Rodney Cordova

Midterm Project

COMP 4449

**Housing Price Prediction:**

**Multi-Input Neural Network**

Between 2009 and 2020 the median sale price of a home in the Denver Metro area more than double from $215,252 to $449,000 (Black, 2021). With such rapidly increasing home prices, evaluating the cost of a home is imperative for buyers and sellers. Buyers seek to maximize the purchasing power of their budget well seller seek to maximize their transactional income. Often the price negotiations occur prior to the completion of an appraisal. As a result, the true value of the home typically is not known until after the purchase contract has been executed. A tool that could accurately estimate home values would be a great benefit to both parties. The objective of this project was to explore the development of a multi-input neural network based model utilizing numeric and image data to predict the sales price of homes in the Denver Metro area.

**I – Dataset:**

There are many open-source housing datasets publicly available. However, these data sets either purely consist of numerical and categorical features or are from housing markets other than the Denver Metro area. Therefore, I decided to generate a new dataset for the purpose of this project. The new dataset was manually compiled using Denver area recently sold homes listed on Zillow.com. The total dataset consists of 639 observations. For each observation four numeric features were collected: number of bedrooms, number of bathrooms total square feet and sale price (the target feature). Additionally, each observation consists of four images: exterior, kitchen, bedroom, and bathroom.

**II – Exploratory Data Analysis:**

To begin EDA, I decided to look at the distribution and box plot for the target variable of price (see Appendix A). The distribution of price appears to be approximately normal with a mean of $602,101.23 and standard deviation of $165,022.82 and the box plot does not indicate any outliers with the range of prices spanning $209,900 to $1,000,000.

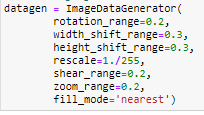
Next joint plots were created between price and each of the other numeric features (see Appendix B). The purpose of the join plot is to visualize any relationships between the target and the other features. Based on the plots, bedrooms and bathrooms take on discrete values between 1 and 6. Those values seem appropriate given the problem space. All three numeric features appear to have a positive relationship with price as illustrated by the upward sloping regression lines. Meaning as the number of bedrooms, bathrooms, and square feet increase the price also increases. This relationship is also evident in the correlation matrix (see Appendix C). The correlation coefficient for each feature range between 0.300 and 0.325 indicating they are all positively correlated.

**III – Feature Engineering:**

Neural networks attempt to learn patterns in the underlying data, as a result major differences in the scale of features may result in the model overestimating the importance of higher ranging features. In our case bedrooms and bathrooms range in the single digits well square feet ranges in the thousands. Therefore, MinMax scaling was applied to each of the numeric features. This scaling converts values to a range of 0 to1 using the formula (Raschka):

.

As previously noted, each observation consists of four images. Training a convolutional neural network (CNN) on four images would require multiple input layers followed by a concatenation layer. Since I will already be concatenating the image inputs with the numeric inputs, I decided to take a different approach and use image mosaicking. Image mosaicking takes multiple input images and outputs a single image that is the union of the other images. This allows us to retain all the information contained in each image well reducing dimensionality. Each observation was represented by a 2 x 2 image set with dimensions (64 x 64 x 3). The image to the right is an example of the mosaic output.

 The next data engineering process used was image augmentation. Convolutional neural networks are very useful in identifying patterns within images. However, they require large amounts of training data, typically tens of thousands of images. When an 80% training 20% test split is applied to the 639 observations in this dataset, we are left with only 511 training observations. Fortunately, image augmentation can be applied to bolster the training set. An image generator is used to apply random combinations of transformations to the existing images. For example, a height shift will move all the pixels up or down and a zoom will randomly zoom in on the image at varying levels. The result is a slightly different image that increases the training variety for the CNN. In total, each of the 511 observations in the training set was used to generate 30 new images resulting in a final training set of 15,330 observations. The figure to the right shows all the transformations I used.

**III – Model Building and tuning:**

Neural networks are a family of machine learning models capable of handling all types of data inputs and modeling many types of problems including regression, classification, and clustering. Different types of neural networks perform better at different tasks. When it comes to image processing convolutional neural networks are among the most prominent thus, I used a CNN for the image data. CNNs are not as ideal for processing numerical data therefore I am using a multi-layer perceptron for that portion.

The general architecture of the model can be seen in Appendix D. The numeric data is input into the multi-layer perceptron of two dense layers with 8 and 4 neurons respectively with a relu activation. At the same time the image data is passed into the CNN. The CNN consists of three filters (16,32,64) each made up of a 2D convolution layer, batch normalization, and a max pooling layer. Again, relu activation is used and 0.5 dropout is applied the final dense layer contains 4 neurons just like the multiple-layer perceptron. Next a concatenation layer is applied. This layer combines the dense layers from the CNN and MLP. The reason 4 neurons are used in the final dense layers of the CNN and MLP was for those inputs to have equal weight in the final model. If one input had more neurons going into the concatenation it would have more weight in the final model. Domain knowledge could be used to tweak these dense layers but for now I decided to keep them equal. Finally, the concatenation is passed to a fully connected layer and a linear activation to generate the output. The model summary can be seen in Appendix E.

For the optimizer I decided to use Adam with a learning rate of 0.1. Adam is the most widely use optimizer for images, so I decided to stick with it here. For a loss function, I decided to use mean absolute percentage error (MAPE). The difference between the actual value and predicted value as a percentage of the actual value. For example, a MAPE of 5% would mean the average prediction is plus or minus 5% of the actual value. When getting a mortgage to purchase a home, one of the most important factors lenders use to determine eligibility is the loan to value ratio (LTV) thus it is important to accurately estimate the LTV based on the home price. The LTV variance on the loan will be minimized when the MAPE is also minimized. The calculation of MAPE is (Kim & Kim 2016):

.

The initial model was trained for 20 epochs with a batch of 10. The resulting MAPE when predicting the test set was 32.88%. Looking at the graph of the loss, the training loss decreased normally but the test loss was wildly erratic and never flattened so further model tuning is needed (see Appendix F).

Typically for model tuning I would use grid search cross-validation. With grid search you can pass a parameter grid of all the hyperparameters to be tested then it will loop over the combinations of hyperparameters preforming k-fold cross-validation and returned the parameter set that preformed the best. Unfortunately, I ran into issues when trying to use grid search cross-validation related to the dimensionality of my input model. I was not able to find a solution to using a multi-input model with sklearn’s implementations of grid search cross-validation, random search cross-validation or cross-validation. Instead, I used a series of loops to train models with different hyperparameters and utilized those that preformed best according to MAPE in the final model. For example, batch sizes of 10,30,50,75,100 were tested with 10 preforming the best. This process was repeated for the tuning of epochs and the learning rate. In the final model, the batch size was set to 10 with 50 epochs and a learning rate of 0.1.

**IV - Results and Conclusion:**

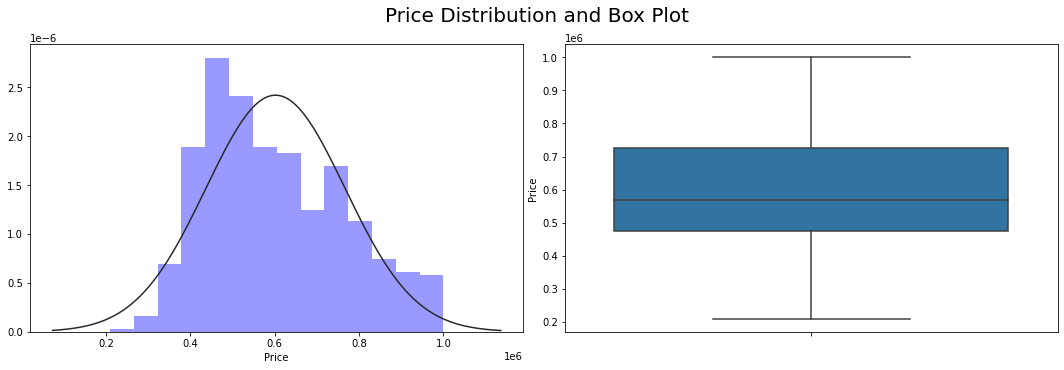
Using the hyperparameters from the previous step the final model was trained. When the model is used to predict the test values, the resulting MAPE was 19.20%. Meaning the average predicted house price varied from the actual price by 19.20%. The average house price in the dataset was $602,101.24 meaning the average predicted price was ± $115,603.43 off the mean. If we examine the plot of the loss (see Appendix G), we see that the training loss decreased at a relatively constant rate before flattening out well the test loss was erratic but centered around 20%. With such a high error, this model does not do a particularly good job at predicting house sales prices.

Despite the poor performance, I was curious about the impact of combining the numerical and image data on the performance of the model. To test this, I trained three models just using the numeric data. The three models used were a support vector regressor, KNN regressor and a decision tree regressor. The blow chart shows that the neural network using numeric and image data outperformed the other three models using just numeric data. This leads me to believe the added complexity of the images does improve the predictive capacity of the model.

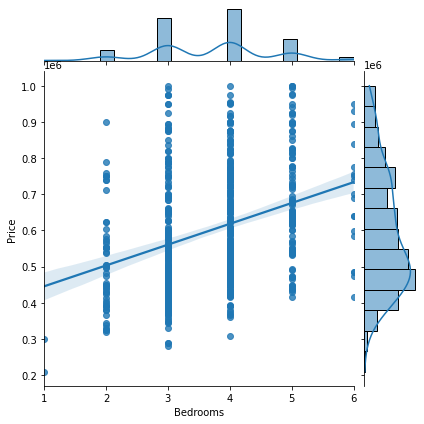


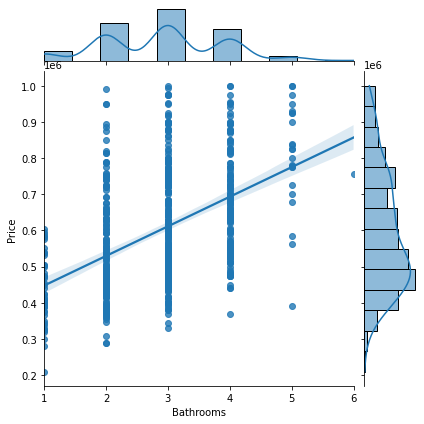
Despite the model’s underperformance, I believe this is a good starting point for future analysis. One of the major limitations on this models’ performance is the dataset. Well the images were useful, the numeric portion only contained three explanatory features. When compared to other housing datasets such as the Bostin and California housing datasets that have 10 and 14 features respectively this dataset falls short. The increased complexity of these datasets improves the ability of models to learn patterns within the data. Geographical data could really bolster the performance of this model. Real estate prices are heavily dictated by location. Even with the Denver Metro area, two identical houses one located in Capital Hill and one located in Aurora would have drastically different prices. Overall, more adding more features would be a benefit to this model.

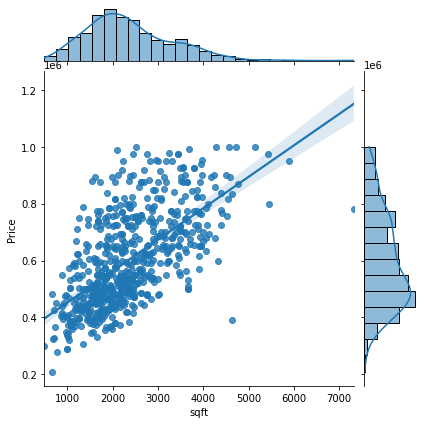
**Appendix A:**

****

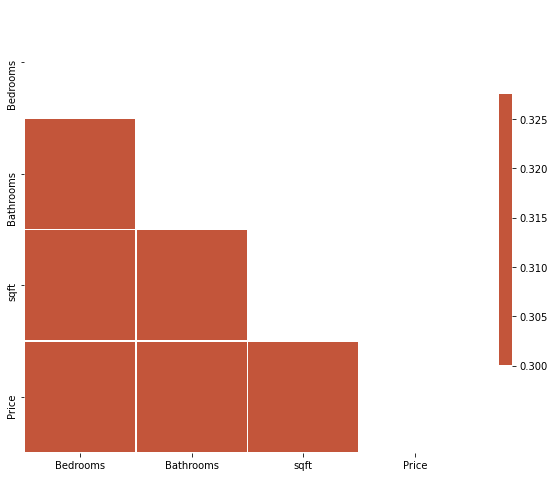
**Appendix B:**

****

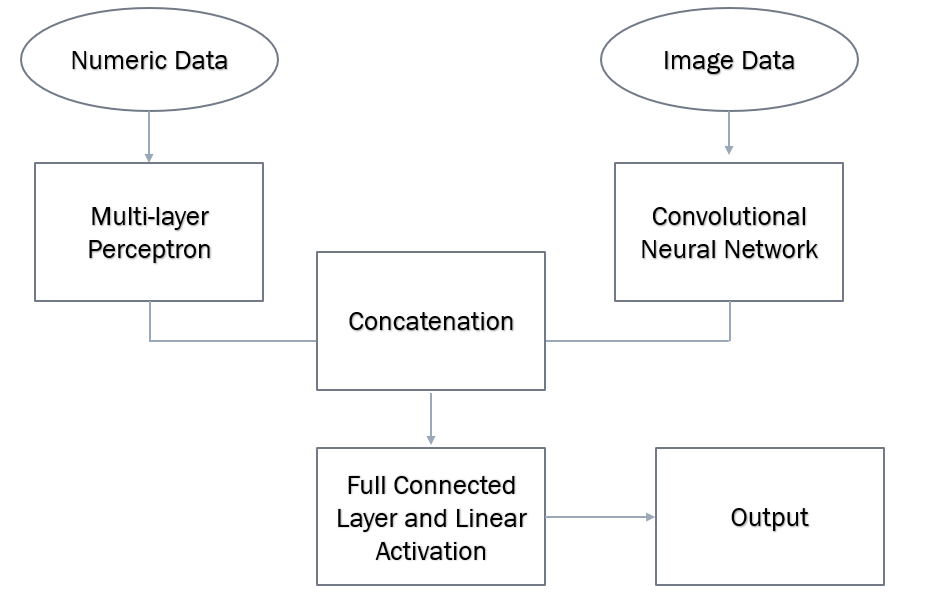
****

****

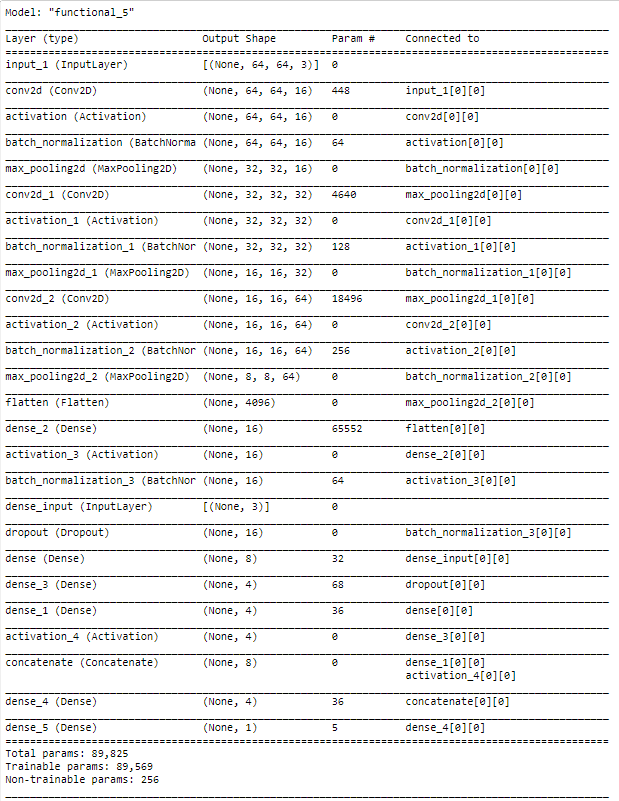
**Appendix C:**

****

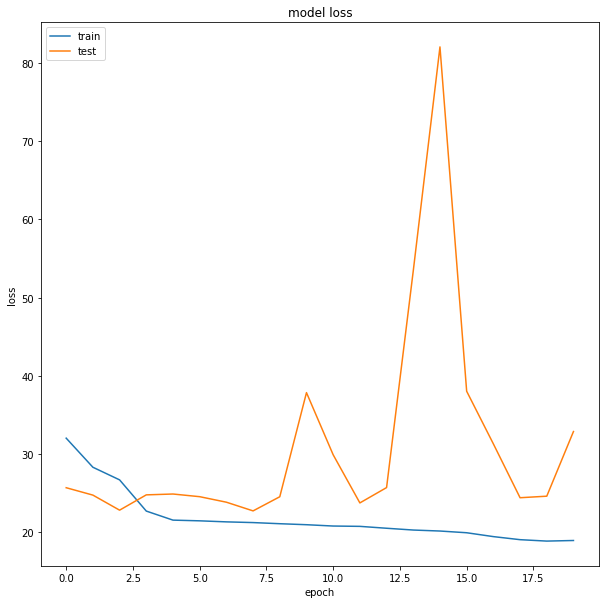
**Appendix D:**



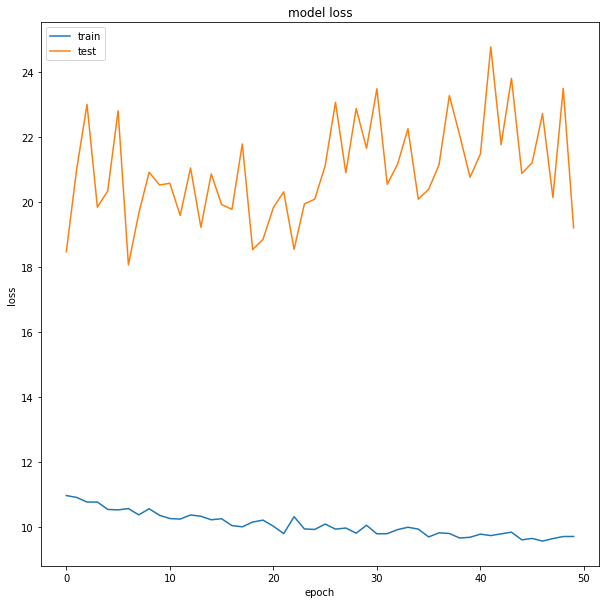
**Appendix E:**

****

**Appendix F:**

****

**Appendix G:**

****

References

Black, J. (2021, February 10). A 10 Year Look at the Denver Real Estate Market. Retrieved May 01, 2021, from https://blog.usajrealty.com/posts/a-look-at-the-denver-real-estate-market

Kim, S., & Kim, H. (2016, March 16). A new metric of absolute percentage error for intermittent demand forecasts. Retrieved May 01, 2021, from https://www.sciencedirect.com/science/article/pii/S0169207016000121

Raschka, S. (n.d.). Minmax scaling. Retrieved May 01, 2021, from http://rasbt.github.io/mlxtend/user\_guide/preprocessing/minmax\_scaling/