



**POST GRADUATE PROGRAM IN ANALYTICS  
AND ARTIFICIAL INTELLIGENCE,  
IMARTICUS LEARNING, BANGALORE.**

## **CAPSTONE PROJECT**

**NAME:** SOWNDARIYA M

**BATCH:** PGAA - 02

**TITLE:** HOUSE IMAGE CLASSIFICATION USING CNN

**PROJECT ASSIGNED:** 28.01.2021

**PROJECT SUBMISSION:** 27.02.2021

## Contents

1.EXECUTIVE SUMMARY: .....	3
2.METHODS AND ANALYSIS .....	4
2.1.SYSTEM ENVIRONMENT.....	4
2.2.LOADING LIBRARIES .....	5
2.3.DATA PREPARATION .....	6
2.3.1.Images — Channels and Sizes: .....	7
2.3.2.Morphological Transformations .....	7
2.3.3.Normalization: .....	8
2.3.4.Augmentation: .....	8
2.4.MODELING .....	10
2.4.1.Sequential: .....	10
2.4.2.Conv2D:.....	10
2.4.3.MaxPool2D.....	11
2.4.4.Flatten .....	11
2.4.5.Dense .....	11
2.4.5.1.Sigmoid Activation Function .....	12
2.5.COMPILING .....	12
2.5.1 Optimizer .....	12
2.5.2.Loss .....	14
BinaryCrossentropy loss: .....	14
2.6.MODEL FITTING.....	14
2.7.MODEL ANALYSING.....	15
2.8.MODEL ARCHITECTURE.....	17
2.9. MODEL ACCURACY AND LOSS.....	19
2.10. MODEL SAVING .....	21
3.PREDICTION OF TEST IMAGES.....	21
4.CONCLUSION.....	23
5.REFERENCES .....	23

## **1.EXECUTIVE SUMMARY:**

Dream Housing Property Listings Company has hosted a website where users can search for their dream houses and also sell their houses. To enable better filtering of properties for the buyers, the website mandates the sellers to list their properties only after they upload the images of their houses to attract the relevant buyers. However, it has been noticed that some of the sellers are violating the rules and uploading random images instead of their houses. To overcome this problem, we are building a simple classification model to classify captured images into a house image or a non-house image which can help with the automatic check such that the user will be restricted from adding any non-house image, only images relevant to the house will get uploaded to the website.

### **1.1.GOAL :**

To build a simple classification model which will classify captured images into a house image or a non-house image. Your model can help with the automatic check such that the user will be restricted from adding any non-house image, only images relevant to the house will get uploaded to the website.

## 2.METHODS AND ANALYSIS

First we introduce the system environment used for this project, then we introduce and load the necessary libraries. After that we proceed with the data wrangling to classify the images and their respective relation to the label.

### 2.1.SYSTEM ENVIRONMENT

To ease the reproducibility of this project, we take a look at the (technical) infrastructure used for this project:

```
In [1]: 1 import sys
        2 print("python version", sys.version)

python version 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
```

```
In [71]: 1 import pkg_resources
        2 import types
        3 def get_imports():
        4     for name, val in globals().items():
        5         if isinstance(val, types.ModuleType):
        6             name = val.__name__.split(".")[0]
        7         elif isinstance(val, type):
        8             name = val.__module__.split(".")[0]
        9         poorly_named_packages = {"PIL": "Pillow", "sklearn": "scikit-learn"}
       10         if name in poorly_named_packages.keys():
       11             name = poorly_named_packages[name]
       12         yield name
       13 imports = list(set(get_imports()))
       14 requirements = []
       15 for m in pkg_resources.working_set:
       16     if m.project_name in imports and m.project_name!="pip":
       17         requirements.append((m.project_name, m.version))
       18 for r in requirements:
       19     print("{}=={}".format(*r))

tqdm==4.42.1
tensorflow==2.3.1
spacy==2.3.2
scikit-learn==0.24.0
pyforest==1.0.3
pydot==1.4.1
py==1.8.1
path==13.1.0
numpy==1.19.5
nltk==3.4.5
matplotlib==3.1.3
gensim==3.8.3
bokeh==1.4.0
```

## 2.2.LOADING LIBRARIES

```
In [4]: 1 from keras.preprocessing.image import ImageDataGenerator
        2 import numpy as np
        3 import keras
        4 import matplotlib.pyplot as plt
        5 from keras.models import Sequential
        6 from keras.layers import Dense, Conv2D, Flatten, MaxPool2D
```

**import numpy as np**- 'Numpy' is used for mathematical operations on large, multi-dimensional arrays and matrices.

**import keras** - It wraps the efficient numerical computation **libraries** Theano and TensorFlow and allows you to define and train neural network models in just a few lines of code.

**import matplotlib.pyplot as plt** - 'Matplotlib' is a data visualization library for 2D and 3D plots, built on numpy.

**from keras.preprocessing.image import ImageDataGenerator** - Generate batches of tensor image data with real-time data augmentation.

**from keras.models import Sequential** - A **Sequential** model is appropriate for a plain stack of **layers** where each **layer** has exactly one input tensor and one output tensor.

**from keras.layers import Dense, Conv2D, Flatten, MaxPool2D:**

**Dense** – Dense implements the operation:  $\text{output} = \text{activation}(\text{dot}(\text{input}, \text{kernel}) + \text{bias})$  where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use\_bias is True).

**Conv2D**- This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use\_bias is True, a bias vector is created and added to the outputs. Finally, if activation is not None, it is applied to the outputs as well.

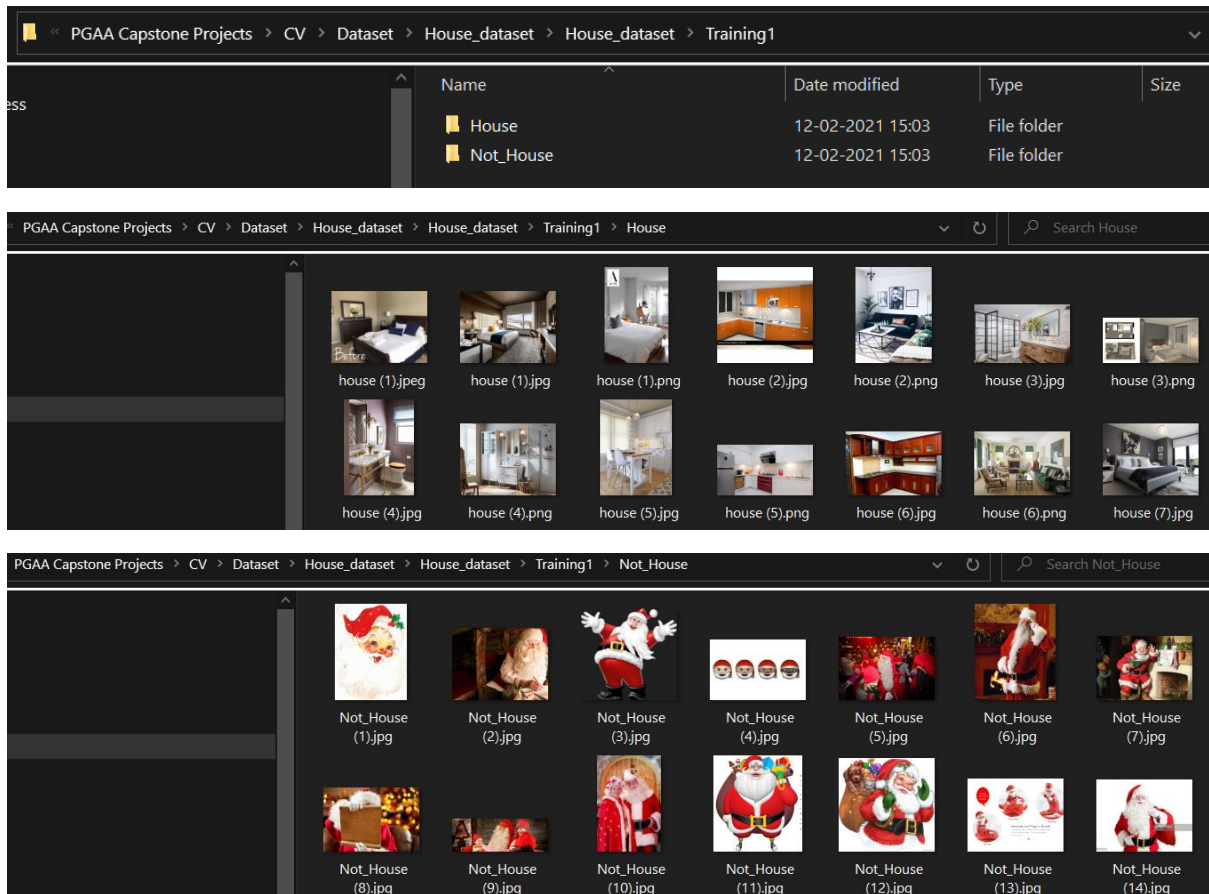
**Flatten**- Flattens the input. Does not affect the batch size. Note: If inputs are shaped (batch,) without a feature axis, then flattening adds an extra channel dimension and output shape is (batch, 1).

**MaxPool2D**- Max pooling operation for 2D spatial data. Downsamples the input representation by taking the maximum value over the window defined by pool\_size for each dimension along the features axis. The window is shifted by strides in each dimension. The resulting output when using "valid" padding option has a shape(number of rows or columns) of:  $\text{output\_shape} = (\text{input\_shape} - \text{pool\_size} + 1) / \text{strides}$ . The resulting output shape when using the "same" padding option is:  $\text{output\_shape} = \text{input\_shape} / \text{strides}$ .

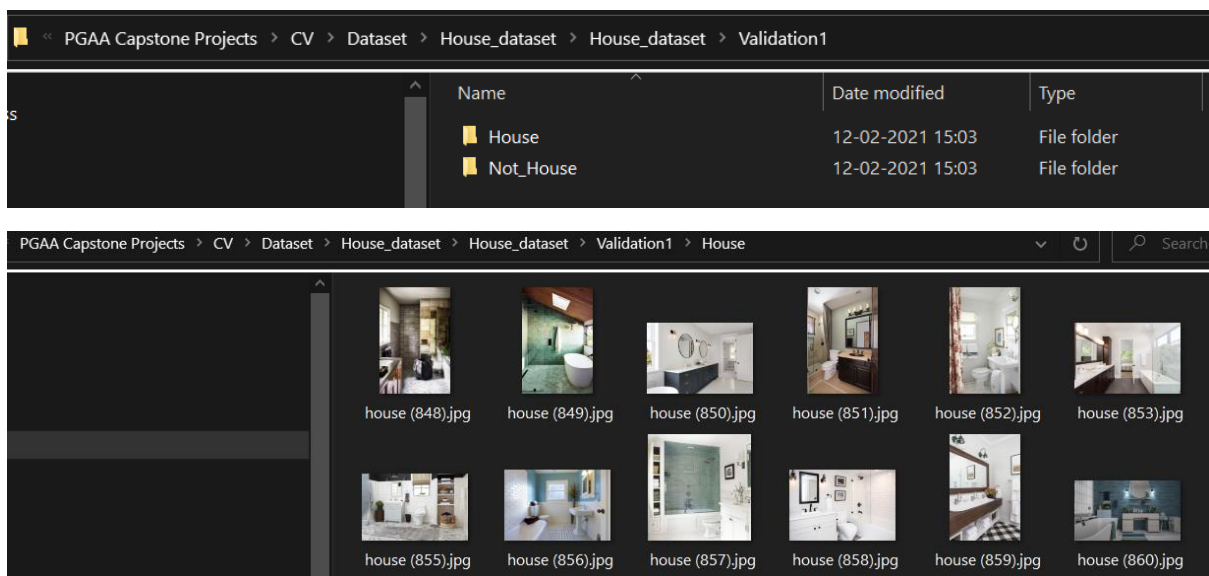
## 2.3.DATA PREPARATION

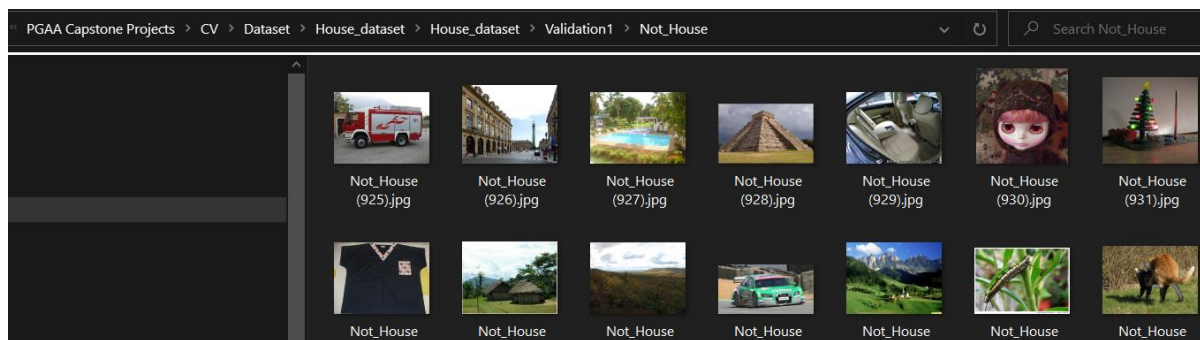
Our starting point is the creation of a new folder, in which the dataset will be downloaded into:

### TRAINING IMAGES:



### VALIDATION IMAGES:





The images are in different sizes, we have to do data preprocessing to build a generalized model.

### 2.3.1.Images — Channels and Sizes:

Images come in different shapes and sizes. They also come through **different sources**. For example, some images are what we call “natural images”, which means they are taken in **color**, in the **real world**. For example:

- A picture of a flower is a natural image.
- An X-ray image is *not* a natural image.

Taking all these variations into consideration, we need to perform some pre-processing on any image data. RGB is the most popular encoding format, and most “natural images” we encounter are in RGB. Also, among the first step of data pre-processing is **to make the images of the same size**. Let’s move on to how we can **change the shape and form of images**.

## 2. Define image properties:

```
In [5]: 1 Image_Width=64
        2 Image_Height=64
        3 target_size=(Image_Width,Image_Height)
        4 Image_Channels=3
```

### 2.3.2.Morphological Transformations

The term *morphological transformation* refers to any modification involving the **shape and form** of the images. These are very often used in image analysis tasks. Although they are used with all types of images, they are especially powerful for images that are not natural (come from a source other than a picture of the real world). The typical transformations are erosion, dilation, opening, and closing.

#### 1.Thresholding

One of the simpler operations where we take all the pixels whose intensities are above a certain threshold and convert them to ones; the pixels having value less than the threshold is converted to zero. This results in a *binary image*.

#### 2.Erosion, Dilation, Opening & Closing

**Erosion** shrinks bright regions and enlarges dark regions. **Dilation** on the other hand is exact opposite side, it shrinks dark regions and enlarges the bright regions.

**Opening** is erosion followed by dilation. Opening can remove small bright spots (i.e. “salt”) and connect small dark cracks. This tends to “open” up (dark) gaps between (bright) features.

**Closing** is dilation followed by erosion. Closing can remove small dark spots (i.e. “pepper”) and connect small bright cracks. This tends to “close” up (dark) gaps between (bright) features. All these can be done using the `skimage.morphology` module. The basic idea is to have a **circular disk** of a certain size (3 below) move around the image and apply these transformations using it.

### 2.3.3. Normalization:

Normalization is the most crucial step in the pre-processing part. This refers to rescaling the pixel values so that they lie within a confined range. One of the reasons to do this is to help with the issue of propagating gradients.

### 2.3.4. Augmentation:

This brings us to the next aspect of data pre-processing — data augmentation. Many times, the **quantity of data that we have is not sufficient** to perform the task of classification well enough. In such cases, we perform **data augmentation**. As an example, if we are working with a dataset of classifying gemstones into their different types, we may not have enough number of images (since high-quality images are difficult to obtain). In this case, we can perform augmentation to increase the size of your dataset. Augmentation is often used in image-based deep learning tasks to increase the amount and variance of training data. Augmentation should only be done on the training set, never on the validation set.

As you know that **pooling** increases **invariance**. If a picture of a dog is in the top left corner of an image, with pooling, you would be able to recognize if the dog is in a little bit left/right/up/down around the top left corner. But with training data consisting of data augmentation like *flipping, rotation, cropping, translation, illumination, scaling, adding noise*, etc., the model learns all these variations. This significantly boosts the accuracy of the model. So, even if the dog is there at any corner of the image, the model will be able to recognize it with high accuracy.

There are multiple types of augmentations possible. The basic ones transform the original image using one of the following types of transformations:

1. Linear transformations
2. Affine transformations



### 2.3.5.Run a data generator

Data generator supports preprocessing — it normalizes the images (dividing by 255) and crops the center (**100 x 100**) portion of the image.

There was no specific reason to include **100** as the dimension but it has been chosen so that we can process all the images which are of greater than 100\*100 dimension. If any dimension (height or width) of an image is less than 100 pixels, then that image is deleted automatically. You can change it to 150 or 200 according to your need.

Let's now set up the **data generator**. The code below sets up a custom data generator that is slightly different than the one that comes with the Keras API. The reason to use a custom generator is to be able to modify it according to the problem at hand (customizability).

Finally, the method `__data_generation` returns the batch of images as the pair X, y where X is of shape (batch\_size, height, width, channels) and y is of shape (batch size, ). Note that `__data_generation` also does some preprocessing - it normalizes the images (divides by 255) and crops the center 100 x 100 portion of the image. Thus, each image has a shape (100, 100, num\_channels). If any dimension (height or width) of an image less than 100 pixels, that image is deleted.

### 3. Training and Validation Data Generator:

```
In [6]: 1 #Loading training dataset
2 train_datagen = ImageDataGenerator(rescale= 1./255,
3                                     shear_range=0.2,
4                                     zoom_range=0.2,
5                                     horizontal_flip=True)
6 train_generator = train_datagen.flow_from_directory('Training1',target_size=(64,64), batch_size=32, class_mode='categorical')
```

Found 1818 images belonging to 2 classes.

```
In [7]: 1 train_generator.class_indices
```

```
Out[7]: {'House': 0, 'Not_House': 1}
```

```
In [8]: 1 #Loading testing data
2 validation = train_datagen.flow_from_directory('Validation1',target_size=(64,64), batch_size=32, class_mode='categorical')
```

Found 321 images belonging to 2 classes.

```
In [9]: 1 validation.class_indices
```

```
Out[9]: {'House': 0, 'Not_House': 1}
```

Directory – “Training1” having **1818 images** in 2 folders (House and Not\_House) and “Validation” having **321 images** in 2 folders (House and Not\_House).

## 2.4.MODELING

### 4. Create the neural net model:

```
In [10]: 1 #initilizing the sequential kernel model and adding layers
2 model = Sequential()
3 model.add(Conv2D(filters=48, kernel_size=3, activation='relu', input_shape=[Image_Width,Image_Height,Image_Channels]))
4 model.add(MaxPool2D(pool_size = 2, strides=2))
5 model.add(keras.layers.Conv2D(filters=40, kernel_size=3, activation='relu'))
6 model.add(keras.layers.MaxPool2D(pool_size=2, strides=2))
7 model.add(Conv2D(filters=32, kernel_size=3, activation='relu'))
8 model.add(MaxPool2D(pool_size = 2, strides=2))
9 model.add(Flatten())
10 model.add(Dense(128, activation='relu'))
11 model.add(Dense(64, activation='relu'))
12 model.add(Dense(64, activation='relu'))
13 model.add(Dense(2, activation='sigmoid'))
14 #compile and train the cnn
15 model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=["accuracy"])
```

#### 2.4.1.Sequential:

A Sequential model is appropriate for a **plain stack of layers** where each layer has **exactly one input tensor and one output tensor**. We can create a Sequential model by passing a list of layers to the Sequential constructor.

The list of layers is as follow:

#### 2.4.2.Conv2D:

```
model.add(Conv2D(filters=48, kernel_size=3, activation='relu', input_shape=[Image_Width,Image_Height,Image_Channels]))
```

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If `use\_bias` is True, a bias vector is created and added to the outputs. Finally,if `activation` is not `None`, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide the keyword argument `input\_shape` (tuple of integers, does not include the sample axis), e.g. `input\_shape=(128, 128, 3)` for 128x128 RGB pictures in `data\_format="channels\_last"`.

**“Model built by using 48 filters, kernel size of 3 with Relu activation function and the given input shape followed the channel last data format”**

**Arguments:**

**filters:** Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).

**kernel\_size:** An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

**data\_format:** A string, one of `channels\_last` (default) or `channels\_first`.The ordering of the dimensions in the inputs. `channels\_last` corresponds to inputs with shape `(batch\_size, height, width, channels)` while `channels\_first` corresponds to inputs with shape `(batch\_size, channels, height, width)`.

### Activation Function:

Applies the rectified linear unit activation function. With default values, this returns the standard ReLU activation:  $\max(x, 0)$ , the element-wise maximum of 0, and the input tensor. Modifying default parameters allows you to use non-zero thresholds, change the max value of the activation, and use a non-zero multiple of the input for values below the threshold.

### 2.4.3.MaxPool2D

```
model.add(MaxPool2D(pool_size = 2, strides=2))
```

Max pooling operation for 2D spatial data.

Downsamples the input representation by taking the maximum value over the window defined by `pool_size` for each dimension along the features axis. The window is shifted by `strides` in each dimension

#### Arguments

- **pool\_size**: integer or tuple of 2 integers, window size over which to take the maximum. (2, 2) will take the max value over a 2x2 pooling window. If only one integer is specified, the same window length will be used for both dimensions.
- **strides**: Integer, tuple of 2 integers, or None. Strides values. Specifies how far the pooling window moves for each pooling step. If None, it will default to `pool_size`.

### 2.4.4.Flatten

```
model.add(Flatten())
```

Flattens the input. Does not affect the batch size.

Note: If inputs are shaped (batch,) without a feature axis, then flattening adds an extra channel dimension and output shape is (batch, 1).

### 2.4.5.Dense

```
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(2, activation='sigmoid'))
```

Dense implements the operation:  $\text{output} = \text{activation}(\text{dot}(\text{input}, \text{kernel}) + \text{bias})$  where `activation` is the element-wise activation function passed as the `activation` argument, `kernel` is a weights matrix created by the layer, and `bias` is a bias vector created by the layer (only applicable if `use_bias` is True).

### 2.4.5.1.Sigmoid Activation Function

We are building a model for binary classification ie) 0 or 1.

```
{ 'House': 0, 'Not_House': 1 }
```

Sigmoid activation function,  $\text{sigmoid}(x) = 1 / (1 + \exp(-x))$ .

Applies the sigmoid activation function. For small values ( $<-5$ ), sigmoid returns a value close to zero, and for large values ( $>5$ ) the result of the function gets close to 1.

Sigmoid is equivalent to a 2-element Softmax, where the second element is assumed to be zero. The sigmoid function always returns a value between 0 and 1.

## 2.5.COMPILING

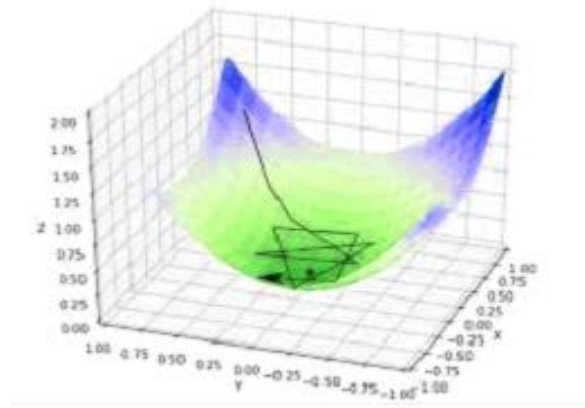
By stacking the above layers in Sequential we have built a model and then we have to compile it. Configures the model for training.

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=["accuracy"])
```

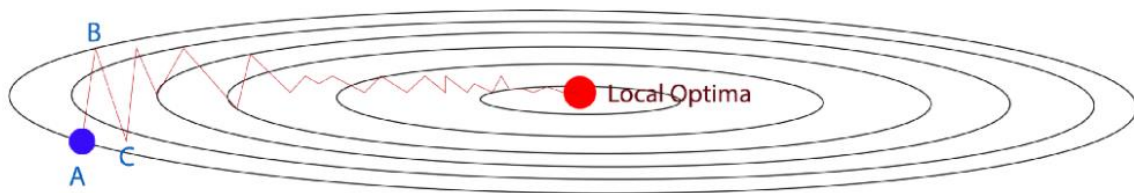
### 2.5.1 Optimizer

RMSprop(Root Mean Square propagation): It is one of the most famous optimization methods in machine learning and deep learning, which is used to boost the speed and accuracy of our model.

Optimization algorithms are used to generate better results by updating the model parameters such as the Weight(W) and bias(b). The most important of them is Gradient Descent. The objective of this algorithm is to reach the global minima where the cost function attains the least possible value. Let's try to visualize the cost function in 3D using the following figure.



**Fig. Convergence**



**Fig. Contour diagram for above convergence**

Here the red point is the minimum. When we start gradient descent from point 'A', after one iteration of gradient descent we may end up at point 'B', the other side of the ellipse. Then another step of gradient descent may end up at point 'C'. So, it will take a lot of steps and move slowly towards the minimum.

But why is there such a bizarre motion? The reason behind this is the presence of a lot of local optima due to high dimension (as in reality the cost-function depends on a lot of weights, therefore increasing the dimension.)

When trying to optimize the parameters in the case of multi-dimension, there are a lot of local optima (one in each dimension). So, it is easier for Gradient Descent to get stuck in a local optimum rather than moving to the global optimum. Therefore the algorithm keeps moving from one local optimum to others and delays reaching the global optimum.

**To overcome this, we want slower learning in the vertical direction and faster learning in the horizontal direction. So here we use RMSProp.**

RMSProp uses the concept of the Exponentially Weighted Average(EWA) of the gradients. So it has been chosen to build a model.

EWA is used to find the moving average. It involves holding the past values in a memory buffer and constantly updating the buffer whenever a new observation is read. This is achieved by using this recursive formula-

$$V_t = \beta V_{t-1} + (1 - \beta) \theta_t$$

Where,

$V_t$ : Moving average value at 't' i.e. averaging  $\theta_t$  over  $1/(1-\beta)$  units (approx).

### 2.5.2.Loss

The purpose of loss functions is to compute the quantity that a model should seek to minimize during training.

BinaryCrossentropy loss:

Computes the cross-entropy loss between true labels and predicted labels. Use this cross-entropy loss when there are only two label classes (assumed to be 0 and 1). For each example, there should be a single floating-point value per prediction. In the snippet below, each of the four examples has only a single floating-pointing value, and both  $y_{pred}$  and  $y_{true}$  have the shape `[batch_size]`.

Here we have to classify only two label classes House or Not House, so we have chosen this loss.

### 2.5.3.Evaluation Metric

A metric is a function that is used to judge the performance of your model.

Metric functions are similar to loss functions, except that the results from evaluating a metric are not used when training the model. Note that you may use any loss function as a metric.

In the given problem statement we have to evaluate our model with **accuracy**.

**Accuracy:**

Calculates how often predictions equal labels.

This metric creates two local variables, total and count that is used to compute the frequency with which  $y_{pred}$  matches  $y_{true}$ .

## 2.6.MODEL FITTING

Trains the model for 20 epochs (iterations on a dataset).

```
hist = model.fit(x=train_generator, validation_data=validation, epochs=20)
```

## 2.7.MODEL ANALYSING

Created model is analyzed by using "model. summary()".

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 62, 48)	1344
max_pooling2d (MaxPooling2D)	(None, 31, 31, 48)	0
conv2d_1 (Conv2D)	(None, 29, 29, 40)	17320
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 40)	0
conv2d_2 (Conv2D)	(None, 12, 12, 32)	11552
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 32)	0
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 128)	147584
dense_1 (Dense)	(None, 64)	8256
dense_2 (Dense)	(None, 64)	4160
dense_3 (Dense)	(None, 2)	130
Total params: 190,346		
Trainable params: 190,346		
Non-trainable params: 0		

Model is fitted in the hist variable, hist. history attribute is a record of training loss values and metrics values at successive epochs, as well as validation loss values and validation metrics values (if applicable).

```
2 hist.history.keys()
```

```
]: dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

To know about the operation, data dimensions, and weights we have to use the keras2ascii.

```
1 from keras_sequential_ascii import keras2ascii
2 keras2ascii(model, sparser=True)
```

OPERATION		DATA DIMENSIONS			WEIGHTS(N)	WEIGHTS(%)
Input	#####	64	64	3		
Conv2D	\ /	-----			1344	0.7%
relu	#####	62	62	48		
MaxPooling2D	Y max	-----			0	0.0%
	#####	31	31	48		
Conv2D	\ /	-----			17320	9.1%
relu	#####	29	29	40		
MaxPooling2D	Y max	-----			0	0.0%
	#####	14	14	40		
Conv2D	\ /	-----			11552	6.1%
relu	#####	12	12	32		
MaxPooling2D	Y max	-----			0	0.0%
	#####	6	6	32		
Flatten		-----			0	0.0%
	#####	1152				
Dense	XXXXX	-----			147584	77.5%

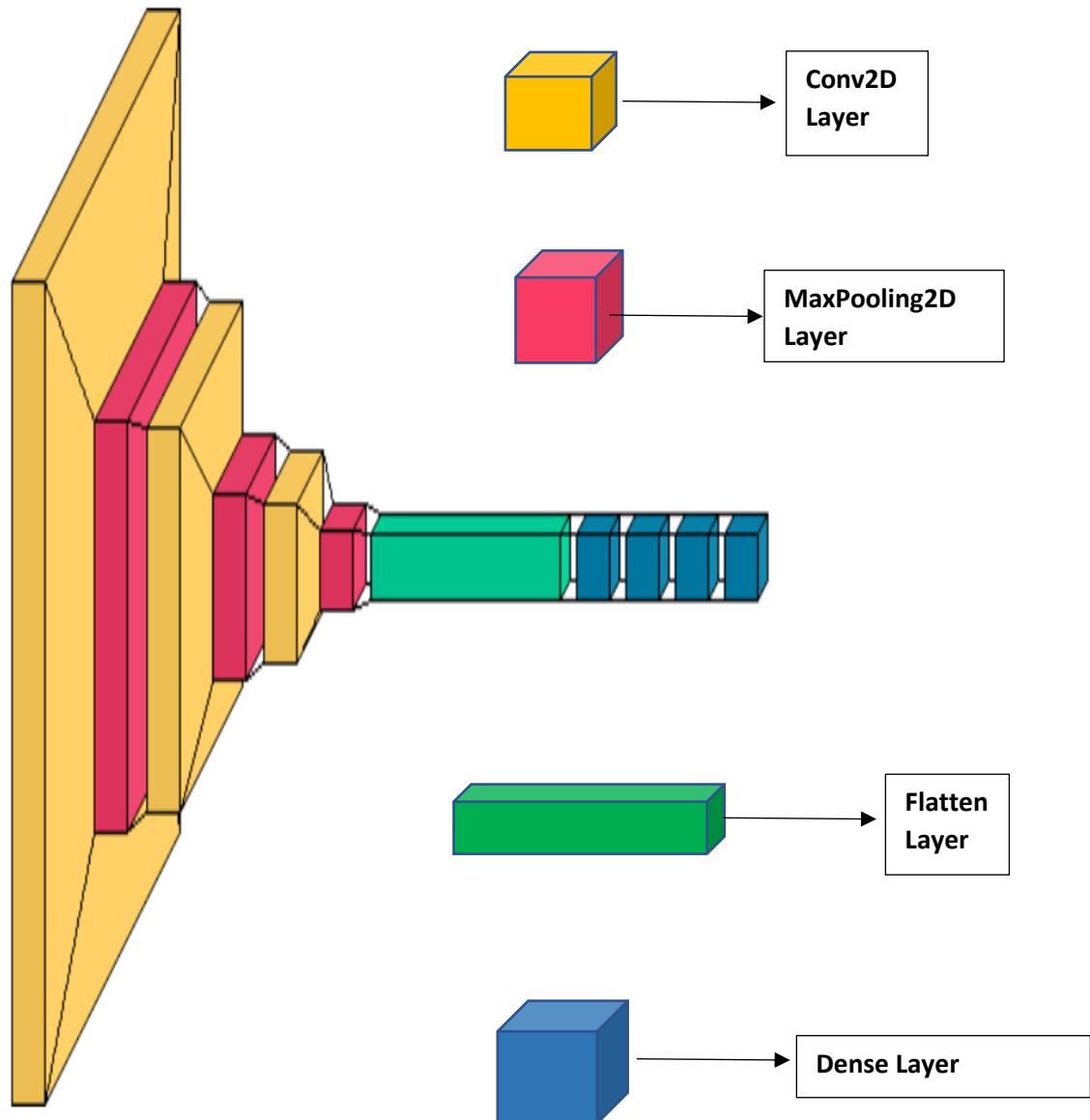
**Fig. Created model with data dimensions, weights, and its operation**



## 2.8.MODEL ARCHITECTURE

```
In [3]: 1 import visulkeras  
2 visulkeras.layered_view(model)
```

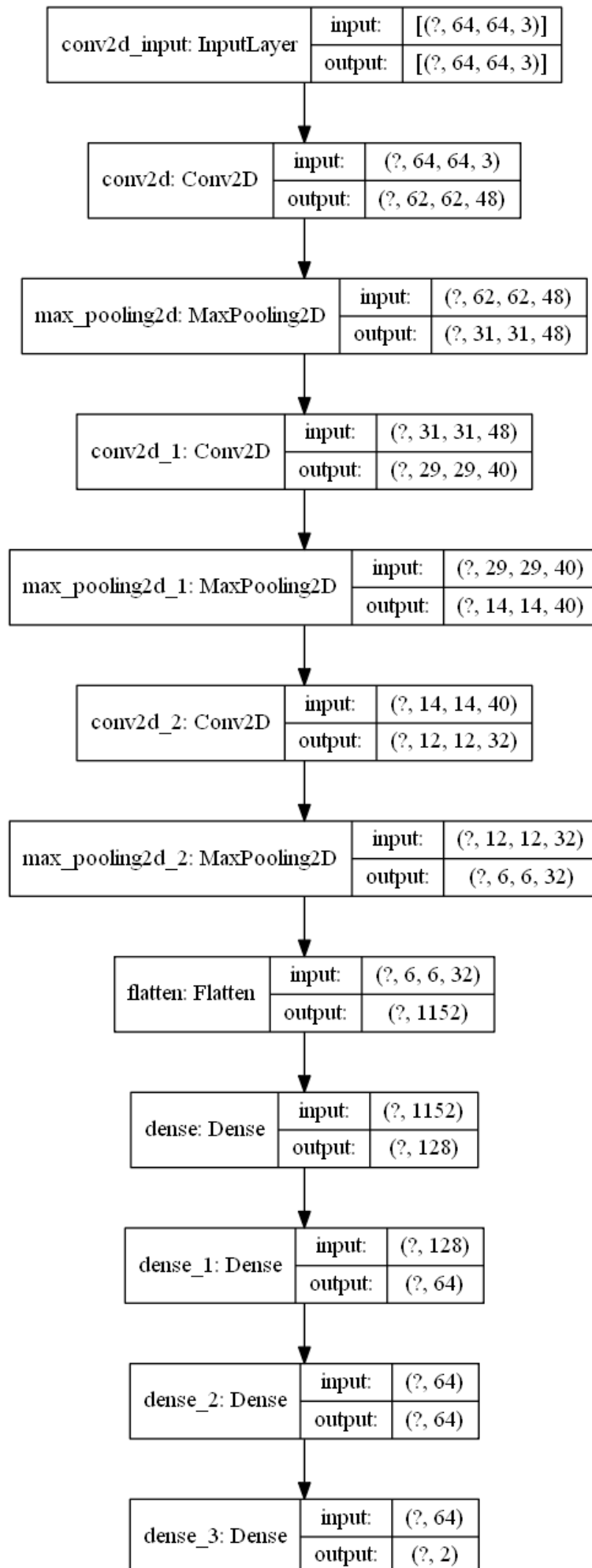
Out[3]:



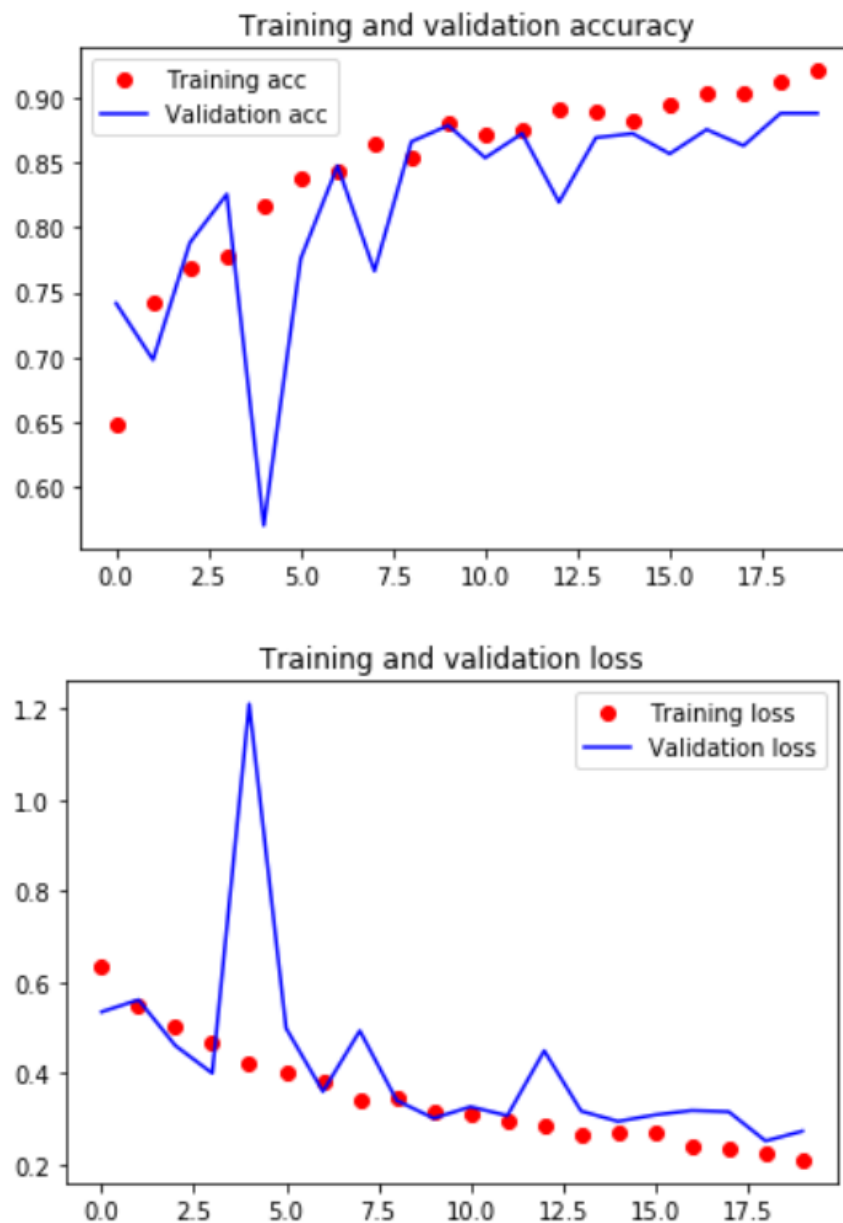
Plot\_model is used to understand the dimension conversion of input along with the different layers.

```
1 from keras.utils.vis_utils import plot_model  
2  
3 plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

The output of the above code is as follow:

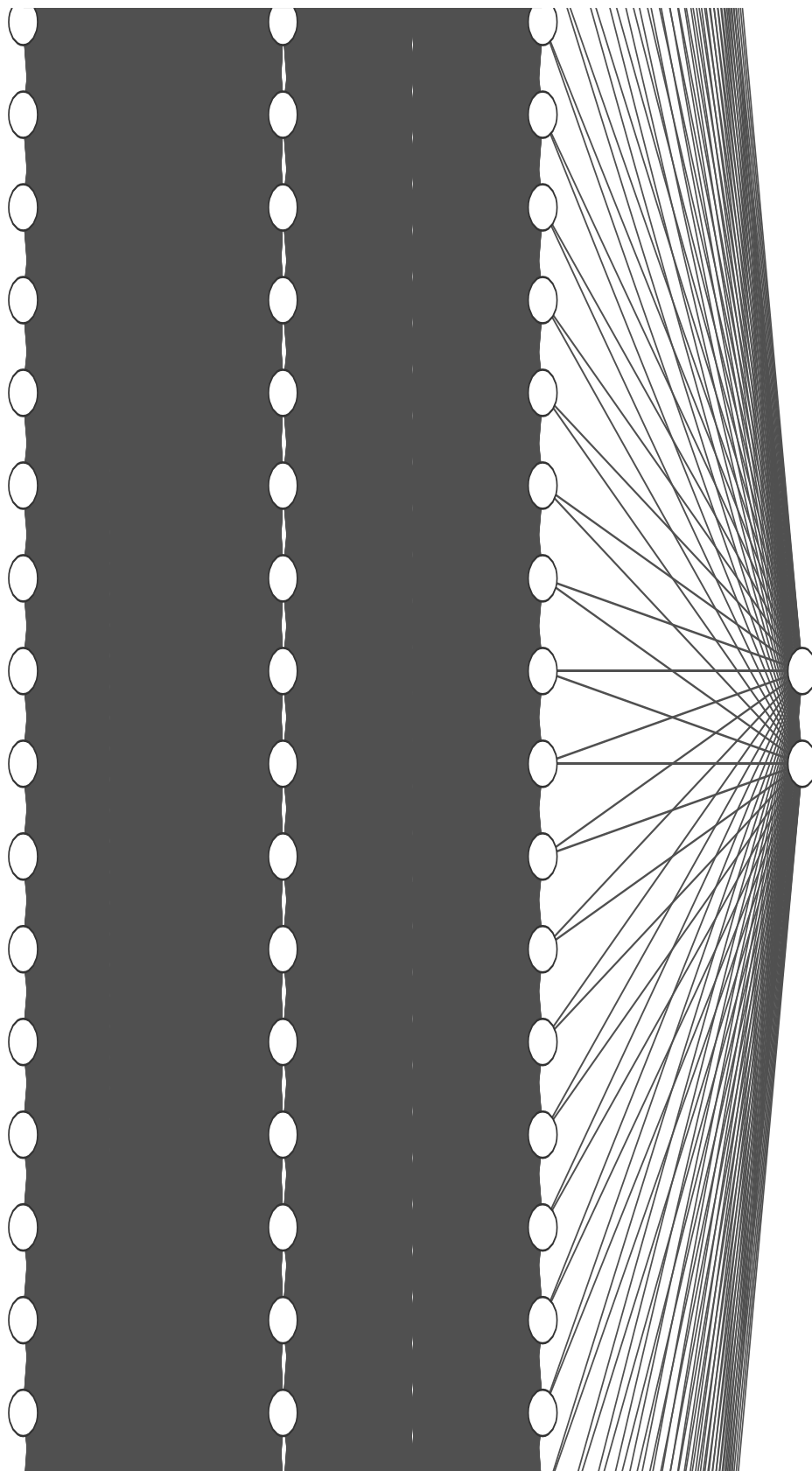


## 2.9. MODEL ACCURACY AND LOSS



**Fig. Training Vs Validation – Accuracy and Loss**

From the above plot, we can observe that the validation accuracy is increasing and losses decreasing linearly, so we can say it is a good model to classify the images of house.



**Fig. Visualizing neurons of Dense Layer (source: nn. svg)**

## 2.10. MODEL SAVING

Training a **neural network/deep learning model** usually takes a lot of time, particularly if the hardware capacity of the system doesn't match up to the requirement. Once the training is done, we **save** the **model** to a file. To reuse the **model** at a later point of time to make predictions, we load the **saved model**.

**Hence**, After fitting the model, we have to save it using `‘.h5’`.

**H5** is a file format to store structured data, it's not a **model** by itself. Keras saves **models** in this format as it can easily store the weights and **model** configuration in a single file.

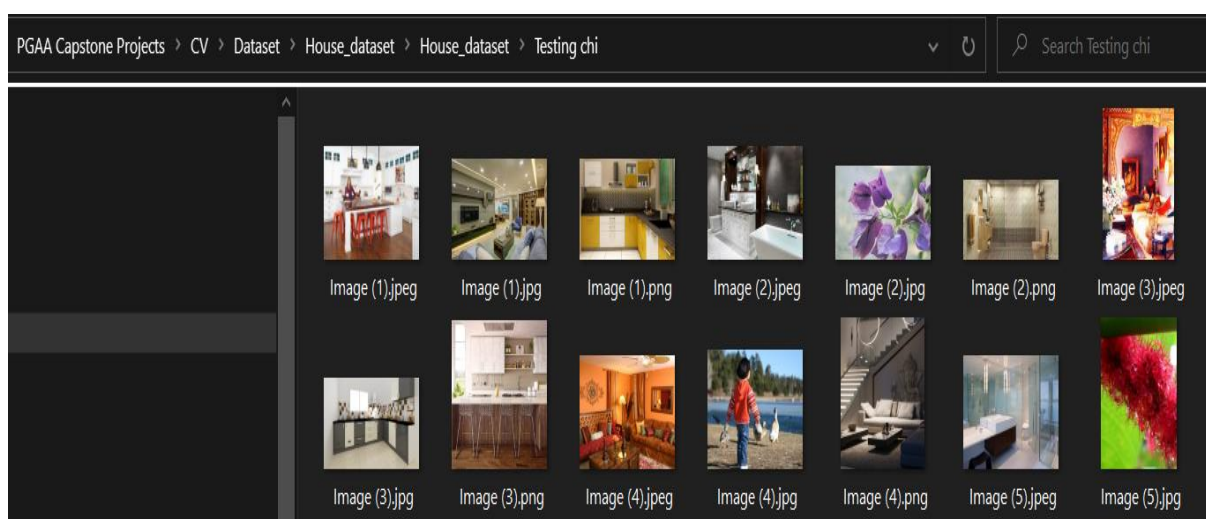
```
model.save("model1_House Vs Non House_20epoch.h5")
```

The saved model is accessed by using `load_model`,

```
1 # Load and evaluate a saved model
2 from numpy import loadtxt
3 from keras.models import load_model
4
5 # Load model
6 model = load_model('model1_House Vs Non House_20epoch.h5')
7 # summarize model.
8 model.summary()
9
```

## 3.PREDICTION OF TEST IMAGES

**Testing Images which having both house and non-house images:**



We have to load the images from the testing directory and predict the images using `“model.predict(test_image)”`


## 10. Make categorical prediction and Convert labels to categories:

```
In [28]: 1 from keras.preprocessing import image
2         # Defining function to testing the model
3         def testing_image(image_directory):
4             test_image = image.load_img(image_directory, target_size = (64, 64))
5             test_image = image.img_to_array(test_image)
6             test_image = np.expand_dims(test_image, axis = 0)
7             result = model.predict(test_image)
8             result = np.argmax(result)
9             if result == 0:
10                prediction = 'HOUSE'
11            else:
12                prediction = 'NOT HOUSE'
13            return prediction
```

## 11. Visualize the prediction results:

### Testing Eg) 1

```
In [29]: 1 new = image.load_img('Image (1).jpeg', target_size = (64,64))
2         new
```

Out[29]: 

```
In [30]: 1 testing_image('Image (1).jpeg')
```

Out[30]: 'HOUSE'

### Testing Eg) 2

```
In [31]: 1 new = image.load_img('Image (4).jpg', target_size = (64,64))
2         new
```

Out[31]: 

```
In [32]: 1 testing_image('Image (4).jpg')
```

Out[32]: 'NOT HOUSE'

From the above image, we can say that created model predicted correctly. The given images Image (1).jpeg and Image (4).jpg from the testing dataset.

## 4.CONCLUSION

The ultimate goal of the project is to not allow the user to upload random images instead of their house on the Dream Housing Property Listings Company website. We have built a model which helps with the automatic check such that the user will be restricted from adding any non-house image, only images relevant to the house will get uploaded to the website.

For Eg)

```
In [38]: 1 # the project goal is to not allow the user to upload an image into website.
2 website = []
3 sample_test = data.head(20)
4 sample_test.head()
5 for index, row in sample_test.iterrows():
6     if row['category'] == "HOUSE":
7         website.append(row['filename'])
8     else:
9         print("Uploaded image is not house")
```

```
Uploaded image is not house
Uploaded image is not house
Uploaded image is not house
Uploaded image is not house
Uploaded image is not house
```

```
In [39]: 1 website
```

```
Out[39]: ['C:\\Users\\sowndariya\\Desktop\\PGAA Capstone Projects-20210205T113733Z-001\\PGAA Capstone Projects\\CV\\Dataset\\House_data
et\\House_dataset\\Testing chi\\Image (1).jpg',
'C:\\Users\\sowndariya\\Desktop\\PGAA Capstone Projects-20210205T113733Z-001\\PGAA Capstone Projects\\CV\\Dataset\\House_data
et\\House_dataset\\Testing chi\\Image (10).jpg',
'C:\\Users\\sowndariya\\Desktop\\PGAA Capstone Projects-20210205T113733Z-001\\PGAA Capstone Projects\\CV\\Dataset\\House_data
et\\House_dataset\\Testing chi\\Image (100).jpg',
```

## 5.REFERENCES

- 5.1.Lin, K., Yang, H. F., Hsiao, J. H., & Chen, C. S. (2015, June). Deep learning of binary hash codes for fast image retrieval. In Computer Vision and Pattern Recognition Workshops (CVPRW), 2015 IEEE Conference on (pp. 27-35).
- 5.2.IEEE. Fu, R., Li, B., Gao, Y., & Wang, P. (2016, October). Content-based image retrieval based on CNN and SVM. In Computer and Communications (ICCC), 2016 2nd IEEE International Conference on (pp. 638-642). IEEE.
- 5.3. Yu, W., Yang, K., Yao, H., Sun, X., & Xu, P. (2017). Exploiting the complementary strengths of multi-layer CNN features for image retrieval. Neurocomputing, 237, 235-241.

You can find the entire coding in github link: [SowndaRiya-M \(SOWNDARIYA M\) \(github.com\)](https://github.com/SowndaRiya-M/SOWNDARIYA-M)