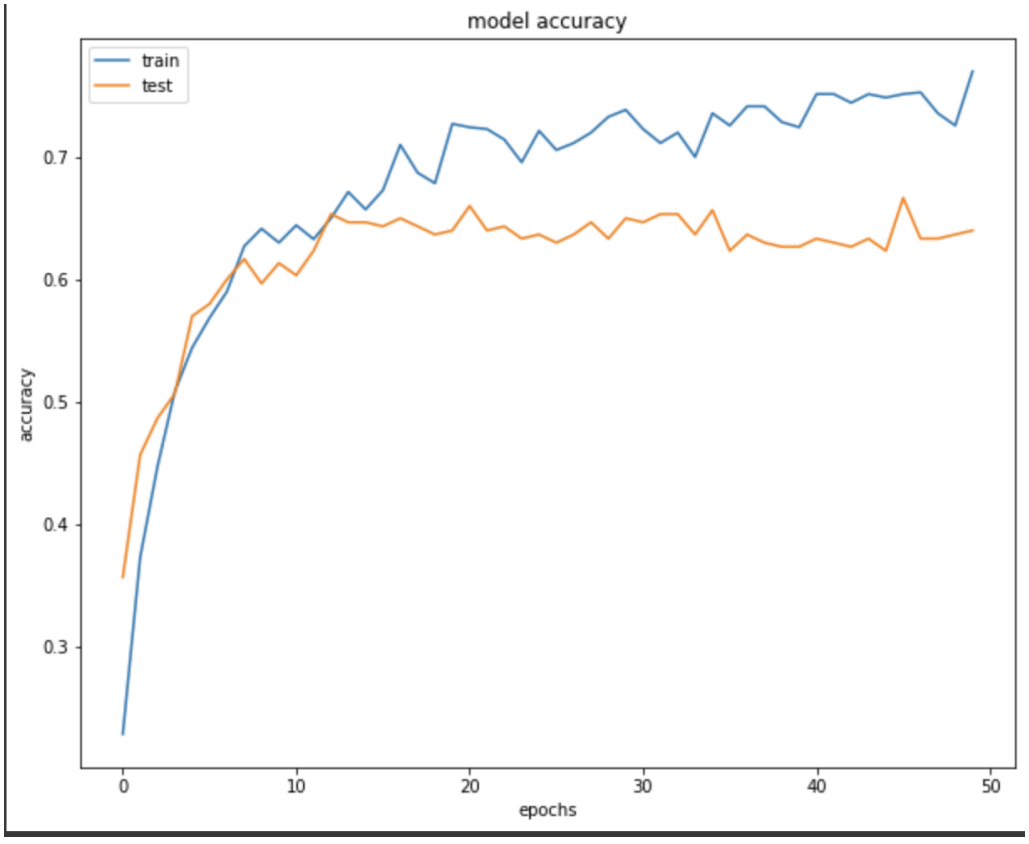
Part A

Question 1)

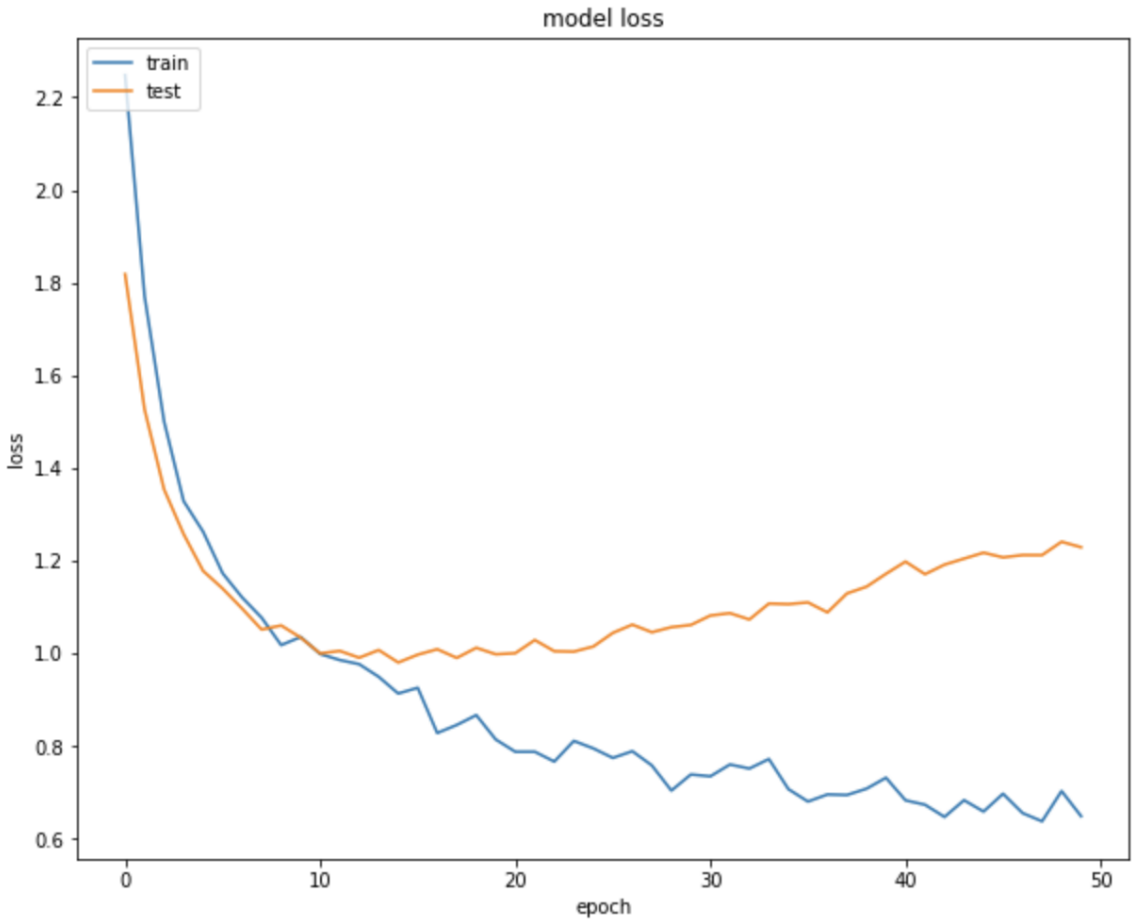
a) In this part, we first divide the dataset into 70:30 ratio for training and testing as well as scale the input features. Then we create a feed-forward deep neural network which consists of an input layer, one hidden layer with 16 neurons and relu activation, and an output softmax layer. We use stochastic gradient descent with adam optimizer and batch size of 1, alongside that , we have a dropout of 0.3 probability to the hidden layer.

b) Below is the plot for training and test accuracy against the training epochs



Comments - The test accuracy increases till approximately **13 epochs** and then converges and does not change much till the 50th epoch. While on the other hand, the train accuracy has a similar trend till the 13th epoch but it continues to increase, nevertheless, it more or less converges till the **20th epoch** as we notice only slight increase or decrease in the train accuracy trend after the **20th epoch**.

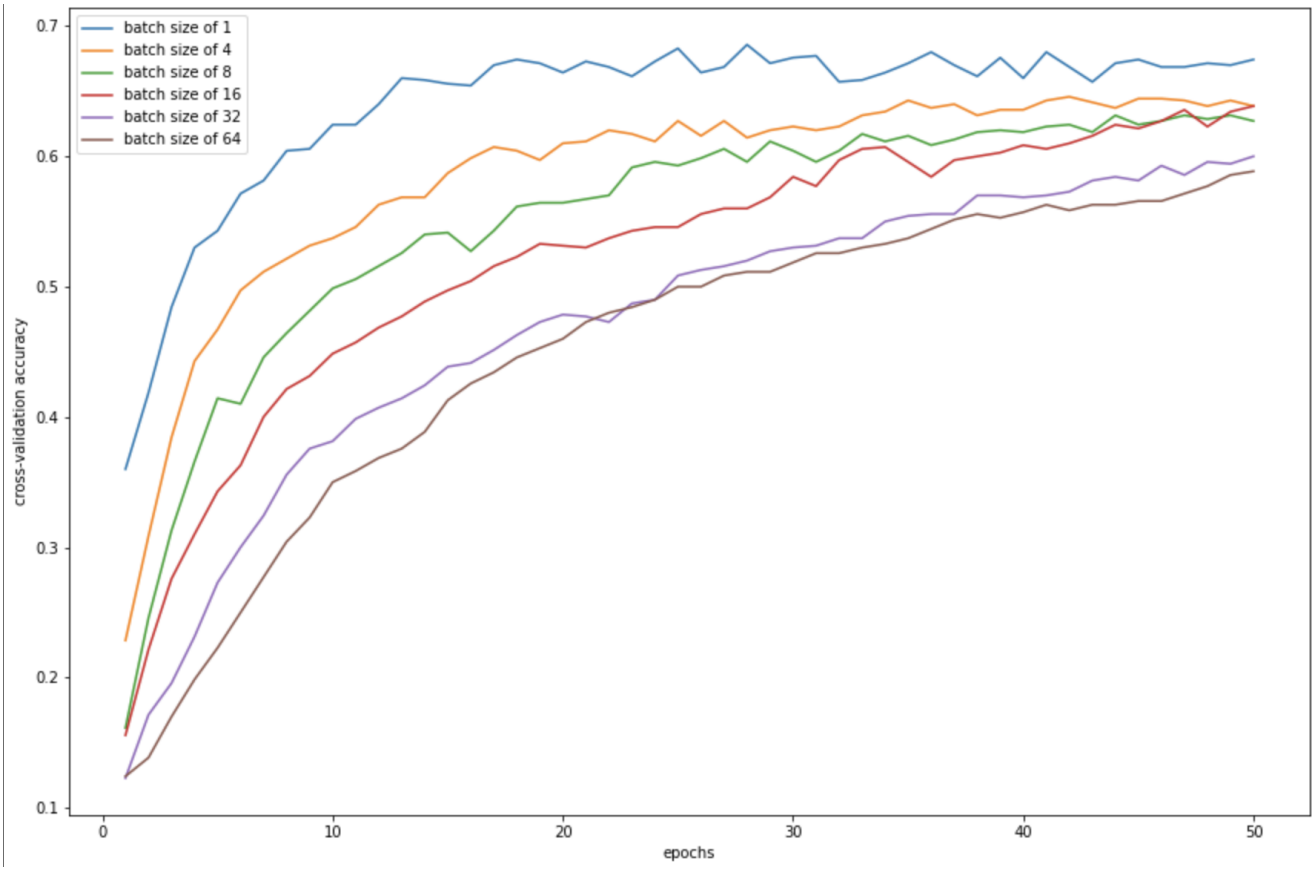
c) Below is the plot for training and test loss against the training epochs



It is quite evident in the plot, test error begins to converge at approximately the **10th epoch**.

Question 2)

a) Below is the plot for cross-validation accuracies over the training epochs for batch sizes - **1,4,8,16,32,64**



b) Below is the median time taken to train the network for one epoch against batch sizes - **1,4,8,16,32,64**

	Batch_size	Median epoch time
1	1	1.43
2	4	0.41
3	8	0.26
4	16	0.15
5	32	0.11
6	64	0.09

In order to find the median time in the above table, we first find the mean time over 3 cross folds for different epochs over different batch sizes. Then we calculate the median epoch time for the corresponding batch size using the results computed previously.The median epoch time **decreases** as we **increase** the batch size.

c) Batch sizes 32 and 64 - Higher batch sizes like **32** and **64** lead to lower asymptotic test accuracies, hence we reject batch sizes **32** and **64** as their test accuracies are reasonably lower as compared to the rest of the batch sizes, additionally batch sizes that are too large lead to poor generalization.

Batch size 1 - Even though lower batch size like 1 has the best test accuracy, yet looking at the results of the median epoch time for training a model of batch size 1, it is quite evident that this batch size is the most computationally expensive out of the lot, hence we reject this batch size solely due to the fact that it is the most computationally expensive batch size.

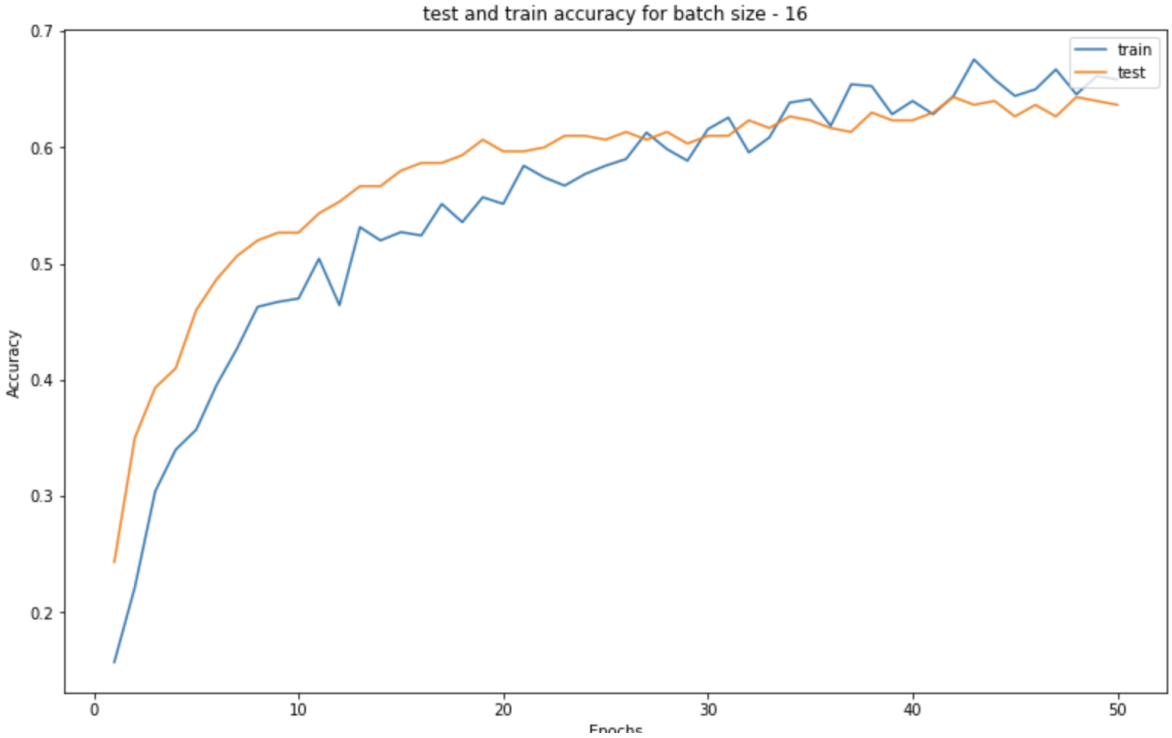
Batch size 4 - Batch size 4 has a test accuracy similar to its counterparts 8 and 16. Moreover, as we are getting similar test accuracies with higher batch sizes , hence we reject batch size **4** because we get similar accuracies for bigger batch sizes that are less computationally expensive as compared to batch size **4**.

Final Verdict - The test accuracies for batch sizes **8** and **16** are quite similar , but given the fact that the model for batch size **16** is less computationally intensive than **8**, hence the optimal batch size for our model should be **16**.

d) **Stochastic gradient descent** - Stochastic gradient descent, often abbreviated SGD, is a variation of the gradient descent algorithm that calculates the error and updates the model for each sample in the training dataset. Updating the model so frequently is more computationally expensive than other configurations of gradient descent thereby taking significantly longer to train models on large datasets. However, the increased model update frequency can result in faster learning on some problems.

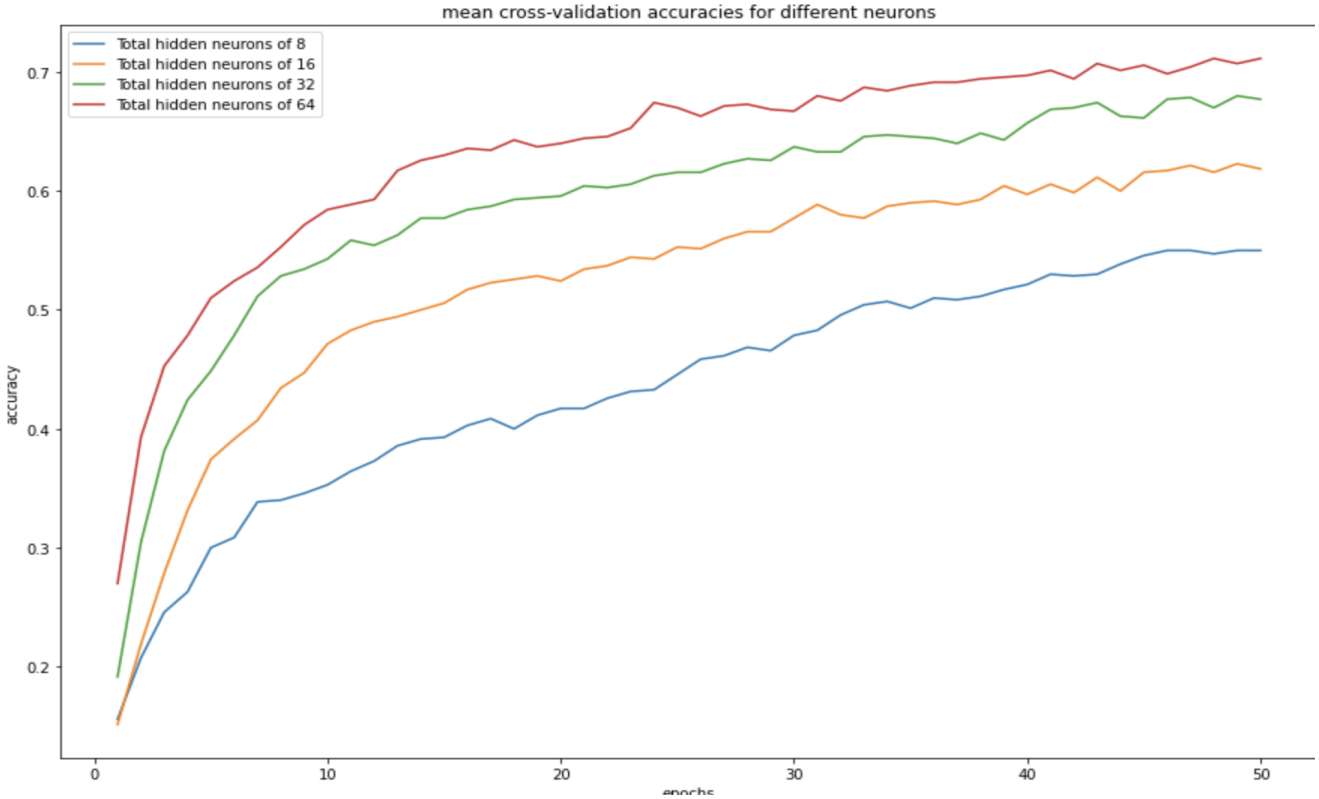
Batch gradient descent - Mini-batch gradient descent is a variation of the gradient descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model weights. The batched updates provide a computationally more efficient process than stochastic gradient descent. Average of the training samples produces stable error gradients and convergence

e) Below is the plot for test and train accuracies against epochs for the chosen **optimal batch size - 16.**



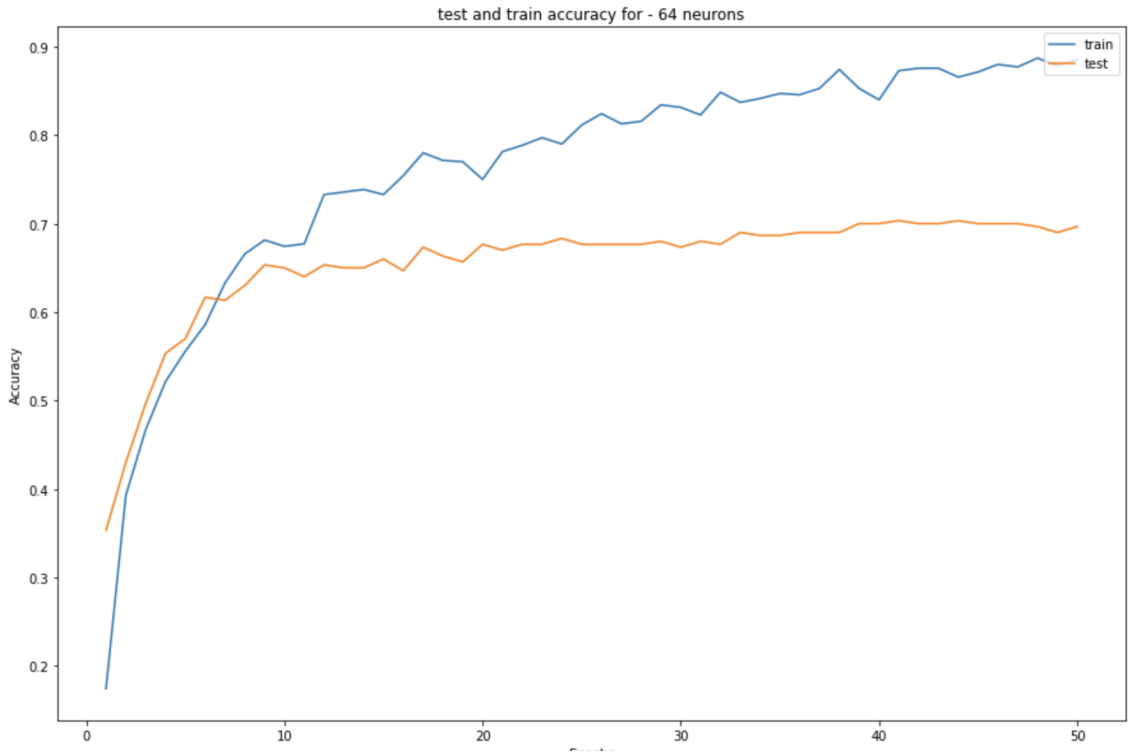
Question 3)

a) Below is the plot for cross-validation accuracies over the training epochs for different number of neurons in the hidden layer - **8,16,32,64**



b) Looking at the above plot we can observe that the mean-cross validation increases with an increase in the total number of neurons in the hidden layer. Hence, it is quite evident that the model with 64 neurons in its hidden layer outperforms the rest of the neuron sizes in the search space in terms of mean cross-validation accuracy. Hence, the optimal number of neurons in the hidden layer of the model is **64**.

c) Below is the plot for train and test accuracies for against training epochs for the optimal number of neurons we selected earlier **(64 neurons).**



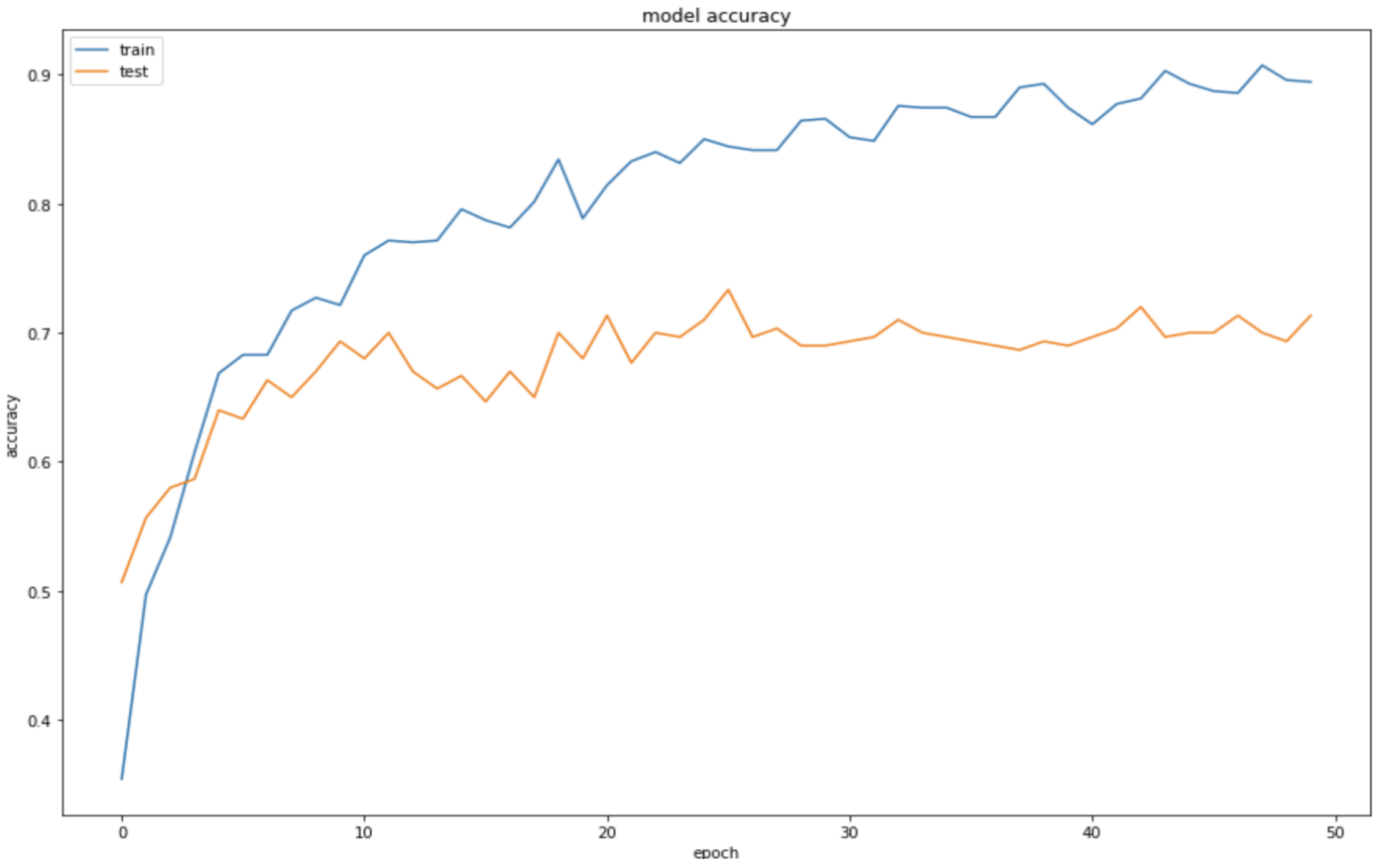
d) Some of the parameters that can be tuned in our current model are -

Learning rate - We can further tune the learning rate so that the model has a higher probability of achieving global minima. For this, we can go one step further by introducing a callback that has a patience of **p,** which reduces the **learning rate** if there is no reduction in loss for **p** continuous epochs.

Activation function - We can tune the activation function of our model since relu can suffer from **dying relu** problem, that is, a large gradient flowing through a **ReLU** neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the **ReLU** units can irreversibly die during training since they can get knocked off the data manifold. Moreover, we can use a popular extension of **ReLU** known as **leaky ReLU** to allow smaller negative values when the input is less than zero.

Question 4)

a) Below is the plot for the train and test accuracies of the 3 layer network against training epochs

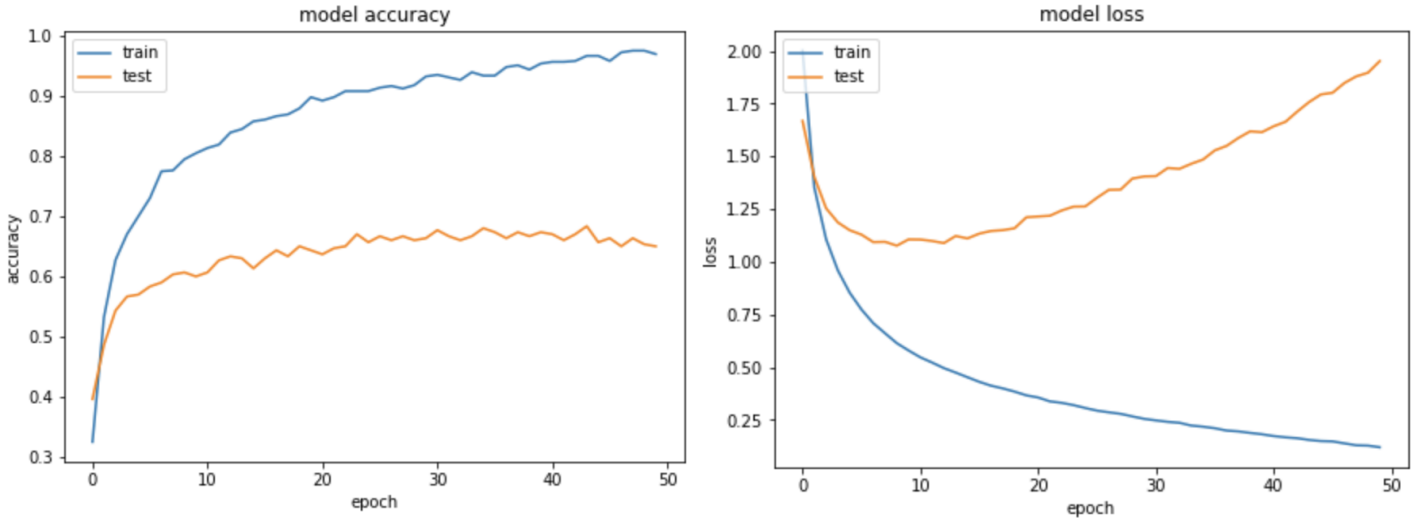


b) The 3-layer model had a slightly higher test accuracy as compared to our optimal 2-layer model, the reason behind the 3 layer model performing better can be due to the fact that a greater number of hidden layers are able to fit more complex decision boundaries. However the optimal 2-layer model had a smoother convergence as compared the 3-layer model, the reason behind this is the fact that in our 3-layer model we use a batch-size of 1 which enables the model to update the weights for every sample , thus making it more susceptible to outliers ,hence as a result we experience fluctuations in the test accuracies graph.While on the other hand, our optimal 2-layer model makes use of the optimal batch-size which enables it to generalise better.

Question 5)

a) Dropout is a technique where randomly selected neurons are ignored during training. They are dropped-out randomly. This means that their contribution to the activation of downstream neurons is temporarily removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.The effect is that the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data.

Below are the plots for train and test accuracy as well as losses for our neural network model without a dropout layer



b) On removing the dropouts, we notice that the test loss of the model reduces till the 10th epoch but then begins to rise steadily from 10th epoch onwards. This effect is coupled with the steady increase in model's train accuracy without any significant change in test accuracy. This goes to show that our model, without the dropout, tends to get overfit.

c) Another approach that we can use to tackle overfitting is to make use of weight regularization, that is, penalize the model during training based on the magnitude of the weights.This will encourage the model to map the inputs to the outputs of the training dataset in such a way that the weights of the model are kept small.

Possible discussion pointers for conclusion

1. Some of the limitations of the current approach (using feed-forward neural network model) are -

    a) Adding more layers in the architecture will enable the model to fit very complex functions but consequently it might cause the model to overfit.
    b) Feed-forward neural networks do not take into consideration the information from the neighbourhood weights which prevents the weights to propagate their learnings

2. The tuning of the hidden layers neurons had the most impact in improving the model's performance , the reason behind the significant improvement was because an increase in the total number of neurons in the hidden layer enables the model to fit more complex functions for the decision boundaries.

3. The alternative approach to achieve genre classification can be to use kernels of varying dimensions to perform feature extraction from the waveforms. Some state of the art machine learning models that we can use for genre classification are Convolutional Neural networks and Recurrent Neural Networks

4.The other potential tasks that modelling waveform data be used for are -

    a) **Sentiment Analysis using the person's voice** - this can be quite helpful in further automating customer service experience by integrating voice commands with automated chat bots for customer service. For this we need waveform data of the customer's voice.

    b) **Drawing comparisons between multiple speakers** - We can use waveform data from the sound produced by different speakers to evaluate their performance.

Some changes that are needed to be made in the pipeline are -

    a) Try out different kernels as well as other parameters like strides and batch normalization to tune a CNN model.

    b) Use techniques to counter vanishing gradients in case a RNN model needs to be used.

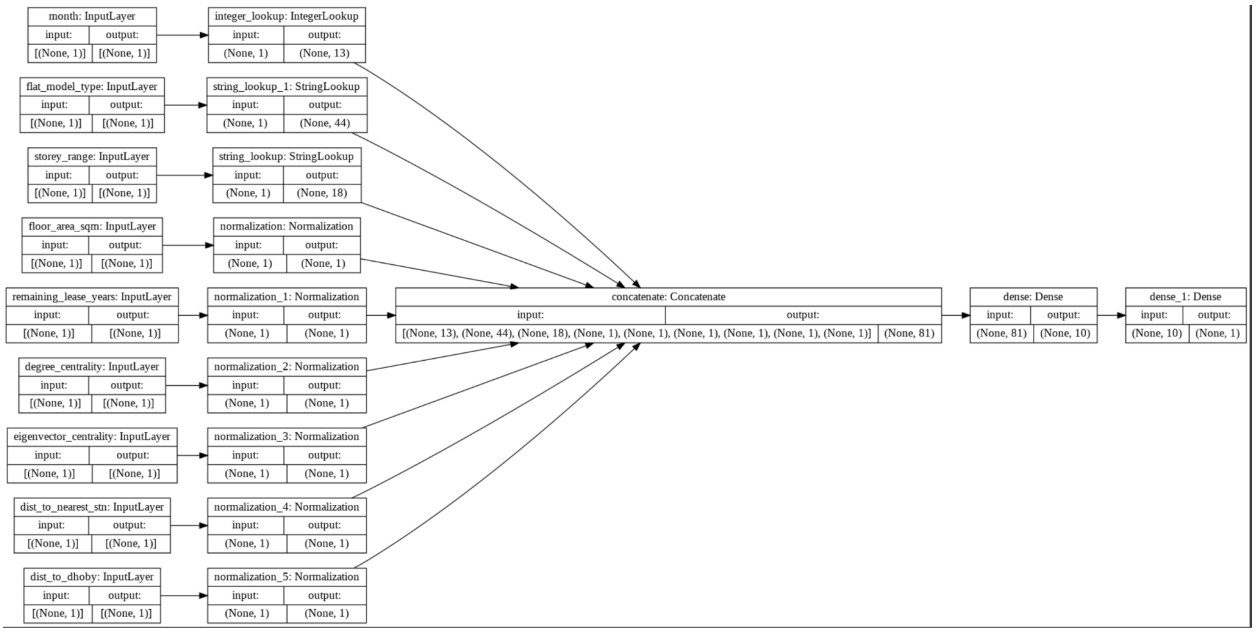
    c) Introduce waveform normalisation as well as scaling. Additionally we can use filters to remove noise or distortions in the waveform data.

Part B

Question 1)

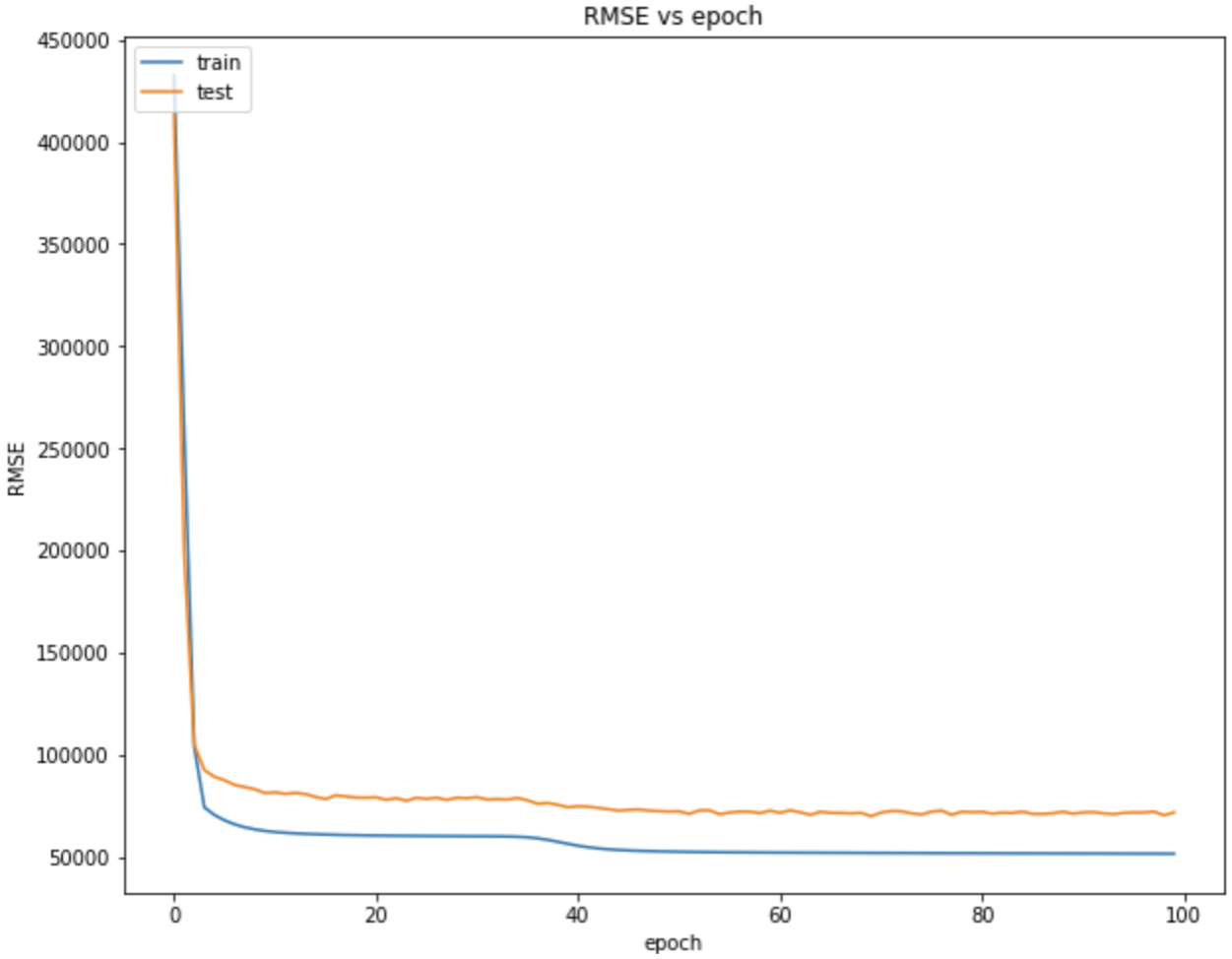
a) The reason behind using data before 2021 as training data and data from 2021 as test data is that the model will be trained on past data and will be tested on the new data that is available. This will enable the model to be robust enough to learn from the past and make predictions for the future. This way we will be able to use any available past data to make future price predictions rather than training the model everytime using the new data and making the predictions as we will never be having data from 2021 if we are currently living in 2021, hence in order to make predictions for the year 2022, we will require data before 2021 and we will be able to test our model when the data from the year 2022 is available.

b) Below is the architecture of the model we built in this question



c) In this part ,we train the model for 100 epochs using mini-batch gradient descent with batch size = 128, using ‘adam’ optimiser with a learning rate of α = 0.05 and mean square error as cost function.

d) Below is the plot for the train and test root mean squared errors against epochs

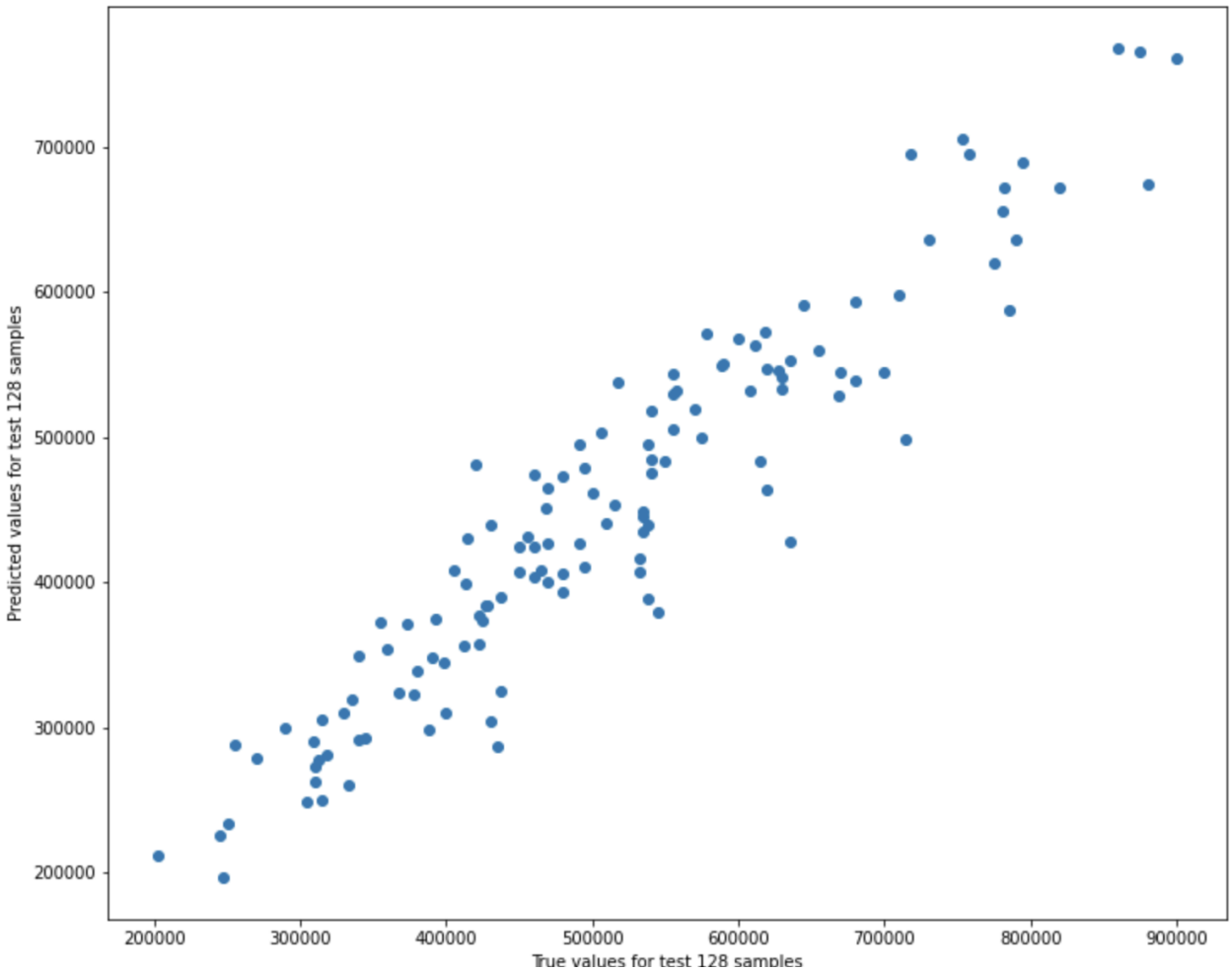


e) The **epoch** with the lowest test error is **70**

R2 score at the best epoch is **0.8073295950889587**
RMSE at the best epoch **70049.67745821533**

f) Below is a scatterplot for the predicted and true values

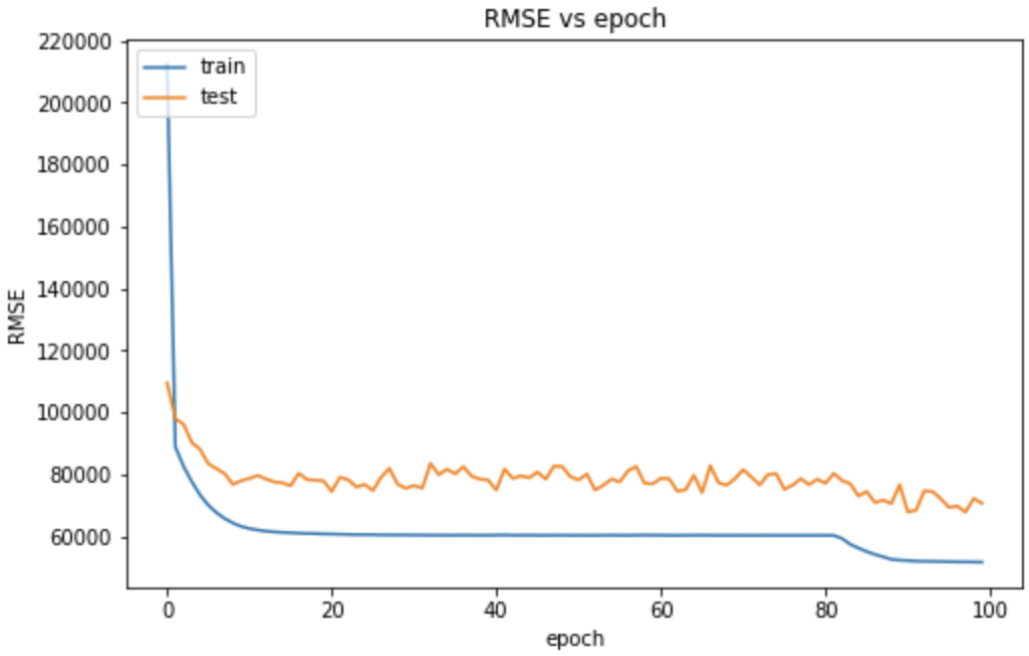
In order to extract a batch from val_ds , we make use of the iterator function available in tensorflow.data.Iterator.



Question 2)

a) In this part, we add an embedding layer output_dim = floor(num_categories/2) after encoding the categorical features.

b) In this part , we make use of the Embedding layer provided in keras and furthermore, we make use of a reshape layer in order for these features to be concatenated with other features. The plot for the train and test RMSE is attached below



c) Result at the best epoch for the model with embeddings
R2 score - **0.8209825754165649** , RMSE - **67840.96980438885**

Result at best epoch for the previous model
R2 score is **0.8073295950889587 ,**RMSE is **70049.67745821533**

The performance of the model got better with the introduction of embeddings for categorical features as the r2 score of the new model increased as well as the rmse score saw a decrease in value as compared to the previous model. This goes to show that the embeddings helped in boosting the performance of our model, the reason behind that is because an embedding captures some of the semantics of the input by placing semantically similar inputs close together in the embedding space, moreover the euclidean distance between similar feature representation will be comparatively smaller as compared to dissimilar feature representations.Thereby enabling more meaningful relationships to be discovered among the categories.

Question 3)

a) In this part we add a callback to introduce early stopping with a patience of 10

b) In this part we perform recursive feature elimination to examine the impact of the removal of a feature on our model's performance.

	Eliminated features	R2_Score	Root_Mean_Square_Error
0	[month]	0.846402	62737.289454
1	[flat_model_type]	0.797243	72014.818031
2	[storey_range]	-9.700381	523243.208980
3	[floor_area_sqm]	0.818185	68233.046305
4	[remaining_lease_years]	0.799237	71601.027926
5	[degree_centrality]	0.828036	66382.821874
6	[eigenvector_centrality]	0.825676	66914.306990
7	[dist_to_nearest_stn]	0.771206	76735.846539
8	[dist_to_dhoby]	0.696147	88409.794932
9	[month, flat_model_type]	0.828193	66270.686612
10	[month, storey_range]	0.829609	65813.331172
11	[month, floor_area_sqm]	0.835079	64864.530832
12	[month, remaining_lease_years]	0.799930	71398.809178
13	[month, degree_centrality]	0.856285	60468.896335
14	[month, eigenvector_centrality]	0.840737	63889.606792
15	[month, dist_to_nearest_stn]	0.783341	74525.585191
16	[month, dist_to_dhoby]	0.730207	83051.350284
17	[month, flat_model_type, degree_centrality]	0.803293	71060.058429
18	[month, storey_range, degree_centrality]	0.820460	68107.341161
19	[month, floor_area_sqm, degree_centrality]	0.820787	67742.378819
20	[month, remaining_lease_years, degree_centrality]	0.782761	74478.111254
21	[month, degree_centrality, eigenvector_central...	0.845159	63025.415572
22	[month, degree_centrality, dist_to_nearest_stn]	0.772755	76353.458494
23	[month, degree_centrality, dist_to_dhoby]	0.688148	89676.730003

c) Results of the model from question 2

R2 score - **0.8209825754165649** , RMSE - **67840.96980438885**

` After running **recursive feature elimination**, we can see that the removal of **month** as the 1st feature led to an increment in the performance as the **r2 score** increased to **0.846402** and rmse decreased to **62737.289454**. Furthermore, the removal of degree_centrality, saw a further increase in **r2 score** to **0.856285** and a decrease in **rmse** to **60468.896335**. Hence, the model trained with the removal of **month** and **degree_centrality** features together turned out to be the best with a **r2 score** of **0.856285** and a **rmse** of **60468.896335**.

d) **Features and their usefulness -**

month - the removal of month had a big impact on the increase in model performance, this goes to show that this feature had **very little correlation** with the output, hence its removal led to a boost in performance.The logical reason behind this is because people don't look at the specific month of the year while buying or selling houses.

flat_model_type - the removal of this feature led to a dip in the performance, thereby making it a **worthy feature** to be considered as a training feature of the model. The logical reason behind this is because different types of flats fall into different price brackets.

storey_range - the removal of this feature led to a high dip in performance, thereby making it one of the **most important input features**. The logical reason behind this is that people take into consideration the storey range as there might be some hdb complexes without an elevator or some people fear height as well.

floor_area_sqm - the removal of this feature led to a dip in the performance, thereby making it a **worthy feature** to be considered as a training feature of the model. The logical reason behind this is that the area of a property is one of the key criterias when it comes to putting a price on that property.

remaining_lease_years - the removal of this feature led to a dip in the performance, thereby making it a **worthy feature** to be considered as a training feature of the model. The logical reason behind this is that the remaining-lease years is a key factor while considering the price of a property.

degree_centrality - the removal of this feature led to an increase in the performance thereby signifying that it has very little or no correlation with the output label. Hence making it a **worthy feature to be removed** from the training features list. The logical reason behind this is that degree centrality is a very vague parameter and it is mostly not considered while coming up with a price for a property.

Eigenvector_centrality - the removal of this feature led to an increase in the performance thereby signifying that it has very little or no correlation with the output label. Hence making it a **worthy feature to be removed from the training features list**. The logical reason behind

this is that eigenvector centrality is a very vague parameter and it is mostly not considered while coming up with a price for a property.

Dist_to_nearest_stn - the removal of this feature led to a dip in the performance, thereby making it a **worthy feature** to be considered as a training feature of the model. The logical reason behind this is that the nearest station is a key criteria for people who wish to go out more often or have to travel to office everyday.

Dist_to_dhoby - the removal of this feature led to a high dip in performance, thereby making it **one of most important input features**. Its removal led to the highest dip making it the feature having the highest correlation with the output label.

Possible discussion pointers for conclusion

1. We can make use of past data analysis studies to come up with the factors responsible for the rise in prices. We can also intuitively use features which a person considers before buying a property like the locality, nearby facilities, distance to the nearest MRT station , distance to the heart of the city ,etc. Alongside that we can use heatmaps for different features against prices to find out the features that have a positive correlation with the increase in the price.

2 .We can add a callback that decays the learning rate if the test error does not decrease for a certain patience p. We can also add more layers to our neural networks so that our model is able to fit more complex functions , and we do not have to worry much about vanishing gradients as relu helps us counter that issue.