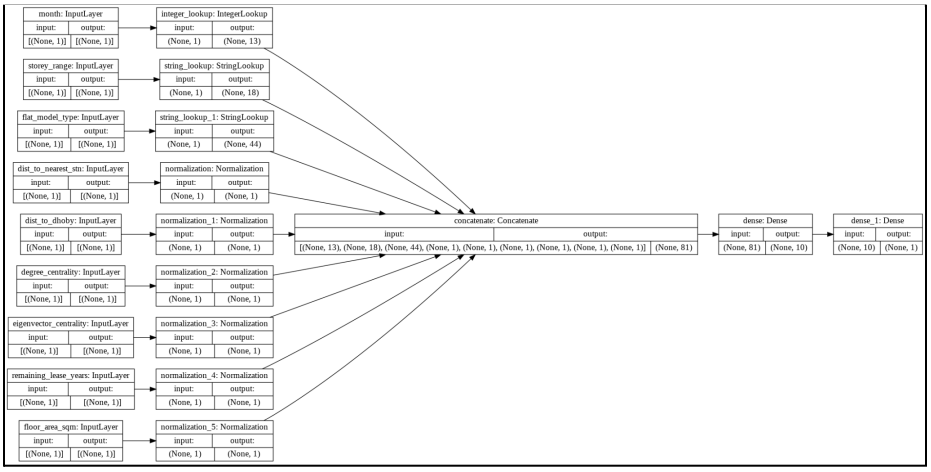

Figure 29: *Model architecture showing the inputs and outputs of every layer.*

The model is now trained on the dataset with the data having a batch size of 128. The hidden layers use the "relu" activation function with a learning rate of 0.05. Mean squared error is used as the cost function for training the model. The model is trained for 100 epochs with the above mentioned architecture and model parameters.

QUESTION 1d)

Figure 30: *Regression model RMSE values (plot starts from 5 epochs)*

QUESTION 1e)

```
def epoch_with_lowest_test_error(removed_feature_model):
    epoch_number = np.argmin(removed_feature_model.history['val_loss'])
    val_r2_value = removed_feature_model.history['val_r2_score'][epoch_number]
    r2_value = removed_feature_model.history['r2_score'][epoch_number]

    print("The training epoch with lowest test loss is: ",epoch_number+1)
    print("The values for train r2: ",r2_value)
    print("The value for validation r2: ",val_r2_value)
```

✓ 0.5s

```
epoch_with_lowest_test_error(regression_model_results[0])
```

✓ 0.9s

```
The training epoch with lowest test loss is:  75
The values for train r2:  0.8823171854019165
The value for validation r2:  0.8024139404296875
```

Figure 31: *function to get the epoch with lowest test loss*

The epoch with the best validation loss is 75. With a r2 score of 0.88 and validation r2 score of 0.80

QUESTION 1f)

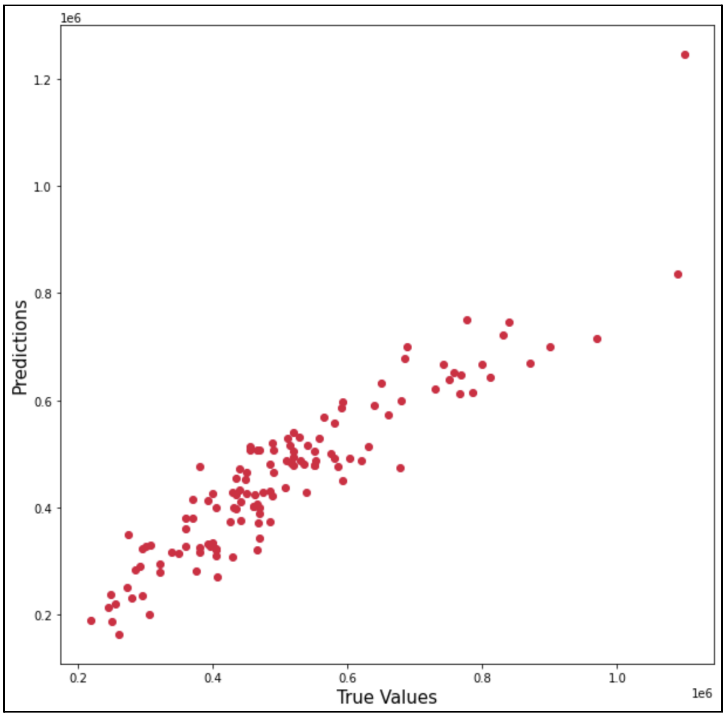

Figure 32: *scatter plot for a single batch of 128 samples*

QUESTION 2

QUESTION 2a)

An alternative model architecture that could be used is instead of using the one hot encoding, the model makes use of an embedding layer. The number of output dimensions for each of the embedding layer is the number of categories divided by two.

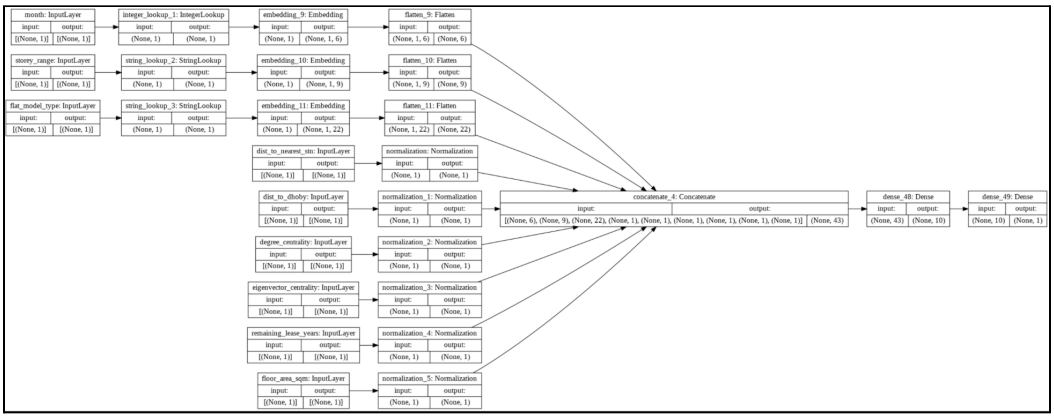

Figure 33: *Model architecture with embedding layer*

In order for the model

architecture to make sense it is necessary to have a flatten layer after the embedding layer. This is a necessary layer that needs to be added to the model since the output from the embedding layer is 3 dimensional while the output from the encodings of the numeric variables is 2 dimensional. The flatten layer reduces the dimension from 3D to 2D to confirm with the inputs for the hidden layer.

The reason to use the embedding layer is so that the model has access to better representations of the same categorical variable. This can be attributed to the fact that in one hot encoding the word vectors are represented in 2 Dimensions. Furthermore, if the vocabulary size is 100 then the vector representation for any word consists of 99 zeros and a single one from which the model needs to learn. So in the case that the vocabulary size is larger the essential features that the model can learn from becomes highly sparse when compared to the non essential features.

The above problem can be addressed to a considerable extent with the use of embedding for the categorical variables since the dimensionality isn't restricted to 2 dimensions. With this method the model could possibly have more features to learn from rather which may not always lead to better model performance.

QUESTION 2c)

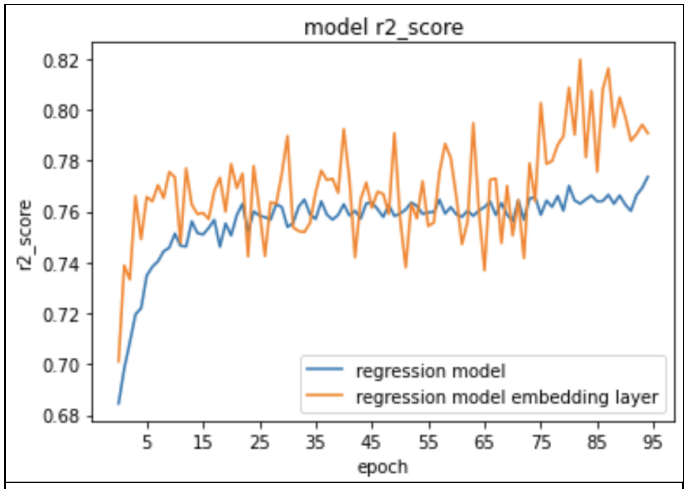

Figure 34: *r2 score plots for model with and without embedding layer.*

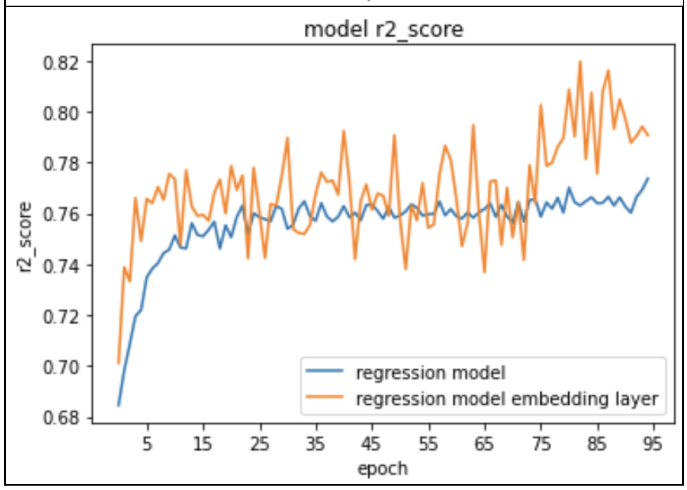

Figure 35: *root mean squared error with and without embedding layer.*

The change in the plots can be reasoned due to the addition of the embedding layer. It is evident that the model has more features to learn from since the embedding layer doesn't restrict its feature vectors to 2d space. Due to the extra dimension it can be said that the model has more information from the same data that is inputted from the dataset.

QUESTION 3

QUESTION 3a)

Early stopping is introduced to stop model training when there is no improvement in the value being monitored. In this case early stopping is implemented to monitor the model performance based on validation loss. The parameter "patience" is set to 10 which means that if there is no improvement in the validation loss after 10 epochs, model training is halted.

QUESTION 3b)

With the above implementation, Recursive feature elimination is used to determine the features that are required for model training and those that aren't required. This method basically involves training the model N times if there N number of features wherein each recursive call trains the model with N-1 features. The next recursive call uses the model with the best N-1 features and trains the model removing one feature. Now the model trains on N-2 features and such a recursive call continues till the model doesn't perform as good as the previous best model. After completion, it can be determined as to which features have a significant impact on the model and those features which reduce model performance.

QUESTION 3c)

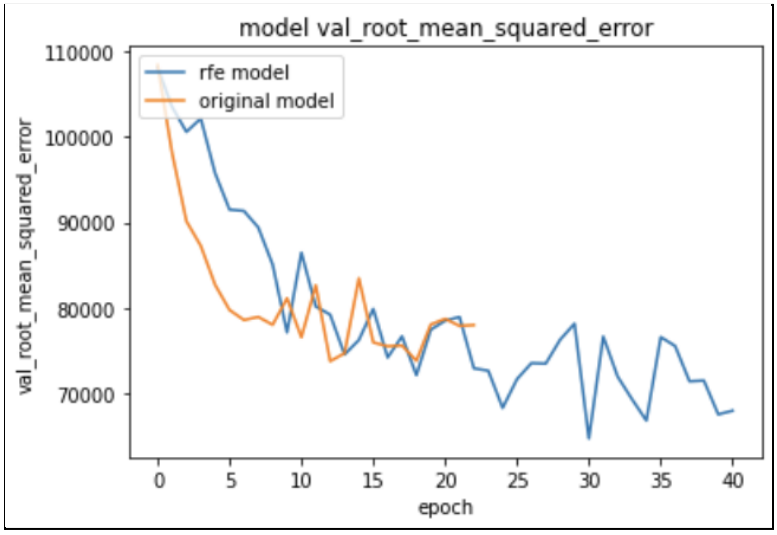

Figure 36: *model validation r2 score*

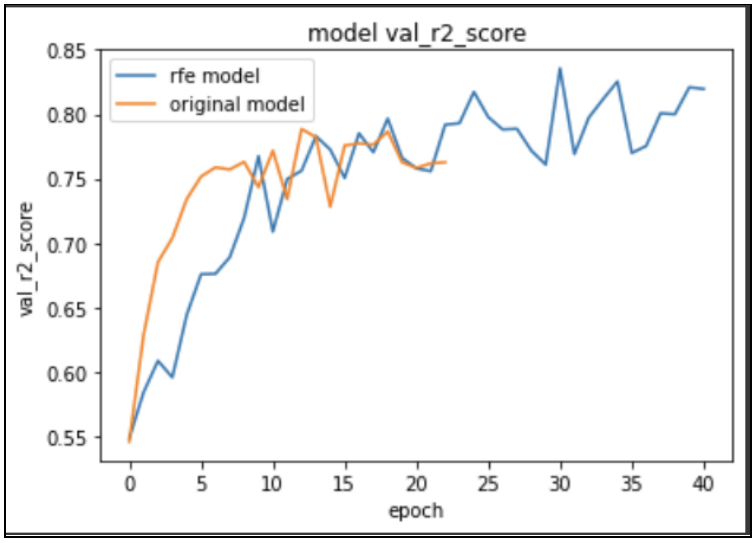

Figure 37: *model validation rmse*

The model obtained from rfe is the model with 8 features, that is the feature “degree centrality” is removed.

The figures 36 and 37 show the performance of the model obtained from recursive feature elimination and the original model. As we can see the model performance has improved greatly as the r2 score value increases. This indicates the features now better fit the model when compared to the model which uses all the features.The root mean squared error also shows improvement in model performance since the value reduces.

QUESTION 3d)

	Features_used_for_training	Feature_removed	Total_number_of_features	validation_loss	root_mean_squared_error	val_root_mean_squared_error
0	[storey_range, flat_model_type, dist_to_neares...	month	9	5.337171e+09	60650.593750	73055.945312
1	[month, flat_model_type, dist_to_nearest_stn, ...	storey_range	9	6.169030e+09	65784.953125	78543.171875
2	[month, storey_range, dist_to_nearest_stn, dis...	flat_model_type	9	5.730205e+09	57365.332031	75698.117188
3	[month, storey_range, flat_model_type, dist_to...	dist_to_nearest_stn	9	6.268960e+09	66050.625000	79176.765625
4	[month, storey_range, flat_model_type, dist_to...	dist_to_dhoby	9	8.002279e+09	80781.734375	89455.460938
5	[month, storey_range, flat_model_type, dist_to...	degree_centrality	9	4.202641e+09	53982.121094	64827.777344
6	[month, storey_range, flat_model_type, dist_to...	eigenvector_centrality	9	4.375159e+09	52717.816406	66144.984375
7	[month, storey_range, flat_model_type, dist_to...	remaining_lease_years	9	5.519778e+09	59252.871094	74295.210938
8	[month, storey_range, flat_model_type, dist_to...	floor_area_sqm	9	4.589663e+09	55681.003906	67747.054688
9	[storey_range, flat_model_type, dist_to_neares...	month	8	4.425053e+09	54255.441406	66521.070312
10	[month, flat_model_type, dist_to_nearest_stn, ...	storey_range	8	4.885740e+09	57956.925781	69898.062500
11	[month, storey_range, dist_to_nearest_stn, dis...	flat_model_type	8	5.299158e+09	58477.925781	72795.312500
12	[month, storey_range, flat_model_type, dist_to...	dist_to_nearest_stn	8	5.030885e+09	60091.480469	70928.734375
13	[month, storey_range, flat_model_type, dist_to...	dist_to_dhoby	8	8.661423e+09	82659.554688	93066.765625
14	[month, storey_range, flat_model_type, dist_to...	eigenvector_centrality	8	4.406172e+09	54282.835938	66379.007812
15	[month, storey_range, flat_model_type, dist_to...	remaining_lease_years	8	5.361463e+09	60532.882812	73222.007812
16	[month, storey_range, flat_model_type, dist_to...	floor_area_sqm	8	5.065774e+09	57929.750000	71174.250000

Figure 38: *Result of each iteration.*

From the table above a few conclusions can be drawn about each of the features. The feature “month” seems to impact model performance in a negative way, since the validation root mean

square error seems to be lower when compared to other models. It can be concluded that the feature can be removed to improve model performance.

On the other hand removing the feature “dist_to_dhoby” results in a higher validation root mean square error. This means that this is an important feature that decides the price of the HDB flat prices.

From the figure it can be inferred removing more than one feature doesn't help improve model performance since the model validation root mean squared error doesn’t improve further.
A similar inference can be made from the second recursive interaction, since removing the feature ‘dist_to_dhoby” impacts model performance greatly, in a negative way.

DISCUSSION POINTERS

1. With the information from RFE we can understand the impact that each of the features have on the model performance.
2. To understand the impact on the price, the model's validation loss can be taken into consideration. On removing a features if the validation loss is lower than when it was present then we can conclude that the feature results in increasing the price. Similarly if the loss is lower it would mean the feature tends to reduce price.
3. The model can be implemented using word vector embeddings such as word to vec or TF-IDF and test out model performance.
4. R2 scores can be considered while deciding the importance of each feature in performing regression.
5. The model can also be trained on random shuffling of data to get an idea as to how it performs on such random shuffling. This may very well help to predict prices of HDB whose values have not been logged.
6. Additional features such as geographical landscape, nearby stores or the view from the house can be included since they would definitely impact the price of the HDB.
7. Hyper-parameters such as loss functions, batch size, optimizer uses, regularization techniques, standardisation techniques can be changed to better analyse the impact of such features.
8. For better regression techniques, the model can be trained to fit data which probably follows a quadratic or logarithmic curve rathe than just to fit a straight line.