DEEP LEARNING AND NEURAL NETWORK
AIML302

Practical file

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

AMITY UNIVERSITY, UTTAR PRADESH

Submitted by:
RISHABH MISHRA
A2305220566
6CSE8X

Submitted to:
Dr Ashok Kumar Yadav

INDEX

S.No	Topic	Date	Teachers' Sign/Remarks
1.	Design a densenet for temperature conversion.	02/02/23	
2.	Design a densenet for classifying images using fashion MNIST dataset.	09/02/23	
3.	Design a convnet for classifying images using fashion MNIST dataset.	16/02/23	
4.	Design a convnet for classifying dog and cat images.	23/02/23	
5.	Image classification using transfer learning for dogs vs cats dataset.	02/03/23	
6.	Implementation of inflated 3D CNN for action recognition.	09/03/23	
7.	Object detection using CNN.	16/03/23	
8.	Generating images with Big GAN.	23/03/23	
9.	Implement transformer network for translating language.	06/04/23	
10.	Implement MLOps techniques for predicting Bangalore Housing Prices: An experiment using ML models on real Estate data	13/04/23	

EXPERIMENT 1

AIM : To design a densenet for temperature conversion.

Theory: This is a simple temperature conversion problem where we need to convert temperature values fromCelsius to Fahrenheit. The formula for the conversion is:

$$f = c \times 1.8 + 32$$

Where f is the temperature in Fahrenheit and c is the temperature in Celsius.
We can use this formula to create a machine learning model that can learn to perform the temperature conversion automatically. The input to the model will be the temperature in Celsius and the output will be thetemperature in Fahrenheit.

CODE AND OUTPUT

Import Dependencies :

```
[2] import tensorflow as tf
```

```
[3] import numpy as np
    import logging
    logger = tf.get_logger()
    logger.setLevel(logging.ERROR)
```

Set Up Training Data:

```
[4] celsius_q    = np.array([-40, -10,  0,  8, 15, 22,  38],  dtype=float)
    fahrenheit_a = np.array([-40,  14, 32, 46, 59, 72, 100],  dtype=float)

    for i,c in enumerate(celsius_q):
      print("{} degrees Celsius = {} degrees Fahrenheit".format(c, fahrenheit_a[i]))

-40.0 degrees Celsius = -40.0 degrees Fahrenheit
-10.0 degrees Celsius = 14.0 degrees Fahrenheit
0.0 degrees Celsius = 32.0 degrees Fahrenheit
8.0 degrees Celsius = 46.0 degrees Fahrenheit
15.0 degrees Celsius = 59.0 degrees Fahrenheit
22.0 degrees Celsius = 72.0 degrees Fahrenheit
38.0 degrees Celsius = 100.0 degrees Fahrenheit
```

Create the Model : We will use the simplest possible model we can, a Dense network. Since the problem is straightforward, this network will require only a single layer, with a single neuron.

```
[5] l0 = tf.keras.layers.Dense(units=1, input_shape=[1])
```

Assemble layers into the model :
Once layers are defined, they need to be assembled into a model. The Sequential model definition takes a list of layers as an argument, specifying the calculation order from the input to the output.
This model has just a single layer, l0.

```
[6] model = tf.keras.Sequential([l0])
```

Compile the model, with loss and optimizer functions : Before training, the model has to be compiled. When compiled for training, the model is given:
Loss function - A way of measuring how far off predictions are from the desired outcome. (The measured difference is called the "loss".)

Optimizer function - A way of adjusting internal values in order to reduce the loss.

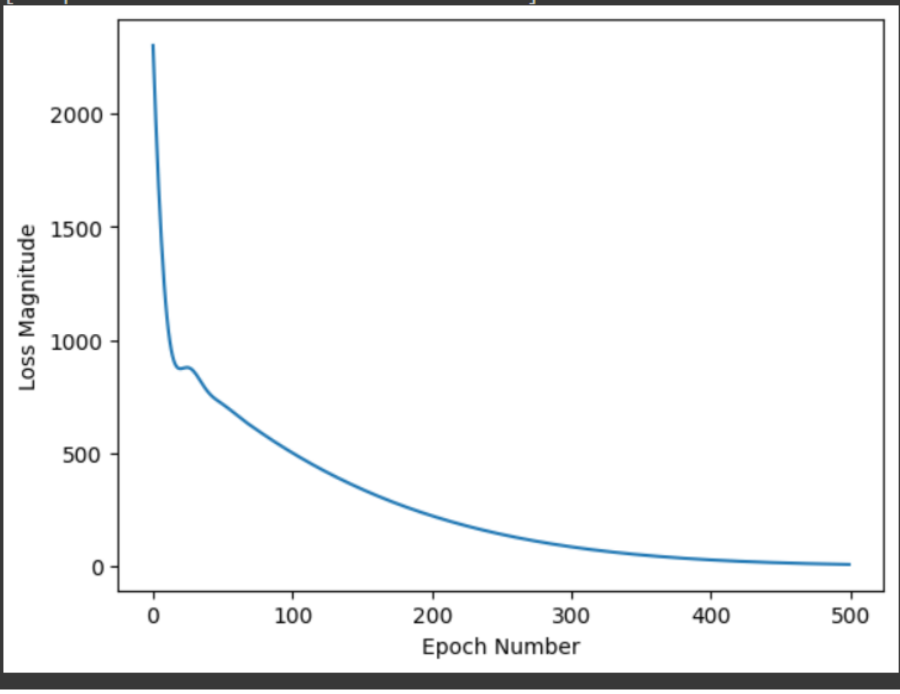
```
[7] model.compile(loss='mean_squared_error',
                  optimizer=tf.keras.optimizers.Adam(0.1))
```

Train the model:

```
[8] history = model.fit(celsius_q, fahrenheit_a, epochs=500, verbose=False)
    print("Finished training the model")

    Finished training the model
```

Display Training Statistics:

```
[9] import matplotlib.pyplot as plt
    plt.xlabel('Epoch Number')
    plt.ylabel("Loss Magnitude")
    plt.plot(history.history['loss'])

    [<matplotlib.lines.Line2D at 0x7fed7810e9d0>]
```



Use the Model to Predict Values:

```
[10] print(model.predict([100.0]))
```

```
1/1 [==============================] - 0s 77ms/step
[[211.3295]]
```

Looking at other layer weights: Finally, let's print the internal variables of the Dense layer.

```
[11] print("These are the layer variables: {}".format(l0.get_weights()))
```

CONCLUSION: A densenet for temperature conversion has been designed.

EXPERIMENT 2

AIM : Design a densenet for classifying images using fashion MNIST dataset.

Thoery: DenseNet is a deep neural network architecture that has been used successfully in a variety of image classification tasks, including the classification of the Fashion MNIST dataset. In this architecture, each layer is connected to every other layer in a feed-forward fashion, resulting in a dense and highly connected network.

To design a DenseNet for classifying images using the Fashion MNIST dataset, we need to consider several factors:

Input size: The Fashion MNIST images have a resolution of 28x28 pixels and are grayscale, so the input size of our network should be (28, 28, 1).

Depth: The depth of our network, i.e., the number of layers, is an important factor that can affect the performance of the network.

Growth rate: The growth rate determines how many new feature maps are added to the network at each layer. A higher growth rate can lead to better performance but also increases the number of parameters andthe computational cost of the network.

Compression factor: The compression factor controls the amount of compression applied to the feature maps in the transition layer. A higher compression factor results in more aggressive compression and asmaller number of feature maps, which can reduce the computational cost of the network.

Activation function: We can experiment with different activation functions, such as ReLU or LeakyReLU, tofind the optimal function for our network.

Dropout: We can use dropout regularization to prevent overfitting and improve the generalizationperformance of our network.

Once we have determined the hyperparameters of our DenseNet, we can train the network using the Fashion MNIST dataset and evaluate its performance on a separate test set. We can also use techniques such as data augmentation and transfer learning to improve the performance of our network and reduce overfitting.

CODE AND OUTPUT

Importing the functionalities

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

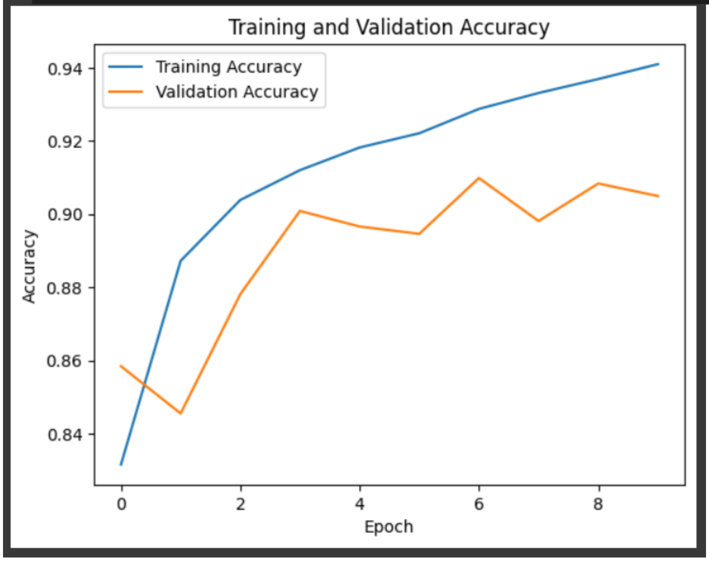
```
# Loading data
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()

# Normalizing pixel values
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshaping data
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)
```

```python
# Defining DenseNet model
def dense_net():
    inputs = keras.Input(shape=(28, 28, 1))
    x = layers.Conv2D(64, 7, padding='same')(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling2D(2)(x)
    num_blocks = 3
    for i in range(num_blocks):
        x, nb_filters = dense_block(x, 32)
        x = transition_layer(x, nb_filters)
    x = layers.GlobalAveragePooling2D()(x)
    outputs = layers.Dense(10, activation='softmax')(x)
    model = keras.Model(inputs=inputs, outputs=outputs, name='DenseNet')
    return model

def dense_block(x, nb_filters):
    concat = x
    for i in range(4):
        x = layers.BatchNormalization()(x)
        x = layers.Activation('relu')(x)
        x = layers.Conv2D(nb_filters, 3, padding='same')(x)
        concat = layers.Concatenate()([concat, x])
    return concat, nb_filters + 32

def transition_layer(x, nb_filters):
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.Conv2D(nb_filters // 2, 1)(x)
    x = layers.AveragePooling2D(2)(x)
    return x
```

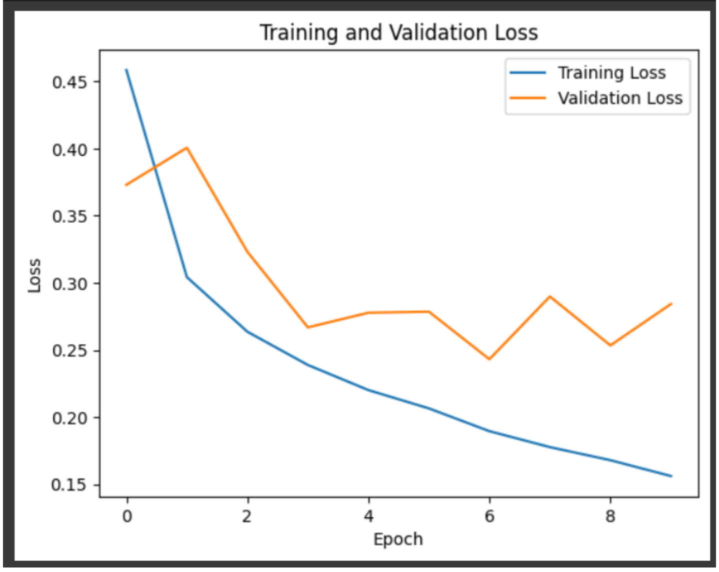
```python
# Creating model instance
model = dense_net()

# Compiling model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Training model
history = model.fit(x_train, y_train, batch_size=64, epochs=10, validation_split=0.2)

# Evaluating model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test loss: {test_loss}, Test accuracy: {test_acc}')
```

```
Epoch 1/10
750/750 [==============================] - 21s 14ms/step - loss: 0.4607 - accuracy: 0.8322 - val_loss: 0.4142 - val_accuracy: 0.8443
Epoch 2/10
750/750 [==============================] - 10s 13ms/step - loss: 0.3097 - accuracy: 0.8857 - val_loss: 0.4038 - val_accuracy: 0.8551
Epoch 3/10
750/750 [==============================] - 10s 13ms/step - loss: 0.2698 - accuracy: 0.8992 - val_loss: 0.3368 - val_accuracy: 0.8759
Epoch 4/10
750/750 [==============================] - 10s 13ms/step - loss: 0.2455 - accuracy: 0.9079 - val_loss: 0.3523 - val_accuracy: 0.8745
Epoch 5/10
750/750 [==============================] - 9s 13ms/step - loss: 0.2280 - accuracy: 0.9154 - val_loss: 0.3256 - val_accuracy: 0.8870
Epoch 6/10
750/750 [==============================] - 10s 13ms/step - loss: 0.2119 - accuracy: 0.9213 - val_loss: 0.2464 - val_accuracy: 0.9098
Epoch 7/10
750/750 [==============================] - 10s 13ms/step - loss: 0.1976 - accuracy: 0.9270 - val_loss: 0.2574 - val_accuracy: 0.9083
Epoch 8/10
750/750 [==============================] - 10s 13ms/step - loss: 0.1852 - accuracy: 0.9313 - val_loss: 0.2589 - val_accuracy: 0.9011
Epoch 9/10
750/750 [==============================] - 10s 13ms/step - loss: 0.1736 - accuracy: 0.9348 - val_loss: 0.3042 - val_accuracy: 0.8925
Epoch 10/10
750/750 [==============================] - 10s 13ms/step - loss: 0.1622 - accuracy: 0.9392 - val_loss: 0.2906 - val_accuracy: 0.8984
313/313 [==============================] - 1s 4ms/step - loss: 0.3024 - accuracy: 0.8981
Test loss: 0.30238908529281616, Test accuracy: 0.8981000185012817
```

```
[2] # Plot the training and validation accuracy and loss
    import matplotlib.pyplot as plt

    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
```




```
[4] plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
```



CONCLUSION : A densenet for classifying images using fashion MNIST dataset has been implemented

EXPERIMENT 3

AIM : To design a convnet for classifying images using fashion MNIST dataset

Theory: Convolutional neural networks (CNNs) are a powerful class of neural networks that are widely used for image classification tasks. The Fashion MNIST dataset is a collection of images of clothing items, where each image is a 28x28 grayscale image. The goal is to classify each image into one of ten categories, such as T-shirts, dresses, shoes, etc.
To design a CNN for classifying images using the Fashion MNIST dataset, we can follow the following steps:

- **Preprocessing**: Before we can train the CNN, we need to preprocess the data. This involves normalizing the pixel values of the images and converting the class labels into one-hot encoded vectors.

- **Model architecture:** We can design the model architecture by stacking a sequence of convolutional layers, followed by pooling layers, and then a few fully connected layers. The convolutional layers use a set of filters to scan across the input image, looking for patterns that are relevant to the classification task. The pooling layers down sample the output of the convolutional layers to reduce the computational complexity of the model. The fully connected layers take the output of the convolutional layers and perform a series of nonlinear transformations to produce the final output.

- **Training:** Once the model architecture is defined, we can train CNN using the Fashion MNIST training set. During training, the model learns to adjust the weights of the filters to minimize the difference between the predicted class labels and the true class labels.

- **Evaluation**: After training, we can evaluate the performance of the model on the Fashion MNIST test set. We can compute metrics such as accuracy, precision, recall, and F1 score to measure how well the model is able to classify the images.

CODE AND OUTPUT

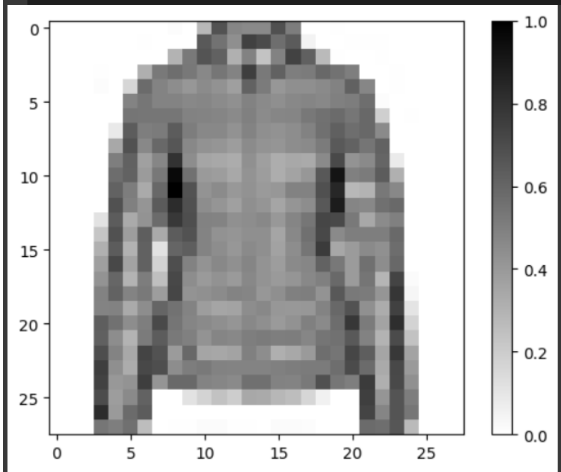
```
[3] import tensorflow as tf
    # Import TensorFlow Datasets
    import tensorflow_datasets as tfds
    tfds.disable_progress_bar()

    # Helper libraries
    import math
    import numpy as np
    import matplotlib.pyplot as plt

    import logging
    logger = tf.get_logger()
    logger.setLevel(logging.ERROR)

    dataset, metadata = tfds.load('fashion_mnist', as_supervised=True, with_info=True)
    train_dataset, test_dataset = dataset['train'], dataset['test']
    class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
    'Sandal', 'Shirt',  'Sneaker', 'Bag', 'Ankle boot']
```

```
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size) to /root/tensorflow_datasets/fashion_mnist/3.0.1...
Dataset fashion_mnist downloaded and prepared to /root/tensorflow_datasets/fashion_mnist/3.0.1. Subsequent calls will reuse this data.
```

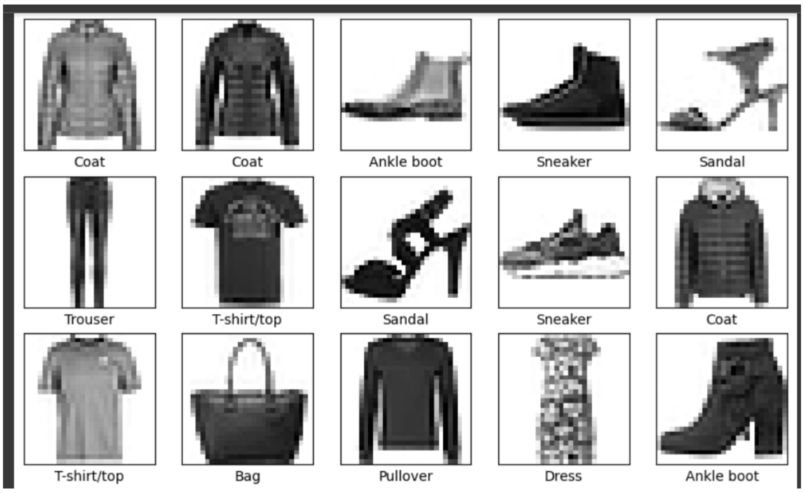
```
[4] num_train_examples = metadata.splits['train'].num_examples
    num_test_examples = metadata.splits['test'].num_examples
    print("Number of training examples: {}".format(num_train_examples))
    print("Number of test examples: {}".format(num_test_examples))
    def normalize(images, labels):
      images = tf.cast(images, tf.float32)
      images /= 255
      return images, labels

    # The map function applies the normalize function to each element in the train # and test datasets
    train_dataset = train_dataset.map(normalize)
    test_dataset = test_dataset.map(normalize)

    # The first time you use the dataset, the images will be loaded from disk # Caching will keep them in memory, making training faster
    train_dataset = train_dataset.cache()
    test_dataset = test_dataset.cache()
```

```
Number of training examples: 60000
Number of test examples:     10000
```

```
[8] # Take a single image, and remove the color dimension by reshaping
    for image, label in test_dataset.take(1):
      break
    image = image.numpy().reshape((28,28))

    # Plot the image - voila a piece of fashion clothing
    plt.figure()
    plt.imshow(image, cmap=plt.cm.binary)
    plt.colorbar()
    plt.grid(False)
    plt.show()
```




```
[9] plt.figure(figsize=(10,10))
    i = 0
    for (image, label) in test_dataset.take(25):
      image = image.numpy().reshape((28,28))
      plt.subplot(5,5,i+1)
      plt.xticks([])
      plt.yticks([])
      plt.grid(False)
      plt.imshow(image, cmap=plt.cm.binary)
      plt.xlabel(class_names[label])
      i += 1
    plt.show()
```




```
[13] model = tf.keras.Sequential([
     tf.keras.layers.Conv2D(32, (3,3), padding='same', activation=tf.nn.relu, input_shape=(28, 28, 1)),
     tf.keras.layers.MaxPooling2D((2, 2), strides=2),
     tf.keras.layers.Conv2D(64, (3,3), padding='same', activation=tf.nn.relu), tf.keras.layers.MaxPooling2D((2, 2), strides=2),
     tf.keras.layers.Flatten(),
     tf.keras.layers.Dense(128, activation=tf.nn.relu), tf.keras.layers.Dense(10, activation=tf.nn.softmax)
     ])

     model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(), metrics=['accuracy'])
     BATCH_SIZE = 32
     train_dataset = train_dataset.cache().repeat().shuffle(num_train_examples).batch(BATCH_SIZE)
     test_dataset = test_dataset.cache().batch(BATCH_SIZE)

     model.fit(train_dataset, epochs=10, steps_per_epoch=math.ceil(num_train_examples/BATCH_SIZE))
```

```
Epoch 1/10
1875/1875 [==============================] - 24s 4ms/step - loss: 0.3902 - accuracy: 0.8581
Epoch 2/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2574 - accuracy: 0.9063
Epoch 3/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2052 - accuracy: 0.9252
Epoch 4/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1764 - accuracy: 0.9349
Epoch 5/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1482 - accuracy: 0.9442
Epoch 6/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1228 - accuracy: 0.9552
Epoch 7/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1038 - accuracy: 0.9619
Epoch 8/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0879 - accuracy: 0.9672
Epoch 9/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0732 - accuracy: 0.9733
Epoch 10/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0627 - accuracy: 0.9768
<keras.callbacks.History at 0x7fefcf230610>
```

```
[15] test_loss, test_accuracy = model.evaluate(test_dataset, steps=math.ceil(num_test_examples/BATCH_SIZE))
     print('Accuracy on test dataset:', test_accuracy)

     313/313 [==============================] - 2s 7ms/step - loss: 0.3178 - accuracy: 0.9188
     Accuracy on test dataset: 0.9187999963760376
```

CONCLUSION: A convnet for classifying images using fashion MNIST dataset has been designed.

EXPERIMENT 4

AIM : To design a convnet for classifying dog and cat images

Theory: To design a convolutional neural network (convnet) for classifying dog and cat images, the following steps can be followed:

- Load the dataset: The dog and cat image dataset can be downloaded from various sources, such as Kaggle, and stored in a directory structure where each class has its own folder. The tensorflow.keras.preprocessing.image module provides a function ImageDataGenerator that can be used to load and preprocess the images.

- Preprocess the data: Before training the convnet, the data should be preprocessed. This includes resizing the images to a standard size (e.g., 150x150 pixels), normalizing the pixel values to a range of 0 to 1, and splitting the data into training and validation sets.

- Design the model: The convnet can be designed using the tensorflow.keras API. The model should start with a convolutional layer with a small kernel size (e.g., 3x3) and a small number of filters (e.g., 32). This should be followed by a max pooling layer to reduce the spatial dimensions of the feature maps. This process of convolution and pooling can be repeated multiple times, with increasing number of filters in each layer. The final output of the convolutional layers can be flattened and fed into a fully connected (dense) layer, followed by a final output layer with 2 neurons (one for each class).

- Compile the model: After designing the model, it needs to be compiled with an appropriate loss function, optimizer, and evaluation metric. Since this is a binary classification problem, binary cross-entropy can be used as the loss function, and the Adam optimizer can be used for training. The accuracy metric can be used to evaluate the performance of the model.

- Train the model: The compiled model can be trained on the preprocessed training data using the model.fit() method. The number of epochs and batch size can be adjusted to optimize the performance of the model.

- Evaluate the model: After training the model, its performance can be evaluated on the preprocessed validation data using the model.evaluate() method. This will provide the accuracy of the model on the validation data.

- Make predictions: Finally, the trained model can be used to make predictions on new, unseen data using the model.predict() method. The predicted class can be the one with the highest probability.

CODE AND OUTPUT:

```
!wget --no-check-certificate \
  https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip \
  -O /tmp/cats_and_dogs_filtered.zip
```

--2023-04-14 13:49:34-- https://storage.googleapis.com/mledu-datasets/cat s_and_dogs_filtered.zip (https://storage.googleapis.com/mledu-datasets/cat s_and_dogs_filtered.zip)
Resolving storage.googleapis.com (storage.googleapis.com)... 142.250.98.12 8, 142.251.107.128, 74.125.196.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|142.250.98.1 28|:443... connected.
HTTP request sent, awaiting response... 200 OK Length: 68606236 (65M) [application/zip]
Saving to: ‘/tmp/cats_and_dogs_filtered.zip’

/tmp/cats_and_dogs_ 100%[===================>] 65.43M   188MB/s   in 0.3 s

2023-04-14 13:49:35 (188 MB/s) - ‘/tmp/cats_and_dogs_filtered.zip’ saved [68606236/68606236]

In [ ]:

```
import os
import zipfile

local_zip='/tmp/cats_and_dogs_filtered.zip'
zip_ref=zipfile.ZipFile(local_zip,'r')
zip_ref.extractall('/tmp')
zip_ref.close()
os.listdir('/tmp')
```

Out[17]:

['pyright-732-aQJulZJouSRb',
'cats_and_dogs_filtered.zip',
'dap_multiplexer.ff827b0d1590.root.log.INFO.20230414-131418.106', 'python-languageserver-cancellation',
'dap_multiplexer.INFO', 'debugger_1m37wi21wi',
'initgoogle_syslog_dir.0', 'pyright-732-a6U43jHmasbP', 'cats_and_dogs_filtered']

```
base_dir='/tmp/cats_and_dogs_filtered'

#training and validation directory
train_dir=os.path.join(base_dir,'train')
validation_dir=os.path.join(base_dir,'validation')

#training directory
train_cats_dir=os.path.join(train_dir,'cats')
train_dogs_dir=os.path.join(train_dir,'dogs')

#validation directory
validation_cats_dir = os.path.join(validation_dir, 'cats')
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
```

In [ ]:

```
train_cat_fnames=os.listdir(train_cats_dir)
train_dog_fnames=os.listdir(train_dogs_dir)

print(train_cat_fnames[:10])
print(train_dog_fnames[:10])


print('Total training cat images.   ',len(os.listdir(train_cats_dir)))
print('Total training dog images.   ',len(os.listdir(train_dogs_dir)))

print('Total Validation cat images.   ',len(os.listdir(validation_cats_dir)))
print('Total Validation dog images.   ',len(os.listdir(validation_dogs_dir)))
```

['cat.546.jpg', 'cat.134.jpg', 'cat.435.jpg', 'cat.816.jpg', 'cat.383.jp
g', 'cat.204.jpg', 'cat.465.jpg', 'cat.64.jpg', 'cat.513.jpg', 'cat.783.jp g']
['dog.939.jpg', 'dog.218.jpg', 'dog.192.jpg', 'dog.43.jpg', 'dog.791.jpg',
'dog.467.jpg', 'dog.408.jpg', 'dog.602.jpg', 'dog.14.jpg', 'dog.624.jpg']
Total training cat images. 1000

Total training dog images.	1000
Total Validation cat images.	500
Total Validation dog images.	500

```
import tensorflow as tf
model = tf.keras.models.Sequential([
    # Note the input shape is the desired size of the image 150x150 with 3 bytes color
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    # Only 1 output neuron. It will contain a value from 0-1 where 0 for 1 class ('cats'
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

In [ ]:

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 16)	448
max_pooling2d (MaxPooling2D)	(None, 74, 74, 16)	0
conv2d_1 (Conv2D)	(None, 72, 72, 32)	4640
max_pooling2d_1 (MaxPooling 2D)	(None, 36, 36, 32)	0
conv2d_2 (Conv2D)	(None, 34, 34, 64)	18496
max_pooling2d_2 (MaxPooling 2D)	(None, 17, 17, 64)	0
flatten (Flatten)	(None, 18496)	0
dense (Dense)	(None, 512)	9470464
dense_1 (Dense)	(None, 1)	513

===

Total params: 9,494,561
Trainable params: 9,494,561
Non-trainable params: 0

```
from tensorflow.keras.optimizers import RMSprop

model.compile(optimizer=RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics = ['acc'])
```

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_r ate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.

In [ ]:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled here
train_datagen = ImageDataGenerator( rescale = 1.0/255. )
test_datagen  = ImageDataGenerator( rescale = 1.0/255. )

train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=20,
                                                    class_mode='binary',
                                                    target_size=(150, 150))

validation_generator =  test_datagen.flow_from_directory(validation_dir,
                                                         batch_size=20,
                                                         class_mode  = 'binary',
                                                         target_size = (150, 150))
```

Found 2000 images belonging to 2 classes. Found 1000 images belonging to 2 classes.

```
history = model.fit_generator(train_generator,
                              validation_data=validation_generator,
                              steps_per_epoch=10,
                              epochs=10,
                              validation_steps=50,
                              verbose=2)
```

Epoch 1/10

<ipython-input-26-73665b94786d>:1: UserWarning: `Model.fit_generator` is d eprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
history = model.fit_generator(train_generator,

10/10 - 3s - loss: 0.6946 - acc: 0.515 - val_loss: 0.692 - val_acc: 0.5
0 2 0
00 - 3s/epoch - 345ms/step
Epoch 2/10
10/10 - 4s - loss: 0.7237 - acc: 0.540 - val_loss: 0.693 - val_acc: 0.5
0 3 1
10 - 4s/epoch - 448ms/step
Epoch 3/10
10/10 - 3s - loss: 0.6966 - acc: 0.550 - val_loss: 0.693 - val_acc: 0.5
0 5 0
00 - 3s/epoch - 345ms/step
Epoch 4/10
10/10 - 3s - loss: 0.6861 - acc: 0.570 - val_loss: 0.693 - val_acc: 0.5
0 0 0
00 - 3s/epoch - 346ms/step
Epoch 5/10
10/10 - 4s - loss: 0.6967 - acc: 0.525 - val_loss: 0.686 - val_acc: 0.5
0 3 8
80 - 4s/epoch - 440ms/step
Epoch 6/10
10/10 - 3s - loss: 0.6871 - acc: 0.575 - val_loss: 0.683 - val_acc: 0.5
0 6 9
10 - 3s/epoch - 349ms/step
Epoch 7/10
10/10 - 3s - loss: 0.6894 - acc: 0.555 - val_loss: 0.679 - val_acc: 0.5
0 0 9
50 - 3s/epoch - 342ms/step
Epoch 8/10
10/10 - 4s - loss: 0.6712 - acc: 0.575 - val_loss: 0.678 - val_acc: 0.6
0 3 1
60 - 4s/epoch - 399ms/step
Epoch 9/10
10/10 - 3s - loss: 0.7452 - acc: 0.625 - val_loss: 0.681 - val_acc: 0.5
0 0 5
60 - 3s/epoch - 344ms/step
Epoch 10/10
10/10 - 4s - loss: 0.7155 - acc: 0.615 - val_loss: 0.672 - val_acc: 0.6
0 6 4
00 - 4s/epoch - 351ms/step

In [ ]:

```
#------------------------------------------------------------
# Retrieve a list of list results on training and test data
# sets for each training epoch
#------------------------------------------------------------
acc      = history.history[     'acc' ]
val_acc  = history.history[ 'val_acc' ]
loss     = history.history[    'loss' ]
val_loss = history.history['val_loss' ]
```

```
import matplotlib.pyplot as plt
# Plot training and validation accuracy per epoch
plt.plot  ( epochs,     acc )
plt.plot  ( epochs, val_acc )
plt.title ('Training and validation accuracy')
plt.figure()

# Plot training and validation loss per epoch
plt.plot  ( epochs,     loss )
plt.plot  ( epochs, val_loss )
plt.title ('Training and validation loss'   )
```

Out[30]:

Text(0.5, 1.0, 'Training and validation loss')

CONCLUSION : A convnet for classifying dog and cat images has been designed

EXPERIMENT 5

AIM : To implement image classification using transfer learning for dogs vs cats dataset

Theory: Image classification using transfer learning for dogs vs cats dataset is a popular application of deep learning in computer vision. In this task, we use a pre-trained neural network to classify images of dogs and cats.
Transfer learning refers to the use of a pre-trained model as a starting point for a new model and fine-tuning it to fit a new task.
The VGG16 model, pre-trained on the ImageNet dataset, is commonly used as a base model for this task. The VGG16 model consists of 13 convolutional layers and 3 fully connected layers and has been shown to achieve high accuracy on image classification tasks.
To use transfer learning with the VGG16 model, we can load the pre-trained model, freeze the layers, and add some fully connected layers on top of it. We can then train the added layers on our own dataset of dogs and cats. By doing this, we can leverage the knowledge learned by the VGG16 model on the ImageNet dataset and use it to improve the accuracy of our own model.

CODE AND OUTPUT :

In [ ]:

```
import tensorflow as tf
import numpy as np
import os
import cv2
import matplotlib.pyplot as plt
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout
```

In [ ]:

```
import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

```
_URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'
zip_dir = tf.keras.utils.get_file('cats_and_dogs_filterted.zip', origin=_URL, extract=Tr
```

Downloading data from https://storage.googleapis.com/mledu-datasets/cats_a nd_dogs_filtered.zip (https://storage.googleapis.com/mledu-datasets/cats_a nd_dogs_filtered.zip)
68606236/68606236 [==============================] - 3s 0us/step

In [ ]:

```
zip_dir_base = os.path.dirname(zip_dir)
!find $zip_dir_base -type d -print
```

/root/.keras/datasets
/root/.keras/datasets/cats_and_dogs_filtered
/root/.keras/datasets/cats_and_dogs_filtered/train
/root/.keras/datasets/cats_and_dogs_filtered/train/dogs
/root/.keras/datasets/cats_and_dogs_filtered/train/cats
/root/.keras/datasets/cats_and_dogs_filtered/validation

/root/.keras/datasets/cats_and_dogs_filtered/validation/dogs
/root/.keras/datasets/cats_and_dogs_filtered/validation/cats

In [ ]:

```
base_dir = os.path.join(os.path.dirname(zip_dir), 'cats_and_dogs_filtered')
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
```

In [ ]:

```
# Set parameters
batch_size = 32
epochs = 10
img_height = 150
img_width = 150
```

In [ ]:

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary')
```

Found 2000 images belonging to 2 classes.

```
val_datagen = ImageDataGenerator(rescale=1./255)
val_generator = val_datagen.flow_from_directory(
    validation_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary')
```

Found 1000 images belonging to 2 classes

.In [ ]:

```
# Get a batch of images from the validation generator
x_batch, y_batch = val_generator.next()

# Display the first few images
num_images = 5
for i in
    range(
```

In [ ]:

```
# Load the pre-trained model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(img_height, img_w
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-appl
ications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5 (https://
storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_t
f_dim_ordering_tf_kernels_notop.h5)
58889256/58889256 [==============================] - 3s 0us/step

In [ ]:

```
# Freeze the base model
base_model.trainable = False
```

In [ ]:

```
# Build the model
model = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```

In [ ]:

```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [ ]:

```
epochs=2
# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=val_generator,
    validation_steps=val_generator.samples // batch_size)
```

Epoch 1/2
62/62 [==============================] - 801s 13s/step - loss: 0.3558 - ac
curacy: 0.8379 - val_loss: 0.2890 - val_accuracy: 0.8669 Epoch 2/2
62/62 [==============================] - 791s 13s/step - loss: 0.3091 - ac
curacy: 0.8694 - val_loss: 0.2634 - val_accuracy: 0.8821

```
# Save the model
model.save('dogs_vs_cats.h5')
```

In [ ]:

```
# Evaluate the model
test_generator = val_datagen.flow_from_directory(
    validation_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=False)
test_loss, test_acc = model.evaluate(test_generator, steps=test_generator.samples // bat
```

Found 1000 images belonging to 2 classes.
31/31 [==============================] - 257s 8s/step - loss: 0.2654 - acc
uracy: 0.8810

In [ ]:

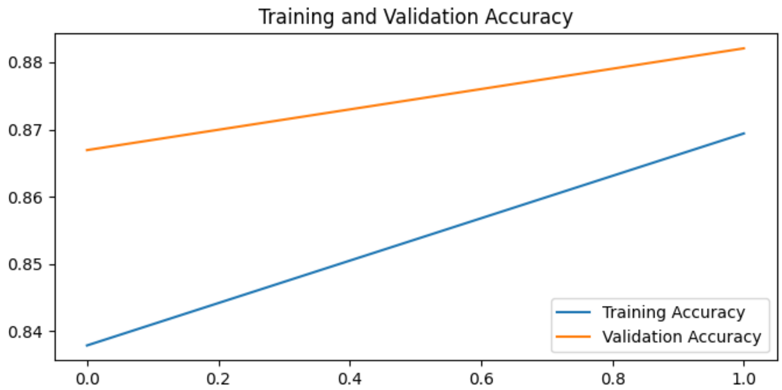
```
print('Test accuracy:', test_acc)
```

Test accuracy: 0.8810483813285828

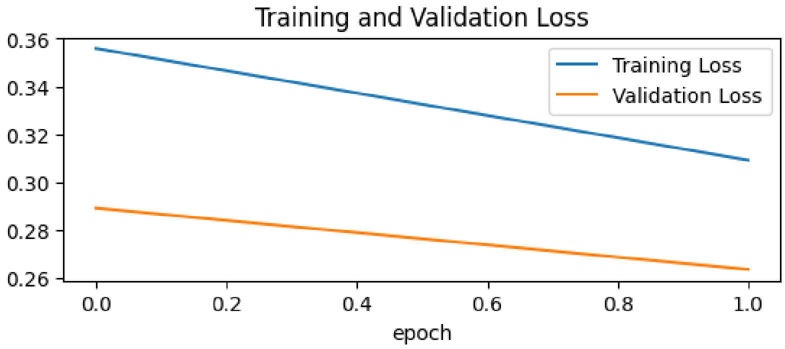
```
# Plot the accuracy and loss

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
```

Out[23]:

Text(0.5, 1.0, 'Training and Validation Accuracy')




```
plt.subplot(2, 1, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Conclusion : Image classification using transfer learning for dogs vs cats dataset is successfully implemented

EXPERIMENT 6

AIM : To implement Action Recognition Using Inflated 3D CNN

Theory : Action recognition is the process of recognizing human actions in videos. It is a challenging task due to the variability in human actions and the complexity of the video data. One approach to action recognition uses deep learning techniques such as convolutional neural networks (CNNs). One popular CNN architecture for action recognition is the I3D (Inflated 3D) CNN, which is an extension of the popular Inception CNN architecture. The I3D CNN has achieved state-of-the-art performance on several action recognition benchmarks.

The I3D CNN is a two-stream network that consists of a temporal stream and a spatial stream. The temporal stream processes the motion information in videos by stacking multiple frames together to form a "clip" of the video. The spatial stream processes the appearance information in videos by processing individual frames. The two streams are combined through fusion to make the final prediction. The temporal stream uses 3D convolutional layers to capture the motion information in videos. The 3D convolutional layers have three dimensions: height, width, and time. They slide over a 3D volume of data, capturing both spatial and temporal information. The spatial stream uses 2D convolutional layers to capture the appearance information in videos. The 2D convolutional layers have two dimensions: height and width. They slide over a 2D image, capturing only spatial information. The two streams are combined through fusion to make the final prediction. There are several ways to fuse the two streams, including early fusion, late fusion, and intermediate fusion. Early fusion combines the two streams at the input level, while late fusion combines the two streams at the output level. Intermediate fusion combines the two streams at an intermediate level, such as after the last layer of the two streams.

In TensorFlow, the I3D CNN can be implemented using the pre-trained model provided by Google, which was trained on the Kinetics dataset. The pre-trained model can be fine-tuned on a new dataset by replacing the last classification layer with a new layer that matches the number of classes in the new dataset. The pre-trained model can also be used as a feature extractor by removing the last classification layer and using the output of the second last layer as the feature representation of the video data. Overall, the I3D CNN is a powerful architecture for action recognition, and its two-stream network with temporal and spatial streams allows it to capture both motion and appearance information in videos.

CODE AND OUTPUT:

In [ ]:

```
!pip install -q imageio
!pip install -q opencv-python
!pip install -q git+https://github.com/tensorflow/docs
```

Preparing metadata (setup.py) ... done
Building wheel for tensorflow-docs (setup.py) ... done

In [ ]:

```
#@title Import
the necessary
modules#
TensorFlow and
TF-Hub modules.
from absl import logging

import tensorflow as tf
import tensorflow_hub as hub
from

tensorflow_docs.

vis import embed

logging.set_verb

osity(logging.ER

ROR)
```

In [ ]:

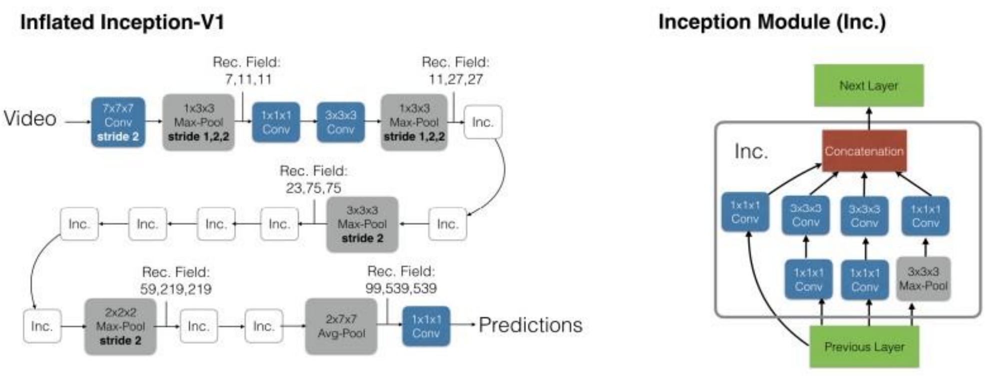
```
#@title Helper functions for the UCF101 dataset

# Utilities to fetch videos from UCF101 dataset
UCF_ROOT = "https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/"
_VIDEO_LIST = None
_CACHE_DIR = tempfile.mkdtemp()
# As of July 2020, crcv.ucf.edu doesn't use a certificate accepted by the # default Colab environment
anymore.
unverified_context = ssl._create_unverified_context()

def list_ucf_videos():
"""Lists videos available in UCF101 dataset."""
global _VIDEO_LIST
if not _VIDEO_LIST:
index = request.urlopen(UCF_ROOT, context=unverified_context).read().decode("utf-8") videos =
re.findall("(v_[\w_]+\.avi)", index)
_VIDEO_LIST = sorted(set(videos))
return list(_VIDEO_LIST)

def fetch_ucf_video(video):
"""Fetchs a video and cache into local filesystem.""" cache_path = os.path.join(_CACHE_DIR, video)
if not os.path.exists(cache_path):
urlpath = request.urljoin(UCF_ROOT, video)
print("Fetching %s => %s" % (urlpath, cache_path))
data = request.urlopen(urlpath, context=unverified_context).read() open(cache_path, "wb").write(data)
return cache_path

# Utilities to open video files using CV2
def crop_center_square(frame): y, x = frame.shape[0:2]
min_dim = min(y, x)
start_x = (x // 2) - (min_dim // 2) start_y = (y // 2) - (min_dim // 2)
return frame[start_y:start_y+min_dim,start_x:start_x+min_dim]

def load_video(path, max_frames=0, resize=(224, 224)): cap = cv2.VideoCapture(path)
frames = []
try:
```

while True:
ret, frame = cap.read()
if not ret:
break
frame = crop_center_square(frame) frame = cv2.resize(frame, resize) frame = frame[:, :, [2, 1, 0]]
frames.append(frame)

if len(frames) == max_frames:
break finally:
cap.release()
return np.array(frames) / 255.0

def to_gif(images):
converted_images = np.clip(images * 255, 0, 255).astype(np.uint8) imageio.mimsave('./animation.gif', converted_images, fps=25)

```
  return
```



In [ ]:

```
#@title Get the kinetics-400 labels
# Get the kinetics-400 action labels from the GitHub repository.
KINETICS_URL = "https://raw.githubusercontent.com/deepmind/kinetics-i3d/master/data/labe
with request.urlopen(KINETICS_URL) as obj:
  labels = [line.decode("utf-8").strip() for line in obj.readlines()]
print("Found %d labels." % len(labels))
```

Found 400 labels.

Using the UCF101 dataset
In [ ]:

```
# Get the list of videos in the dataset.
ucf_videos = list_ucf_videos()

categories = {}
for video in ucf_videos:
  category = video[2:-12]
  if category not in categories:
    categories[category] = []
  categories[category].append(video)
print("Found %d videos in %d categories." % (len(ucf_videos), len(categories)))

for category, sequences in categories.items():
  summary = ", ".join(sequences[:2])
  print("%-20s %4d videos (%s, ...)" % (category, len(sequences), summary))
```

In [ ]:

```
# Get a sample cricket video.
video_path = fetch_ucf_video("v_CricketShot_g04_c02.avi")
sample_video = load_video(video_path)
```

Fetching https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/v_CricketShot_g04_c02.avi (https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/v_CricketShot_g0 4_c02.avi) => /tmp/tmpd_nzisfu/v_CricketShot_g04_c02.avi

```
sample_video.shape
```

Out[7]:

(116, 224, 224, 3)

In [ ]:

```
i3d = hub.load("https://tfhub.dev/deepmind/i3d-kinetics-400/1").signatures['default']
```

Run the id3 model and print the top-5 action predictions.

In [ ]:

```
def predict(sample_video):
  # Add a batch axis to the sample video.
  model_input = tf.constant(sample_video, dtype=tf.float32)[tf.newaxis, ...]

  logits = i3d(model_input)['default'][0]
  probabilities = tf.nn.softmax(logits)

  print("Top 5 actions:")
  for i in np.argsort(probabilities)[::-1][:5]:
    print(f"  {labels[i]:22}: {probabilities[i] * 100:5.2f}%")
```

In [ ]:

```
predict(sample_video)
```

Top 5 actions:

playing cricket	:	97.77 %
skateboarding	:	0.71 %
robot dancing	:	0.56 %
roller skating	:	0.56 %
golf putting	:	0.13 %

In [ ]:

```
!curl -O https://upload.wikimedia.org/wikipedia/commons/8/86/End_of_a_jam.ogv
```

% Total % Received % Xferd Average Speed Time Time Time Cu rrent
Dload Upload Total Spent Left Sp
eed
100 55.0M 100 55.0M 0 0 25.4M 0 0:00:02 0:00:02 --:--:-- 2
5.4M

In [ ]:

```
video_path = "End_of_a_jam.ogv"
```

```
sample_video = load_video(video_path)[:100]
sample_video.shape
```

Out[13]:

(100, 224, 224, 3)

In [ ]:

```

```

In [ ]:

```
predict(sample_video)
```

Top 5 actions:

roller skating	:	96.85 %
playing volleyball	:	1.63 %
skateboarding	:	0.21 %
playing ice hockey	:	0.20 %
playing basketball	:	0.16 %

In [ ]:

```
import matplotlib.pyplot as plt

def predict(sample_video):
  # Add a batch axis to the sample video.
  model_input = tf.constant(sample_video, dtype=tf.float32)[tf.newaxis, ...]

  logits = i3d(model_input)['default'][0]
  probabilities = tf.nn.softmax(logits)

  print("Top 5 actions:")
  for i in np.argsort(probabilities)[::-1][:5]:
    print(f"  {labels[i]:22}: {probabilities[i] * 100:5.2f}%")

  return probabilities.numpy()
```

In [ ]:

```
probabilities = predict(sample_video)
```

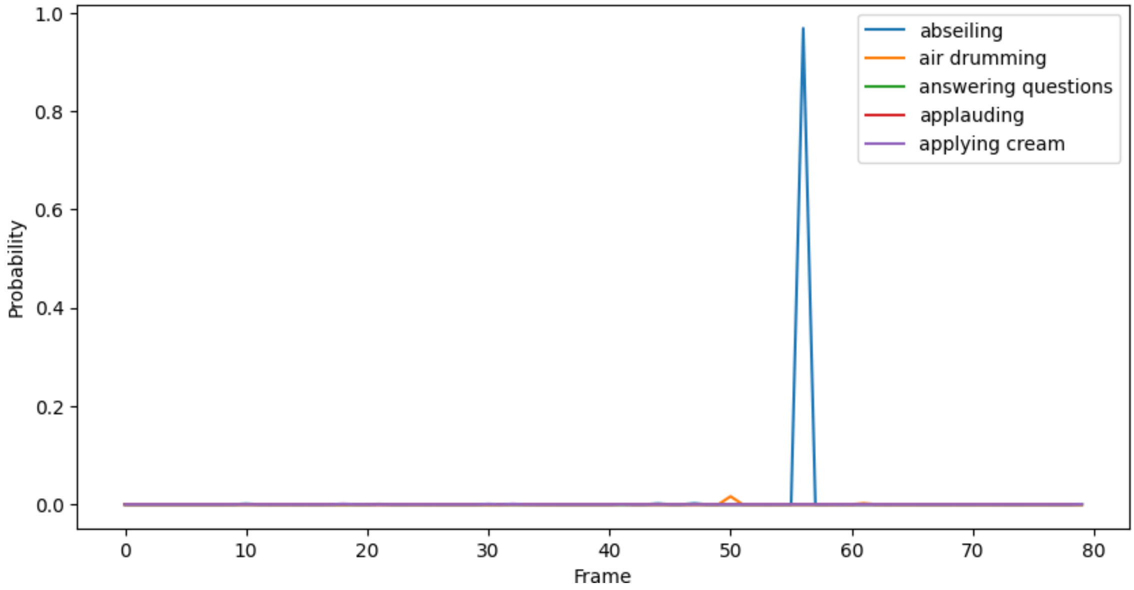
Top 5 actions:

roller skating	:	96.85 %
playing volleyball	:	1.63%
skateboarding	:	0.21%
playing ice hockey	:	0.20%
playing basketball	:	0.16%

```
plt.figure(figsize=(10, 5))
plt.plot(probabilities.reshape(-1, 5))
plt.legend(labels[:5])
plt.xlabel('Frame')
plt.ylabel('Probability')
```

Out[17]:

Text(0, 0.5, 'Probability')



In [ ]:

```
# Load a set of videos.
videos = ["v_CricketShot_g04_c02.avi", "v_HorseRiding_g09_c03.avi", "v_PlayingPiano_g06_
video_paths = [fetch_ucf_video(video) for video in videos]
sample_videos = [load_video(video_path) for video_path in video_paths]
```

Fetching https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/v_HorseRiding_g09 _c03.avi (https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/v_HorseRiding_g0 9_c03.avi) => /tmp/tmpd_nzisfu/v_HorseRiding_g09_c03.avi
Fetching https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/v_PlayingPiano_g0 6_c02.avi (https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/v_PlayingPiano_ g06_c02.avi) => /tmp/tmpd_nzisfu/v_PlayingPiano_g06_c02.avi
Fetching https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/v_SoccerJuggling_ g25_c01.avi (https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/v_SoccerJuggl ing_g25_c01.avi) => /tmp/tmpd_nzisfu/v_SoccerJuggling_g25_c01.avi
Fetching https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/v_Typing_g01_c02. avi (https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/v_Typing_g01_c02.avi) => /tmp/tmpd_nzisfu/v_Typing_g01_c02.avi

```
# Make predictions on the sample videos.
top5_labels_list = []
top1_probs_list = []
```

In [ ]:

```
for sample_video in sample_videos:
  result = predict(sample_video)
  if len(result) == 4:
    top5_labels, _, _, top1_prob = result
    top5_labels_list.append(top5_labels)
    top1_probs_list.append(top1_prob)
```

Top 5 actions:
playing cricket : 97.77 %
skateboarding : 0.71 %
robot dancing : 0.56 %
roller skating : 0.56 %
golf putting : 0.13 %
Top 5 actions:
riding or walking with horse: 98.30% riding mule : 1.59%
riding camel : 0.09%
jogging : 0.00%
walking the dog : 0.00%
Top 5 actions:
bandaging : 36.60%
smoking : 2.57%
stretching arm : 2.54%
air drumming : 2.23%
rock scissors paper : 2.13%
Top 5 actions:
juggling soccer ball : 98.70%
dribbling basketball : 1.17%
kicking soccer ball : 0.07%
shooting goal (soccer): 0.05%
playing basketball : 0.01%
Top 5 actions:
using computer : 100.00%
drumming fingers : 0.00%
texting : 0.00%
using remote controller (not gaming): 0.00 %
recording music : 0.00%

```
to_gif(sample_video)
```

Out[21]:

Conclusion : Action Recognition Using Inflated 3D CNN is successfully implemented

EXPERIMENT 7

AIM : To implement object detection using CNN

Theory: Object detection is a computer vision task that involves detecting and localizing objects within an image orvideo. Convolutional Neural Networks (CNNs) have been shown to be highly effective in solving object detection tasks. In this context, CNNs are used to learn features that can distinguish different objects in animage and localize their position within the image.

The general pipeline for object detection using CNNs involves the following steps:

Image pre-processing: In order to prepare the images for input into the CNN model, pre-processing is oftenrequired. This may include resizing the images to a specific size, normalizing pixel values, and applying other transformations such as cropping or rotation to increase the robustness of the model.

Feature extraction: CNNs are used to learn features that represent objects in the image. These features are learned by applying convolutional filters to the input image and pooling the results to produce a feature map.The feature map captures spatial information about the image, such as the location of edges, corners, and other salient features that can be used to identify objects.

Object proposal generation: Once features have been extracted from the image, object proposals are generated to identify potential objects in the image. This involves selecting regions of the image that are likely to contain objects, based on the features extracted from the previous step. Object proposal generation can be performed using techniques such as selective search or region proposal networks (RPNs).

Classification and localization: Finally, the object proposals are classified and localized. This involves assigning a label to each object proposal (e.g., "car", "person", "tree", etc.) and determining the preciselocation of the object within the image. This is typically done using a combination of CNNs and other machine learning techniques, such as regression.

In [ ]:

```
import tensorflow as tf
from tensorflow import keras
```

In [ ]:

```
# Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
```

Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz)
170498071/170498071 [==============================] - 2s 0us/step

```
# Normalize the pixel values to be between 0 and 1
x_train = x_train / 255.0
x_test = x_test / 255.0
```

In [ ]:

```
# Convert the labels to one-hot encoded vectors
y_train = keras.utils.to_categorical(y_train, num_classes=10)
y_test = keras.utils.to_categorical(y_test, num_classes=10)
```

In [ ]:

```
# Define the CNN model
model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.Flatten(),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

In [ ]:

```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Train the model
history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))
```

Epoch 1/10
1563/1563 [==============================] - 23s 6ms/step - loss: 1.5609 - accuracy: 0.4314 - val_loss: 1.3032 - val_accuracy: 0.5318Epoch 2/10
1563/1563 [==============================] - 9s 6ms/step - loss: 1.1964 - accuracy: 0.5756 - val_loss: 1.1481 - val_accuracy: 0.5925Epoch 3/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.0340 - accuracy: 0.6378 - val_loss: 1.0058 - val_accuracy: 0.6506Epoch 4/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.9318 - accuracy: 0.6730 - val_loss: 1.0019 - val_accuracy: 0.6531Epoch 5/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.8586 - accuracy: 0.6993 - val_loss: 0.9353 - val_accuracy: 0.6696Epoch 6/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.8057 - accuracy: 0.7186 - val_loss: 0.9184 - val_accuracy: 0.6888Epoch 7/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7544 -

accuracy: 0.7337 - val_loss: 0.9378 - val_accuracy: 0.6779Epoch
8/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.7108 -
accuracy: 0.7526 - val_loss: 0.8890 - val_accuracy: 0.6959Epoch
9/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.6734 -
accuracy: 0.7634 - val_loss: 0.8742 - val_accuracy: 0.7081Epoch
10/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.6412 -
accuracy: 0.7737 - val_loss: 0.9162 - val_accuracy: 0.6903

In [ ]:

```
# Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)
print(f'Test accuracy: {accuracy}')
```

313/313 [==============================] - 1s 3ms/step - loss: 0.9162 - ac
curacy: 0.6903
Test accuracy: 0.6902999877929688

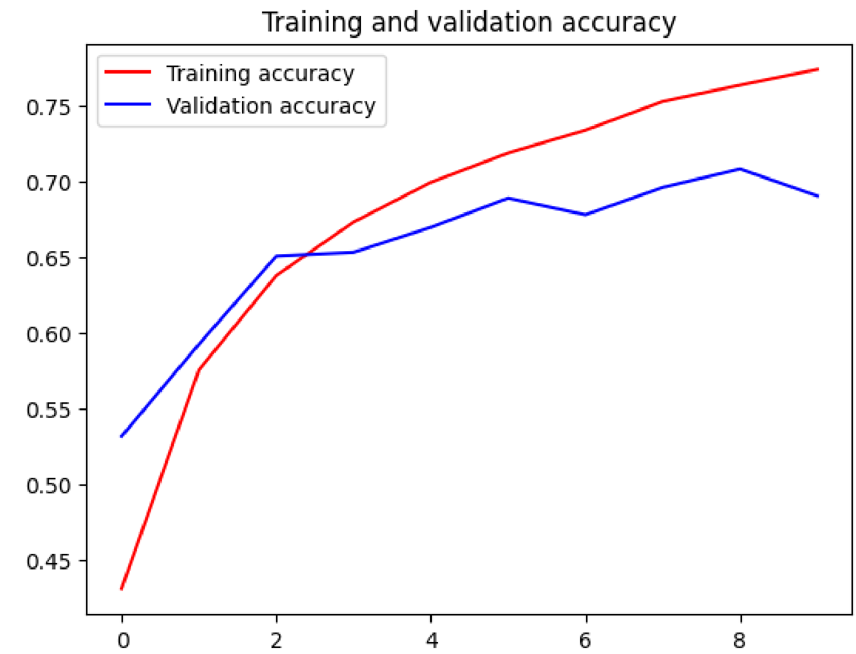
In [ ]:

```
# Plot the accuracy and loss curves
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

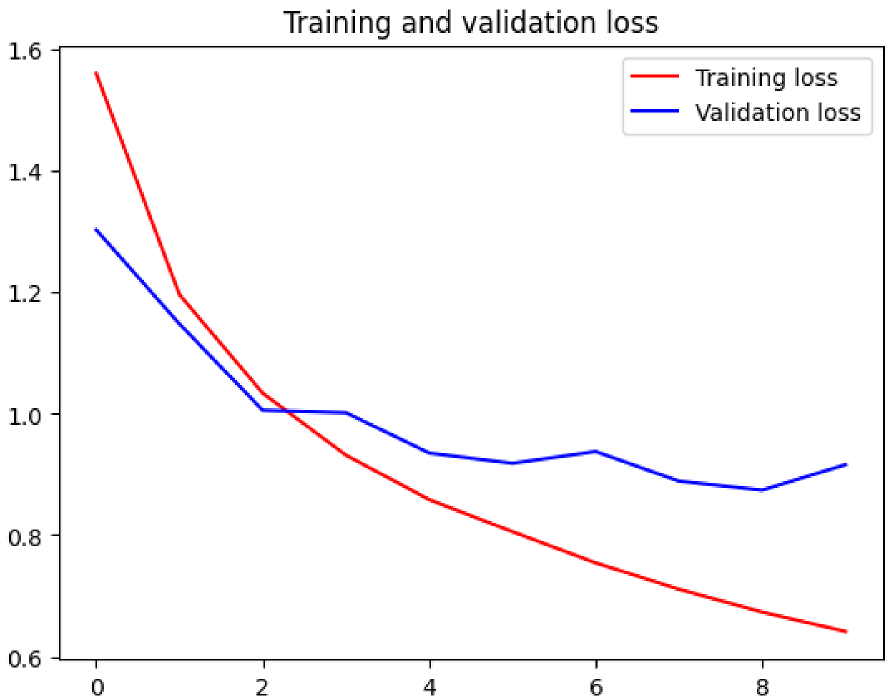
```
epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
```

Out[12]:

<Figure size 640x480 with 0 Axes>



<Figure size 640x480 with 0 Axes>



CONCLUSION: To implement object detection using CNN

EXPERIMENT 8

AIM: To generate images with Big GAN

THEORY: BigGAN is a generative adversarial network (GAN) that can produce high-quality and diverse images of various classes. A GAN consists of two neural networks: a generator and a discriminator. The generator triesto create realistic images from random noise, while the discriminator tries to distinguish between real and fake images. The generator and the discriminator are trained in an adversarial manner, where the generator aims to fool the discriminator and the discriminator aims to correctly classify the images.

CODE AND OUTPUT

In [1]:

```
module_path = 'https://tfhub.dev/deepmind/biggan-deep-256/1'
```

In [2]:

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

import os
import io
import IPython.display
import numpy as np
import PIL.Image
from scipy.stats import truncnorm
import tensorflow_hub as hub
```

WARNING:tensorflow:From /usr/local/lib/python3.9/dist-packages/tensorflow/python/compat/v2_compat.py:107: disable_resource_variables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed in a futureversion.
Instructions for updating:
non-resource variables are not supported in the long term

```
tf.reset_default_graph()
print('Loading BigGAN module from:', module_path)
module = hub.Module(module_path)
inputs = {k: tf.placeholder(v.dtype, v.get_shape().as_list(), k)
          for k, v in module.get_input_info_dict().items()}
output = module(inputs)

print()
print('Inputs:\n', '\n'.join(
    ' {}: {}'.format(*kv) for kv in inputs.items()))
print()
print('Output:', output)
```

Loading BigGAN module from: https://tfhub.dev/deepmind/biggan-deep-256/1 (https://tfhub.dev/deepmind/biggan-deep-256/1)

Inputs:
 truncation: Tensor("truncation:0", shape=(), dtype=float32)y:
Tensor("y:0", shape=(?, 1000), dtype=float32)
z: Tensor("z:0", shape=(?, 128), dtype=float32)

Output: Tensor("module_apply_default/G_trunc_output:0", shape=(?, 256, 25
6, 3), dtype=float32)

```
input_z = inputs['z']
input_y = inputs['y']
input_trunc = inputs['truncation']

dim_z = input_z.shape.as_list()[1]
vocab_size = input_y.shape.as_list()[1]

def truncated_z_sample(batch_size, truncation=1., seed=None):
state = None if seed is None else np.random.RandomState(seed)
values = truncnorm.rvs(-2, 2, size=(batch_size, dim_z), random_state=state)
return truncation * values

def one_hot(index, vocab_size=vocab_size):index
= np.asarray(index)
if len(index.shape) == 0:
  index = np.asarray([index])assert
len(index.shape) == 1 num =
index.shape[0]
output = np.zeros((num, vocab_size), dtype=np.float32)
output[np.arange(num), index] = 1
return output

def one_hot_if_needed(label, vocab_size=vocab_size):label =
np.asarray(label)
if len(label.shape) <= 1:
label = one_hot(label, vocab_size)
assert len(label.shape) == 2
return label

sample(sess,    noise,    label,    truncation=1.,    batch_size=8,
```

```
vocab_size=vocab_size):
noise = np.asarray(noise)label =
np.asarray(label)      num       =
noise.shape[0]
if len(label.shape) == 0:
label = np.asarray([label] * num)
if label.shape[0] != num:
raise ValueError('Got # noise samples ({}) != # label samples ({})'
                              .format(noise.shape[0],   label.shape[0]))  label   =
one_hot_if_needed(label, vocab_size)
ims = []
for batch_start in range(0, num, batch_size):
s = slice(batch_start, min(num, batch_start + batch_size))
feed_dict    =    {input_z:    noise[s],    input_y:    label[s],    input_trunc:    truncation}
ims.append(sess.run(output, feed_dict=feed_dict))
ims = np.concatenate(ims, axis=0)
assert ims.shape[0] == num
ims = np.clip(((ims + 1) / 2.0) * 256, 0, 255)ims =
np.uint8(ims)
return ims

def interpolate(A, B, num_interps):
if A.shape != B.shape:
   raise ValueError('A and B must have the same shape to interpolate.')alphas =
np.linspace(0, 1, num_interps)
return np.array([(1-a)*A + a*B for a in alphas])

def imgrid(imarray, cols=5, pad=1):
if imarray.dtype != np.uint8:
   raise ValueError('imgrid input imarray must be uint8')pad =
int(pad)
assert pad >= 0 cols
= int(cols)assert cols
>= 1
N, H, W, C = imarray.shape
rows = N // cols + int(N % cols != 0)
batch_pad = rows * cols - N
assert batch_pad >= 0
post_pad = [batch_pad, pad, pad, 0] pad_arg =
[[0, p] for p in post_pad]
imarray = np.pad(imarray, pad_arg, 'constant', constant_values=255)H += pad
W += pad
grid = (imarray
.reshape(rows, cols, H, W, C)
.transpose(0, 2, 1, 3, 4)
.reshape(rows*H, cols*W, C))
if pad:
grid = grid[:-pad, :-pad]
return grid

def imshow(a, format='png', jpeg_fallback=True):a =
np.asarray(a, dtype=np.uint8)
data = io.BytesIO()
PIL.Image.fromarray(a).save(data, format)im_data
= data.getvalue()
```

```
try:
disp = IPython.display.display(IPython.display.Image(im_data))
except IOError:
if jpeg_fallback and format != 'jpeg':
  print(('Warning: image was too large to display in format "{}"; "trying jpeg
instead.').format(format))
return imshow(a, format='jpeg')
else:
raise
   return disp
```

In [5]:

```
initializer = tf.global_variables_initializer()
sess = tf.Session()
sess.run(initializer)
```

```
num_samples = 5
truncation = 0.4
noise_seed = 0
category = "937) broccoli"

z = truncated_z_sample(num_samples, truncation, noise_seed)
y = int(category.split(')')[0])

ims = sample(sess, z, y, truncation=truncation)
imshow(imgrid(ims, cols=min(num_samples, 5)))
```

```
num_samples = 2
num_interps = 5
truncation = 0.2
noise_seed_A = 0
category_A = "39) common iguana, iguana, Iguana iguana"
noise_seed_B = 0
category_B = "130) flamingo"

def interpolate_and_shape(A, B, num_interps):
    interps = interpolate(A, B, num_interps)
  return (interps.transpose(1, 0, *range(2, len(interps.shape)))
                .reshape(num_samples * num_interps, *interps.shape[2:]))

z_A, z_B = [truncated_z_sample(num_samples, truncation, noise_seed)
            for noise_seed in [noise_seed_A, noise_seed_B]]
y_A, y_B = [one_hot([int(category.split(')')[0])] * num_samples)
            for category in [category_A, category_B]]

z_interp = interpolate_and_shape(z_A, z_B, num_interps)
y_interp = interpolate_and_shape(y_A, y_B, num_interps)

ims = sample(sess, z_interp, y_interp, truncation=truncation)
imshow(imgrid(ims, cols=num_interps))
```

Conclusion: We have successfully generated images with Big GAN.

EXPERIMENT 9

AIM : To implement transformer network for translating language

Theory: The Transformer network is a powerful deep learning model that has been widely used for machinetranslation tasks. Here's an overview of the theory behind implementing a Transformer network for translating languages:

Encoder and Decoder Architecture: The Transformer network consists of two main components: an encoderand a decoder. The encoder processes the input sentence and generates a representation of it, while the decoder takes this representation and generates the output sentence. Both the encoder and decoder consistof multiple layers of self-attention and feedforward neural networks.

Self-Attention Mechanism: The self-attention mechanism is the heart of the Transformer network. It allows the model to focus on different parts of the input sentence when generating the output. Self-attention is calculated by multiplying the input sentence by three matrices (queries, keys, and values), which are learnedduring training. The resulting scores are normalized and used to weight the values, which are then combinedto produce the output of the self-attention layer.

Multi-Head Attention: The self-attention mechanism is extended to a multi-head attention mechanism to allow the model to attend to multiple parts of the input sentence simultaneously. This is achieved by splittingthe input sentence into multiple heads, each of which is processed by a separate self-attention layer. The resulting outputs are concatenated and passed through a linear layer to generate the final output.

Positional Encoding: Since the Transformer network does not have any recurrence or convolutional layers, itlacks any explicit notion of the order of the words in the input sentence. To overcome this, positional encoding is used to inject information about the position of each word in the sentence into the input embeddings.

Loss Function: The training of the Transformer network involves minimizing a loss function that measures the difference between the predicted and actual output sentences. The most commonly used loss functionfor machine translation is the cross-entropy loss, which measures the difference between the predicted probability distribution over the output vocabulary and the true distribution.

Training Process: The Transformer network is trained using stochastic gradient descent (SGD) or one of its variants. During training, the model is fed input-output pairs, and the parameters are adjusted to minimize the loss function. The model is typically trained for several epochs, with the learning rate annealed over timeto improve convergence.

CODE AND OUTPUT:

```
import tensorflow_datasets as tfds
import tensorflow as tf

import time
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:

```
examples, metadata = tfds.load('ted_hrlr_translate/pt_to_en', with_info=True,
                               as_supervised=True)
train_examples, val_examples = examples['train'], examples['validation']
```

In [ ]:

```
tokenizer_en = tfds.deprecated.text.SubwordTextEncoder.build_from_corpus(
    (en.numpy() for pt, en in train_examples), target_vocab_size=2**13)

tokenizer_pt = tfds.deprecated.text.SubwordTextEncoder.build_from_corpus(
    (pt.numpy() for pt, en in train_examples), target_vocab_size=2**13)
```

In [ ]:

```
sample_string = 'Transformer is awesome.'

tokenized_string = tokenizer_en.encode(sample_string)
print ('Tokenized string is {}'.format(tokenized_string))

original_string = tokenizer_en.decode(tokenized_string)
print ('The original string: {}'.format(original_string))

assert original_string == sample_string
```

Tokenized string is [7915, 1248, 7946, 7194, 13, 2799, 7877]The original string:
Transformer is awesome.

In [ ]:

```
for ts in tokenized_string:
  print ('{} ----> {}'.format(ts, tokenizer_en.decode([ts])))
```

```
7915 ----> T
1248 ----> ran
7946 ----> s
7194 ----> former
13 ----> is
2799 ----> awesome
7877 ----> .
```

In [ ]:

```
BUFFER_SIZE = 20000
BATCH_SIZE = 64
```

```
def encode(lang1, lang2):
  lang1 = [tokenizer_pt.vocab_size] + tokenizer_pt.encode(
      lang1.numpy()) + [tokenizer_pt.vocab_size+1]

  lang2 = [tokenizer_en.vocab_size] + tokenizer_en.encode(
      lang2.numpy()) + [tokenizer_en.vocab_size+1]

  return lang1, lang2
```

In [ ]:

```
def tf_encode(pt, en):
  result_pt, result_en = tf.py_function(encode, [pt, en], [tf.int64, tf.int64])
  result_pt.set_shape([None])
  result_en.set_shape([None])

  return result_pt, result_en
```

In [ ]:

```
MAX_LENGTH = 40
```

In [ ]:

```
def filter_max_length(x, y, max_length=MAX_LENGTH):
  return tf.logical_and(tf.size(x) <= max_length,
                        tf.size(y) <= max_length)
```

In [ ]:

```
train_dataset = train_examples.map(tf_encode)
train_dataset = train_dataset.filter(filter_max_length)
# cache the dataset to memory to get a speedup while reading from it.
train_dataset = train_dataset.cache()
train_dataset = train_dataset.shuffle(BUFFER_SIZE).padded_batch(BATCH_SIZE)
train_dataset = train_dataset.prefetch(tf.data.experimental.AUTOTUNE)


val_dataset = val_examples.map(tf_encode)
val_dataset = val_dataset.filter(filter_max_length).padded_batch(BATCH_SIZE)
```

In [ ]:

```
pt_batch, en_batch = next(iter(val_dataset))
pt_batch, en_batch
```

In [ ]:

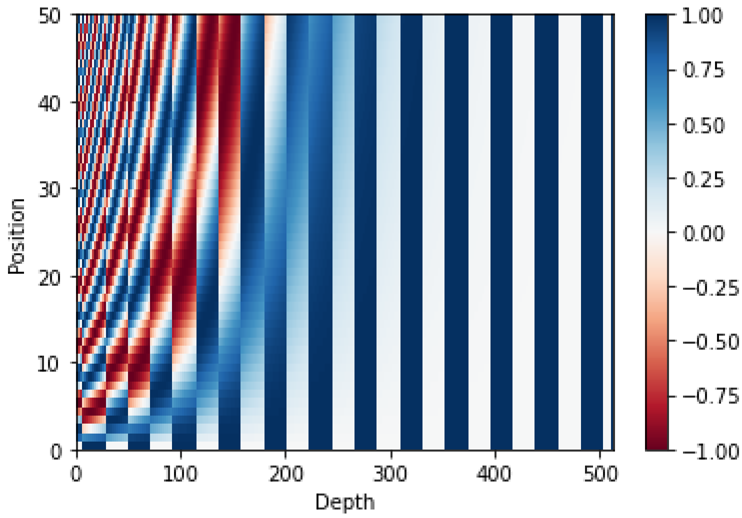
```
def get_angles(pos, i, d_model):
  angle_rates = 1 / np.power(10000, (2 * (i//2)) / np.float32(d_model))
  return pos * angle_rates
```

```
def positional_encoding(position, d_model):
  angle_rads = get_angles(np.arange(position)[:, np.newaxis],
                          np.arange(d_model)[np.newaxis, :],
                          d_model)

  # apply sin to even indices in the array; 2i
  angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])

  # apply cos to odd indices in the array; 2i+1
  angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])

  pos_encoding = angle_rads[np.newaxis, ...]

  return tf.cast(pos_encoding, dtype=tf.float32)
```

In [ ]:

```
pos_encoding = positional_encoding(50, 512)
print (pos_encoding.shape)

plt.pcolormesh(pos_encoding[0], cmap='RdBu')
plt.xlabel('Depth')
plt.xlim((0, 512))
plt.ylabel('Position')
plt.colorbar()
plt.show()
```

(1, 50, 512)



In [ ]:

```
def create_padding_mask(seq):
  seq = tf.cast(tf.math.equal(seq, 0), tf.float32)

  # add extra dimensions to add the padding
  # to the attention logits.
  return seq[:, tf.newaxis, tf.newaxis, :]  # (batch_size, 1, 1, seq_len)
```

```
x = tf.constant([[7, 6, 0, 0, 1], [1, 2, 3, 0, 0], [0, 0, 0, 4, 5]])
create_padding_mask(x)
```

In [ ]:

```
def create_look_ahead_mask(size):
  mask = 1 - tf.linalg.band_part(tf.ones((size, size)), -1, 0)
  return mask  # (seq_len, seq_len)
```

In [ ]:

```
x = tf.random.uniform((1, 3))
temp = create_look_ahead_mask(x.shape[1])
temp
```

Out[21]:

```
<tf.Tensor: shape=(3, 3), dtype=float32, numpy=
       array([[0.,   1.,   1.],
              [0.,   0.,   1.],
```

[0.,   0.,   0.]], dtype=float32)>

In [ ]:

```
def scaled_dot_product_attention(q, k, v, mask):

  matmul_qk = tf.matmul(q, k, transpose_b=True)  # (..., seq_len_q, seq_len_k)

  # scale matmul_qk
  dk = tf.cast(tf.shape(k)[-1], tf.float32)
  scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)

  # add the mask to the scaled tensor.
  if mask is not None:
    scaled_attention_logits += (mask * -1e9)

  # softmax is normalized on the last axis (seq_len_k) so that the scores
  # add up to 1.
  attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1)  # (..., seq_len_q

  output = tf.matmul(attention_weights, v)  # (..., seq_len_q, depth_v)

  return output, attention_weights
```

In [ ]:

```
def print_out(q, k, v):
  temp_out, temp_attn = scaled_dot_product_attention(
      q, k, v, None)
  print ('Attention weights are:')
  print (temp_attn)
  print ('Output is:')
  print (temp_out)
```

```
np.set_printoptions(suppress=True)

temp_k = tf.constant([[10,0,0],
                      [0,10,0],
                      [0,0,10],
                      [0,0,10]], dtype=tf.float32)  # (4, 3)

temp_v = tf.constant([[   1,0],
                      [  10,0],
                      [ 100,5],
                      [1000,6]], dtype=tf.float32)  # (4, 2)

# This `query` aligns with the second `key`,
# so the second `value` is returned.
temp_q = tf.constant([[0, 10, 0]], dtype=tf.float32)  # (1, 3)
print_out(temp_q, temp_k, temp_v)
```

Attention weights are:
tf.Tensor([[0. 1. 0. 0.]], shape=(1, 4), dtype=float32)Output is:
tf.Tensor([[10.   0.]], shape=(1, 2), dtype=float32)

In [ ]:

```
# This query aligns with a repeated key (third and fourth),
# so all associated values get averaged.
temp_q = tf.constant([[0, 0, 10]], dtype=tf.float32)  # (1, 3)
print_out(temp_q, temp_k, temp_v)
```

Attention weights are:
tf.Tensor([[0. 0. 0.5 0.5]], shape=(1, 4), dtype=float32)Output is:
tf.Tensor([[550.   5.5]], shape=(1, 2), dtype=float32)

In [ ]:

```
# This query aligns equally with the first and second key,
# so their values get averaged.
temp_q = tf.constant([[10, 10, 0]], dtype=tf.float32)  # (1, 3)
print_out(temp_q, temp_k, temp_v)
```

Attention weights are:
tf.Tensor([[0.5 0.5 0. 0.]], shape=(1, 4), dtype=float32)Output is:
tf.Tensor([[5.5 0.]], shape=(1, 2), dtype=float32)

In [ ]:

```
temp_q = tf.constant([[0, 0, 10], [0, 10, 0], [10, 10, 0]], dtype=tf.float32)  # (3, 3)
print_out(temp_q, temp_k, temp_v)
```

```
class MultiHeadAttention(tf.keras.layers.Layer):
  def__init__(self, d_model, num_heads):
    super(MultiHeadAttention, self)._init_()self.num_heads =
    num_heads
    self.d_model = d_model

    assert d_model % self.num_heads == 0

    self.depth = d_model // self.num_heads

    self.wq  =  tf.keras.layers.Dense(d_model) self.wk  =
    tf.keras.layers.Dense(d_model)          self.wv           =
    tf.keras.layers.Dense(d_model)

    self.dense = tf.keras.layers.Dense(d_model)

  def split_heads(self, x, batch_size):

    x = tf.reshape(x, (batch_size, -1, self.num_heads, self.depth))
    return tf.transpose(x, perm=[0, 2, 1, 3])

  def call(self, v, k, q, mask):batch_size =
    tf.shape(q)[0]

    q = self.wq(q)   # (batch_size, seq_len, d_model)k = self.wk(k)   #
    (batch_size,  seq_len,  d_model) v  =  self.wv(v)     #  (batch_size,
    seq_len, d_model)

    q  =  self.split_heads(q,  batch_size)     #  (batch_size,  num_heads,  seq_len_q,  depth) k  =  self.split_heads(k,
    batch_size)   # (batch_size, num_heads, seq_len_k, depth)v = self.split_heads(v, batch_size)    # (batch_size,
    num_heads, seq_len_v, depth)

    # scaled_attention.shape == (batch_size, num_heads, seq_len_q, depth)
```

```
        # attention_weights.shape == (batch_size, num_heads, seq_len_q, seq_len_k)
        scaled_attention, attention_weights = scaled_dot_product_attention(q, k, v, mask)

        scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1, 3])  # (batch_size,

        concat_attention = tf.reshape(scaled_attention,
                                      (batch_size, -1, self.d_model))  # (batch_size, seq_le

        output = self.dense(concat_attention)  # (batch_size, seq_len_q, d_model)

        return output, attention_weights
```

In [ ]:

```
temp_mha = MultiHeadAttention(d_model=512, num_heads=8)
y = tf.random.uniform((1, 60, 512))  # (batch_size, encoder_sequence, d_model)
out, attn = temp_mha(y, k=y, q=y, mask=None)
out.shape, attn.shape
```

Out[29]:

(TensorShape([1, 60, 512]), TensorShape([1, 8, 60, 60]))

```
def point_wise_feed_forward_network(d_model, dff):
  return tf.keras.Sequential([
      tf.keras.layers.Dense(dff, activation='relu'),  # (batch_size, seq_len, dff)
      tf.keras.layers.Dense(d_model)  # (batch_size, seq_len, d_model)
  ])
```

In [ ]:

```
sample_ffn = point_wise_feed_forward_network(512, 2048)
sample_ffn(tf.random.uniform((64, 50, 512))).shape
```

Out[31]:

TensorShape([64, 50, 512])

In [ ]:

```
class EncoderLayer(tf.keras.layers.Layer):
    def _init_(self, d_model, num_heads, dff, rate=0.1):super(EncoderLayer, self)._init_()

        self.mha = MultiHeadAttention(d_model, num_heads)
        self.ffn = point_wise_feed_forward_network(d_model, dff)

        self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)self.layernorm2 =
        tf.keras.layers.LayerNormalization(epsilon=1e-6)

        self.dropout1 = tf.keras.layers.Dropout(rate)self.dropout2 =
        tf.keras.layers.Dropout(rate)

    def call(self, x, training, mask):

        attn_output, _ = self.mha(x, x, x, mask)  # (batch_size, input_seq_len, d_model)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(x + attn_output)  # (batch_size, input_seq_len, d_model)

        ffn_output = self.ffn(out1)  # (batch_size, input_seq_len, d_model)
        ffn_output = self.dropout2(ffn_output, training=training)
        out2 = self.layernorm2(out1 + ffn_output)  # (batch_size, input_seq_len, d_model)
```

```
        return out2
```

In [ ]:

```
sample_encoder_layer = EncoderLayer(512, 8, 2048)

sample_encoder_layer_output = sample_encoder_layer(
    tf.random.uniform((64, 43, 512)), False, None)

sample_encoder_layer_output.shape  # (batch_size, input_seq_len, d_model)
```

Out[33]:

TensorShape([64, 43, 512])

```
class DecoderLayer(tf.keras.layers.Layer):
    def __init (self, d_model, num_heads, dff, rate=0.1):super(DecoderLayer, self)._init_()

        self.mha1 = MultiHeadAttention(d_model, num_heads)self.mha2 =
        MultiHeadAttention(d_model, num_heads)

        self.ffn = point_wise_feed_forward_network(d_model, dff)

        self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6) self.layernorm2 =
        tf.keras.layers.LayerNormalization(epsilon=1e-6) self.layernorm3 =
        tf.keras.layers.LayerNormalization(epsilon=1e-6)

        self.dropout1 = tf.keras.layers.Dropout(rate) self.dropout2 =
        tf.keras.layers.Dropout(rate) self.dropout3 =
        tf.keras.layers.Dropout(rate)


    def call(self, x, enc_output, training,
              look_ahead_mask, padding_mask):
        # enc_output.shape == (batch_size, input_seq_len, d_model)

        attn1, attn_weights_block1 = self.mha1(x, x, x, look_ahead_mask)  # (batch_size, tar
        attn1 = self.dropout1(attn1, training=training)out1 =
        self.layernorm1(attn1 + x)

        attn2, attn_weights_block2 = self.mha2(
            enc_output, enc_output, out1, padding_mask)  # (batch_size, target_seq_len, d_mo
        attn2 = self.dropout2(attn2, training=training)
        out2 = self.layernorm2(attn2 + out1)  # (batch_size, target_seq_len, d_model)

        ffn_output = self.ffn(out2)  # (batch_size, target_seq_len, d_model)
        ffn_output = self.dropout3(ffn_output, training=training)
        out3 = self.layernorm3(ffn_output + out2)  # (batch_size, target_seq_len, d_model)

        return out3, attn_weights_block1, attn_weights_block2
```

In [ ]:

```
sample_decoder_layer = DecoderLayer(512, 8, 2048)

sample_decoder_layer_output, _, _ = sample_decoder_layer(
    tf.random.uniform((64, 50, 512)), sample_encoder_layer_output,
    False, None, None)

sample_decoder_layer_output.shape  # (batch_size, target_seq_len, d_model)
```

Out[35]:

TensorShape([64, 50, 512])

```
class Encoder(tf.keras.layers.Layer):
  def_init_(self, num_layers, d_model, num_heads, dff, input_vocab_size,
               maximum_position_encoding, rate=0.1):
    super(Encoder, self).__init__()

    self.d_model = d_model
    self.num_layers = num_layers

    self.embedding = tf.keras.layers.Embedding(input_vocab_size, d_model)
    self.pos_encoding = positional_encoding(maximum_position_encoding,
                                            self.d_model)


    self.enc_layers = [EncoderLayer(d_model, num_heads, dff, rate)
                       for _ in range(num_layers)]

    self.dropout = tf.keras.layers.Dropout(rate)

  def call(self, x, training, mask):

    seq_len = tf.shape(x)[1]

    # adding embedding and position encoding.
    x = self.embedding(x)  # (batch_size, input_seq_len, d_model)
    x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
    x += self.pos_encoding[:, :seq_len, :]

    x = self.dropout(x, training=training)

    for i in range(self.num_layers):
      x = self.enc_layers[i](x, training, mask)

    return x  # (batch_size, input_seq_len, d_model)
```

In [ ]:

```
sample_encoder = Encoder(num_layers=2, d_model=512, num_heads=8,
                         dff=2048, input_vocab_size=8500,
                         maximum_position_encoding=10000)
temp_input = tf.random.uniform((64, 62), dtype=tf.int64, minval=0, maxval=200)

sample_encoder_output = sample_encoder(temp_input, training=False, mask=None)

print (sample_encoder_output.shape)  # (batch_size, input_seq_len, d_model)
```

(64, 62, 512)

```python
class Decoder(tf.keras.layers.Layer):
  def_init_(self, num_layers, d_model, num_heads, dff, target_vocab_size,
               maximum_position_encoding, rate=0.1):
    super(Decoder, self).__init__()

    self.d_model = d_model
    self.num_layers = num_layers

    self.embedding = tf.keras.layers.Embedding(target_vocab_size, d_model)
    self.pos_encoding = positional_encoding(maximum_position_encoding, d_model)

    self.dec_layers = [DecoderLayer(d_model, num_heads, dff, rate)
                       for _ in range(num_layers)]
    self.dropout = tf.keras.layers.Dropout(rate)

  def call(self, x, enc_output, training,
           look_ahead_mask, padding_mask):

    seq_len = tf.shape(x)[1]
    attention_weights = {}

    x = self.embedding(x)  # (batch_size, target_seq_len, d_model)
    x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
    x += self.pos_encoding[:, :seq_len, :]

    x = self.dropout(x, training=training)

    for i in range(self.num_layers):
      x, block1, block2 = self.dec_layers[i](x, enc_output, training,
                                             look_ahead_mask, padding_mask)

      attention_weights['decoder_layer{}_block1'.format(i+1)] = block1
      attention_weights['decoder_layer{}_block2'.format(i+1)] = block2

    # x.shape == (batch_size, target_seq_len, d_model)
    return x, attention_weights
```

In [ ]:

```python
sample_decoder = Decoder(num_layers=2, d_model=512, num_heads=8,
                         dff=2048, target_vocab_size=8000,
                         maximum_position_encoding=5000)
temp_input = tf.random.uniform((64, 26), dtype=tf.int64, minval=0, maxval=200)

output, attn = sample_decoder(temp_input,
                              enc_output=sample_encoder_output,
                              training=False,
                              look_ahead_mask=None,
                              padding_mask=None)

output.shape, attn['decoder_layer2_block2'].shape
```

Out[39]:

(TensorShape([64, 26, 512]), TensorShape([64, 8, 26, 62]))

```python
class Transformer(tf.keras.Model):
  def_init_(self, num_layers, d_model, num_heads, dff, input_vocab_size,
               target_vocab_size, pe_input, pe_target, rate=0.1):
    super(Transformer, self).__init__()

    self.encoder = Encoder(num_layers, d_model, num_heads, dff,
                           input_vocab_size, pe_input, rate)

    self.decoder = Decoder(num_layers, d_model, num_heads, dff,
                           target_vocab_size, pe_target, rate)

    self.final_layer = tf.keras.layers.Dense(target_vocab_size)

  def call(self, inp, tar, training, enc_padding_mask,
           look_ahead_mask, dec_padding_mask):

    enc_output = self.encoder(inp, training, enc_padding_mask)  # (batch_size, inp_seq_l

    # dec_output.shape == (batch_size, tar_seq_len, d_model)
    dec_output, attention_weights = self.decoder(
        tar, enc_output, training, look_ahead_mask, dec_padding_mask)

    final_output = self.final_layer(dec_output)  # (batch_size, tar_seq_len, target_voca

    return final_output, attention_weights
```

In [ ]:

```python
sample_transformer = Transformer(
    num_layers=2, d_model=512, num_heads=8, dff=2048,
    input_vocab_size=8500, target_vocab_size=8000,
    pe_input=10000, pe_target=6000)

temp_input = tf.random.uniform((64, 38), dtype=tf.int64, minval=0, maxval=200)
temp_target = tf.random.uniform((64, 36), dtype=tf.int64, minval=0, maxval=200)

fn_out, _ = sample_transformer(temp_input, temp_target, training=False,
                               enc_padding_mask=None,
                               look_ahead_mask=None,
                               dec_padding_mask=None)

fn_out.shape  # (batch_size, tar_seq_len, target_vocab_size)
```

Out[41]:

TensorShape([64, 36, 8000])

```
num_layers = 4
d_model = 128
dff = 512
num_heads = 8

input_vocab_size = tokenizer_pt.vocab_size + 2
target_vocab_size = tokenizer_en.vocab_size + 2
dropout_rate = 0.1
```

In [ ]:

```
class CustomSchedule(tf.keras.optimizers.schedules.LearningRateSchedule):
  def __init_(self, d_model, warmup_steps=4000):
    super(CustomSchedule, self)._init_()

    self.d_model = d_model
    self.d_model = tf.cast(self.d_model, tf.float32)

    self.warmup_steps = warmup_steps

  def __call__(self, step):
    arg1 = tf.math.rsqrt(step)
    arg2 = step * (self.warmup_steps ** -1.5)

    return tf.math.rsqrt(self.d_model) * tf.math.minimum(arg1, arg2)
```

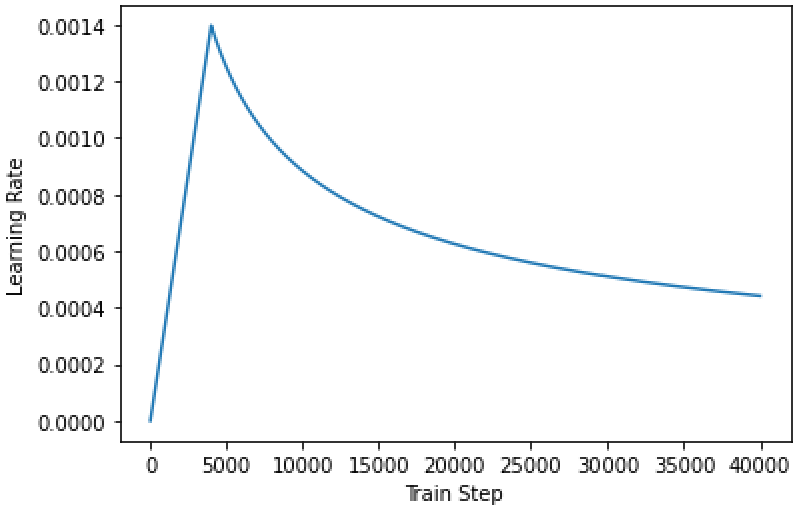
In [ ]:

```
learning_rate = CustomSchedule(d_model)

optimizer = tf.keras.optimizers.Adam(learning_rate, beta_1=0.9, beta_2=0.98,
                                     epsilon=1e-9)
```

```
temp_learning_rate_schedule = CustomSchedule(d_model)

plt.plot(temp_learning_rate_schedule(tf.range(40000, dtype=tf.float32)))
plt.ylabel("Learning Rate")
plt.xlabel("Train Step")
```

Out[45]:

Text(0.5, 0, 'Train Step')



In [ ]:

```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')
```

In [ ]:

```
def loss_function(real, pred):
  mask = tf.math.logical_not(tf.math.equal(real, 0))
  loss_ = loss_object(real, pred)

  mask = tf.cast(mask, dtype=loss_.dtype)
  loss_ *= mask

  return tf.reduce_sum(loss_)/tf.reduce_sum(mask)
```

In [ ]:

```
train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(
    name='train_accuracy')
```

```
transformer = Transformer(num_layers, d_model, num_heads, dff,
                          input_vocab_size, target_vocab_size,
                          pe_input=input_vocab_size,
                          pe_target=target_vocab_size,
                          rate=dropout_rate)
```

In [ ]:

```
def create_masks(inp, tar):
  # Encoder padding mask
  enc_padding_mask = create_padding_mask(inp)

  # Used in the 2nd attention block in the decoder.
  # This padding mask is used to mask the encoder outputs.
  dec_padding_mask = create_padding_mask(inp)

  # Used in the 1st attention block in the decoder.
  # It is used to pad and mask future tokens in the input received by
  # the decoder.
  look_ahead_mask = create_look_ahead_mask(tf.shape(tar)[1])
  dec_target_padding_mask = create_padding_mask(tar)
  combined_mask = tf.maximum(dec_target_padding_mask, look_ahead_mask)

  return enc_padding_mask, combined_mask, dec_padding_mask
```

In [ ]:

```
checkpoint_path = "./checkpoints/train"

ckpt = tf.train.Checkpoint(transformer=transformer,
                           optimizer=optimizer)

ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=5)

# if a checkpoint exists, restore the latest checkpoint.
if ckpt_manager.latest_checkpoint:
  ckpt.restore(ckpt_manager.latest_checkpoint)
  print ('Latest checkpoint restored!!')
```

In [ ]:

```
EPOCHS = 20
```

```
train_step_signature = [
    tf.TensorSpec(shape=(None, None), dtype=tf.int64),
    tf.TensorSpec(shape=(None, None), dtype=tf.int64),
]

@tf.function(input_signature=train_step_signature)
def train_step(inp, tar):
  tar_inp = tar[:, :-1]
  tar_real = tar[:, 1:]

  enc_padding_mask, combined_mask, dec_padding_mask = create_masks(inp, tar_inp)

  with tf.GradientTape() as tape:
    predictions, _ = transformer(inp, tar_inp,
                                 True,
                                 enc_padding_mask,
                                 combined_mask,
                                 dec_padding_mask)
    loss = loss_function(tar_real, predictions)

  gradients = tape.gradient(loss, transformer.trainable_variables)
  optimizer.apply_gradients(zip(gradients, transformer.trainable_variables))

  train_loss(loss)
  train_accuracy(tar_real, predictions)
```

In [ ]:

```
for epoch in range(EPOCHS):
  start = time.time()

  train_loss.reset_states()
  train_accuracy.reset_states()

  # inp -> portuguese, tar -> english
  for (batch, (inp, tar)) in enumerate(train_dataset):
    train_step(inp, tar)

    if batch % 50 == 0:
      print ('Epoch {} Batch {} Loss {:.4f} Accuracy {:.4f}'.format(
          epoch + 1, batch, train_loss.result(), train_accuracy.result()))

  if (epoch + 1) % 5 == 0:
    ckpt_save_path = ckpt_manager.save()
    print ('Saving checkpoint for epoch {} at {}'.format(epoch+1,
                                                         ckpt_save_path))

  print ('Epoch {} Loss {:.4f} Accuracy {:.4f}'.format(epoch + 1,
                                                train_loss.result(),
                                                train_accuracy.result()))

  print ('Time taken for 1 epoch: {} secs\n'.format(time.time() - start))
```

```python
def evaluate(inp_sentence):
  start_token = [tokenizer_pt.vocab_size]
  end_token = [tokenizer_pt.vocab_size + 1]

  # inp sentence is portuguese, hence adding the start and end token
  inp_sentence = start_token + tokenizer_pt.encode(inp_sentence) + end_token
  encoder_input = tf.expand_dims(inp_sentence, 0)

  # as the target is english, the first word to the transformer should be the
  # english start token.
  decoder_input = [tokenizer_en.vocab_size]
  output = tf.expand_dims(decoder_input, 0)

  for i in range(MAX_LENGTH):
    enc_padding_mask, combined_mask, dec_padding_mask = create_masks(
        encoder_input, output)

    # predictions.shape == (batch_size, seq_len, vocab_size)
    predictions, attention_weights = transformer(encoder_input,
                                                 output,
                                                 False,
                                                 enc_padding_mask,
                                                 combined_mask,
                                                 dec_padding_mask)

    # select the last word from the seq_len dimension
    predictions = predictions[: ,-1:, :]  # (batch_size, 1, vocab_size)

    predicted_id = tf.cast(tf.argmax(predictions, axis=-1), tf.int32)

    # return the result if the predicted_id is equal to the end token
    if predicted_id == tokenizer_en.vocab_size+1:
      return tf.squeeze(output, axis=0), attention_weights

    # concatentate the predicted_id to the output which is given to the decoder
    # as its input.
    output = tf.concat([output, predicted_id], axis=-1)

  return tf.squeeze(output, axis=0), attention_weights
```

```
def plot_attention_weights(attention, sentence, result, layer):
  fig = plt.figure(figsize=(16, 8))

  sentence = tokenizer_pt.encode(sentence)

  attention = tf.squeeze(attention[layer], axis=0)

  for head in range(attention.shape[0]):
    ax = fig.add_subplot(2, 4, head+1)

    # plot the attention weights
    ax.matshow(attention[head][:-1, :], cmap='viridis')

    fontdict = {'fontsize': 10}

    ax.set_xticks(range(len(sentence)+2))
    ax.set_yticks(range(len(result)))

    ax.set_ylim(len(result)-1.5, -0.5)

    ax.set_xticklabels(
        ['<start>']+[tokenizer_pt.decode([i]) for i in sentence]+['<end>'],
        fontdict=fontdict, rotation=90)

    ax.set_yticklabels([tokenizer_en.decode([i]) for i in result
                        if i < tokenizer_en.vocab_size],
                       fontdict=fontdict)

    ax.set_xlabel('Head {}'.format(head+1))

  plt.tight_layout()
  plt.show()
```

In [ ]:

```
def translate(sentence, plot=''):
  result, attention_weights = evaluate(sentence)

  predicted_sentence = tokenizer_en.decode([i for i in result
                                            if i < tokenizer_en.vocab_size])

  print('Input: {}'.format(sentence))
  print('Predicted translation: {}'.format(predicted_sentence))

  if plot:
    plot_attention_weights(attention_weights, sentence, result, plot)
```

In [ ]:

```
translate("este é um problema que temos que resolver.")
print ("Real translation: this is a problem we have to solve .")
```

Input: este é um problema que temos que resolver.
Predicted translation: this is a problem we have to solve it .Real translation: this is a problem we have to solve .

```
translate("os meus vizinhos ouviram sobre esta ideia.")
print ("Real translation: and my neighboring homes heard about this idea .")
```

Input: os meus vizinhos ouviram sobre esta ideia.
Predicted translation: my neighbors heard about this idea .
Real translation: and my neighboring homes heard about this idea .

In [ ]:

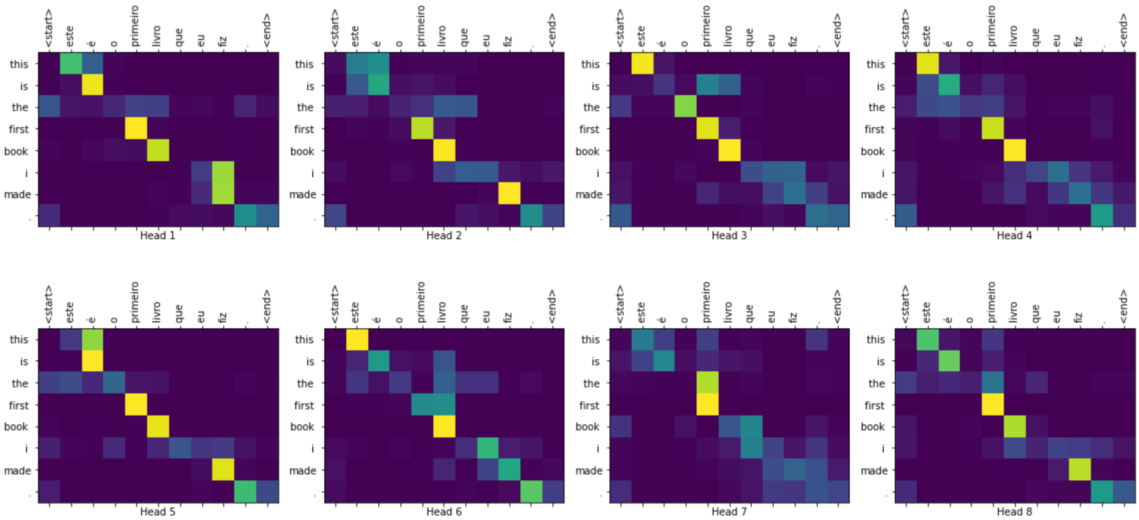
```
translate("vou então muito rapidamente partilhar convosco algumas histórias de algumas c
print ("Real translation: so i 'll just share with you some stories very quickly of some
```

Input: vou então muito rapidamente partilhar convosco algumas histórias dealgumas coisas mágicas que aconteceram.
Predicted translation: so i 'm going to very quickly to share with you some of the magic stories that have happened .
Real translation: so i 'll just share with you some stories very quickly of some magical things that have happened .

In [ ]:

```
translate("este é o primeiro livro que eu fiz.", plot='decoder_layer4_block2')
print ("Real translation: this is the first book i've ever done.")
```

Input: este é o primeiro livro que eu fiz.
Predicted translation: this is the first book i made .



Real translation: this is the first book i've ever done.

Conclusion : Transformer network for translating language is successfully implemented

EXPERIMENT 10

Aim : TO implement MLops

Theory : MLOps (Machine Learning Operations) is a set of practices for collaboration and communication between data scientists and operations professionals. Applying these practices increases the quality, simplifies the management process, and automates the deployment of Machine Learning and Deep Learning models in large-scale production environments. It’s easier to align models with business needs, as well as regulatoryrequirements.

MLOps cycle

MLOps is slowly evolving into an independent approach to ML lifecycle management. It applies to the entire lifecycle – data gathering, model creation (software development lifecycle, continuous integration/continuous delivery), orchestration, deployment, health, diagnostics, governance, and business metrics.

The key phases of MLOps are:

Data gathering Data analysis Data transformation/preparation Model training & development Model validation Model serving Model monitoring Model re-training. DevOps vs MLOps DevOps and MLOps havefundamental similarities because MLOps principles were derived from DevOps principles. But they’re quitedifferent in execution:

Unlike DevOps, MLOps is much more experimental in nature. Data Scientists and ML/DL engineers have totweak various features – hyperparameters, parameters, and models – while also keeping track of and managing the data and the code base for reproducible results. Besides all the efforts and tools, the ML/DL

industry still struggles with the reproducibility of experiments. This topic is out of the scope of this article, so for more information check the reproducibility subsection in references at the end. Hybrid team composition:the team needed to build and deploy models in production won’t be composed of software engineers only. Inan ML project, the team usually includes data scientists or ML researchers, who focus on exploratory data analysis, model development, and experimentation. They might not be experienced software engineers whocan build production-class services. Testing: testing an ML system involves model validation, model training,and so on – in addition to the conventional code tests, such as unit testing and integration testing.

Automated Deployment: you can’t just deploy an offline-trained ML model as a prediction service. You’ll need a multi-step pipeline to automatically retrain and deploy a model. This pipeline adds complexity because you need to automate the steps that data scientists do manually before deployment to train and validate new models. Production performance degradation of the system due to evolving data profiles or simply Training-Serving Skew: ML models in production can have reduced performance not only due to suboptimal coding but also due to constantly evolving data profiles. Models can decay in more ways than conventional software systems, and you need to plan for it. This can be caused by: A discrepancy between how you handle data in the training and serving pipelines. A change in the data between when you train andwhen you serve. Feedback loop – when you choose the wrong hypothesis (i.e. objective) to optimize, which makes you collect biased data for training your model. Then, without knowing, you collect newer data pointsusing this flawed hypothesis, it’s fed back in to retrain/fine-tune future versions of the model, making the model even more biased, and the snowball keeps growing. For more information read Fastbook’s section on

Limitations Inherent To Machine Learning. Monitoring: models in production need to be monitored. Similarly, the summary statistics of data that built the model need to be monitored so that you can refresh the model when needed. These statistics can and will change over time, you need notifications or a roll-back process

DATA PREPROCEESING

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:

```
path = "https://drive.google.com/uc?export=download&id=1rnNTruX-8rSq89DUavC0enRbzU3sMBCE
df_raw = pd.read_csv(path)
print(df_raw.shape)
```

(13320, 9)

In [ ]:

```
df_raw.head()
```

Out[4]:

	area_type	availability	location	size	society	total_sqft	bath	balcony	p
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	3
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	12
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	6
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3.0	1.0	9
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0	1.0	5

Out[5]:

	area_type	availability	location	size	society	total_sqft	bath	balcony
13315	Built-up Area	Ready To Move	Whitefield	5 Bedroom	ArsiaEx	3453	4.0	0.0
13316	Super built-up Area	Ready To Move	Richards Town	4 BHK	NaN	3600	5.0	NaN
13317	Built-up Area	Ready To Move	Raja Rajeshwari Nagar	2 BHK	Mahla T	1141	2.0	1.0
13318	Super built-up Area	18-Jun	Padmanabhanagar	4 BHK	SollyCl	4689	4.0	1.0
13319	Super built-up Area	Ready To Move	Doddathoguru	1 BHK	NaN	550	1.0	1.0

In [ ]:

```
df = df_raw.copy()
```

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13320 entries, 0 to 13319 Data
columns (total 9 columns):
#   Column        Non-Null Count   Dtype
---  ------        --------------   -----
0   area_type     13320 non-null   object
1   availability  13320 non-null   object
2   location      13319 non-null   object
3   size          13304 non-null   object
4   society       7818 non-null    object
5   total_sqft    13320 non-null   object
6   bath          13247 non-null   float64
7   balcony       12711 non-null   float64
8   price         13320 non-null   float64
dtypes: float64(3), object(6)memory
usage: 936.7+ KB
```

```
df.describe()
```

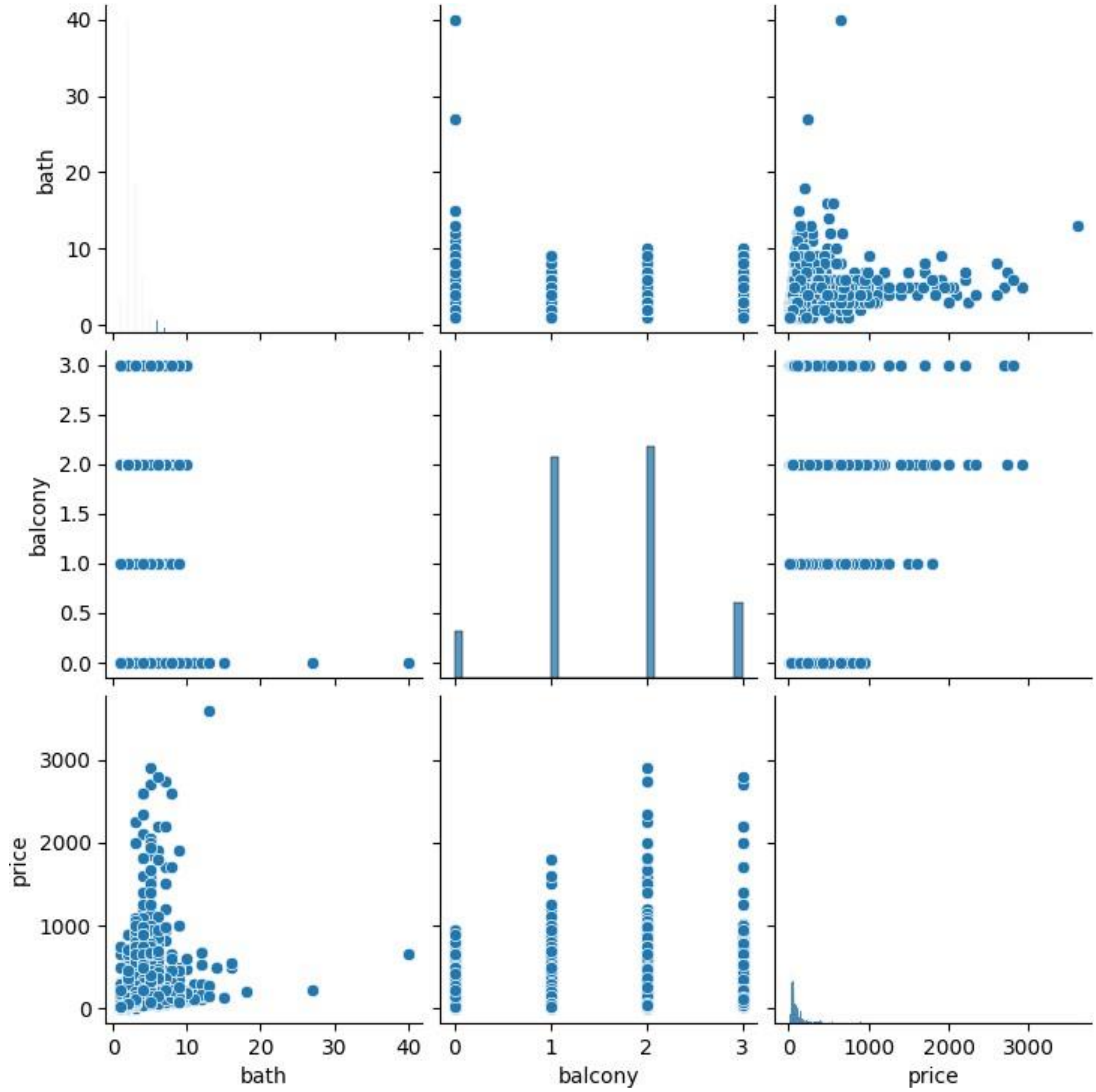
Out[8]:

	bath	balcony	price
count	13247.000000	12711.000000	13320.000000
mean	2.692610	1.584376	112.565627
std	1.341458	0.817263	148.971674
min	1.000000	0.000000	8.000000
25%	2.000000	1.000000	50.000000
50%	2.000000	2.000000	72.000000
75%	3.000000	2.000000	120.000000
max	40.000000	3.000000	3600.000000

```
sns.pairplot(df)
```

Out[9]:

<seaborn.axisgrid.PairGrid at 0x7f586388f8e0>



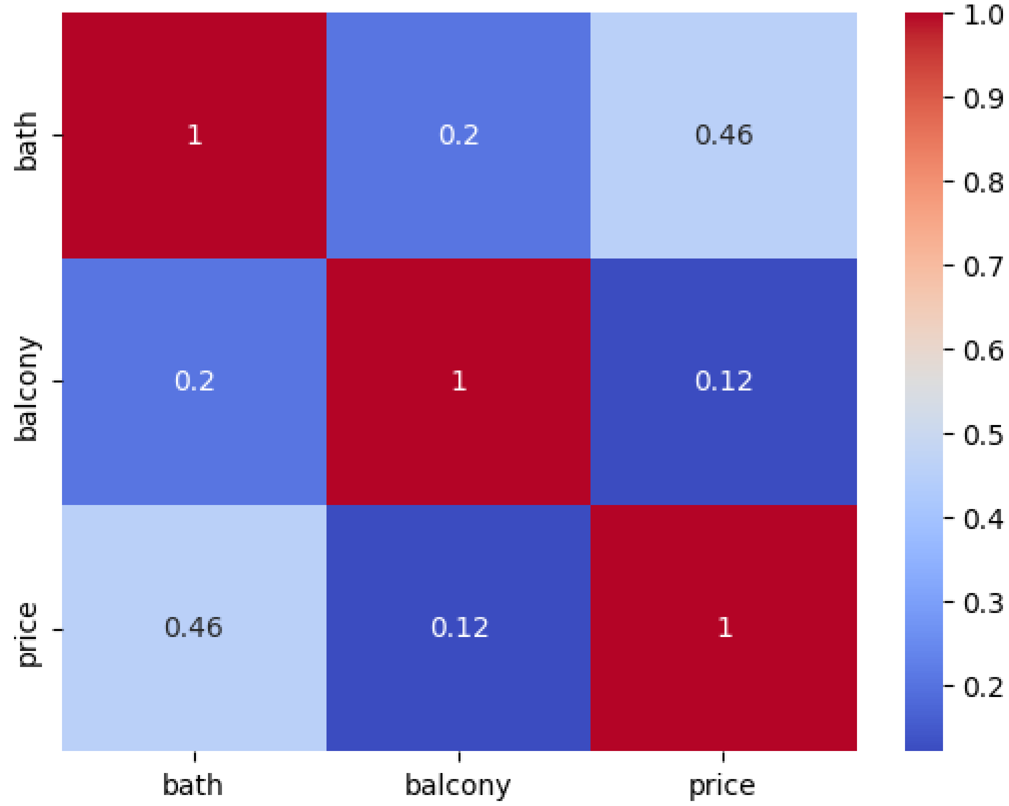
```python
# value count of each feature
def value_count(df):
  for var in df.columns:
    print(df[var].value_counts())
    print("--------------------------------------------------")

value_count(df)
```

```
Super built-up  Area    8790
Built-up Area           2418
Plot Area               2025
Carpet  Area              87
Name: area_type, dtype: int64
--------------------------------
Ready To Move    10581
18-Dec             307
18-May             295
18-Apr             271
18-Aug             200
                 ...
15-Aug               1
17-Jan               1
16-Nov               1
16-Jan               1
14-Jul               1
Name: availability, Length: 81, dtype: int64
--------------------------------------------------
```

```
# correlation heatmap
num_vars = ["bath", "balcony", "price"]
sns.heatmap(df[num_vars].corr(),cmap="coolwarm", annot=True)
```

Out[11]:

<Axes: >



In [ ]:

```
df.isnull().sum() # find the homuch missing data available

df.isnull().mean()*100 # % of measing value
```

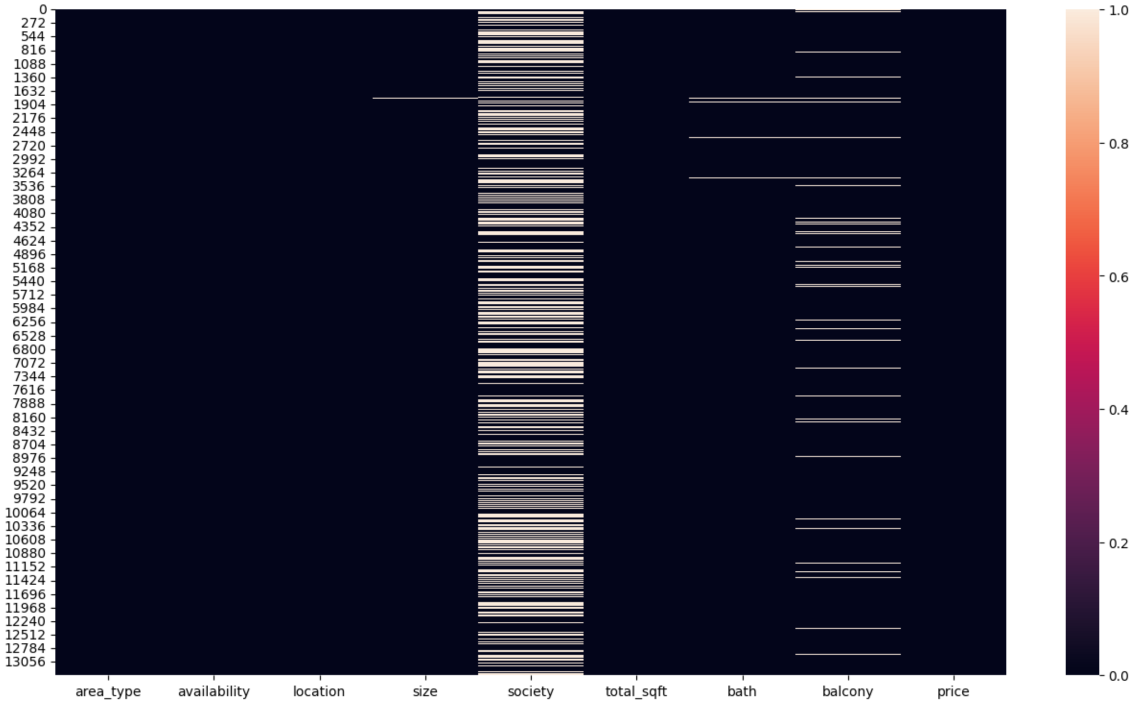
Out[12]:

```
area_type        0.000000
availability     0.000000
location         0.007508
size             0.120120
society         41.306306
total_sqft       0.000000
bath             0.548048
balcony          4.572072
price            0.000000
dtype: float64
```

```
# visualize missing value using heatmap to get idea where is the value missing

plt.figure(figsize=(16,9))
sns.heatmap(df.isnull())
```

Out[13]:

<Axes: >



In [ ]:

```
# Drop ----------> society feature
# because 41.3% missing value
df2 = df.drop('society', axis='columns')
df2.shape
```

Out[14]:

(13320, 8)

```
# because it contain 4.5% missing value
df2['balcony'] = df2['balcony'].fillna(df2['balcony'].mean())
df2.isnull().sum()
```

Out[16]:

```
area_type        0
availability     0
location         1
size            16
total_sqft       0
bath            73
balcony          0
price            0
dtype: int64
```

In [ ]:

```
# drop na value rows from df2
# because there is very less % value missing
df3 = df2.dropna()
df3.shape
```

Out[17]:

(13246, 8)

In [ ]:

```
df3.isnull().sum()
```

In [ ]:

```
df3.head()
```

Out[19]:

	area_type	availability	location	size	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	1056	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	2600	5.0	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	1440	2.0	3.0	62.00
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	1521	3.0	1.0	95.00
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	1200	2.0	1.0	51.00

```
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", None)
```

In [ ]:

```
df3['total_sqft'].value_counts()
```

In [ ]:

```
# best strategy is to convert it into number by spliting it

total_sqft_int = []
for str_val in df3['total_sqft']:
  try:
    total_sqft_int.append(float(str_val)) # if '123.4' like this value in str then conve
  except:
    try:
      temp = []
      temp = str_val.split('-')
      total_sqft_int.append((float(temp[0])+float(temp[-1]))/2) # '123 - 534' this str v
    except:
      total_sqft_int.append(np.nan) # if value not contain in above format then consider
```

In [ ]:

```
# reset the index of dataframe
df4 = df3.reset_index(drop=True) # drop=True - don't add index column in df
```

In [ ]:

```
# join df4 and total_srft_int list
df5 = df4.join(pd.DataFrame({'total_sqft_int':total_sqft_int}))
df5.head()
```

Out[24]:

	area_type	availability	location	size	total_sqft	bath	balcony	price	total
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	1056	2.0	1.0	39.07	
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	2600	5.0	3.0	120.00	
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	1440	2.0	3.0	62.00	
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	1521	3.0	1.0	95.00	
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	1200	2.0	1.0	51.00	

```
df5.tail()
```

Out[25]:

	area_type	availability	location	size	total_sqft	bath	balcony	price
13241	Built-up Area	Ready To Move	Whitefield Bedroom	5	3453	4.0	0.000000	231.0
13242	Super built-up Area	Ready To Move	Richards Town	4 BHK	3600	5.0	1.584376	400.0
13243	Built-up Area	Ready To Move	Raja Rajeshwari Nagar	2 BHK	1141	2.0	1.000000	60.0
13244	Super built-up Area	18-Jun	Padmanabhanagar	4 BHK	4689	4.0	1.000000	488.0
13245	Super built-up Area	Ready To Move	Doddathoguru	1 BHK	550	1.0	1.000000	17.0

In [ ]:

```
df5.isnull().sum()
```

Out[26]:

```
area_type          0
availability       0
location           0
size               0
total_sqft         0
bath               0
balcony            0
price              0
total_sqft_int    46
dtype: int64
```

In [ ]:

```
# drop na value
df6 = df5.dropna()
df6.shape
```

Out[27]:

(13200, 9)

```
df6.info()
```

<class 'pandas.core.frame.DataFrame'>Int64Index: 13200 entries, 0 to 13245Data columns (total 9 columns):

#	Column	Non-Null Count	Dtype
---	------	--------------	-----
0	area_type	13200 non-null	object
1	availability	13200 non-null	object
2	location	13200 non-null	object
3	size	13200 non-null	object
4	total_sqft	13200 non-null	object
5	bath	13200 non-null	float64
6	balcony	13200 non-null	float64
7	price	13200 non-null	float64
8	total_sqft_int	13200 non-null	float64

dtypes: float64(4), object(5)
memory usage: 1.0+ MB

```
## Working on <<<< Size >>>> feature"""

df6['size'].value_counts()
```

Out[29]:

2 BHK	5192
3 BHK	4277
4 Bedroom	816
4 BHK	574
3 Bedroom	541
1 BHK	527
2 Bedroom	325
5 Bedroom	293
6 Bedroom	190
1 Bedroom	100
7 Bedroom	83
8 Bedroom	83
5 BHK	56
9 Bedroom	45
6 BHK	30
7 BHK	17
1 RK	13
10 Bedroom	12
9 BHK	7
8 BHK	5
11 BHK	2
11 Bedroom	2
10 BHK	2
14 BHK	1
13 BHK	1
12 Bedroom	1
27 BHK	1
43 Bedroom	1

16	BHK	1
19	BHK	1
18	Bedroom	1

Name: size, dtype: int64

```
"""
in  size feature we assume that
2 BHK = 2 Bedroom == 2 RK
so takes only number and remove sufix text
"""
size_int = []
for str_val in df6['size']:
  temp=[]
  temp = str_val.split(" ")
  try:
    size_int.append(int(temp[0]))
  except:
    size_int.append(np.nan)
    print("Noice = ",str_val)
```

In [ ]:

```
df6 = df6.reset_index(drop=True)
```

In [ ]:

```
# join df6 and list size_int
df7 = df6.join(pd.DataFrame({'bhk':size_int}))
df7.shape
```

Out[32]:

(13200, 10)

In [ ]:

```
df7.tail()
```

Out[33]:

	area_type	availability	location	size	total_sqft	bath	balcony	price
13195	Built-up Area	Ready To Move	Whitefield	5 Bedroom	3453	4.0	0.000000	231.0
13196	Super built-up Area	Ready To Move	Richards Town	4 BHK	3600	5.0	1.584376	400.0
13197	Built-up Area	Ready To Move	Raja Rajeshwari Nagar	2 BHK	1141	2.0	1.000000	60.0
13198	Super							

13199	built-up Area	18-Jun	Padmanabhanagar	4 BHK	4689	4.0	1.000000	488.0
	Super built-up Area	Ready To Move	Doddathoguru	1 BHK	550	1.0	1.000000	17.0

```python
# for Q-Q plots
import scipy.stats as stats
```

In [ ]:

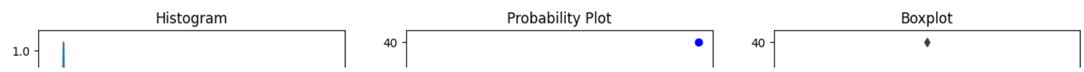
```python
def diagnostic_plots(df, variable):
    # function takes a dataframe (df) and
    # the variable of interest as arguments

    # define figure size
    plt.figure(figsize=(16, 4))

    # histogram
    plt.subplot(1, 3, 1)
    sns.distplot(df[variable], bins=30)
    plt.title('Histogram')

    # Q-Q plot
    plt.subplot(1, 3, 2)
    stats.probplot(df[variable], dist="norm", plot=plt)
    plt.ylabel('Variable quantiles')

    # boxplot
    plt.subplot(1, 3, 3)
    sns.boxplot(y=df[variable])
    plt.title('Boxplot')

    plt.show()
```

In [ ]:

```python
plt.show()
```

```
num_var = ["bath","balcony","total_sqft_int","bhk","price"]
for var in num_var:
  print("******* {} *******".format(var))
  diagnostic_plots(df7, var)
```

```
******* bath *******

<ipython-input-40-e65df554a832>:12: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.1
4.0.

Please adapt your code to use either `displot` (a figure-level function
with
similar flexibility) or `histplot` (an axes-level function for histogra
ms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (http
s://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df[variable], bins=30)
```



In [ ]:

```
# here we consider  1 BHK requierd min 350 sqft are
df7[df7['total_sqft_int']/df7['bhk'] < 350].head()
```

Out[43]:

	area_type	availability	location	size	total_sqft	bath	balcony	price	total_s
9	Plot Area	Ready To Move	Gandhi Bazar	6 Bedroom	1020	6.0	1.584376	370.0	
26	Super built-up Area	Ready To Move	Electronic City	2 BHK	660	1.0	1.000000	23.1	
29	Super built-up Area	Ready To Move	Electronic City	3 BHK	1025	2.0	1.000000	47.0	
45	Plot Area	Ready To Move	HSR Layout	8 Bedroom	600	9.0	1.584376	200.0	
57	Plot Area	Ready To Move	Murugeshpalya	6 Bedroom	1407	4.0	1.000000	150.0	

```
# no we found outliers

# if 1 BHK total_sqft are < 350 then we ae going to remove them
df8 = df7[~(df7['total_sqft_int']/df7['bhk'] < 350)]
df8.shape
```

Out[44]:

(12106, 10)

In [ ]:

```
# create new feature that is price per squre foot
# it help to find the outliers

#price in lakh so conver into rupee and then / by total_sqft_int
df8['price_per_sqft'] = df8['price']*100000 / df8['total_sqft_int']
df8.head()
```

<ipython-input-45-ce9ebd953aa1>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df8['price_per_sqft'] = df8['price']*100000 / df8['total_sqft_int']Out[45]:

	area_type	availability	location	size	total_sqft	bath	balcony	price	total
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	1056	2.0	1.0	39.07	
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	2600	5.0	3.0	120.00	
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	1440	2.0	3.0	62.00	
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	1521	3.0	1.0	95.00	
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	1200	2.0	1.0	51.00	

```
df8.price_per_sqft.describe()
```

Out[46]:

```
count    12106.000000
mean      6184.466889
std       4019.983503
min        267.829813
25%       4200.030048
50%       5261.108523
75%       6800.000000
max     176470.588235
Name: price_per_sqft, dtype: float64
```

In [ ]:

```
#here we can see huge difference between min and max price_per_sqft
# min 6308.502826 max 176470.588235

# Removing outliers using help of 'price per sqrt'  taking std and mean per location
def remove_pps_outliers(df):
  df_out = pd.DataFrame()
  for key, subdf in df.groupby('location'):
    m=np.mean(subdf.price_per_sqft)
    st=np.std(subdf.price_per_sqft)
    reduced_df = subdf[(subdf.price_per_sqft>(m-st))&(subdf.price_per_sqft<=(m+st))]
    df_out = pd.concat([df_out, reduced_df], ignore_index = True)
  return df_out
```

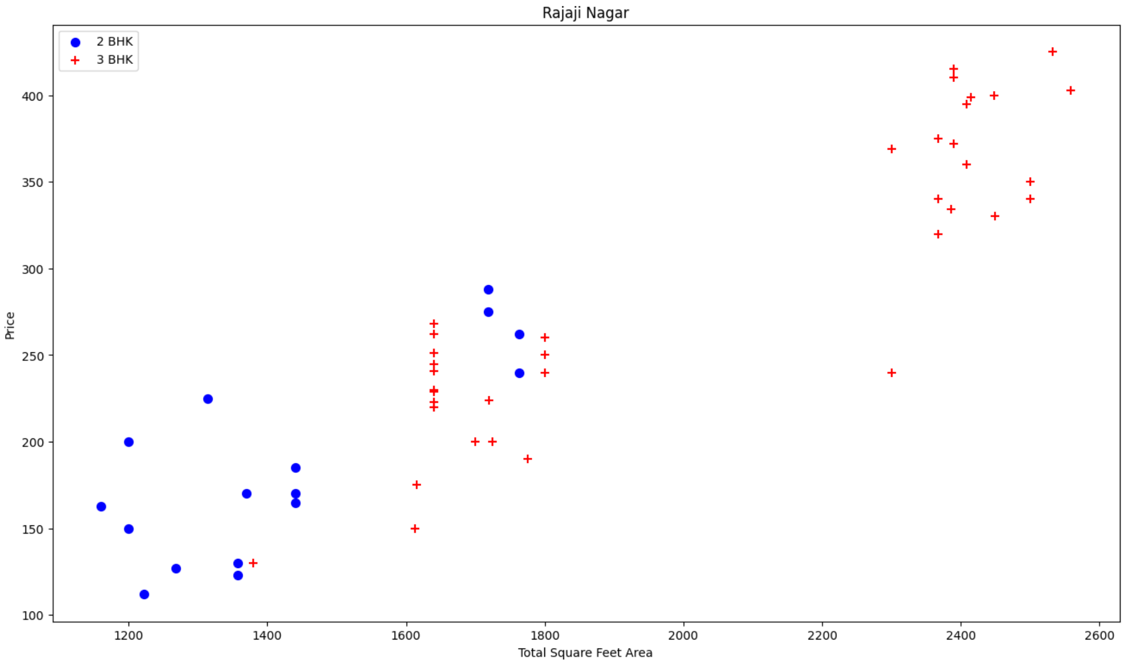
In [ ]:

```
df9 = remove_pps_outliers(df8)
df9.shape
```

Out[48]:

(8888, 11)

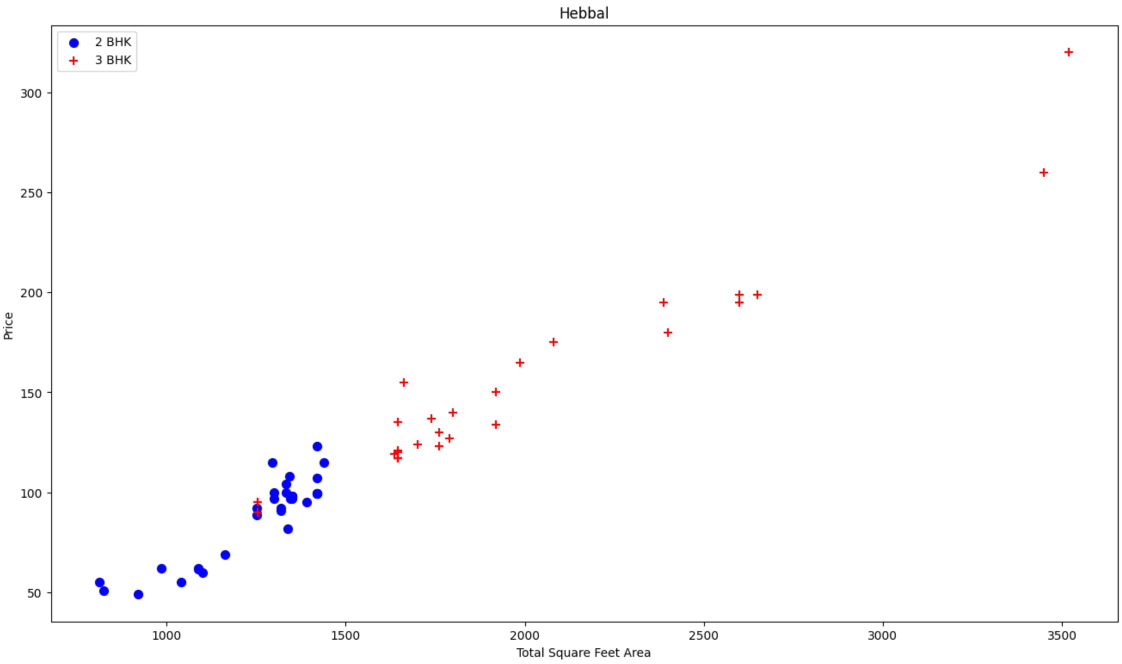
```python
def plot_scatter_chart(df,location):
  bhk2 = df[(df.location==location) & (df.bhk==2)]
  bhk3 = df[(df.location==location) & (df.bhk==3)]
  plt.figure(figsize=(16,9))
  plt.scatter(bhk2.total_sqft_int, bhk2.price, color='Blue', label='2 BHK', s=50)
  plt.scatter(bhk3.total_sqft_int, bhk3.price, color='Red', label='3 BHK', s=50, marker=
  plt.xlabel("Total Square Feet Area")
  plt.ylabel("Price")
  plt.title(location)
  plt.legend()

plot_scatter_chart(df9, "Rajaji Nagar")
```



In [ ]:

```python
# 3 bhk house is less than 2 bhk so it is outlier

# Removing BHK outliers
def remove_bhk_outliers(df):
  exclude_indices = np.array([])
  for location, location_df in df.groupby('location'):
    bhk_stats = {}
    for bhk, bhk_df in location_df.groupby('bhk'):
      bhk_stats[bhk]={
          'mean':np.mean(bhk_df.price_per_sqft),
          'std':np.std(bhk_df.price_per_sqft),
          'count':bhk_df.shape[0]}
    for bhk, bhk_df in location_df.groupby('bhk'):
      stats=bhk_stats.get(bhk-1)
      if stats and stats['count']>5:
        exclude_indices = np.append(exclude_indices, bhk_df[bhk_df.price_per_sqft<(stats
  return df.drop(exclude_indices, axis='index')
```

```python
df10 = remove_bhk_outliers(df9)
df10.shape

plot_scatter_chart(df10, "Hebbal")
# In below scatter plot most of the red data point remove fron blue points
```



In [ ]:

```python
"""### Remove outliers using the help of 'bath' feature"""

df10.bath.unique()

df10[df10.bath > df10.bhk+2]
```

Out[52]:

	area_type	availability	location	size	total_sqft	bath	balcony	price	tota
1861	Built-up Area	Ready To Move	Chikkabanavar	4 Bedroom	2460	7.0	2.000000	80.0	
5836	Built-up Area	Ready To Move	Nagasandra	4 Bedroom	7000	8.0	1.584376	450.0	
7098	Super built-up Area	Ready To Move	Sathya Sai Layout	6 BHK	11338	9.0	1.000000	1000.0	
7569	Super built-up Area	Ready To Move	Thanisandra	3 BHK	1806	6.0	2.000000	116.0	

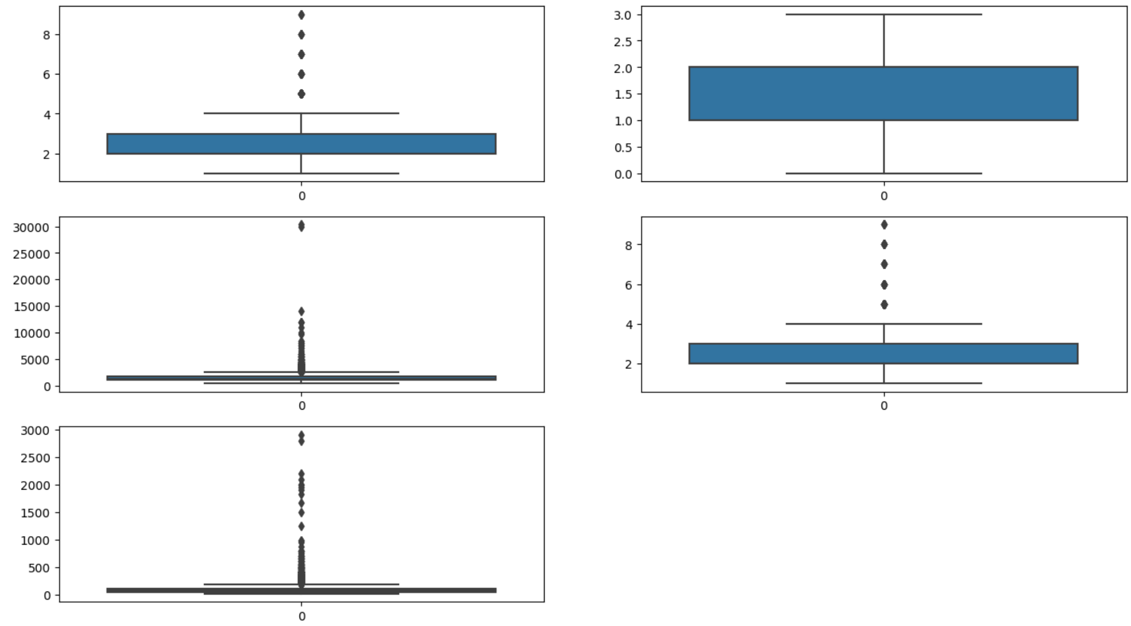
```python
# here we are considering data only total no. bathroom =  bhk + 1
df11 = df10[df10.bath < df10.bhk+2]
df11.shape
```

Out[53]:

(7120, 11)

In [ ]:

```python
plt.figure(figsize=(16,9))
for i,var in enumerate(num_var):
  plt.subplot(3,2,i+1)
  sns.boxplot(df11[var])
```



In [ ]:

```python
df11.head()
```

Out[55]:

	area_type	availability	location	size	total_sqft	bath	balcony	price	total
0	Super built-up Area	Ready To Move	Devarabeesana Halli	3 BHK	1672	3.0	2.0	150.0	
1	Built-up Area	Ready To Move	Devarabeesana Halli	3 BHK	1750	3.0	3.0	149.0	
2	Super built-up Area	Ready To Move	Devarabeesana Halli	3 BHK	1750	3.0	2.0	150.0	
4	Super built-upArea	Ready To Move	Devarachikkanahalli	2 BHK	1250	2.0	2.0	40.0	
5	Plot Area	Ready To Move	Devarachikkanahalli	2 Bedroom	1200	2.0	2.0	83.0	

```
df12 = df11.drop(['area_type', 'availability',"location","size","total_sqft"], axis =1)
df12.head()
```

Out[56]:

	bath	balcony	price	total_sqft_int	bhk	price_per_sqft
0	3.0	2.0	150.0	1672.0	3	8971.291866
1	3.0	3.0	149.0	1750.0	3	8514.285714
2	3.0	2.0	150.0	1750.0	3	8571.428571
4	2.0	2.0	40.0	1250.0	2	3200.000000
5	2.0	2.0	83.0	1200.0	2	6916.666667

In [ ]:

```
df12.to_csv("clean_data.csv", index=False) # test ml model on this data
# ML model train on this data and got best score >>>> XGBoost=0.914710
```

In [ ]:

```
"""# Categorical Variable Encoding"""

df13 = df11.drop(["size","total_sqft"], axis =1)
df13.head()
```

Out[58]:

	area_type	availability	location	bath	balcony	price	total_sqft_int	bhk	price_
0	Super built-up Area	Ready To Move	Devarabeesana Halli	3.0	2.0	150.0	1672.0	3	897
1	Built-up Area	Ready To Move	Devarabeesana Halli	3.0	3.0	149.0	1750.0	3	851
2	Super built-up Area	Ready To Move	Devarabeesana Halli	3.0	2.0	150.0	1750.0	3	857
4	Super built-up Area	Ready To Move	Devarachikkanahalli	2.0	2.0	40.0	1250.0	2	320
5	Plot Area	Ready To Move	Devarachikkanahalli	2.0	2.0	83.0	1200.0	2	691

```
df14 = pd.get_dummies(df13, drop_first=True, columns=['area_type','availability','locati
df14.shape

df14.head()
```

Out[60]:

	bath	balcony	price	total_sqft_int	bhk	price_per_sqft	area_type_Carpet Area	area_type_Plot Area
0	3.0	2.0	150.0	1672.0	3	8971.291866	0	0
1	3.0	3.0	149.0	1750.0	3	8514.285714	0	0
2	3.0	2.0	150.0	1750.0	3	8571.428571	0	0
4	2.0	2.0	40.0	1250.0	2	3200.000000	0	0
5	2.0	2.0	83.0	1200.0	2	6916.666667	0	1

In [ ]:

```
df14.to_csv('oh_encoded_data.csv', index=False)
```

In [ ]:

```
## Working on <<<<<< area_type >>>>> feature


df13['area_type'].value_counts()

df15 = df13.copy()
```

In [ ]:

```
# appy Ohe-Hot  encoding on 'area_type' feature
for cat_var in ["Super built-up Area","Built-up Area","Plot Area"]:
  df15["area_type"+cat_var] = np.where(df15['area_type']==cat_var, 1,0)
df15.shape

df15.head(2)
```

Out[63]:

	area_type	availability	location	bath	balcony	price	total_sqft_int	bhk	price_per
0	Super built-up Area	Ready To Move	Devarabeesana Halli	3.0	2.0	150.0	1672.0	3	8971.29
1	Built-up Area	Ready To Move	Devarabeesana Halli	3.0	3.0	149.0	1750.0	3	8514.28

```python
"""## Working with <<<<< availability >>>>> Feature"""

df15["availability"].value_counts()

# in availability feature, 10525 house 'Ready to Move" and remaining will be redy on per
# so we crate new feature ""availability_Ready To Move"" and add vale 1 if availability
df15["availability_Ready To Move"] = np.where(df15["availability"]=="Ready To Move",1,0)
df15.shape

df15.tail()
```

Out[64]:

	area_type	availability	location	bath	balcony	price	total_sqft_int	bhk	price_per_
8883	Super built-up Area	Ready To Move	frazertown	3.0	2.0	325.00	2900.0	3	11206.89
8884	Super built-up Area	18-Nov	manyata park	3.0	1.0	84.83	1780.0	3	4765.73
8885	Plot Area	Ready To Move	tc.palya	2.0	1.0	48.00	880.0	2	5454.54
8886	Plot Area	18-Apr	tc.palya	2.0	1.0	55.00	1000.0	2	5500.00
8887	Plot Area	18-Apr	tc.palya	2.0	1.0	78.00	1400.0	3	5571.42

```python
"""## Working on <<<< Location >>>> feature"""

location_value_count = df15['location'].value_counts()
location_value_count

location_gert_20 = location_value_count[location_value_count>=20].index
location_gert_20
```

Out[65]:

Index(['Whitefield', 'Sarjapur Road', 'Electronic City', 'Marathahalli','Raja Rajeshwari Nagar', 'Haralur Road',
'Hennur Road',
'Bannerghatta Road', 'Uttarahalli', 'Thanisandra',
'Electronic City Phase II', 'Hebbal', 'Yelahanka', '7th Phase JP Na
gar',
'Kanakpura Road', 'KR Puram', 'Sarjapur', 'Rajaji Nagar', 'Bellandu'Kasavanhalli', 'Begur
r',
Road', 'Banashankari', 'Kothanur', 'Hormav
u',
'Harlur', 'Akshaya Nagar', 'Electronics City Phase 1', 'Jakkur', 'Varthur', 'HSR Layout',
'Hennur', 'Ramamurthy Nagar', 'Chandapur
a',
'Koramangala', 'Kaggadasapura', 'Kundalahalli', 'Ramagondanahalli','Budigere', 'Hulimavu',
'Hoodi', 'Malleshwaram', 'Hegde Nagar',
'Yeshwanthpur', 'Gottigere', '8th Phase JP Nagar', 'JP Nagar', 'Channasandra',
'Bisuvanahalli', 'Vittasandra', 'Indira Nagar','Old Airport Road', 'Sahakara Nagar',
'Brookefield', 'Kengeri','Hosa Road', 'Vijayanagar', 'Balagere', 'Green Glen Layout',
'Bommasandra', 'Rachenahalli', 'Panathur', 'Old Madras Road',
'Kudlu Gate', 'Mysore Road', 'Thigalarapalya', 'Talaghattapura','Kadugodi', 'Ambedkar
Nagar', 'Jigani', 'Yelahanka New Town',
'Frazer Town', 'Kanakapura', 'Attibele', 'Dodda Nekkundi','Devanahalli',
'Lakshminarayana Pura', 'Nagarbhavi',
'5th Phase JP Nagar', 'TC Palaya', 'Ananth Nagar', 'Anekal', 'Kudl 'CV Raman Nagar',

u', 'Jalahalli', 'Kengeri Satellite Town', 'Doddathog'Bhoganhalli', 'Subramanyapura', 'Kalena

uru', Agrahara', 'Horamavu Agar 'Vidyaranyapura', 'Hosur Road', 'Hebbal Kempapura', 'BTM

a', 2nd Stag

e', 'Domlur', 'Horamavu Banaswadi', 'Tumkur Road', 'Mahadevpura'],dtype='object')

```
# location count is greter than 19 then we create column of that feature
# then if this location present in location feature then set value 1 else 0 ( ohe hot en
df16 = df15.copy()
for cat_var in location_gert_20:
  df16['location_'+cat_var]=np.where(df16['location']==cat_var, 1,0)
df16.shape
```

<ipython-input-66-549e2d4d64be>:5: PerformanceWarning: DataFrame is highlyfragmented. This is usually the result of calling `frame.insert` many times, which has poor performance. Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
df16['location_'+cat_var]=np.where(df16['location']==cat_var, 1,0)Out[66]:

(7120, 111)

In [ ]:

```
df16.head()
```

Out[67]:

	area_type	availability	location	bath	balcony	price	total_sqft_int	bhk	price_
0	Super built-up Area	Ready To Move	Devarabeesana Halli	3.0	2.0	150.0	1672.0	3	897
1	Built-up Area	Ready To Move	Devarabeesana Halli	3.0	3.0	149.0	1750.0	3	851
2	Super built-up Area	Ready To Move	Devarabeesana Halli	3.0	2.0	150.0	1750.0	3	857
4	Super built-up Area	Ready To Move	Devarachikkanahalli	2.0	2.0	40.0	1250.0	2	320
5	Plot Area	Ready To Move	Devarachikkanahalli	2.0	2.0	83.0	1200.0	2	691

In [ ]:

```
"""## Drop categorical variable"""

df17 = df16.drop(["area_type","availability",'location'], axis =1)
df17.shape
```

Out[68]:

(7120, 108)

```
df17.head()

df17.to_csv('df_processed.csv', index=False)
```

MODEL BUILDING

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", None)
```

In [3]:

```
from google.colab import files
files=files.upload()
df = pd.read_csv('ohe_data.csv')
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session.Please rerun this cell to enable.

Saving ohe_data.csv to ohe_data.csv

In [4]:

```
df.shape
```

Out[4]:

(7120, 108)

In [5]:

```
df.head()
```

Out[5]:

bath	balcony	price	total_sqft_int	bhk	price_per_sqft	area_typeSuper built-up Area	area_typeBuilt-up Area

0	3.0	2.0	150.0	1672.0	3	8971.291866	1	0
1	3.0	3.0	149.0	1750.0	3	8514.285714	0	1
2	3.0	2.0	150.0	1750.0	3	8571.428571	1	0
3	2.0	2.0	40.0	1250.0	2	3200.000000	1	0
4	2.0	2.0	83.0	1200.0	2	6916.666667	0	0

```
df.head()
```

Out[6]:

	bath	balcony	price	total_sqft_int	bhk	price_per_sqft	area_typeSuper built-up Area	area_typeBuilt-up Area
0	3.0	2.0	150.0	1672.0	3	8971.291866	1	0
1	3.0	3.0	149.0	1750.0	3	8514.285714	0	1
2	3.0	2.0	150.0	1750.0	3	8571.428571	1	0
3	2.0	2.0	40.0	1250.0	2	3200.000000	1	0
4	2.0	2.0	83.0	1200.0	2	6916.666667	0	0

In [ ]:

```

```

In [7]:

```
"""## Split Dataset in train and test"""

X = df.drop("price", axis=1)
y = df['price']
print('Shape of X = ', X.shape)
print('Shape of y = ', y.shape)
```

Shape of X = (7120, 107)Shape of y =
(7120,)

In [8]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state
print('Shape of X_train = ', X_train.shape)
print('Shape of y_train = ', y_train.shape)
print('Shape of X_test = ', X_test.shape)
print('Shape of y_test = ', y_test.shape)
```

Shape of X_train = (5696, 107)Shape of
y_train = (5696,)
Shape of X_test = (1424, 107)Shape of
y_test = (1424,)

```
"""## Feature Scaling"""

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train= sc.transform(X_train)
X_test = sc.transform(X_test)
```

In [10]:

```
"""## Machine Learning Model Training

## Linear Regression
"""

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
lr = LinearRegression()
lr_lasso = Lasso()
lr_ridge = Ridge()
```

In [11]:

```
def rmse(y_test, y_pred):
  return np.sqrt(mean_squared_error(y_test, y_pred))

lr.fit(X_train, y_train)
lr_score = lr.score(X_test, y_test) # with all num var 0.7842744111909903
lr_rmse = rmse(y_test, lr.predict(X_test))
lr_score, lr_rmse
```

Out[11]:

(0.7903837092682255, 64.89843531105602)

In [12]:

```
# Lasso
lr_lasso.fit(X_train, y_train)
lr_lasso_score=lr_lasso.score(X_test, y_test) # with balcony 0.5162364637824872
lr_lasso_rmse = rmse(y_test, lr_lasso.predict(X_test))
lr_lasso_score, lr_lasso_rmse
```

Out[12]:

(0.8036372973672521, 62.813242204691555)

```
"""## Support Vector Machine"""

from sklearn.svm import SVR
svr = SVR()
svr.fit(X_train,y_train)
svr_score=svr.score(X_test,y_test) # with 0.2630802200711362
svr_rmse = rmse(y_test, svr.predict(X_test))
svr_score, svr_rmse
```

Out[16]:

(0.20638035840828173, 126.27806378079055)

In [13]:

```
"""## Random Forest Regressor"""

from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()
rfr.fit(X_train,y_train)
rfr_score=rfr.score(X_test,y_test) # with 0.8863376025408044
rfr_rmse = rmse(y_test, rfr.predict(X_test))
rfr_score, rfr_rmse
```

Out[13]:

(0.8934165866198863, 46.277141237623795)

In [14]:

```
"""## XGBoost"""

import xgboost
xgb_reg = xgboost.XGBRegressor()
xgb_reg.fit(X_train,y_train)
xgb_reg_score=xgb_reg.score(X_test,y_test) # with 0.8838865742273464
xgb_reg_rmse = rmse(y_test, xgb_reg.predict(X_test))
xgb_reg_score, xgb_reg_rmse
```

Out[14]:

(0.8866071985706575, 47.73252984729787)

In [17]:

```
print(pd.DataFrame([{'Model': 'Linear Regression','Score':lr_score, "RMSE":lr_rmse},
              {'Model': 'Lasso','Score':lr_lasso_score, "RMSE":lr_lasso_rmse},
              {'Model': 'Support Vector Machine','Score':svr_score, "RMSE":svr_rmse},
              {'Model': 'Random Forest','Score':rfr_score, "RMSE":rfr_rmse},
              {'Model': 'XGBoost','Score':xgb_reg_score, "RMSE":xgb_reg_rmse}],
             columns=['Model','Score','RMSE']))
```

	Model	Score	RMSE
0	Linear Regression	0.790384	64.898435
1	Lasso	0.803637	62.813242
2	Support Vector Machine	0.206380	126.278064
3	Random Forest	0.893417	46.277141
4	XGBoost	0.886607	47.732530

```
"""## Cross Validation"""

from sklearn.model_selection import KFold,cross_val_score
cvs = cross_val_score(xgb_reg, X_train,y_train, cv = 10)
cvs, cvs.mean() # 0.9845963377450353)
```

Out[18]:

(array([0.98991766, 0.98632505, 0.99685279, 0.98296558, 0.97656955,
0.99525125, 0.97360094, 0.94556716, 0.99408487, 0.98854857]),
0.9829683423733027)

In [19]:

```
cvs_rfr = cross_val_score(rfr, X_train,y_train, cv = 10)
cvs_rfr, cvs_rfr.mean() # 0.9652425691235843)
```

Out[19]:

(array([0.991548 , 0.95684631, 0.9977923 , 0.97422004, 0.96530165,
0.93147588, 0.94293705, 0.90864149, 0.99675013, 0.98948002]),
0.9654992875911675)

In [20]:

```
from sklearn.model_selection import cross_val_score
cvs_rfr2 = cross_val_score(RandomForestRegressor(), X_train,y_train, cv = 10)
cvs_rfr2, cvs_rfr2.mean() # 0.9652425691235843)
```

Out[20]:

(array([0.99154604, 0.96823157, 0.99734878, 0.96817488, 0.9680747 ,
0.9477437 , 0.92612231, 0.91591544, 0.99606147, 0.98811826]),
0.9667337155042659)

In [21]:

```
"""# Hyper Parmeter Tuning"""

from sklearn.model_selection import GridSearchCV
from xgboost.sklearn import XGBRegressor
```

In [22]:

```
xgb1 = XGBRegressor()
earning_rate': [0.1,0.03, 0.05, 0.07], #so called `eta` value, # [defaul'min_child_weight': [1,3,5], #[default=1] Defines the
minimum sum of weigh'max_depth': [4, 6, 8], #[default=6] The maximum depth of a tree,
'gamma':[0,0.1,0.001,0.2], #Gamma specifies the minimum loss reduction req'subsample': [0.7,1,1.5], #Denotes the
fraction of observations to be rand'colsample_bytree': [0.7,1,1.5], #Denotes the fraction of columns to be ra
'objective':['reg:linear'], #This defines the loss function to be minimize

'n_estimators': [100,300,500]}
```

```
xgb_grid = GridSearchCV(xgb1,
                        parameters,
                        cv = 2,
                        n_jobs = -1,
                        verbose=True)
```

In [28]:

```
xgb_tune = xgb_grid.estimator

xgb_tune.fit(X_train,y_train) # 0.9117591385438816
xgb_tune.score(X_test,y_test)
```

Out[28]:

0.8866071985706575

In [29]:

```
cvs = cross_val_score(xgb_tune, X_train,y_train, cv = 10)
cvs, cvs.mean() # 0.9645582338461773
```

Out[29]:

(array([0.98991766, 0.98632505, 0.99685279, 0.98296558, 0.97656955,
0.99525125, 0.97360094, 0.94556716, 0.99408487, 0.98854857]),
0.9829683423733027)

In [30]:

```
[i/10.0 for i in range(1,6)]

xgb_grid.estimator
```

Out[30]:

gressor(base_score=None, booster=None, callbacks=None,colsample_bylevel=None,
colsample_bynode=None,
colsample_bytree=None, early_stopping_rounds=None,
enable_categorical=False, eval_metric=None, feature_types=Non
e,
gamma=None, gpu_id=None, grow_policy=None, importance_type=No
ne,
interaction_constraints=None, learning_rate=None, max_bin=Non
e,
max_cat_threshold=None, max_cat_to_onehot=None,
max_delta_step=None, max_depth=None, max_leaves=None,
min_child_weight=None, missing=nan, monotone_constraints=Non
e,
n_estimators=100, n_jobs=None, num_parallel_tree=None,
predictor=None, random_state=None, ...)

In a Jupyter environment, please rerun this cell to show the HTML representation ortrust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this pagewith nbviewer.org.

```
cvs = cross_val_score(xgb_tune2, X_train,y_train, cv = 5)
cvs, cvs.mean() # 0.9706000326331659'''
```

[15:02:38] WARNING: ../src/objective/regression_obj.cu:213: reg:linearis now deprecated in favor of reg:squarederror.

/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.py:794: UserWarning: Scoring failed. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
File "/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_scorer.py", line 117, in_call
score = scorer(estimator, *args, **kwargs)
File "/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_scorer. py", line 444, in _passthrough_scorer
return estimator.score(*args, **kwargs)
File "/usr/local/lib/python3.9/dist-packages/sklearn/base.py", line 722, in score
y_pred = self.predict(X)
File "/usr/local/lib/python3.9/dist-packages/xgboost/sklearn.py", line 1114, in predict
predts = self.get_booster().inplace_predict(
File "/usr/local/lib/python3 9/dist-packages/xgboost/core py" line 2

```
"""## Test Model"""

list(X.columns)

# it help to get predicted value of hosue  by providing features value
def predict_house_price(model,bath,balcony,total_sqft_int,bhk,price_per_sqft,area_type,a

  x =np.zeros(len(X.columns)) # create zero numpy array, len = 107 as input value for mo

  # adding feature's value accorind to their column index
  x[0]=bath
  x[1]=balcony
  x[2]=total_sqft_int
  x[3]=bhk
  x[4]=price_per_sqft

  if "availability"=="Ready To Move":
    x[8]=1

  if 'area_type'+area_type in X.columns:
    area_type_index = np.where(X.columns=="area_type"+area_type)[0][0]
    x[area_type_index] =1

    #print(area_type_index)

  if 'location_'+location in X.columns:
    loc_index = np.where(X.columns=="location_"+location)[0][0]
    x[loc_index] =1

    #print(loc_index)

  #print(x)

  # feature scaling
  x = sc.transform([x])[0] # give 2d np array for feature scaling and get 1d scaled np a
  #print(x)

  return model.predict([x])[0] # return the predicted value by train XGBoost model
```

In [34]:

```
predict_house_price(model=xgb_tune2, bath=3,balcony=2,total_sqft_int=1672,bhk=3,price_pe
```

/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(

Out[34]:

146.30998

```
##test sample
#area_type   availability            location           bath          balcony price          total_sqft_int bhk pric#2   Super
built-up Area Ready To Move          Devarabeesana Halli 3.0 2.0 150.0                       1750.0 3

predict_house_price(model=xgb_tune2, bath=3,balcony=2,total_sqft_int=1750,bhk=3,price_pe
```

/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(

Out[35]:

146.11647

In [36]:

```
"""# Save model & load model"""

import joblib
# save model
joblib.dump(xgb_tune2, 'bangalore_house_price_prediction_model.pkl')
joblib.dump(rfr, 'bangalore_house_price_prediction_rfr_model.pkl')
```

Out[36]:

['bangalore_house_price_prediction_rfr_model.pkl']

In [37]:

```
# load model
bangalore_house_price_prediction_model = joblib.load("bangalore_house_price_prediction_m
```

In [38]:

```
# predict house price
predict_house_price(bangalore_house_price_prediction_model,bath=3,balcony=3,total_sqft_i
```

/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(

Out[38]:

67.8374

MODEL.PY

```
#load data
df = pd.read_csv("D:/banglore house price/.venv/Scripts/ohe_data.csv")

# Split data
X= df.drop('price', axis=1)y=
df['price']
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_state=51)

# feature scaling
sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)X_test =
sc.transform(X_test)


###### Load Model

model = joblib.load('D:/banglore house price/.venv/Scripts/2.pkl')


# it help to get predicted value of house by providing features valuedef
predict_house_price(bath,balcony,total_sqft_int,bhk,price_per_sqft,area_type,availability,location):

    x =np.zeros(len(X.columns)) # create zero numpy array, len = 107 as inputvalue for model

# adding feature's value accorind to their column indexx[0]=bath
x[1]=balcony x[2]=total_sqft_int
x[3]=bhk x[4]=price_per_sqft

if "availability"=="Ready To Move":x[8]=1

if 'area_type'+area_type in X.columns:
area_type_index = np.where(X.columns=="area_type"+area_type)[0][0]x[area_type_index] =1

if 'location_'+location in X.columns:
loc_index = np.where(X.columns=="location_"+location)[0][0]
```

```
    x[loc_index] =1


  x = sc.transform([x])[0]

  return model.predict([x])[0]
```

APP.PY

```
#Import Libraries
from flask import Flask, request, render_template

import sys

# Add the directory containing the module to the Python path
sys.path.append(r'D:/banglore house price/.venv\static/model.py')

# Import the module
import model

app = Flask(__name__)
app = Flask(__name__, template_folder='D:/banglore house
price/.venv/template')


# render htmp page
@app.route('/')
def home():
    return render_template('index.html')



# get user input and the predict the output and return to user
@app.route('/predict',methods=['POST'])
def predict():

    #take data from form and store in each feature
    input_features = [x for x in request.form.values()]
    bath = input_features[0]
    balcony = input_features[1]
    total_sqft_int = input_features[2]
    bhk = input_features[3]
    price_per_sqft = input_features[4]
    area_type = input_features[5]
    availability = input_features[6]
    location = input_features[7]

    # predict the price of house by calling model.py
```

```
    predicted_price =
model.predict_house_price(bath,balcony,total_sqft_int,bhk,price_per_sqft,area_
type,availability,location)


    # render the html page and show the output
    return render_template('index.html', prediction_text='Predicted Price of
Bangalore House is {}'.format(predicted_price))



if_name_== "_main_":
     app.run(host="0.0.0.0", port="8080")
```

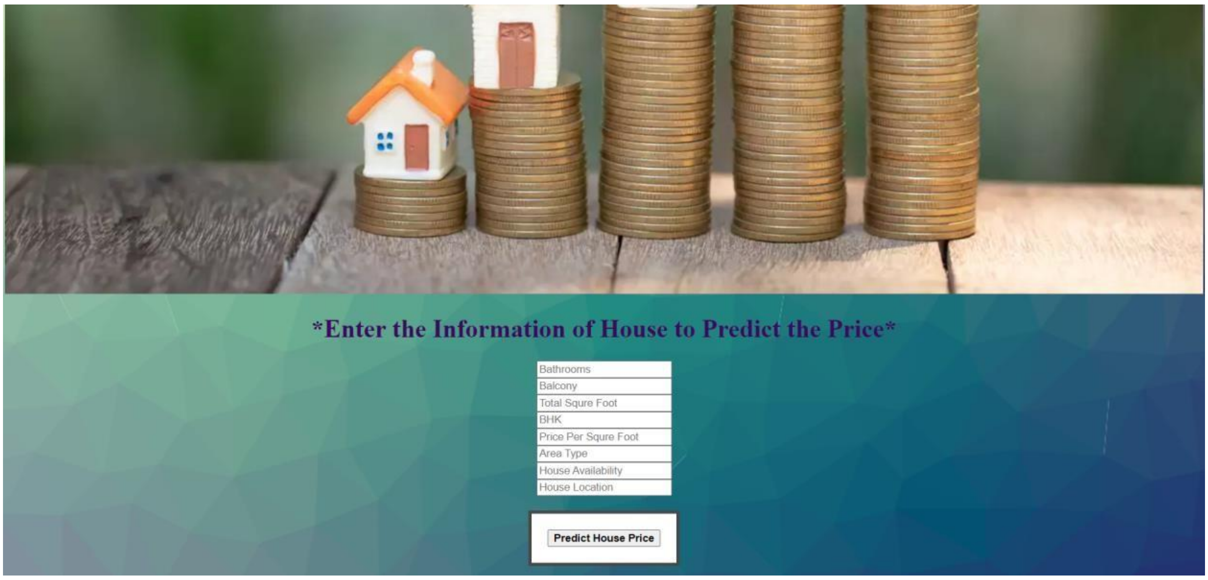
HTML CODE

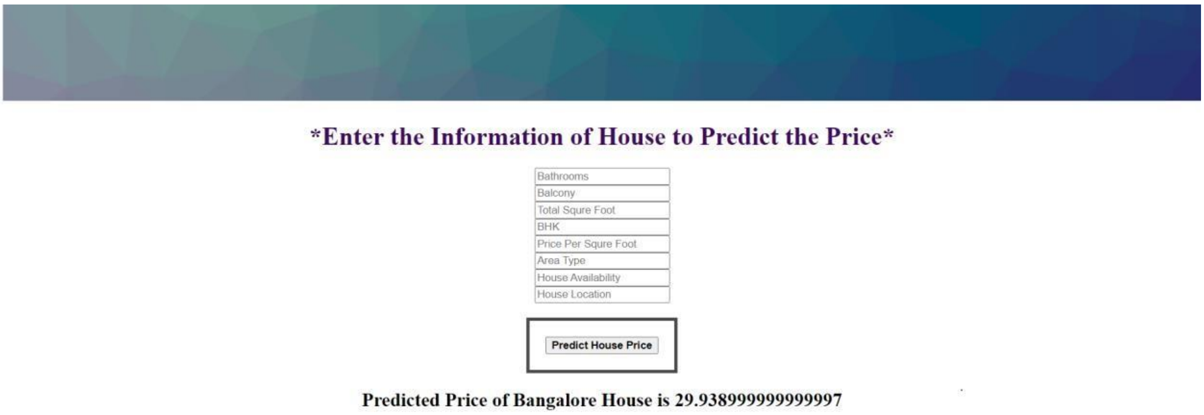
```
<!-- Bangalore House Price Predictor -->

<!DOCTYPE html>
<html >
<head>
  <meta charset="UTF-8">
  <title>Bangalore House Price Predictor </title>

  <style>

      body {
      background-image: url('D:/banglore house price/.venv/template/2.jpg');
      background-repeat: no-repeat;
      background-attachment: fixed;
      background-size: cover;
    }

    h1   {color: rgb(65, 8, 97);}  /* CSS code for heading h1 */
    p   {color: rgb(8, 212, 199);}  /* CSS code for heading h1 */

    /* CSS code for button */
    .button_css {
    color: #494949 !important;
    text-transform: uppercase;
    text-decoration: none;
    background: #ffffff;
    padding: 20px;
    border: 4px solid #494949 !important;
    display: inline-block;
    transition: all 0.4s ease 0s;
    }
```

```
.button_css:hover   { color: #ffffff
!important;background: #f6b93b;
border-color: #f6b93b !important;transition: all
0.4s ease 0s;
}

    .footer { position:
fixed;left: 0;
bottom: 0;
width: 100%;
background-color: #26e1b9;color:
white;
text-align: center;


}
</style>

</head>

<body>


<div>
        <img src="https://coolbackgrounds.io/images/backgrounds/index/ranger-4df6c1b6.png"
class="w3-border w3-padding" alt="shambhavi test img" style="width:100%">
</div>





<div class="login">

<!-- Form Get input to predict Marks-->
<center>
<form action="{{ url_for('predict')}}"method="post">
<h1>*Enter the Information of House to Predict the Price*</h1>

                <input align="center" type="number" name="bathrooms"
placeholder="Bathrooms" required="required" width="48" height="10"step=".01"/><br>
                <input align="center" type="number" name="balcony"
placeholder="Balcony" required="required" width="48" height="10"step=".01"/><br>
```

```
		<input align="center" type="number" name="total_sqft_int" placeholder="Total Squre
Foot" required="required" width="48" height="10"step=".01"/><br>
		<input align="center" type="number" name="bhk" placeholder="BHK"
required="required" width="48" height="10" step=".01"/><br>
		<input align="center" type="number" name="price_per_sqft" placeholder="Price Per Squre
Foot" required="required" width="48" height="10"step=".01"/><br>
		<input type="text" name="area_type" placeholder="Area Type"
required="required" /><br>
		<input type="text" name="availability" placeholder="HouseAvailability"
required="required" /><br>
		<input type="text" name="location" placeholder="House Location"
required="required" />

<br>

<br>

<!-- Show button -->


</form>
</center>

<!-- Show predicted output using ML model -->
<div>
<center>
<h2>{{ prediction_text }}</h2>
</center>
</div>

</div>


</body>
</html>
```

OUTPUTs





CONCLUSION : MLops is successful implemented