Predicting Home Prices from Zillow Images

Springboard Data Science Final Capstone Project

By Sangeeta Jayakar, PhD - August 28, 2021

Introduction

Real estate has been a hot topic lately, especially with the recent record low interest rates. With more people working from home, many young adults have been leaving big cities with high rental prices and heading to suburbs to purchase homes of their own. One area in the US that has seen a rapid influx of new residents is Austin, TX. This increase in demand for homes has led to a dramatic increase in house prices. The median price of a home in Austin was $470,000 in June 2020 and in one year went up to $575,000, according to realtor.com.

A hobby of many people since the start of the pandemic was to scroll through real estate apps such as Zillow and peruse through the listings, looking at the many pictures of the homes for sale. Zillow saw 9.6 billion visits to its website in 2020, up from the previous year by 1.5 billion. Zillow listings include information about the features of the homes, all pricing and sales history of its properties, and also include pictures of the homes for sale.

While there are many data science datasets for predicting house prices based on a number of features of the homes, I was intrigued by finding this dataset on Kaggle of Zillow listings for Austin, TX. It was created by scraping the first image of each home listing, along with other descriptive data about the homes (ie. number of rooms, square feet, etc). I wanted to use the images from the Zillow listings and use deep neural network modeling try to predict the prices of the homes.

Problem Statement:

Can machine modeling with input from images taken from Zillow listings be used to accurately predict home prices?

Data Wrangling and Cleaning

The dataset was downloaded from: https://www.kaggle.com/ericpierce/austinhousingprices. It consisted of a folder containing 15,171 images pulled from Zillow in JPEG format along with an accompanying CSV file containing an entry for each image. After downloading the entire data set from Kaggle, I first read the CSV file into a Pandas DataFrame to inspect it. There were 47 columns of information, including the name of the image file, the prices of the home, and other descriptors of the property.

The target feature was identified as the column 'latestPrice' which contained a float number indicating the price of the homes. Another very important column in the dataset is the 'ImageFile' column, which contains a string with the name of the image jpeg file. For the purpose of this project, which was to mainly focus on analyzing the images, I created a new DataFrame called 'image_price_df' with only the target feature and the column containing the names of the image files. The names of the columns were renamed to "Image Name" and "Price" for simplicity (Figure 1). The 'image_price_df' DataFrame was saved as a CSV file so it could later be used during pre-processing, where it was used to create a function to load the image. That function will be used in the data generator to load data in batches into the model.

	Image Name	Price
0	111373431_ffce26843283d3365c11d81b8e6bdc6f-p_f...	305000
1	120900430_8255c127be8dcf0a1a18b7563d987088-p_f...	295000
2	2084491383_a2ad649e1a7a098111dcea084a11c855-p_...	256125
3	120901374_b469367a619da85b1f5ceb69b675d88e-p_f...	240000
4	60134862_b1a48a3df3f111e005bb913873e98ce2-p_f.jpg	239900
...	...	...
15166	29512934_ff9b6eefa7e2eb4e9ef983da13a23098-p_f.jpg	330000
15167	241937773_66d3e483bd783eac5a52ff5f938d2a2e-p_f...	550000
15168	29473281_9e90ec4652c4b3b6592a7fdd09f1ea6d-p_f.jpg	875000
15169	29392029_a9a8306ea363d23f37d91d37975a1b96-p_f.jpg	420000
15170	29390174_9580c60e14870498b2ce98b5f889471e-p_f.jpg	374900

15012 rows × 2 columns

Fig 1. 'image_price_df' DataFrame was saved as a CSV file and used in the pre-processing stage for loading images into the model.

Putting aside the CSV file with the information on each listing, I next moved on to processing the images. First, using the Python Image Library (PIL), I found out that the sizes of all of the images were different. They ranged from 900 to 1100 pixels by 500-800 pixels. Since there were all in a rectangular shape, I resized them using PIL's resize function so that they were all 250x300 pixels. Now all the pictures had the shape (250, 300, 3) with the last 3 indicating that they are color RGB images (Figure 2). I directly saved these resized images into a new folder on my computer called "ResizedHomeImages'. Realizing that grayscale images would allow for faster training than RGB, I used PIL's convert function to convert the resized images to grayscale (Figure 3). These images were also saved directly to a new folder, called "ResizedGrayImages". These folders were compressed and uploaded to my Google Drive where I could import and unzip the files in Google Colaboratory. The rest of the project after the initial Data Wrangling and Exploratory Data Analysis sections were all completed in Google Colab.

Fig 2. Examples of six random home images from the dataset, all resized to 250x300 pixels in RGB format.

Fig 3. RGB Images were converted to Grayscale for faster training time.

After initial modeling on the images alone, I realized adding in features data about the home listings would be helpful, so I went back to the CSV file with all of the home features. This dataset contained 47 columns containing information about the homes such as square feet, number of bedrooms/bathrooms, as well as information about schools. There were also boolean columns with information such as 'has garage,' 'has parking,' 'has HOA,' or 'has a view'. Some things to note about this dataset is that there were no null values. There were however some odd values that needed to be removed. For example, there was one row that had a listing with 27 bathrooms. A quick check of that row made it apparent that it was an error, probably meant to say 2 or 2.5 bathrooms. This row was deleted. Another row with 20 bedrooms was removed after checking that entry and concluding that it must be an error. There were 22 rows that had unusually low prices, as low as $5500. This is obviously not possible. I checked Zillow listings for Austin currently and the lowest listings were in the range of $70,000. I decided to drop the 22 rows that had values less than 70,000 for the price. Lastly, I dropped rows that had '0' listed for number of bedrooms or number of bathrooms. After this cleaning, there were 15,102 home listings out of the 15,171 that were used in the analysis.

I also created a new DataFrame containing the features of the homes that I thought would help increase the performance of the model:

['zipcode', 'livingAreaSqFt', 'numOfBathrooms', 'numOfBedrooms',
'numOfPrimarySchools','numOfHighSchools', 'avgSchoolRating']

These features will be explored in the next section.

Exploratory Data Analysis

I first wanted to explore the target feature, 'latestPrice' so I plotted a histogram of the prices to see the distribution. There were some key observations made from plotting the distribution. The majority of the homes are below $1M with the average price around $500K. Then there are a smaller number of homes that are very expensive going up to $13M. The distribution is skewed to the right (Figure 4, top). Additionally, plotting the prices on a log scale made it easier to see the distribution of the lower to mid-range priced homes (Figure 4, bottom). I also counted the unique values for price and there were 2,345 unique values.

Other features were explored next. The majority of the homes were single-family homes, while some were condos or townhouses. The number of bedrooms and bathrooms were plotted in a histogram to see their distribution (Figure 5, top). Both of these were normally distributed with a slight skew to the right. This probably indicates the more expensive homes that have more bedrooms and bathrooms. Most homes had

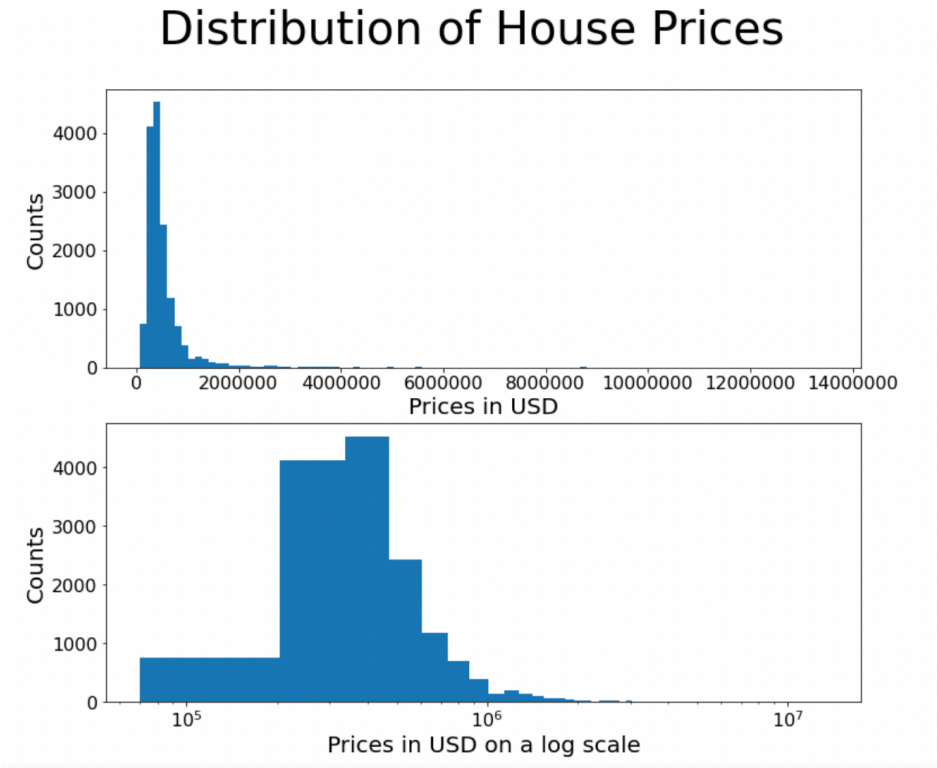

Fig 4. Distribution of home prices on a linear scale (top) and log scale (bottom).

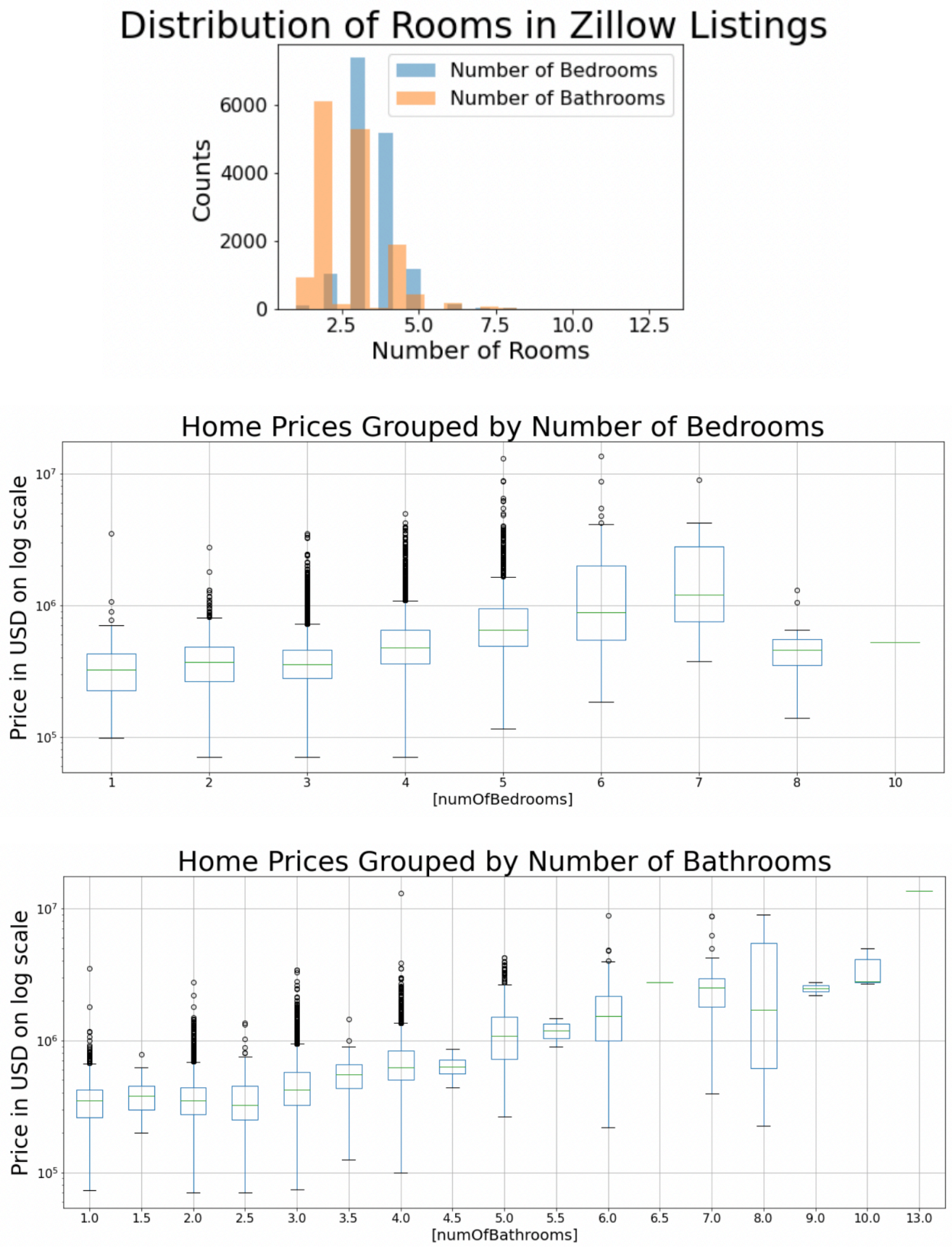

Fig 5. Distribution of bedrooms and bathrooms in Zillow home listings shows most common number of rooms (top). Box plots show prices increase as number of bedrooms (middle) and bathrooms (bottom) increase.

two or three bathrooms, and three or four bedrooms. When home price was plotted as a box plot grouped by the number of bedrooms and bathrooms, there was a trend that home price increases as number of bedrooms and bathrooms increases (Figure 5).

The living area feature was also explored. After some listings with unusually high values were dropped, the distribution of the values were plotted in a histogram Figure 6, left). The home prices were also plotted against the values for living area. There is clearly an upward trend in price with increased living area (Figure 6, right).

I also examined the average school rating and zip code columns, as these may also be related to the home prices. I converted the zip code column to a string format so that it could be treated as a categorical feature. Then, I plotted the price in a box plot grouped by the average school rating and also the zip code. Grouping by average school rating shows that price tends to increase with higher rated schools (Figure 7). Grouping by zip code shows that variation between the price distributions among the different areas in Austin (Figure 8).

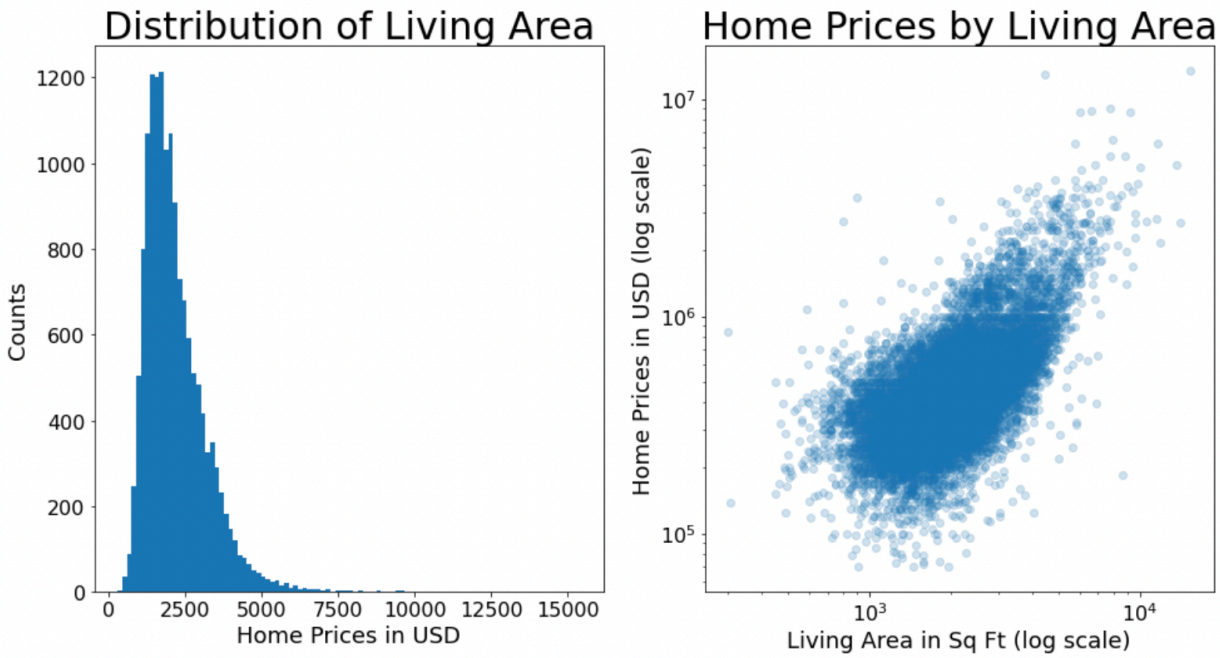

Fig 6. Histogram of living area square feet values shows a normal distribution with a skew to the right (left). Scatterplot of home prices vs living area both on a log scale shows an increase in price as living area increase.

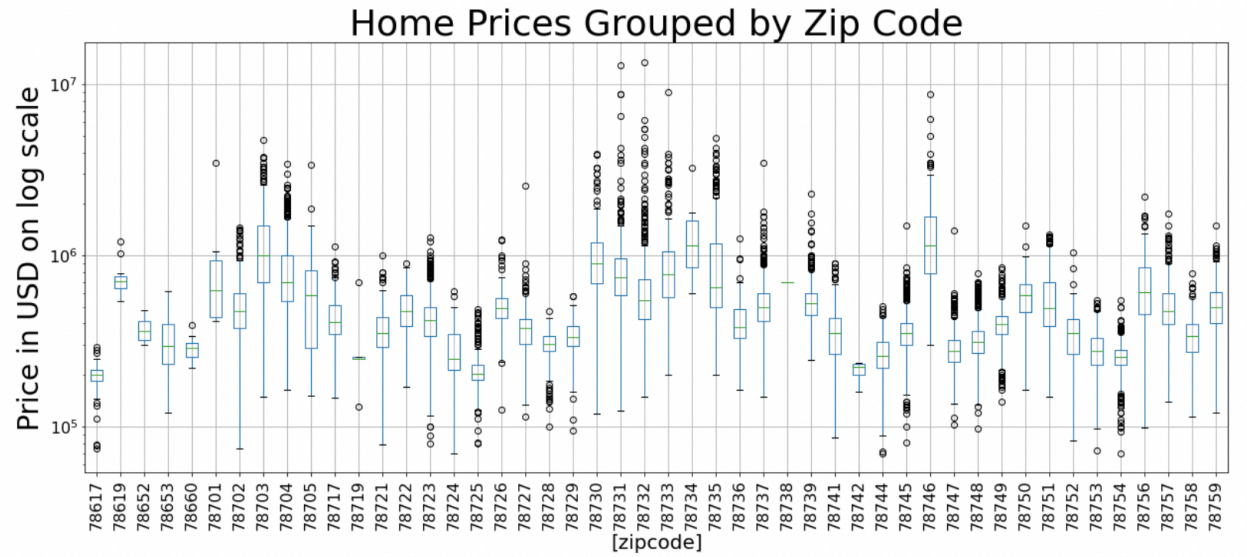

Fig 7. Home prices vary greatly based on zip code.

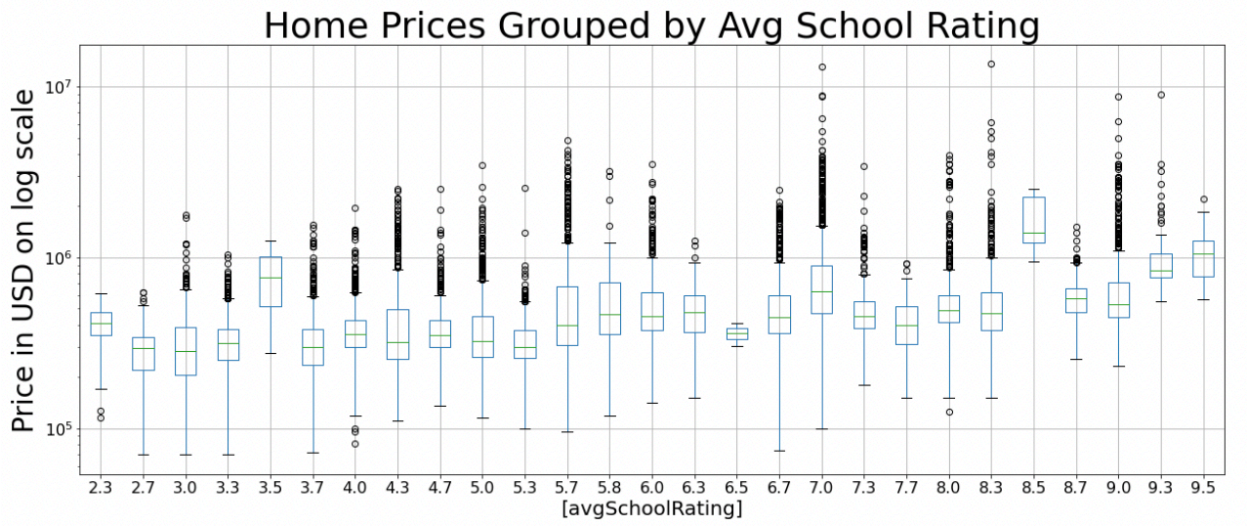

Fig 8. The more expensive homes tend to have higher school ratings.

I created a heatmap to visualize all of the numerical features that were correlated with price (Figure 9). Many of the features had to do with schools. Number of primary, elementary, high schools, and average school rating all appeared to have a relationship to price. Also, number of bedrooms, bathrooms, and living area all seemed to have a correlation with price in the heat map, agreeing with the findings from the EDA. After

confirming the features that are possibly correlated with home price, another new DataFrame was created, called **'features_df'** that included the following columns:

['zipcode', 'livingAreaSqFt', 'numOfBathrooms', 'numOfBedrooms', 'numOfPrimarySchools','numOfHighSchools', 'avgSchoolRating']

This DataFrame was saved as a CSV file to be used in the pre-processing section and modeling section. It will allow additional features for the home listings to be used in the modeling in addition to the images.

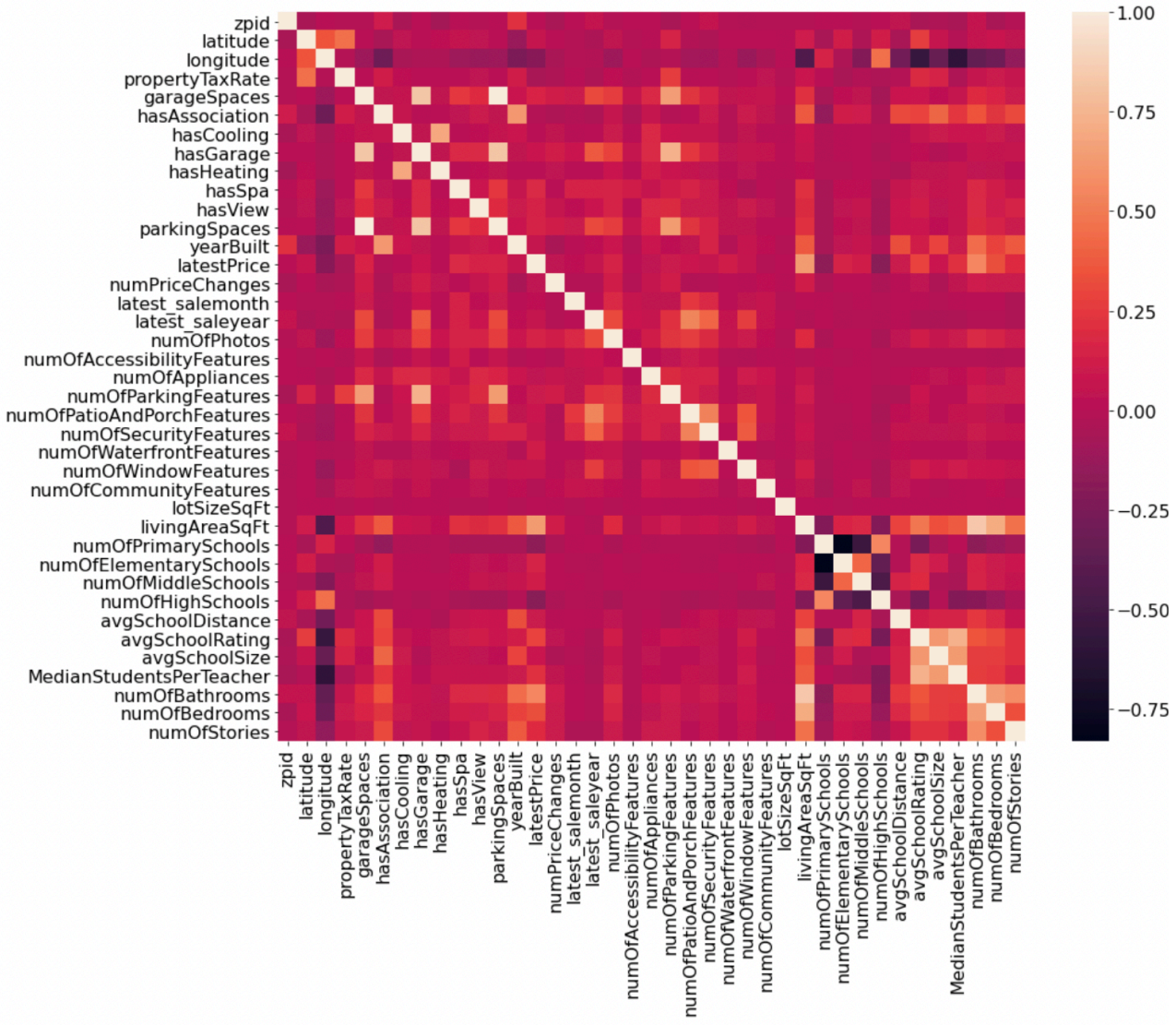

Fig 9. Heatmap shows correlated features with home price.

Pre-processing

The DataFrame **features_df** was pre-processed to be in a format used for machine learning. First, certain features were scaled. The 'livingAreaSqFt' was normalized by dividing each value by the standard deviation of all of the values. The other columns did not need to be scaled. The 'zip code' column had the zip codes in the integer data type. I wanted to use the zip codes as a categorical feature, so I converted them to strings, and then replaced the original zip code column with dummy variables. Appending the dummy variables onto 'features_df' resulted in a DataFrame containing 55 columns to be used in the model.

Modeling

Initial CNN Model (Images only): The first attempt at building a convolutional neural network model utilized only the resized grayscale images. I had all of the images in my Google Drive, but in order to feed the images to the model during training, I created a Data Generator following a tutorial specifically for image data. By

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 250, 300, 32)      320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 125, 150, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 125, 150, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 62, 75, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 62, 75, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 31, 37, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 31, 37, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 15, 18, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 34560)             0
_________________________________________________________________
dropout (Dropout)            (None, 34560)             0
_________________________________________________________________
dense (Dense)                (None, 512)               17695232
_________________________________________________________________
dense_1 (Dense)              (None, 32)                16416
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 17,951,937
Trainable params: 17,951,937
Non-trainable params: 0
_________________________________________________________________
None
```

Fig 10. Baseline CNN architecture used on images only.

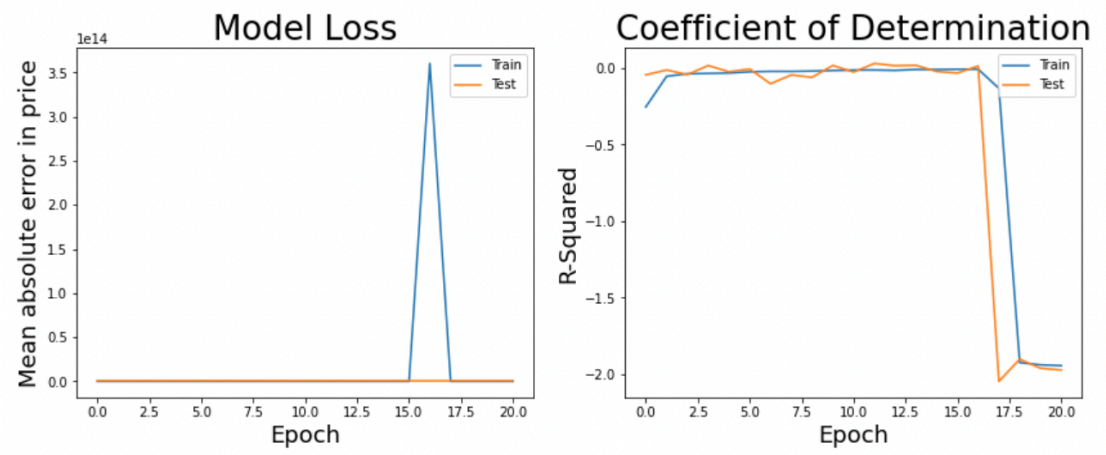

Figure 11. CNN Model using only images performed poorly, with an R-squared score of 0.03.

specifying a batch size of 128, images were fed into the model in small batches. Training the model took place over 40 epochs, where during each epoch the entire training data were passed through the CNN model. Convolutional layers were created with filters of size (3,3) and went from less filters to more filters starting with 32 and going up to 128 (Figure 10). Using only the images resulted in a poor performing model, only getting an R-squared score of 0.03.

MLP Model with (features data only): Artificial Neural Network, or Multi-Later-Perceptron. The features DataFrame was tested in a simple MLP model and resulted in an R-squared score of 0.57 (Figure 12). This was promising, so the next step was to try combining the images and features into one model.

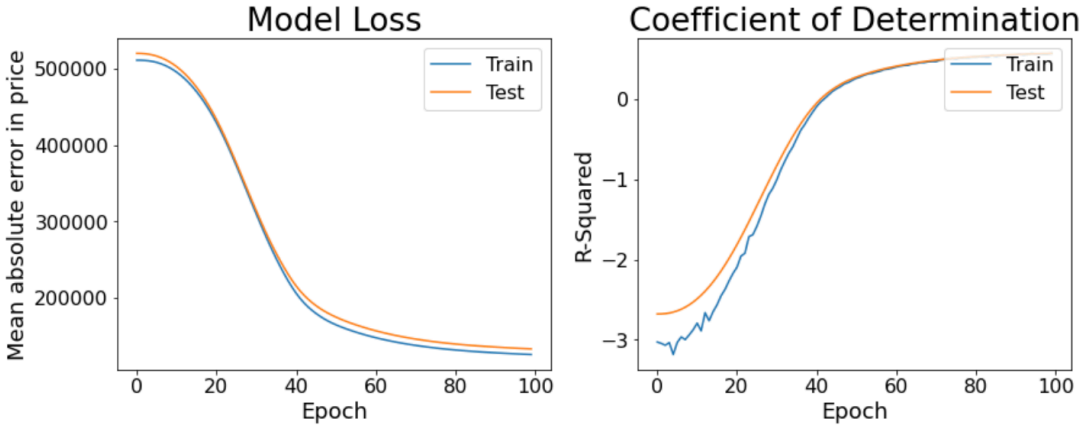

Fig 12. MLP model performance on Home features only was better than images only, resulting in an R-squared score of 0.57.

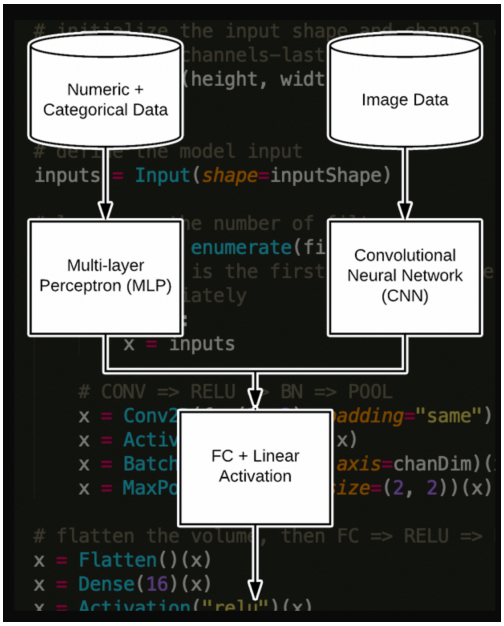

Fig 13. Schematic of neural network model with mixed inputs.

Mixed Inputs Model (images and features): In order to increase the model performance, a new model was created by concatenating the two models, the CNN model for images and the MLP for features. This works by taking the data from the data generator and feeds the image data to the CNN, the features data to the MLP, and then the outputs from those models merge into the final dense layers and then a single output is generated. A schematic for this model can be seen in Figure 13. I found that

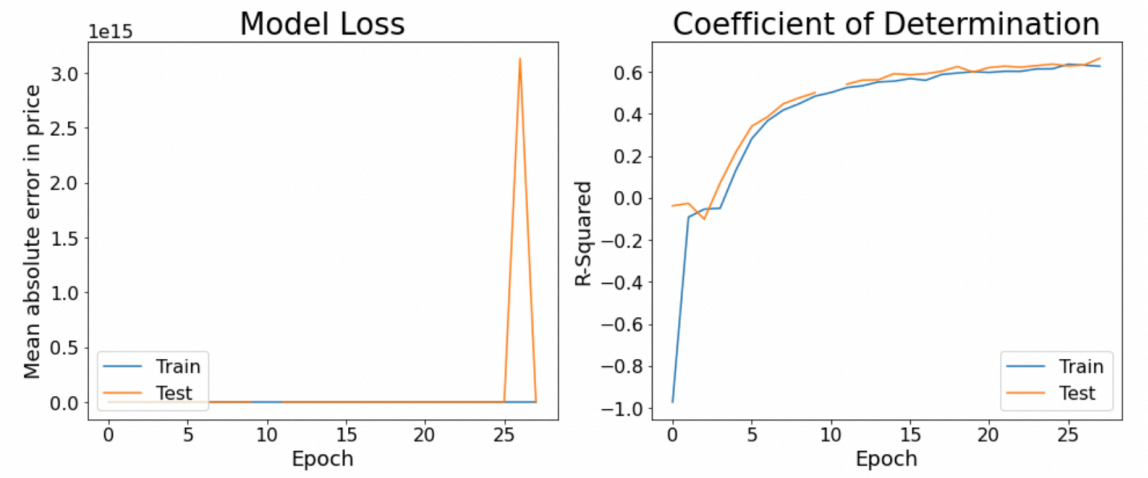

Figure 14. Mixed inputs model performance (notebook 5) with an R-squared score of 0.66 for validation set.

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
conv2d_input (InputLayer)       [(None, 250, 300, 1) 0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 250, 300, 32) 320         conv2d_input[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 125, 150, 32) 0           conv2d[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 125, 150, 64) 18496       max_pooling2d[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 62, 75, 64)   0           conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 62, 75, 128)  73856       max_pooling2d_1[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 31, 37, 128)  0           conv2d_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 31, 37, 128)  147584      max_pooling2d_2[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 15, 18, 128)  0           conv2d_3[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 34560)        0           max_pooling2d_3[0][0]
__________________________________________________________________________________________________
dense_input (InputLayer)        [(None, 54)]         0
__________________________________________________________________________________________________
dropout (Dropout)               (None, 34560)        0           flatten[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 16)           880         dense_input[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 512)          17695232    dropout[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 8)            136         dense[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 32)           16416       dense_3[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 1)            9           dense_1[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 1)            33          dense_4[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 2)            0           dense_2[0][0]
                                                                 dense_5[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 4)            12          concatenate[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                 (None, 1)            5           dense_6[0][0]
==================================================================================================
Total params: 17,952,979
Trainable params: 17,952,979
Non-trainable params: 0
__________________________________________________________________________________________________
```

Fig 15. Architecture of the mixed input CNN model shows how image data and features data were incorporated and compiled into the final model.

this model with mixed inputs performed better than either model individually (Figure 14), with an R-squared score of 0.66. The first model was a basic combination of the first two models tested. The full architecture of the model can be seen in Figure 15.

Mixed Inputs Model (with additional modifications):

After the initial mixed inputs model, I tried making various modifications to see if the performance of the model would improve. First I tried scaling the target feature, by dividing the price column by the standard deviation. This improved the model performance, resulting in an R-squared score of 0.70 (Figure 16).

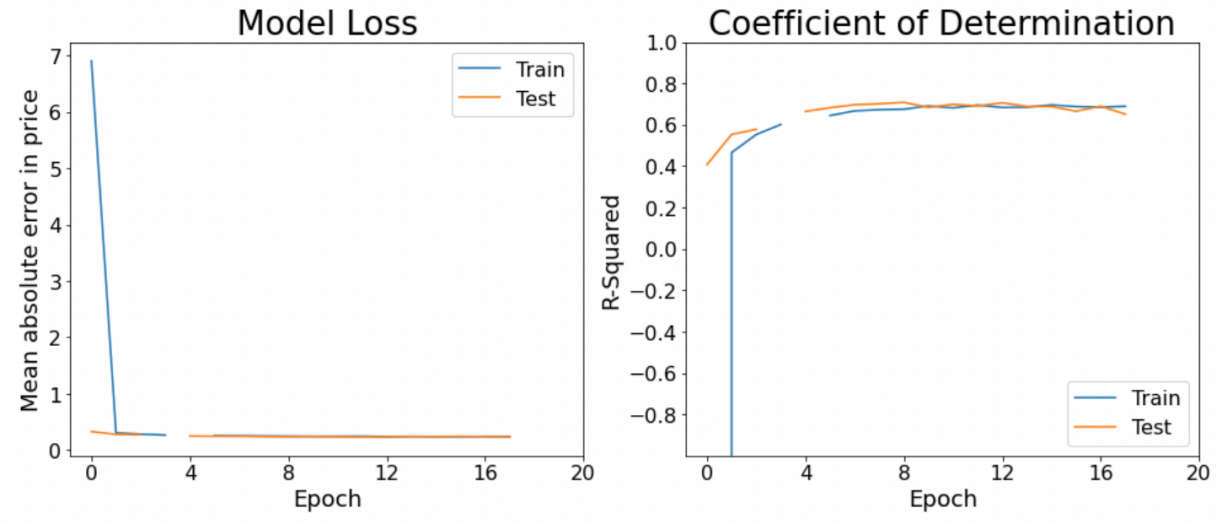

Fig 16. Scaling the target feature resulted in a slight increase in model performance, with an R-squared score of 0.70.

Then I tried adding another dense layer in the MLP model. This resulted in an R-squared score of 0.71 (Figure 17). Next I tried increasing the kernel size from 3x3 to 7x7. This resulted in an R-squared score of 0.69. Increasing the kernel size was done to try to see if any other patterns could be seen in filters (Figure 18).

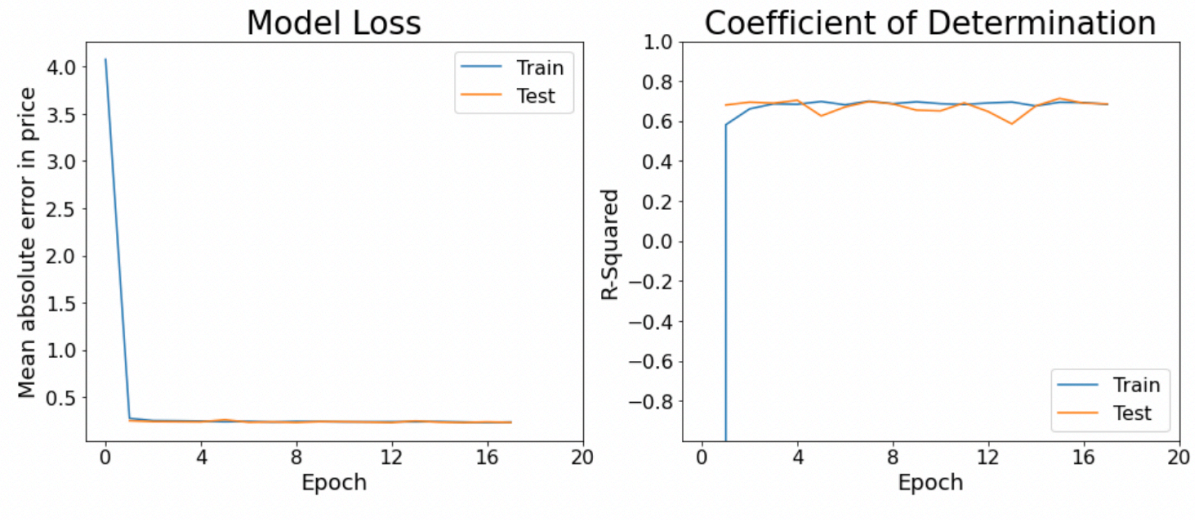

Fig 17. Scaling the target feature and also adding an additional dense layer resulted in the best model performance, with an R-squared score of 0.71.

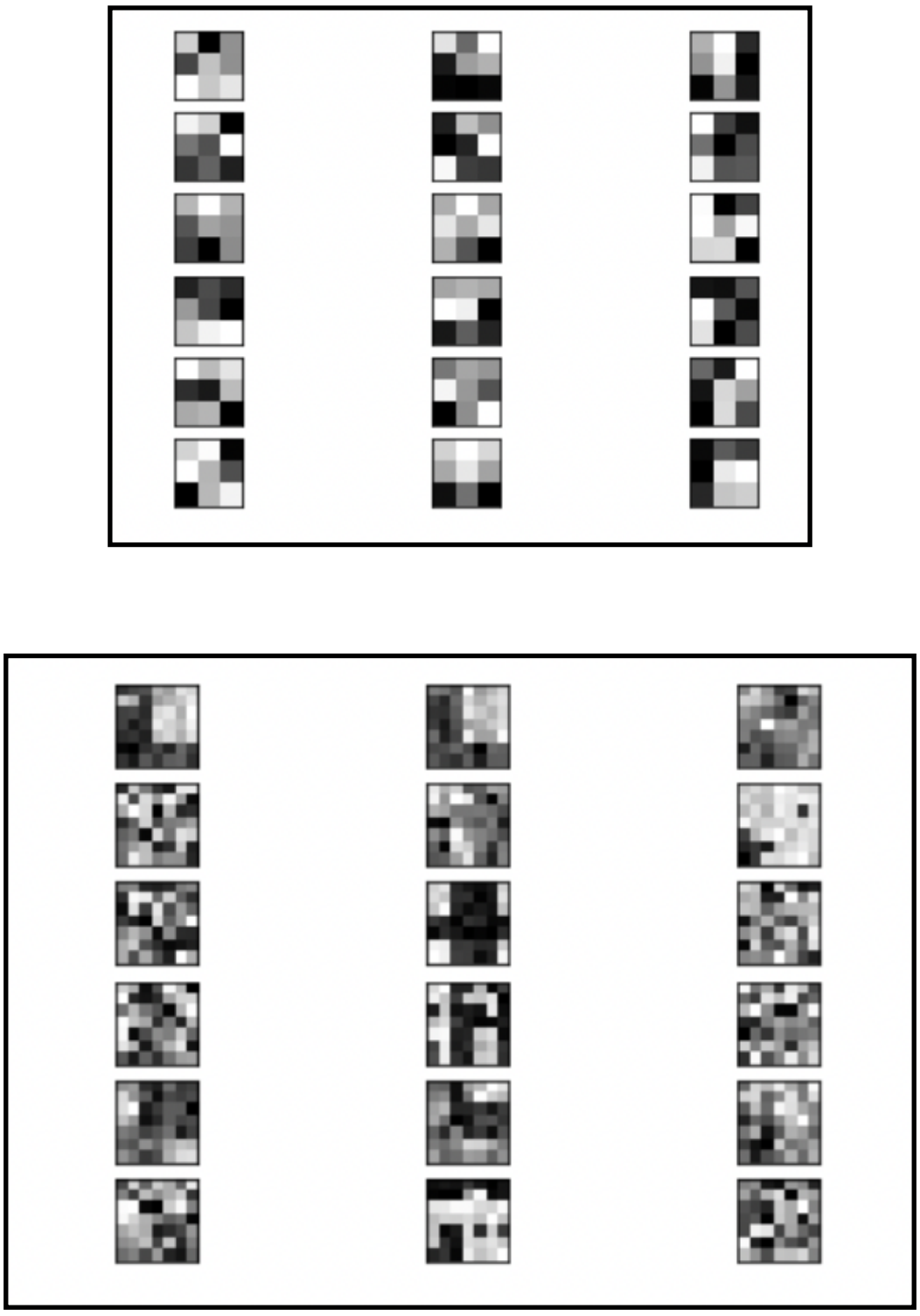

Figure 18. Filters with 3x3 kernel size (top) and filters with 7x7 kernel size (bottom).

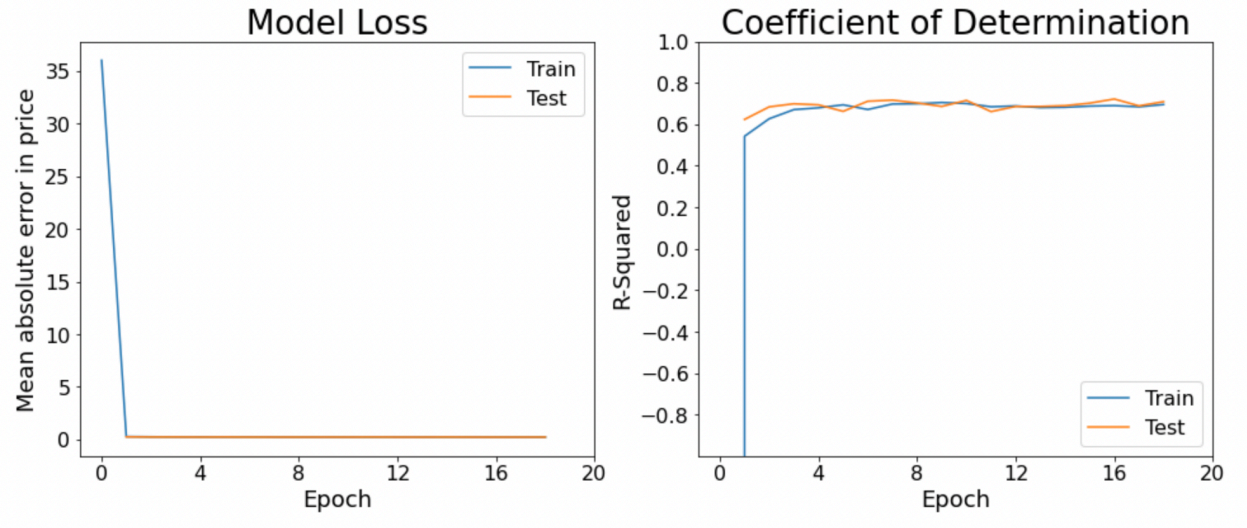

Fig 19. Using Color images in RGB format did not greatly improve model performance, and resulted in an R-squared score of 0.72.

Lastly, I tried using the color images in RGB format. The model performance did not greatly improve, with an R-squared score of 0.72, and training time took much longer. There was no great advantage to using RGB images over the grayscale (Figure 19).

Discussion

During this project, I learned how to effectively use images and numerical/ categorical data in a neural network regression model. Using both types of data improved model performance over using one over the other. The R-squared scores for all of the models tested can be seen in the table below (Figure 20).

One area where I think this project could be improved has to do with the images. This dataset from Kaggle scraped the first image listed on Zillow for each listing. Most listings have many images of the exterior, interior, or views of the home. For most of the listings, the first image is usually a view of the exterior of the home to show its curb appeal, however, there is no way to guarantee that all of the first images for each listing are from the same angle of the house, same distance from the home, etc. There is no way to standardize the first image. One possible solution would be to scrape all of the images for each listing from Zillow, and then somehow filter out the images so that they

are all from comparable views points. One article that I consulted for this project actually used four images for each home listing, one for the outside of the home, one for interior kitchen, interior living room, and interior master bedroom. They then made a collage of the four images to create one image for each listing, but this way, the different areas of the images could be compared to on another. From looking at the plots of the filters (Figure 18), it is hard to discern specific structures that might resemble a house. Ideally, during this step, it would be possible to visualize some features of the homes in the images.

Another way I could have improved the performance of the model was to include all of the features from the dataset. Since we know that zip code is influential on the

Table 1

Model Properties	**R-Squared Score**
images only	0.0300
features only	0.5700
mixed inputs	0.6644
mixed inputs with normalized price	0.7084
mixed inputs with normalized price and extra dense layer	0.7137
mixed inputs with normalized price, extra dense layer, increased kernel size	0.6966
mixed inputs with normalized price and color images	0.7223

Figure 20. Table of R-squared scores for each model.

price of the homes, perhaps the listings could have been grouped by groups of zip codes (ie. 'Desirable areas,' 'good schools,' 'gated communities', etc). There was also a column in the dataset called 'description' which had a paragraph of each listing, usually this is found on Zillow listings describing the property. Using the information from this column for natural language processing could have been helpful for the model, to analyze the language used to describe the pricier homes.

Conclusion

The CNN model utilizing both the images along with the numerical/categorical data from the home features performed the better than using models for either component alone. With further work on the image processing and features engineering, the model could be improved in the future. Overall, neural network modeling proved to be a valuable tool in this project.

References:

The dataset:
Pierce, E. Austin, TX House Listings, Version 4. Retrieved [July 2021] from [https://www.kaggle.com/ericpierce/austinhousingprices]

Other sources:

Amidi, A. 'A detalied example of how to use Data Generators with Keras', <https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly>[accessed August 2021].

Brownlee, J. 'Regression Tutorial with the Keras Deep Learning Library in Python', MachineLearningMastery.com. < https://machinelearningmastery.com/regression-tutorial-keras-deep-learning-library-python/>[accessed August 2021].

Brownlee, J. 'How to Visualize Filters and Feature Maps in Convulutional Neural Networks', MachineLearningMastery.com, <https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/>[accessed August 2021].

'Implement a Custom Metric Function in Keras'. <https://jmlb.github.io/ml/2017/03/20/CoeffDetermination_CustomMetric4Keras/>[accessed August 2021].

Khandelwal, R. 'Convultional Neural Network: Feature Map and Filter Visualization,' Towards Data Science,<https://towardsdatascience.com/convolutional-neural-network-feature-map-and-filter-visualization-f75012a5a49c>[accessed August 2021].

Lewis, L. 'Building a mixed-data neural network in Keras to predict accident locations', Heartbeat. <https://heartbeat.fritz.ai/building-a-mixed-data-neural-network-in-keras-to-predict-accident-locations-d51a63b738cf>[accessed August 2021].

Rosebrock, A. 'Keras: Multiple Inputs and Mixed Data', pyimagesearch.com <https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/>accessed August 2021].