**CALIFORNIA STATE UNIVERSITY, SACRAMENTO (CSUS)**

**COLLEGE OF ENGINEERING AND COMPUTER SCIENCE**

SACRAMENTO STATE

**CSC 215-01: ARTIFICIAL INTELLIGENCE**

**FALL 2020**

# MINI-PROJECT 3: HOUSE PRICE ESTIMATION USING TENSORFLOW FUNCTIONAL API

Due at 4.00 pm, Wednesday, October 28, 2020

Submitted by:

Harshitha Onkar (Sac State Id: 220262706)
Gargi Prabhugaonkar (Sac State Id: 301111525)

# Table of Contents

## Problem Statement

We intend to build a novel house price estimation system by extracting visual features from house photos and combining them with the house's textual information using TensorFlow Functional API.

We intend to build a deep learning model and evaluate the result by displaying RMSE and lift chart on test data set.

Additionally, we intend to implement additional features such as:

1) Implement transfer learning models- VGG16, ResNet50, MobileNetV2
2) Evaluate the output by treating images separately over combining them together as input
3) Take advantage of zip code as a feature
    a. Exclude rows where zip code does not have enough number of houses.
    b. Treat zip code as a separate input channel
    c. Using transfer learning model on dataset without low frequency zip codes.

## Methodology

### Dataset

The dataset used for this project is available at [1]. The dataset contains visual and textual information.

| Dataset type | Features | Total data |
|---|---|---|
| Textual Information | Bedroom, Bathroom, Area, Price, Zip code | 535 |
| Visual Information | Images for Frontal, Bedroom, Bathroom, Kitchen for each house | 2140 images, 4 images for each house |

Table 1: Dataset Information

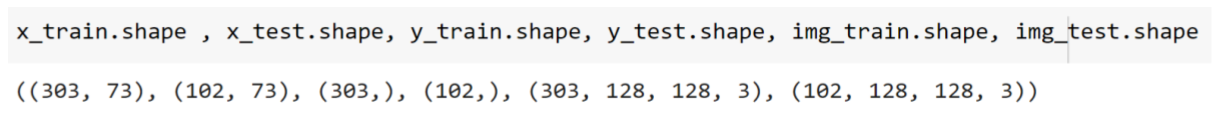
### Helper functions

All common repetitive functions used for data preprocessing, visualizing output and plotting functions have been stored separately and imported wherever required.

# House Price Estimation using TensorFlow Functional API

## Data preprocessing

1) Textual data is loaded from “HousesInfo.txt”
2) Houses with Price range less than 100K and greater than 900K are removed
3) ‘Bedroom’, ‘Bathroom’ and ‘Zip code’ columns are considered as categorical and one hot encoded
4) ‘Area’ is normalized, and Price is the target column
5) Textual data frame is converted to tensors using to_xy function
6) Image URLs are combined into a data frame by looping over the images
7) URLs are used to retrieve and combine frontal, bedroom, bathroom and kitchen images
8) Textual and Image datasets are then split into train and test sets. The shapes are as shown in Fig.1.
9) The preprocessed datasets are saved for future use.



```
x_train.shape , x_test.shape, y_train.shape, y_test.shape, img_train.shape, img_test.shape
((303, 73), (102, 73), (303,), (102,), (303, 128, 128, 3), (102, 128, 128, 3))
```

Fig.1: Train and Test shapes for base model input data

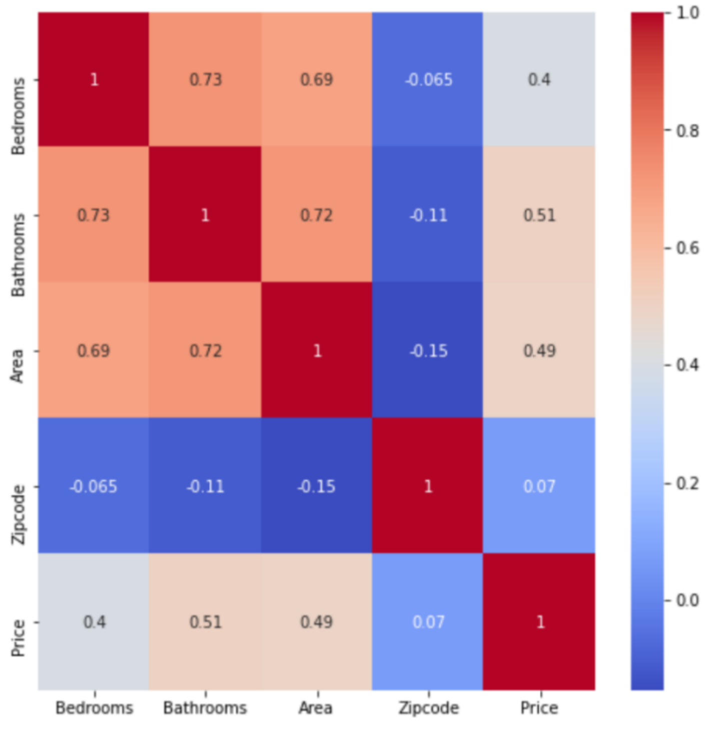


Fig.2 Correlation matrix for House Price Estimation

### Model Design

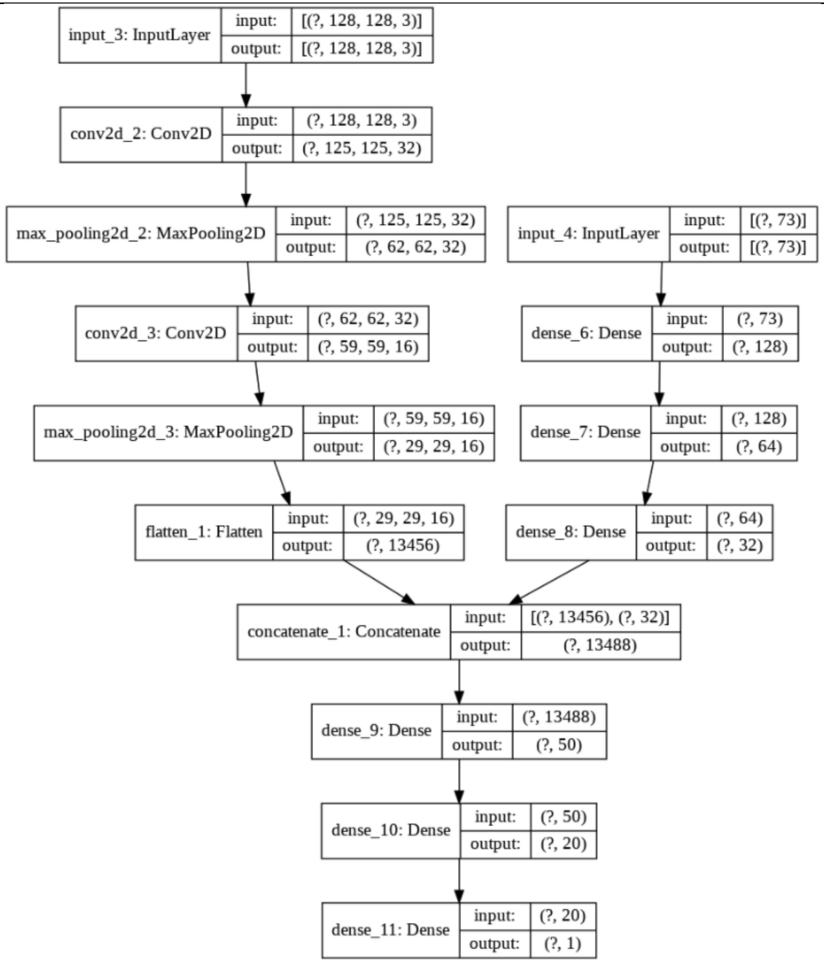
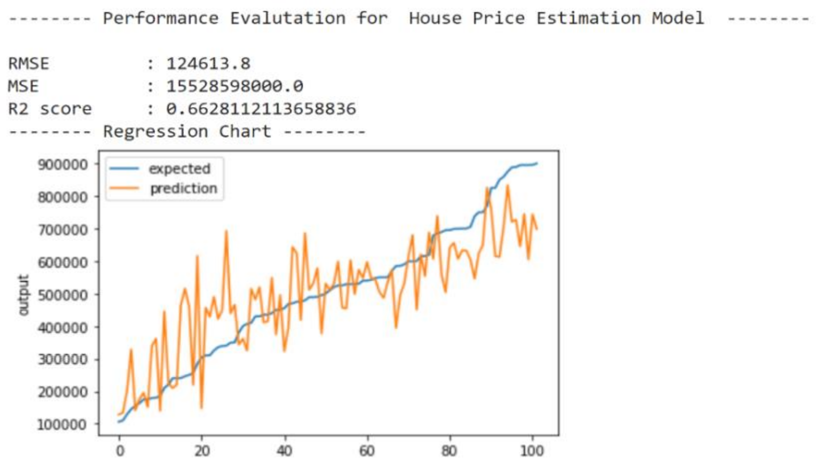

input_3: InputLayer | input: [(?, 128, 128, 3)] | output: [(?, 128, 128, 3)]
conv2d_2: Conv2D | input: (?, 128, 128, 3) | output: (?, 125, 125, 32)
max_pooling2d_2: MaxPooling2D | input: (?, 125, 125, 32) | output: (?, 62, 62, 32)
input_4: InputLayer | input: [(?, 73)] | output: [(?, 73)]
conv2d_3: Conv2D | input: (?, 62, 62, 32) | output: (?, 59, 59, 16)
dense_6: Dense | input: (?, 73) | output: (?, 128)
max_pooling2d_3: MaxPooling2D | input: (?, 59, 59, 16) | output: (?, 29, 29, 16)
dense_7: Dense | input: (?, 128) | output: (?, 64)
flatten_1: Flatten | input: (?, 29, 29, 16) | output: (?, 13456)
dense_8: Dense | input: (?, 64) | output: (?, 32)
concatenate_1: Concatenate | input: [(?, 13456), (?, 32)] | output: (?, 13488)
dense_9: Dense | input: (?, 13488) | output: (?, 50)
dense_10: Dense | input: (?, 50) | output: (?, 20)
dense_11: Dense | input: (?, 20) | output: (?, 1)

Fig.3: Base Model design

## Implementation of Base Model

1) Multi-input with single output model is created using Functional API
2) The preprocessed textual data is given as input to fully connected neural network
3) The preprocessed image dataset (each image of 128x128x3) is given as input to CNN
4) Outputs from both the channels are merged before sending through additional dense layers, the final dense layer with one neuron is used as the output layer of the model as shown in Fig.3.
5) Early Stopping is used to prevent overfitting and Checkpoint is used to save the best weight model.

## Results for Base Model



```
-------- Performance Evalutation for  House Price Estimation Model  --------

RMSE         : 124613.8
MSE          : 15528598000.0
R2 score     : 0.6628112113658836
-------- Regression Chart --------
```

Fig.4: RMSE and Regression Chart for Functional API-Base Model

## Additional Feature: Using Transfer Learning

### Data Preprocessing

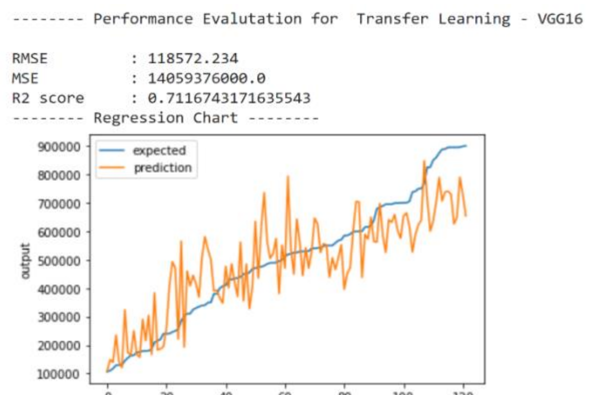
Preprocessed data saved during previous step is used for Transfer Learning.

### Implementation

Transfer Learning is implemented using VGG16, ResNet50 and MobileNetV2.

1) For all three, the models are downloaded from respective keras.applications class
2) The top layer is eliminated by setting include_top = False
3) The pretrained model layer weights are fixed by setting layer.trainable = False
4) Additional dense layers are added to existing layers before merging with the fully connected neural network (FCNN) channel.
5) The input of transfer learning model and FCNN are given as inputs to the final model.
6) Early Stopping is used to prevent overfitting and Checkpoint is used to save the best weight model.

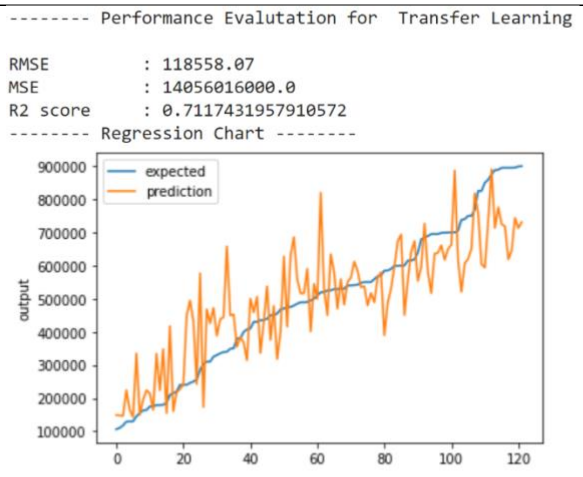
### Result



```
-------- Performance Evalutation for  Transfer Learning - VGG16

RMSE         : 118572.234
MSE          : 14059376000.0
R2 score     : 0.7116743171635543
-------- Regression Chart --------
```

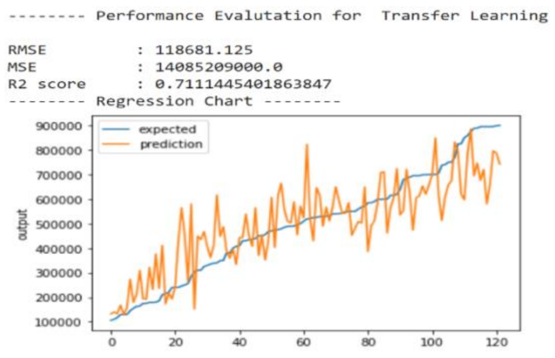
Fig.5: RMSE and Regression Chart for VGG16



```
-------- Performance Evalutation for  Transfer Learning

RMSE         : 118558.07
MSE          : 14056016000.0
R2 score     : 0.7117431957910572
-------- Regression Chart --------
```

Fig.6: RMSE and Regression Chart for Resnet50



```
-------- Performance Evalutation for  Transfer Learning

RMSE         : 118681.125
MSE          : 14085209000.0
R2 score     : 0.7111445401863847
-------- Regression Chart --------
```

Fig.7: RMSE and Regression Chart for MobileNetV2

## Additional Feature: Images as separate input channels

To evaluate deep learning model using TensorFlow Functional API by providing visual information as separate input channel for each type of image corresponding to each house.

### Model Design

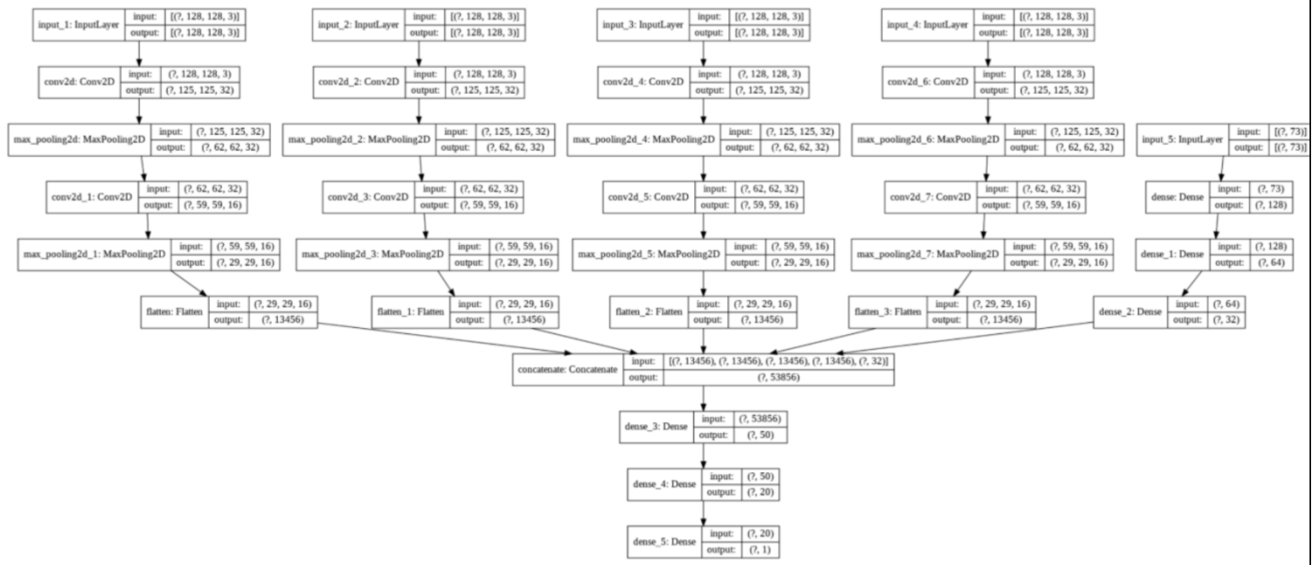
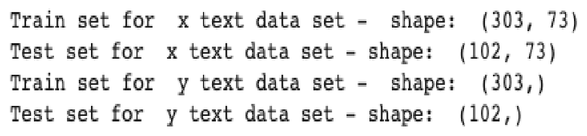

input_1: InputLayer | input: [(?, 128, 128, 3)] | output: [(?, 128, 128, 3)]
input_2: InputLayer | input: [(?, 128, 128, 3)] | output: [(?, 128, 128, 3)]
input_3: InputLayer | input: [(?, 128, 128, 3)] | output: [(?, 128, 128, 3)]
input_4: InputLayer | input: [(?, 128, 128, 3)] | output: [(?, 128, 128, 3)]
conv2d: Conv2D | input: (?, 128, 128, 3) | output: (?, 125, 125, 32)
conv2d_2: Conv2D | input: (?, 128, 128, 3) | output: (?, 125, 125, 32)
conv2d_4: Conv2D | input: (?, 128, 128, 3) | output: (?, 125, 125, 32)
conv2d_6: Conv2D | input: (?, 128, 128, 3) | output: (?, 125, 125, 32)
max_pooling2d: MaxPooling2D | input: (?, 125, 125, 32) | output: (?, 62, 62, 32)
max_pooling2d_2: MaxPooling2D | input: (?, 125, 125, 32) | output: (?, 62, 62, 32)
max_pooling2d_4: MaxPooling2D | input: (?, 125, 125, 32) | output: (?, 62, 62, 32)
max_pooling2d_6: MaxPooling2D | input: (?, 125, 125, 32) | output: (?, 62, 62, 32)
input_5: InputLayer | input: [(?, 73)] | output: [(?, 73)]
conv2d_1: Conv2D | input: (?, 62, 62, 32) | output: (?, 59, 59, 16)
conv2d_3: Conv2D | input: (?, 62, 62, 32) | output: (?, 59, 59, 16)
conv2d_5: Conv2D | input: (?, 62, 62, 32) | output: (?, 59, 59, 16)
conv2d_7: Conv2D | input: (?, 62, 62, 32) | output: (?, 59, 59, 16)
dense: Dense | input: (?, 73) | output: (?, 128)
max_pooling2d_1: MaxPooling2D | input: (?, 59, 59, 16) | output: (?, 29, 29, 16)
max_pooling2d_3: MaxPooling2D | input: (?, 59, 59, 16) | output: (?, 29, 29, 16)
max_pooling2d_5: MaxPooling2D | input: (?, 59, 59, 16) | output: (?, 29, 29, 16)
max_pooling2d_7: MaxPooling2D | input: (?, 59, 59, 16) | output: (?, 29, 29, 16)
dense_1: Dense | input: (?, 128) | output: (?, 64)
flatten: Flatten | input: (?, 29, 29, 16) | output: (?, 13456)
flatten_1: Flatten | input: (?, 29, 29, 16) | output: (?, 13456)
flatten_2: Flatten | input: (?, 29, 29, 16) | output: (?, 13456)
flatten_3: Flatten | input: (?, 29, 29, 16) | output: (?, 13456)
dense_2: Dense | input: (?, 64) | output: (?, 32)
concatenate: Concatenate | input: [(?, 13456), (?, 13456), (?, 13456), (?, 13456), (?, 32)] | output: (?, 53856)
dense_3: Dense | input: (?, 53856) | output: (?, 50)
dense_4: Dense | input: (?, 50) | output: (?, 20)
dense_5: Dense | input: (?, 20) | output: (?, 1)

Fig.8: Model design- Images as separate input channels

### Implementation

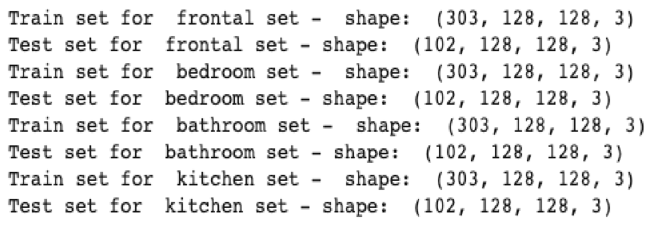
1) Load preprocessed textual information
2) Create data frame containing columns- ‘House#’, ‘frontal_img’, ‘bedroom_img’, ‘bathroom_img’, ‘kitchen_img’ and rows containing path to each image
3) Identify images matching textual information and exclude any unmatched rows. Each house must have textual data as well as visual data
4) Load images into four lists of images for frontal, bedroom, bathroom and kitchen
5) Normalize images in the four lists
6) Create input and output channels for textual data
7) For input channels:
    a. For textual information, create train set (75%) and test set (25%)
    b. For visual information, for each of the four lists create train set (75%) and test set (25%)
8) Create Functional API model with multiple inputs- Fours image inputs and one textual input channels.
9) Train deep learning model
10) Perform model testing and performance evaluation

## Results



```
Train set for  x text data set -  shape:  (303, 73)
Test set for  x text data set - shape:  (102, 73)
Train set for  y text data set -  shape:  (303,)
Test set for  y text data set - shape:  (102,)
```

Fig.9: Train and test set dimension details for textual information



```
Train set for  frontal set -  shape:  (303, 128, 128, 3)
Test set for  frontal set - shape:  (102, 128, 128, 3)
Train set for  bedroom set -  shape:  (303, 128, 128, 3)
Test set for  bedroom set - shape:  (102, 128, 128, 3)
Train set for  bathroom set -  shape:  (303, 128, 128, 3)
Test set for  bathroom set - shape:  (102, 128, 128, 3)
Train set for  kitchen set -  shape:  (303, 128, 128, 3)
Test set for  kitchen set - shape:  (102, 128, 128, 3)
```

Fig.10: Train and test set dimension details for visual information

```
RMSE         : 127148.45
MSE          : 16166730000.0
R2 score     : 0.6489547194772525
-------- Regression Chart --------
```

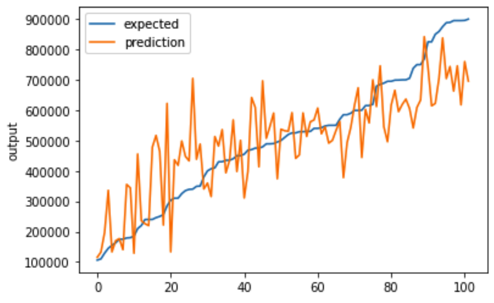


Fig.11: RMSE and Regression Chart for additional feature: Images as separate input channels

## Additional Feature: Zip code Feature Tuning

There is a correlation between zip code and price of the house. Therefore, we used zip code feature tuning to improve our results. We implemented two cases for feature tuning using zip code.

- Case 1: Eliminating zip codes with low frequency and using transfer learning
- Case 2: Eliminating zip codes with low frequency and treating zip code as a separate input channel

### Data preprocessing

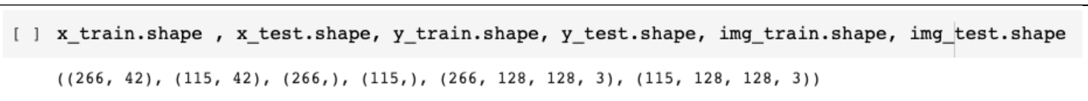
1) After initial textual preprocessing, additional datapoints with low zip code frequency (zip codes with 1-2 houses) are dropped.
2) The above dataset is used to fit the base model and a transfer learning model – MobileNetV2.
3) For Case 2, we further split the textual input into two parts –
    - Input with 'Bedroom', 'Bathroom', 'Area' columns
    - 'Zip code' as separate input
4) Data is then split into train and test for the two cases as shown in Fig.12 and Fig.16, 17.

### Case 1: Removing zip codes with low frequency

#### Implementation

The textual data (with zip code outliers removed) is given as input to FCNN and the image data to CNN. The same dataset is applied to MobileNetV2 and evaluated to compare performance.

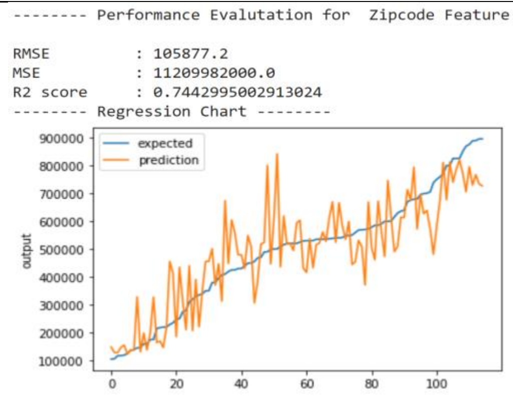
#### Result



```
[ ] x_train.shape , x_test.shape, y_train.shape, y_test.shape, img_train.shape, img_test.shape

    ((266, 42), (115, 42), (266,), (115,), (266, 128, 128, 3), (115, 128, 128, 3))
```

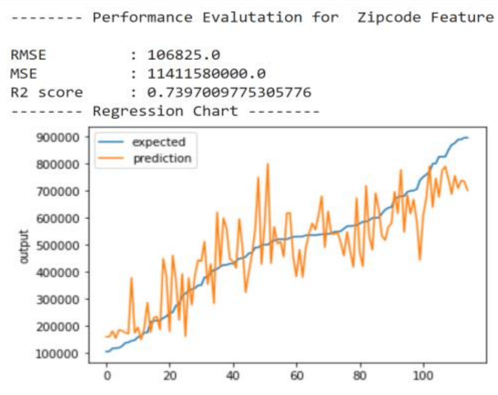
Fig.12: Train and test set dimension details for textual and visual information



```
-------- Performance Evalutation for  Zipcode Feature Tuning

RMSE          : 105877.2
MSE           : 11209982000.0
R2 score      : 0.7442995002913024
-------- Regression Chart --------
```

Fig.13. RMSE and Regression Lift chart Without Transfer Learning



```
-------- Performance Evalutation for  Zipcode Feature Tuning

RMSE          : 106825.0
MSE           : 11411580000.0
R2 score      : 0.7397009775305776
-------- Regression Chart --------
```

Fig. 14. RMSE and Regression Lift chart with Transfer Learning

## Case 2: Zip code as a separate input channel

## Model Design

The model design for this implementation is given below. Zip code feature has been dropped from the original preprocessed data set and provided as a separate input channel to the model.

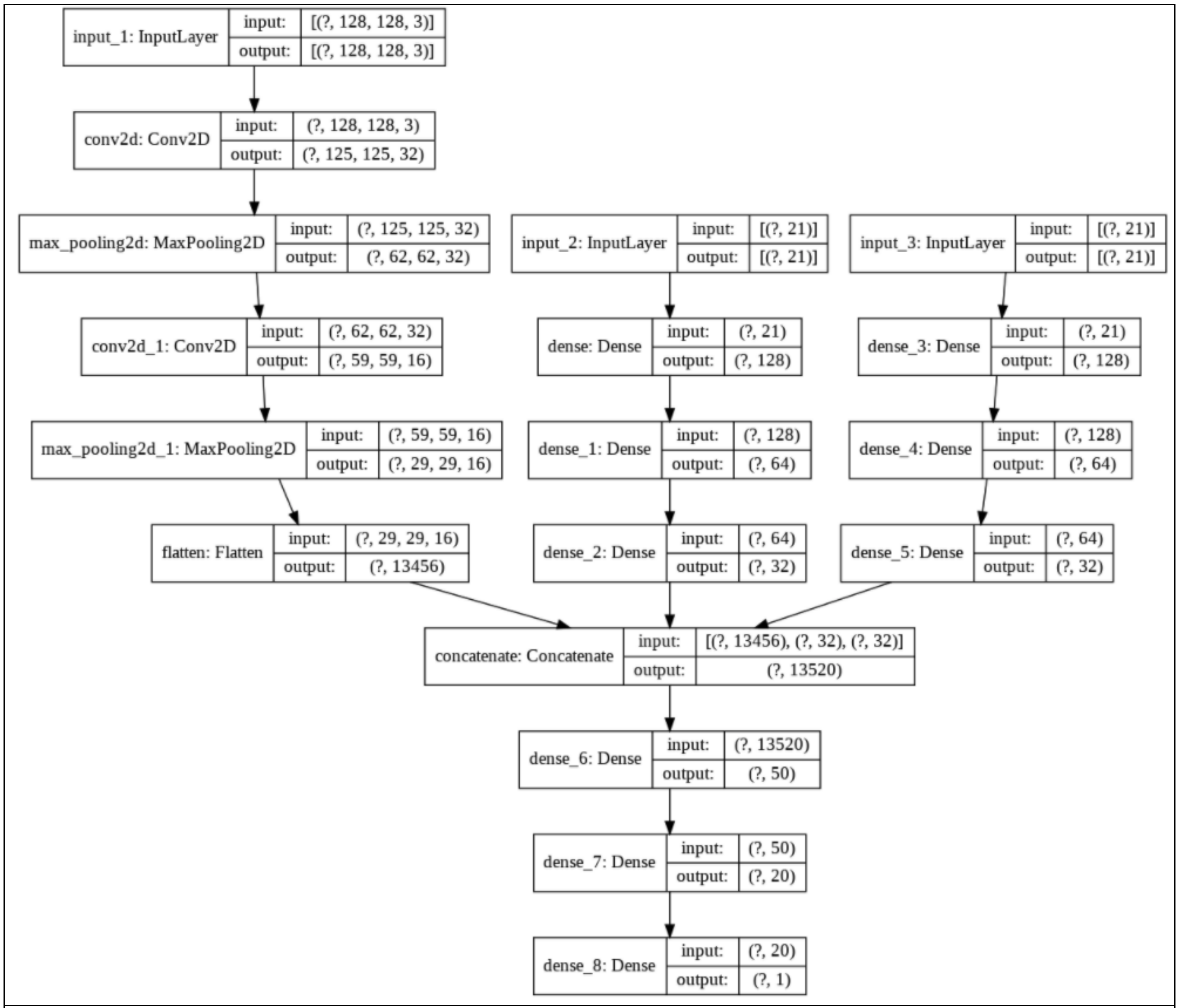


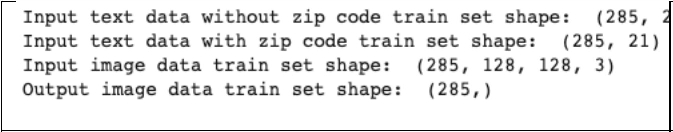
Fig.15. Model design- Zip code as separate input channel

## Implementation

After preprocessing zip code feature, it can be provided as a separate input channel. To implement this, we exclude zip code feature from the original textual data set and create a new data frame using zip code feature. The steps to implement this model are summarized as follows:

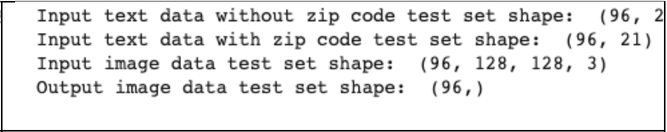
1) Remove outliers using Zip code and Price feature
2) Create a new data frame using Zip code and drop the feature from original data set
3) Encode Zip code feature
4) Load and normalize preprocessed images
5) Identify images matching textual information and exclude any unmatched rows. Each house must have textual data as well as visual data
6) Create input and output channels for textual data
7) For input channels:
    a. For textual information, original dataset and for zip code dataset create train set (75%) and test set (25%)
    b. For visual information, dataset with combined images create train set (75%) and test set (25%)
8) Create Functional API model with multiple inputs- One image input and two textual input channels.
9) Train deep learning model
10) Perform model testing and performance evaluation

## Results



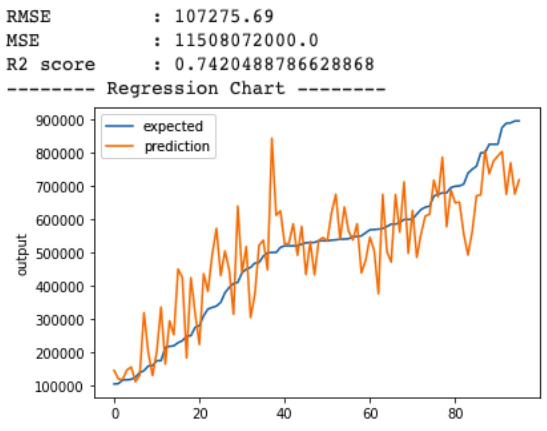
```
Input text data without zip code train set shape:  (285, 2
Input text data with zip code train set shape:  (285, 21)
Input image data train set shape:  (285, 128, 128, 3)
Output image data train set shape:  (285,)
```

Fig. 16. Train set dimension details for textual information



```
Input text data without zip code test set shape:  (96, 2
Input text data with zip code test set shape:  (96, 21)
Input image data test set shape:  (96, 128, 128, 3)
Output image data test set shape:  (96,)
```

Fig. 17. Test set dimension details for visual information



```
RMSE         : 107275.69
MSE          : 11508072000.0
R2 score     : 0.7420488786628868
-------- Regression Chart --------
```

Fig.18. RMSE and Regression Chart for additional feature: Zip code as a separate input channel

## Challenges faced and Takeaways

### Challenges

1. Data preprocessing – visual information had to match corresponding textual information. Using 'House#' as a reference parameter helped to group matching data
2. Saving and loading preprocessed images- Saving the preprocessed images and retrieving the same images was challenging [2]. In some cases, we used original text set with preprocessed images in which case only the required images had to be retained. Using 'House#' as a common parameter allowed to retain only matching text and visual information.
3. Combining the layers of pretrained models: When connecting transfer learning model layers to additional layers, we encountered shape mismatch error, which was resolved by adding GlobalAveragePooling2D layer.
4. Hyperparameter Tuning – As the functional API model contains various inputs, tuning each channel to achieve the best model was time consuming.

### Takeaways

1. Saving processed data especially for images helped in saving a lot of time.
2. We learned that using Functional API, we can build complex models and also use features in separate input channel to improve model performance.
3. We learned to combine base models with transfer learning to achieve better performing models and obtain improved results.

| **Model** | **RMSE** |
|---|---|
| Base model | 124613.8 |
| Images as separate input channels | 127148.45 |
| Transfer learning model | 118558.07 |
| Zip code as separate input channel | 107275.69 |
| Zip code tuning with transfer learning | 106825.0 |
| Zip code tuning without transfer learning | 105877.2 |

## Task Division

| **Tasks** | |
|---|---|
| Data preprocessing, Saving and loading preprocessed data | Harshitha, Gargi |
| House Price Estimation using TensorFlow Functional API | Harshitha, Gargi |
| Additional Feature 1: Transfer learning model | Harshitha |
| Additional Feature 2: Images as separate input channel | Gargi |
| Additional Feature 3: Zip code tuning | Harshitha, Gargi |

## References

[1] "House Price Estimation Data set," [Online]. Available: https://github.com/emanhamed/Houses-dataset.

[2] "Research Paper," [Online]. Available: https://arxiv.org/pdf/1609.08399.pdf.

[3] "Image Preprocessing," [Online]. Available: https://www.pluralsight.com/guides/importing-image-data-into-numpy-arrays.

[4] "Loading images- Tensorflow," [Online]. Available: https://www.tensorflow.org/tutorials/load_data/images.

[5] "Transfer Learning Models," [Online]. Available: https://keras.io/api/applications/.