TABLE OF CONTENTS

EXPERIMENT 1

AIM:

To design a densenet for temperature conversion.

THEORY AND SOURCE CODE:

The problem we will solve is to convert from Celsius to Fahrenheit, where the approximate formula is:

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

We will give TensorFlow some sample Celsius values (0, 8, 15, 22, 38) and their corresponding Fahrenheit values (32, 46, 59, 72, 100). Then, we will train a model using densenets that figures out the above formula through the training process.

Import dependencies

First, import TensorFlow. Here, we're calling it `tf` for ease of use. We also tell it to only display errors.

Next, import NumPy as `np`. Numpy helps us to represent our data as highly performant lists.

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

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

Set up training data

Since the task in this Codelab is to create a model that can give the temperature in Fahrenheit when given the degrees in Celsius, we create two lists `celsius_q` and `fahrenheit_a` that we can use to train our model.

```
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
```

Some Machine Learning terminology

- **Feature** — The input(s) to our model. In this case, a single value — the degrees in Celsius.
- **Labels** — The output our model predicts. In this case, a single value — the degrees in Fahrenheit.
- **Example** — A pair of inputs/outputs used during training. In our case a pair of values from `celsius_q` and `fahrenheit_a` at a specific index, such as `(22,72)`.

Create the model

Next, 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.

Build a layer

We'll call the layer `l0` and create it by instantiating `tf.keras.layers.Dense` with the following configuration:

- `input_shape=[1]` — This specifies that the input to this layer is a single value. That is, the shape is a one-dimensional array with one member. Since this is the first (and only) layer, that input shape is the input shape of the entire model. The single value is a floating point number, representing degrees Celsius.

- `units=1` — This specifies the number of neurons in the layer. The number of neurons defines how many internal variables the layer has to try to learn how to solve the problem (more later). Since this is the final layer, it is also the size of the model's output — a single float value representing degrees Fahrenheit. (In a multi-layered network, the size and shape of the layer would need to match the `input_shape` of the next layer.)

```
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.

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

Note

You will often see the layers defined inside the model definition, rather than beforehand:

```
model = tf.keras.Sequential([
  tf.keras.layers.Dense(units=1, input_shape=[1])
])
```

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.

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

These are used during training (`model.fit()`, below) to first calculate the loss at each point, and then improve it. In fact, the act of calculating the current loss of a model and then improving it is precisely what training is.

During training, the optimizer function is used to calculate adjustments to the model's internal variables. The goal is to adjust the internal variables until the model (which is really a math function) mirrors the actual equation for converting Celsius to Fahrenheit.

TensorFlow uses numerical analysis to perform this tuning, and all this complexity is hidden from you so we will not go into the details here. What is useful to know about these parameters are:

The loss function (mean squared error) and the optimizer (Adam) used here are standard for simple models like this one, but many others are available. It is not important to know how these specific functions work at this point.

One part of the Optimizer you may need to think about when building your own models is the learning rate (`0.1` in the code above). This is the step size taken when adjusting values in the model. If the value is too small, it will take too many iterations to train the model. Too large, and accuracy goes down. Finding a good value often involves some trial and error, but the range is usually within 0.001 (default), and 0.1

Train the model

Train the model by calling the `fit` method.

During training, the model takes in Celsius values, performs a calculation using the current internal variables (called "weights") and outputs values which are meant to be the Fahrenheit equivalent. Since the weights are initially set randomly, the output will not be close to the correct value. The difference between the actual output and the desired output is calculated using the loss function, and the optimizer function directs how the weights should be adjusted.

This cycle of calculate, compare, adjust is controlled by the `fit` method. The first argument is the inputs, the second argument is the desired outputs. The `epochs` argument specifies how many times this cycle should be run, and the `verbose` argument controls how much output the method produces.

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

```
Finished training the model!
```

In later videos, we will go into more detail on what actually happens here and how a Dense layer actually works internally.

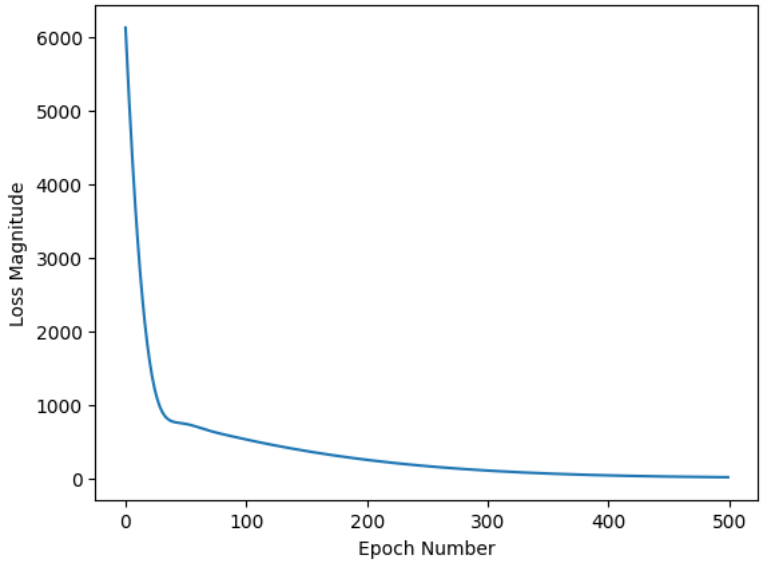
Display training statistics

The `fit` method returns a history object. We can use this object to plot how the loss of our model goes down after each training epoch. A high loss means that the Fahrenheit degrees the model predicts is far from the corresponding value in `fahrenheit_a`.

We'll use Matplotlib to visualize this (you could use another tool). As you can see, our model improves very quickly at first, and then has a steady, slow improvement until it is very near "perfect" towards the end.

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

```
[<matplotlib.lines.Line2D at 0x7fa88261a370>]
```



Use the model to predict values

Now you have a model that has been trained to learn the relationship between `celsius_q` and `fahrenheit_a`. You can use the predict method to have it calculate the Fahrenheit degrees for a previously unknown Celsius degrees.

So, for example, if the Celsius value is 100, what do you think the Fahrenheit result will be? Take a guess before you run this code.

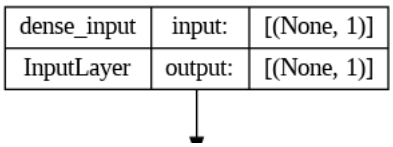
```
print(model.predict([100.0]))
```

```
1/1 [==============================] - 0s 74ms/step
[[211.29253]]
```

```
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1)                 2

=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
None
```

```
import keras
import keras.utils
keras.utils.plot_model(model,show_shapes=True)
```



The correct answer is $100 \times 1.8 + 32 = 212$, so our model is doing really well.

To review

- We created a model with a Dense layer
- We trained it with 3500 examples (7 pairs, over 500 epochs).

Our model tuned the variables (weights) in the Dense layer until it was able to return the correct Fahrenheit value for any Celsius value. (Remember, 100 Celsius was not part of our training data.)

Looking at the layer weights

Finally, let's print the internal variables of the Dense layer.

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

```
    These are the layer variables: [array([[1.8266834]], dtype=float32), array([28.624191], dtype=float32)]
```

The first variable is close to ~1.8 and the second to ~32. These values (1.8 and 32) are the actual variables in the real conversion formula.

This is really close to the values in the conversion formula. We'll explain this in an upcoming video where we show how a Dense layer works, but for a single neuron with a single input and a single output, the internal math looks the same as the equation for a line, $y = mx + b$, which has the same form as the conversion equation, $f = 1.8c + 32$.

Since the form is the same, the variables should converge on the standard values of 1.8 and 32, which is exactly what happened.

With additional neurons, additional inputs, and additional outputs, the formula becomes much more complex, but the idea is the same.

A little experiment

Just for fun, what if we created more Dense layers with different units, which therefore also has more variables?

```
l0 = tf.keras.layers.Dense(units=4, input_shape=[1])
l1 = tf.keras.layers.Dense(units=4)
l2 = tf.keras.layers.Dense(units=1)
model = tf.keras.Sequential([l0, l1, l2])
model.compile(loss='mean_squared_error', optimizer=tf.keras.optimizers.Adam(0.1))
model.fit(celsius_q, fahrenheit_a, epochs=500, verbose=False)
print("Finished training the model")
print(model.predict([100.0]))
print("Model predicts that 100 degrees Celsius is: {} degrees Fahrenheit".format(model.predict([100.0])))
print("These are the l0 variables: {}".format(l0.get_weights()))
print("These are the l1 variables: {}".format(l1.get_weights()))
print("These are the l2 variables: {}".format(l2.get_weights()))
```

```
    Finished training the model
    1/1 [==============================] - 0s 59ms/step
    [[211.74745]]
    1/1 [==============================] - 0s 33ms/step
    Model predicts that 100 degrees Celsius is: [[211.74745]] degrees Fahrenheit
    These are the l0 variables: [array([[ 0.36089694, -0.4090389 , -0.0063686 , -0.3904875 ]],
          dtype=float32), array([ 3.4436357, -3.4865687,  2.046431 , -3.4134338], dtype=float32)]
    These are the l1 variables: [array([[ 0.62515783,  0.7055783 , -0.31273922,  0.81420064],
           [-0.39436975, -0.99827737, -0.54323816,  0.6652985 ],
           [ 0.1947854 ,  0.26946622,  0.40013695, -0.21200895],
           [-0.4280463 , -0.40862733, -0.02305545,  0.7729029 ]],
          dtype=float32), array([ 3.454347 ,  3.4841833, -1.9135866, -3.3435104], dtype=float32)]
    These are the l2 variables: [array([[ 1.1715512],
           [ 1.1638895],
           [-0.311847 ],
           [-0.8288303]], dtype=float32), array([3.4162192], dtype=float32)]
```

As we can see, this model is also able to predict the corresponding Fahrenheit value really well. But when we look at the variables (weights) in the `l0` and `l1` layers, they are nothing even close to ~1.8 and ~32. The added complexity hides the "simple" form of the conversion equation.

EXPERIMENT 2

AIM:

To design a densenet for classifying images using Fashion MNIST dataset

THEORY AND SOURCE CODE:

A DenseNet is a type of convolutional neural network that utilises dense connections between layers, through Dense Blocks, where we connect all layers (with matching feature-map sizes) directly with each other.

In [1]:
```
# Import TensorFlow Datasets
import tensorflow as tf
import tensorflow_datasets as tfds

tfds.disable_progress_bar()

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

In [2]:
```
import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

In [3]:
```
dataset, metadata = tfds.load('fashion_mnist', as_supervised=True, with_info=True)
train_dataset, test_dataset = dataset['train'], dataset['test']
```

```
Downloading and preparing dataset 29.45 MiB (download: 29.45 MiB, generated: 36.42
MiB, total: 65.87 MiB) 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.
```

In [4]:
```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal',      'Shirt',   'Sneaker',  'Bag',   'Ankle boot']
```

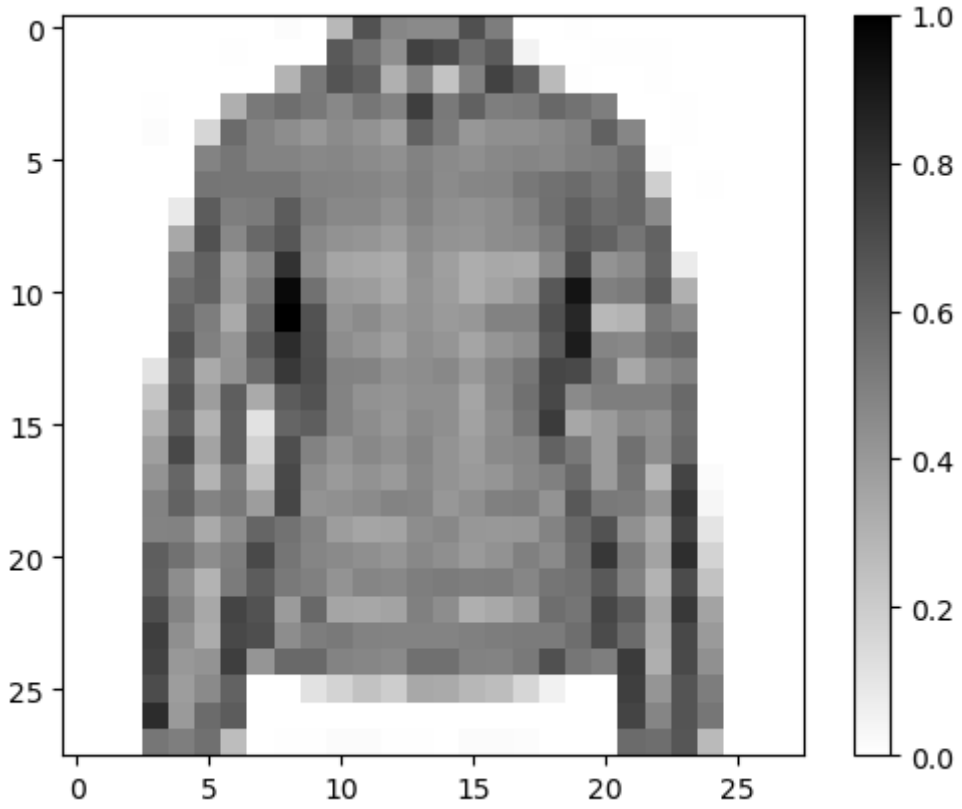
In [5]:
```
num_train_examples = metadata.splits['train'].num_examples
num_test_examples = metadata.splits['test'].num_examples
print("Number of training examples: {}".format(num_train_examples))
print("Number of test examples:     {}".format(num_test_examples))
```

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

In [6]:
```
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()
```

In [7]:
```
# Take a single image, and remove the color dimension by reshaping
for image, label in test_dataset.take(5):
  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()
```



In [8]:
```
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()
```

In [9]:

```python
import tensorflow as tf

# Define DenseNet block
def dense_block(x, num_layers):
    for _ in range(num_layers):
        bn = tf.keras.layers.BatchNormalization()(x)
        conv = tf.keras.layers.Conv2D(64, (3, 3), padding='same', activation='relu
        x = tf.keras.layers.Concatenate(axis=-1)([x, conv])
    return x

# Define input layer
inputs = tf.keras.Input(shape=(28, 28, 1))

# Create DenseNet model
x = tf.keras.layers.Conv2D(32, (3, 3), padding='same', activation='relu')(inputs)
x = tf.keras.layers.MaxPooling2D((2, 2), strides=2)(x)
x = dense_block(x, 4)
x = dense_block(x, 8)
x = dense_block(x, 16)
x = dense_block(x, 8)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(128, activation='relu')(x)
outputs = tf.keras.layers.Dense(10, activation='softmax')(x)
```

```
# Create model
model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

In [10]:
```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0         []

 conv2d (Conv2D)             (None, 28, 28, 32)        320       ['input_1[0][0]']

 max_pooling2d (MaxPooling2D)  (None, 14, 14, 32)      0         ['conv2d[0][0]']

 batch_normalization (BatchNorm  (None, 14, 14, 32)    128       ['max_pooling2d
[0][0]']
 alization)

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        18496     ['batch_normaliza
tion[0][0]']

 concatenate (Concatenate)   (None, 14, 14, 96)        0         ['max_pooling2d
[0][0]',
                                                                  'conv2d_1[0]
[0]']

 batch_normalization_1 (BatchNo  (None, 14, 14, 96)    384       ['concatenate[0]
[0]']
 rmalization)

 conv2d_2 (Conv2D)           (None, 14, 14, 64)        55360     ['batch_normaliza
tion_1[0][0]']

 concatenate_1 (Concatenate)  (None, 14, 14, 160)      0         ['concatenate[0]
[0]',
                                                                  'conv2d_2[0]
[0]']

 batch_normalization_2 (BatchNo  (None, 14, 14, 160)   640       ['concatenate_1
[0][0]']
 rmalization)

 conv2d_3 (Conv2D)           (None, 14, 14, 64)        92224     ['batch_normaliza
tion_2[0][0]']

 concatenate_2 (Concatenate)  (None, 14, 14, 224)      0         ['concatenate_1
[0][0]',
                                                                  'conv2d_3[0]
[0]']

 batch_normalization_3 (BatchNo  (None, 14, 14, 224)   896       ['concatenate_2
[0][0]']
 rmalization)

 conv2d_4 (Conv2D)           (None, 14, 14, 64)        129088    ['batch_normaliza
tion_3[0][0]']

 concatenate_3 (Concatenate)  (None, 14, 14, 288)      0         ['concatenate_2
[0][0]',
                                                                  'conv2d_4[0]
[0]']

 batch_normalization_4 (BatchNo  (None, 14, 14, 288)   1152      ['concatenate_3
```

```
[0][0']
 rmalization)

 conv2d_5 (Conv2D)              (None, 14, 14, 64)   165952      ['batch_normaliza
tion_4[0][0']

 concatenate_4 (Concatenate)    (None, 14, 14, 352)  0           ['concatenate_3
[0][0',
                                                                  'conv2d_5[0]
[0]']

 batch_normalization_5 (BatchNo  (None, 14, 14, 352)  1408       ['concatenate_4
[0][0']
 rmalization)

 conv2d_6 (Conv2D)              (None, 14, 14, 64)   202816      ['batch_normaliza
tion_5[0][0']

 concatenate_5 (Concatenate)    (None, 14, 14, 416)  0           ['concatenate_4
[0][0',
                                                                  'conv2d_6[0]
[0]']

 batch_normalization_6 (BatchNo  (None, 14, 14, 416)  1664       ['concatenate_5
[0][0']
 rmalization)

 conv2d_7 (Conv2D)              (None, 14, 14, 64)   239680      ['batch_normaliza
tion_6[0][0']

 concatenate_6 (Concatenate)    (None, 14, 14, 480)  0           ['concatenate_5
[0][0',
                                                                  'conv2d_7[0]
[0]']

 batch_normalization_7 (BatchNo  (None, 14, 14, 480)  1920       ['concatenate_6
[0][0']
 rmalization)

 conv2d_8 (Conv2D)              (None, 14, 14, 64)   276544      ['batch_normaliza
tion_7[0][0']

 concatenate_7 (Concatenate)    (None, 14, 14, 544)  0           ['concatenate_6
[0][0',
                                                                  'conv2d_8[0]
[0]']

 batch_normalization_8 (BatchNo  (None, 14, 14, 544)  2176       ['concatenate_7
[0][0']
 rmalization)

 conv2d_9 (Conv2D)              (None, 14, 14, 64)   313408      ['batch_normaliza
tion_8[0][0']

 concatenate_8 (Concatenate)    (None, 14, 14, 608)  0           ['concatenate_7
[0][0',
                                                                  'conv2d_9[0]
[0]']

 batch_normalization_9 (BatchNo  (None, 14, 14, 608)  2432       ['concatenate_8
[0][0']
```

```
 rmalization)

 conv2d_10 (Conv2D)             (None, 14, 14, 64)   350272      ['batch_normaliza
tion_9[0][0]']

 concatenate_9 (Concatenate)    (None, 14, 14, 672)  0           ['concatenate_8
[0][0]',
                                                                  'conv2d_10[0]
[0]']

 batch_normalization_10 (BatchN  (None, 14, 14, 672)  2688       ['concatenate_9
[0][0]']
 ormalization)

 conv2d_11 (Conv2D)             (None, 14, 14, 64)   387136      ['batch_normaliza
tion_10[0][0]']

 concatenate_10 (Concatenate)   (None, 14, 14, 736)  0           ['concatenate_9
[0][0]',
                                                                  'conv2d_11[0]
[0]']

 batch_normalization_11 (BatchN  (None, 14, 14, 736)  2944       ['concatenate_10
[0][0]']
 ormalization)

 conv2d_12 (Conv2D)             (None, 14, 14, 64)   424000      ['batch_normaliza
tion_11[0][0]']

 concatenate_11 (Concatenate)   (None, 14, 14, 800)  0           ['concatenate_10
[0][0]',
                                                                  'conv2d_12[0]
[0]']

 batch_normalization_12 (BatchN  (None, 14, 14, 800)  3200       ['concatenate_11
[0][0]']
 ormalization)

 conv2d_13 (Conv2D)             (None, 14, 14, 64)   460864      ['batch_normaliza
tion_12[0][0]']

 concatenate_12 (Concatenate)   (None, 14, 14, 864)  0           ['concatenate_11
[0][0]',
                                                                  'conv2d_13[0]
[0]']

 batch_normalization_13 (BatchN  (None, 14, 14, 864)  3456       ['concatenate_12
[0][0]']
 ormalization)

 conv2d_14 (Conv2D)             (None, 14, 14, 64)   497728      ['batch_normaliza
tion_13[0][0]']

 concatenate_13 (Concatenate)   (None, 14, 14, 928)  0           ['concatenate_12
[0][0]',
                                                                  'conv2d_14[0]
[0]']

 batch_normalization_14 (BatchN  (None, 14, 14, 928)  3712       ['concatenate_13
[0][0]']
 ormalization)
```

```
 conv2d_15 (Conv2D)             (None, 14, 14, 64)   534592      ['batch_normaliza
tion_14[0][0]']

 concatenate_14 (Concatenate)   (None, 14, 14, 992)  0           ['concatenate_13
[0][0]',
                                                                  'conv2d_15[0]
[0]']

 batch_normalization_15 (BatchN  (None, 14, 14, 992)  3968       ['concatenate_14
[0][0]']
 ormalization)

 conv2d_16 (Conv2D)             (None, 14, 14, 64)   571456      ['batch_normaliza
tion_15[0][0]']

 concatenate_15 (Concatenate)   (None, 14, 14, 1056  0           ['concatenate_14
[0][0]',
                                )                                 'conv2d_16[0]
[0]']

 batch_normalization_16 (BatchN  (None, 14, 14, 1056  4224       ['concatenate_15
[0][0]']
 ormalization)                  )

 conv2d_17 (Conv2D)             (None, 14, 14, 64)   608320      ['batch_normaliza
tion_16[0][0]']

 concatenate_16 (Concatenate)   (None, 14, 14, 1120  0           ['concatenate_15
[0][0]',
                                )                                 'conv2d_17[0]
[0]']

 batch_normalization_17 (BatchN  (None, 14, 14, 1120  4480       ['concatenate_16
[0][0]']
 ormalization)                  )

 conv2d_18 (Conv2D)             (None, 14, 14, 64)   645184      ['batch_normaliza
tion_17[0][0]']

 concatenate_17 (Concatenate)   (None, 14, 14, 1184  0           ['concatenate_16
[0][0]',
                                )                                 'conv2d_18[0]
[0]']

 batch_normalization_18 (BatchN  (None, 14, 14, 1184  4736       ['concatenate_17
[0][0]']
 ormalization)                  )

 conv2d_19 (Conv2D)             (None, 14, 14, 64)   682048      ['batch_normaliza
tion_18[0][0]']

 concatenate_18 (Concatenate)   (None, 14, 14, 1248  0           ['concatenate_17
[0][0]',
                                )                                 'conv2d_19[0]
[0]']

 batch_normalization_19 (BatchN  (None, 14, 14, 1248  4992       ['concatenate_18
[0][0]']
 ormalization)                  )
```

```
 conv2d_20 (Conv2D)             (None, 14, 14, 64)   718912      ['batch_normaliza
tion_19[0][0]']

 concatenate_19 (Concatenate)   (None, 14, 14, 1312  0           ['concatenate_18
[0][0]',
                                )                                 'conv2d_20[0]
[0]']

 batch_normalization_20 (BatchN  (None, 14, 14, 1312  5248       ['concatenate_19
[0][0]']
 ormalization)                  )

 conv2d_21 (Conv2D)             (None, 14, 14, 64)   755776      ['batch_normaliza
tion_20[0][0]']

 concatenate_20 (Concatenate)   (None, 14, 14, 1376  0           ['concatenate_19
[0][0]',
                                )                                 'conv2d_21[0]
[0]']

 batch_normalization_21 (BatchN  (None, 14, 14, 1376  5504       ['concatenate_20
[0][0]']
 ormalization)                  )

 conv2d_22 (Conv2D)             (None, 14, 14, 64)   792640      ['batch_normaliza
tion_21[0][0]']

 concatenate_21 (Concatenate)   (None, 14, 14, 1440  0           ['concatenate_20
[0][0]',
                                )                                 'conv2d_22[0]
[0]']

 batch_normalization_22 (BatchN  (None, 14, 14, 1440  5760       ['concatenate_21
[0][0]']
 ormalization)                  )

 conv2d_23 (Conv2D)             (None, 14, 14, 64)   829504      ['batch_normaliza
tion_22[0][0]']

 concatenate_22 (Concatenate)   (None, 14, 14, 1504  0           ['concatenate_21
[0][0]',
                                )                                 'conv2d_23[0]
[0]']

 batch_normalization_23 (BatchN  (None, 14, 14, 1504  6016       ['concatenate_22
[0][0]']
 ormalization)                  )

 conv2d_24 (Conv2D)             (None, 14, 14, 64)   866368      ['batch_normaliza
tion_23[0][0]']

 concatenate_23 (Concatenate)   (None, 14, 14, 1568  0           ['concatenate_22
[0][0]',
                                )                                 'conv2d_24[0]
[0]']

 batch_normalization_24 (BatchN  (None, 14, 14, 1568  6272       ['concatenate_23
[0][0]']
 ormalization)                  )

 conv2d_25 (Conv2D)             (None, 14, 14, 64)   903232      ['batch_normaliza
```

```
tion_24[0][0]']

 concatenate_24 (Concatenate)   (None, 14, 14, 1632  0           ['concatenate_23
[0][0]',
                                )                                 'conv2d_25[0]
[0]']

 batch_normalization_25 (BatchN  (None, 14, 14, 1632  6528       ['concatenate_24
[0][0]']
 ormalization)                  )

 conv2d_26 (Conv2D)             (None, 14, 14, 64)   940096      ['batch_normaliza
tion_25[0][0]']

 concatenate_25 (Concatenate)   (None, 14, 14, 1696  0           ['concatenate_24
[0][0]',
                                )                                 'conv2d_26[0]
[0]']

 batch_normalization_26 (BatchN  (None, 14, 14, 1696  6784       ['concatenate_25
[0][0]']
 ormalization)                  )

 conv2d_27 (Conv2D)             (None, 14, 14, 64)   976960      ['batch_normaliza
tion_26[0][0]']

 concatenate_26 (Concatenate)   (None, 14, 14, 1760  0           ['concatenate_25
[0][0]',
                                )                                 'conv2d_27[0]
[0]']

 batch_normalization_27 (BatchN  (None, 14, 14, 1760  7040       ['concatenate_26
[0][0]']
 ormalization)                  )

 conv2d_28 (Conv2D)             (None, 14, 14, 64)   1013824     ['batch_normaliza
tion_27[0][0]']

 concatenate_27 (Concatenate)   (None, 14, 14, 1824  0           ['concatenate_26
[0][0]',
                                )                                 'conv2d_28[0]
[0]']

 batch_normalization_28 (BatchN  (None, 14, 14, 1824  7296       ['concatenate_27
[0][0]']
 ormalization)                  )

 conv2d_29 (Conv2D)             (None, 14, 14, 64)   1050688     ['batch_normaliza
tion_28[0][0]']

 concatenate_28 (Concatenate)   (None, 14, 14, 1888  0           ['concatenate_27
[0][0]',
                                )                                 'conv2d_29[0]
[0]']

 batch_normalization_29 (BatchN  (None, 14, 14, 1888  7552       ['concatenate_28
[0][0]']
 ormalization)                  )

 conv2d_30 (Conv2D)             (None, 14, 14, 64)   1087552     ['batch_normaliza
tion_29[0][0]']
```

```
 concatenate_29 (Concatenate)   (None, 14, 14, 1952  0           ['concatenate_28
[0][0]',
                                )                                 'conv2d_30[0]
[0]']

 batch_normalization_30 (BatchN  (None, 14, 14, 1952  7808       ['concatenate_29
[0][0]']
 ormalization)                  )

 conv2d_31 (Conv2D)             (None, 14, 14, 64)   1124416     ['batch_normaliza
tion_30[0][0]']

 concatenate_30 (Concatenate)   (None, 14, 14, 2016  0           ['concatenate_29
[0][0]',
                                )                                 'conv2d_31[0]
[0]']

 batch_normalization_31 (BatchN  (None, 14, 14, 2016  8064       ['concatenate_30
[0][0]']
 ormalization)                  )

 conv2d_32 (Conv2D)             (None, 14, 14, 64)   1161280     ['batch_normaliza
tion_31[0][0]']

 concatenate_31 (Concatenate)   (None, 14, 14, 2080  0           ['concatenate_30
[0][0]',
                                )                                 'conv2d_32[0]
[0]']

 batch_normalization_32 (BatchN  (None, 14, 14, 2080  8320       ['concatenate_31
[0][0]']
 ormalization)                  )

 conv2d_33 (Conv2D)             (None, 14, 14, 64)   1198144     ['batch_normaliza
tion_32[0][0]']

 concatenate_32 (Concatenate)   (None, 14, 14, 2144  0           ['concatenate_31
[0][0]',
                                )                                 'conv2d_33[0]
[0]']

 batch_normalization_33 (BatchN  (None, 14, 14, 2144  8576       ['concatenate_32
[0][0]']
 ormalization)                  )

 conv2d_34 (Conv2D)             (None, 14, 14, 64)   1235008     ['batch_normaliza
tion_33[0][0]']

 concatenate_33 (Concatenate)   (None, 14, 14, 2208  0           ['concatenate_32
[0][0]',
                                )                                 'conv2d_34[0]
[0]']

 batch_normalization_34 (BatchN  (None, 14, 14, 2208  8832       ['concatenate_33
[0][0]']
 ormalization)                  )

 conv2d_35 (Conv2D)             (None, 14, 14, 64)   1271872     ['batch_normaliza
tion_34[0][0]']
```

```
 concatenate_34 (Concatenate)   (None, 14, 14, 2272  0           ['concatenate_33
[0][0]',
                                )                                 'conv2d_35[0]
[0]']

 batch_normalization_35 (BatchN  (None, 14, 14, 2272  9088       ['concatenate_34
[0][0]']
 ormalization)                  )

 conv2d_36 (Conv2D)             (None, 14, 14, 64)   1308736     ['batch_normaliza
tion_35[0][0]']

 concatenate_35 (Concatenate)   (None, 14, 14, 2336  0           ['concatenate_34
[0][0]',
                                )                                 'conv2d_36[0]
[0]']

 global_average_pooling2d (Glob  (None, 2336)        0           ['concatenate_35
[0][0]']
 alAveragePooling2D)

 flatten (Flatten)              (None, 2336)         0           ['global_average_
pooling2d[0][0]'
                                                                  ]

 dense (Dense)                  (None, 128)          299136      ['flatten[0][0]']

 dense_1 (Dense)                (None, 10)           1290        ['dense[0][0]']

==================================================================================
================
Total params: 24,356,810
Trainable params: 24,273,866
Non-trainable params: 82,944
__________________________________________________________________________________
________________
```

In [11]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [12]:
```
BATCH_SIZE = 32
train_dataset = train_dataset.cache().repeat().shuffle(num_train_examples).batch(BA
test_dataset = test_dataset.cache().batch(BATCH_SIZE)
```

In [13]:
```
model.fit(train_dataset, epochs=10, steps_per_epoch=math.ceil(num_train_examples/BA
```

```
Epoch 1/10
1875/1875 [==============================] - 262s 120ms/step - loss: 0.5464 - accu
racy: 0.8071
Epoch 2/10
1875/1875 [==============================] - 224s 120ms/step - loss: 0.3087 - accu
racy: 0.8871
Epoch 3/10
1875/1875 [==============================] - 224s 120ms/step - loss: 0.2555 - accu
racy: 0.9083
Epoch 4/10
1875/1875 [==============================] - 224s 119ms/step - loss: 0.2212 - accu
racy: 0.9196
Epoch 5/10
1875/1875 [==============================] - 224s 120ms/step - loss: 0.1960 - accu
racy: 0.9293
Epoch 6/10
1875/1875 [==============================] - 224s 120ms/step - loss: 0.1765 - accu
racy: 0.9376
Epoch 7/10
1875/1875 [==============================] - 225s 120ms/step - loss: 0.1552 - accu
racy: 0.9441
Epoch 8/10
1875/1875 [==============================] - 225s 120ms/step - loss: 0.1351 - accu
racy: 0.9516
Epoch 9/10
1875/1875 [==============================] - 225s 120ms/step - loss: 0.1220 - accu
racy: 0.9567
Epoch 10/10
1875/1875 [==============================] - 223s 119ms/step - loss: 0.1057 - accu
racy: 0.9623
```

Out[13]: <keras.callbacks.History at 0x7a1185d0c2d0>

In [14]:
```
test_loss, test_accuracy = model.evaluate(test_dataset, steps=math.ceil(num_test_ex
print('Accuracy on test dataset:', test_accuracy)
```

```
313/313 [==============================] - 14s 43ms/step - loss: 0.3032 - accurac
y: 0.9118
Accuracy on test dataset: 0.9118000268936157
```

EXPERIMENT 3

AIM :

To design a convnet for classifying images using Fashion MNIST dataset

THEORY AND SOURCE CODE:

A convolutional neural network (CNN or ConvNet) is a network architecture for deep learning that learns directly from data. CNNs are particularly useful for finding patterns in images to recognize objects, classes, and categories. They can also be quite effective for classifying audio, time-series, and signal data.

In [ ]:
```
#Import TensorFlow Datasets
import tensorflow as tf
import tensorflow_datasets as tfds

tfds.disable_progress_bar()

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

In [2]:
```
import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

In [3]:
```
dataset, metadata = tfds.load('fashion_mnist', as_supervised=True, with_info=True)
train_dataset, test_dataset = dataset['train'], dataset['test']
```

```
Downloading and preparing dataset 29.45 MiB (download: 29.45 MiB, generated: 36.42
MiB, total: 65.87 MiB) 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.
```

In [4]:
```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal',      'Shirt',   'Sneaker',  'Bag',   'Ankle boot']
```

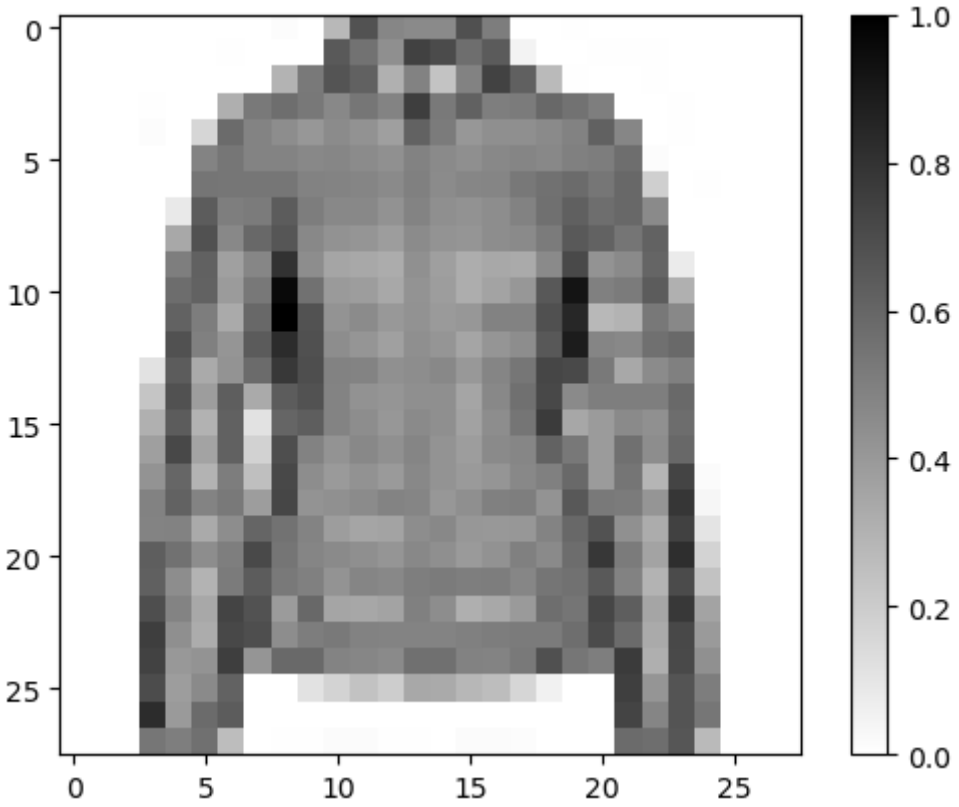
In [5]:
```
num_train_examples = metadata.splits['train'].num_examples
num_test_examples = metadata.splits['test'].num_examples
print("Number of training examples: {}".format(num_train_examples))
print("Number of test examples:     {}".format(num_test_examples))
```

```
Number of training examples: 60000

Number of test examples:     10000
```

In [6]:
```
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()
```

In [7]:
```
# Take a single image, and remove the color dimension by reshaping
for image, label in test_dataset.take(5):
  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()
```



In [8]:
```
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()
```

In [9]:
```
import tensorflow as tf

# Define ConvNet model
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)
])
```

In [10]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 128)               401536

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 421,642
Trainable params: 421,642
Non-trainable params: 0
_________________________________________________________________
```

In [11]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [12]:
```
BATCH_SIZE = 32
train_dataset = train_dataset.cache().repeat().shuffle(num_train_examples).batch(BA
test_dataset = test_dataset.cache().batch(BATCH_SIZE)
```

In [13]:
```
model.fit(train_dataset, epochs=10, steps_per_epoch=math.ceil(num_train_examples/BA
```

```
Epoch 1/10

1875/1875 [==============================] - 19s 4ms/step - loss: 0.3909 - accurac
y: 0.8583

Epoch 2/10

1875/1875 [==============================] - 7s 4ms/step - loss: 0.2527 - accurac
y: 0.9075

Epoch 3/10

1875/1875 [==============================] - 7s 4ms/step - loss: 0.2124 - accurac
y: 0.9215

Epoch 4/10

1875/1875 [==============================] - 7s 4ms/step - loss: 0.1744 - accurac
y: 0.9355

Epoch 5/10

1875/1875 [==============================] - 7s 4ms/step - loss: 0.1482 - accurac
y: 0.9458

Epoch 6/10

1875/1875 [==============================] - 7s 4ms/step - loss: 0.1295 - accurac
y: 0.9514

Epoch 7/10

1875/1875 [==============================] - 7s 4ms/step - loss: 0.1059 - accurac
y: 0.9606

Epoch 8/10

1875/1875 [==============================] - 7s 4ms/step - loss: 0.0872 - accurac
y: 0.9675

Epoch 9/10

1875/1875 [==============================] - 7s 4ms/step - loss: 0.0734 - accurac
y: 0.9726

Epoch 10/10

1875/1875 [==============================] - 7s 4ms/step - loss: 0.0614 - accurac
y: 0.9772
```

Out[13]: <keras.callbacks.History at 0x72773d19bd50>

In [14]:
```
test_loss, test_accuracy = model.evaluate(test_dataset, steps=math.ceil(num_test_e
print('Accuracy on test dataset:', test_accuracy)
```

```
313/313 [==============================] - 2s 5ms/step - loss: 0.3358 - accuracy:
0.9179

Accuracy on test dataset: 0.917900025844574
```

EXPERIMENT 4

AIM:

Design a ConvNet for classifying dog and cat images

THEORY

To classify images of dogs and cats, a CNN would typically consist of several convolutional layers followed by pooling layers, which help reduce the size of the image while retaining important features. These layers are then followed by fully connected layers, which perform the classification task. The network is trained using a large dataset of labeled images, where the weights of the layers are adjusted using backpropagation to minimize the classification error. Additionally, data augmentation techniques such as random cropping and flipping can be used to increase the size of the training dataset and improve the robustness of the model. Overall, a CNN-based approach can achieve high accuracy in classifying dog and cat images.

Importing packages

Let's start by importing required packages:

- os — to read files and directory structure
- numpy — for some matrix math outside of TensorFlow
- matplotlib.pyplot — to plot the graph and display images in our training and validation data

```
import tensorflow as tf
```

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

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

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

Data Loading

To build our image classifier, we begin by downloading the dataset. The dataset we are using is a filtered version of Dogs vs. Cats dataset from Kaggle (ultimately, this dataset is provided by Microsoft Research).

In previous Colabs, we've used TensorFlow Datasets, which is a very easy and convenient way to use datasets. In this Colab however, we will make use of the class `tf.keras.preprocessing.image.ImageDataGenerator` which will read data from disk. We therefore need to directly download *Dogs vs. Cats* from a URL and unzip it to the Colab filesystem.

```
_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=True)
```

```
Downloading data from https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
68606236/68606236 [==============================] - 0s 0us/step
```

The dataset we have downloaded has the following directory structure.

```
cats_and_dogs_filtered
|__ train
    |______ cats: [cat.0.jpg, cat.1.jpg, cat.2.jpg ...]
    |______ dogs: [dog.0.jpg, dog.1.jpg, dog.2.jpg ...]
|__ validation
```

```
|_____ cats: [cat.2000.jpg, cat.2001.jpg, cat.2002.jpg ...]
|_____ dogs: [dog.2000.jpg, dog.2001.jpg, dog.2002.jpg ...]
```

We can list the directories with the following terminal command:

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

We'll now assign variables with the proper file path for the training and validation sets.

```
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')

train_cats_dir = os.path.join(train_dir, 'cats')  # directory with our training cat pictures
train_dogs_dir = os.path.join(train_dir, 'dogs')  # directory with our training dog pictures
validation_cats_dir = os.path.join(validation_dir, 'cats')  # directory with our validation cat pictures
validation_dogs_dir = os.path.join(validation_dir, 'dogs')  # directory with our validation dog pictures
```

Understanding our data

Let's look at how many cats and dogs images we have in our training and validation directory

```
num_cats_tr = len(os.listdir(train_cats_dir))
num_dogs_tr = len(os.listdir(train_dogs_dir))

num_cats_val = len(os.listdir(validation_cats_dir))
num_dogs_val = len(os.listdir(validation_dogs_dir))

total_train = num_cats_tr + num_dogs_tr
total_val = num_cats_val + num_dogs_val
```

```
print('total training cat images:', num_cats_tr)
print('total training dog images:', num_dogs_tr)

print('total validation cat images:', num_cats_val)
print('total validation dog images:', num_dogs_val)
print("--")
print("Total training images:", total_train)
print("Total validation images:", total_val)
```

```
total training cat images: 1000
total training dog images: 1000
total validation cat images: 500
total validation dog images: 500
--
Total training images: 2000
Total validation images: 1000
```

Setting Model Parameters

For convenience, we'll set up variables that will be used later while pre-processing our dataset and training our network.

```
BATCH_SIZE = 100  # Number of training examples to process before updating our models variables
IMG_SHAPE  = 150  # Our training data consists of images with width of 150 pixels and height of 150 pixels
```

Data Preparation

Images must be formatted into appropriately pre-processed floating point tensors before being fed into the network. The steps involved in preparing these images are:

1. Read images from the disk
2. Decode contents of these images and convert it into proper grid format as per their RGB content
3. Convert them into floating point tensors
4. Rescale the tensors from values between 0 and 255 to values between 0 and 1, as neural networks prefer to deal with small input values.

Fortunately, all these tasks can be done using the class **tf.keras.preprocessing.image.ImageDataGenerator**.

We can set this up in a couple of lines of code.

```
train_image_generator      = ImageDataGenerator(rescale=1./255)  # Generator for our training data
validation_image_generator = ImageDataGenerator(rescale=1./255)  # Generator for our validation data
```

After defining our generators for training and validation images, **flow_from_directory** method will load images from the disk, apply rescaling, and resize them using single line of code.

```
train_data_gen = train_image_generator.flow_from_directory(batch_size=BATCH_SIZE,
                                                           directory=train_dir,
                                                           shuffle=True,
                                                           target_size=(IMG_SHAPE,IMG_SHAPE), #(150,150)
                                                           class_mode='binary')
```

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

```
val_data_gen = validation_image_generator.flow_from_directory(batch_size=BATCH_SIZE,
                                                              directory=validation_dir,
                                                              shuffle=False,
                                                              target_size=(IMG_SHAPE,IMG_SHAPE), #(150,150)
                                                              class_mode='binary')
```

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

Visualizing Training images

We can visualize our training images by getting a batch of images from the training generator, and then plotting a few of them using `matplotlib`.

```
sample_training_images, _ = next(train_data_gen)
```

The `next` function returns a batch from the dataset. One batch is a tuple of (*many images*, *many labels*). For right now, we're discarding the labels because we just want to look at the images.

```
# This function will plot images in the form of a grid with 1 row and 5 columns where images are placed in each column.
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()
```

```
plotImages(sample_training_images[:5])  # Plot images 0-4
```

Model Creation

Define the model

The model consists of four convolution blocks with a max pool layer in each of them. Then we have a fully connected layer with 512 units, with a `relu` activation function. The model will output class probabilities for two classes — dogs and cats — using `softmax`.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(2)
])
```

Compile the model

As usual, we will use the `adam` optimizer. Since we output a softmax categorization, we'll use `sparse_categorical_crossentropy` as the loss function. We would also like to look at training and validation accuracy on each epoch as we train our network, so we are passing in the metrics argument.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Model Summary

Let's look at all the layers of our network using **summary** method.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2D  (None, 74, 74, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 17, 17, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 15, 15, 128)       147584

 max_pooling2d_3 (MaxPooling  (None, 7, 7, 128)        0
 2D)

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 512)               3211776

 dense_1 (Dense)             (None, 2)                 1026
```

```
=================================================================
Total params: 3,453,634
Trainable params: 3,453,634
Non-trainable params: 0
_________________________________________________________________
```

Train the model

It's time we train our network.

Since our batches are coming from a generator (`ImageDataGenerator`), we'll use `fit_generator` instead of `fit` .

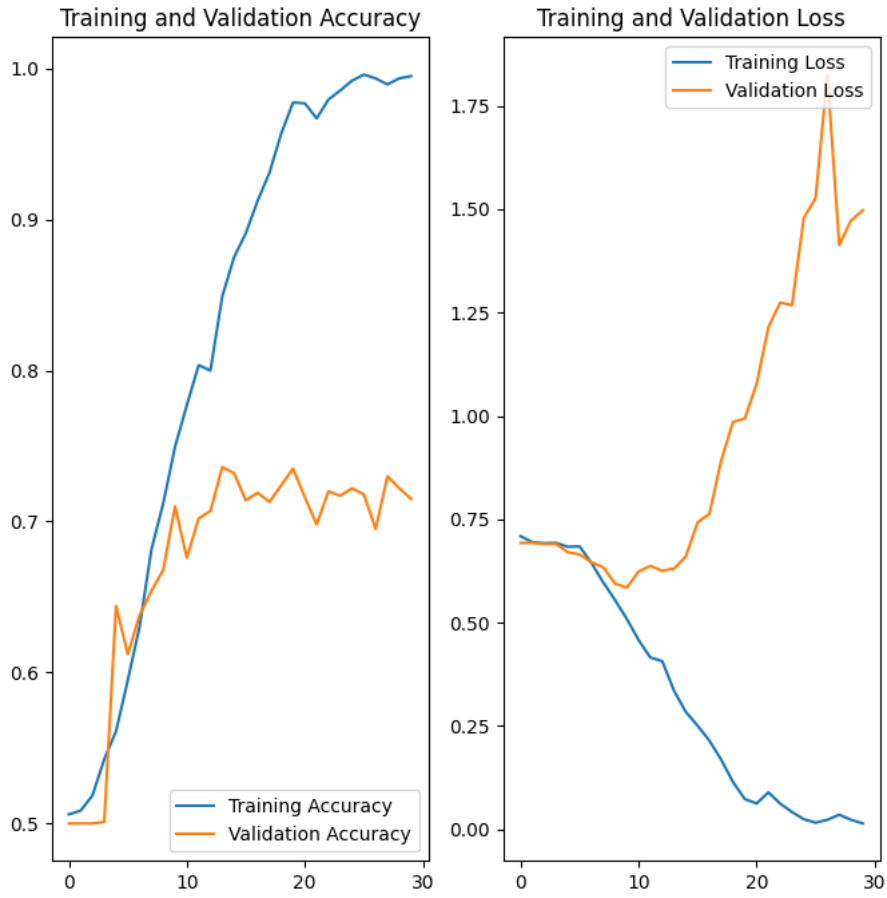
```
EPOCHS = 30
history = model.fit_generator(
    train_data_gen,
    steps_per_epoch=int(np.ceil(total_train / float(BATCH_SIZE))),
    epochs=EPOCHS,
    validation_data=val_data_gen,
    validation_steps=int(np.ceil(total_val / float(BATCH_SIZE)))
)
```

```
<ipython-input-21-3c1367ef56ee>:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Ple
  history = model.fit_generator(
Epoch 1/30
20/20 [==============================] - 26s 451ms/step - loss: 0.7093 - accuracy: 0.5060 - val_loss: 0.6929 - val_accuracy: 0.5
Epoch 2/30
20/20 [==============================] - 10s 482ms/step - loss: 0.6948 - accuracy: 0.5085 - val_loss: 0.6933 - val_accuracy: 0.5
Epoch 3/30
20/20 [==============================] - 10s 486ms/step - loss: 0.6924 - accuracy: 0.5185 - val_loss: 0.6902 - val_accuracy: 0.5
Epoch 4/30
20/20 [==============================] - 10s 518ms/step - loss: 0.6931 - accuracy: 0.5425 - val_loss: 0.6906 - val_accuracy: 0.5
Epoch 5/30
20/20 [==============================] - 9s 462ms/step - loss: 0.6837 - accuracy: 0.5610 - val_loss: 0.6708 - val_accuracy: 0.64
Epoch 6/30
20/20 [==============================] - 9s 444ms/step - loss: 0.6845 - accuracy: 0.5950 - val_loss: 0.6651 - val_accuracy: 0.61
Epoch 7/30
20/20 [==============================] - 10s 493ms/step - loss: 0.6450 - accuracy: 0.6305 - val_loss: 0.6462 - val_accuracy: 0.6
Epoch 8/30
20/20 [==============================] - 10s 487ms/step - loss: 0.5977 - accuracy: 0.6815 - val_loss: 0.6343 - val_accuracy: 0.6
Epoch 9/30
20/20 [==============================] - 10s 487ms/step - loss: 0.5551 - accuracy: 0.7120 - val_loss: 0.5946 - val_accuracy: 0.6
Epoch 10/30
20/20 [==============================] - 9s 444ms/step - loss: 0.5093 - accuracy: 0.7495 - val_loss: 0.5849 - val_accuracy: 0.71
Epoch 11/30
20/20 [==============================] - 10s 495ms/step - loss: 0.4583 - accuracy: 0.7770 - val_loss: 0.6237 - val_accuracy: 0.6
Epoch 12/30
20/20 [==============================] - 10s 487ms/step - loss: 0.4158 - accuracy: 0.8035 - val_loss: 0.6376 - val_accuracy: 0.7
Epoch 13/30
20/20 [==============================] - 10s 489ms/step - loss: 0.4072 - accuracy: 0.8000 - val_loss: 0.6255 - val_accuracy: 0.7
Epoch 14/30
20/20 [==============================] - 9s 451ms/step - loss: 0.3359 - accuracy: 0.8490 - val_loss: 0.6308 - val_accuracy: 0.73
Epoch 15/30
20/20 [==============================] - 9s 441ms/step - loss: 0.2845 - accuracy: 0.8750 - val_loss: 0.6601 - val_accuracy: 0.73
Epoch 16/30
20/20 [==============================] - 10s 488ms/step - loss: 0.2511 - accuracy: 0.8910 - val_loss: 0.7427 - val_accuracy: 0.7
Epoch 17/30
20/20 [==============================] - 10s 491ms/step - loss: 0.2149 - accuracy: 0.9125 - val_loss: 0.7635 - val_accuracy: 0.7
Epoch 18/30
20/20 [==============================] - 10s 497ms/step - loss: 0.1689 - accuracy: 0.9310 - val_loss: 0.8921 - val_accuracy: 0.7
Epoch 19/30
20/20 [==============================] - 11s 564ms/step - loss: 0.1150 - accuracy: 0.9570 - val_loss: 0.9856 - val_accuracy: 0.7
Epoch 20/30
20/20 [==============================] - 9s 465ms/step - loss: 0.0738 - accuracy: 0.9775 - val_loss: 0.9937 - val_accuracy: 0.73
Epoch 21/30
20/20 [==============================] - 10s 493ms/step - loss: 0.0629 - accuracy: 0.9770 - val_loss: 1.0774 - val_accuracy: 0.7
Epoch 22/30
20/20 [==============================] - 10s 488ms/step - loss: 0.0900 - accuracy: 0.9670 - val_loss: 1.2148 - val_accuracy: 0.6
Epoch 23/30
20/20 [==============================] - 9s 458ms/step - loss: 0.0627 - accuracy: 0.9795 - val_loss: 1.2741 - val_accuracy: 0.72
Epoch 24/30
20/20 [==============================] - 9s 468ms/step - loss: 0.0423 - accuracy: 0.9855 - val_loss: 1.2677 - val_accuracy: 0.71
Epoch 25/30
20/20 [==============================] - 10s 490ms/step - loss: 0.0250 - accuracy: 0.9920 - val_loss: 1.4789 - val_accuracy: 0.7
Epoch 26/30
20/20 [==============================] - 10s 494ms/step - loss: 0.0167 - accuracy: 0.9960 - val_loss: 1.5268 - val_accuracy: 0.7
Epoch 27/30
20/20 [==============================] - 10s 491ms/step - loss: 0.0235 - accuracy: 0.9935 - val_loss: 1.8254 - val_accuracy: 0.6
Epoch 28/30
```

Visualizing results of the training

We'll now visualize the results we get after training our network.

```
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(1, 2, 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')

plt.subplot(1, 2, 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.savefig('./foo.png')
plt.show()
```



As we can see from the plots, training accuracy and validation accuracy are off by large margin and our model has achieved only around **70%** accuracy on the validation set (depending on the number of epochs you trained for).

This is a clear indication of overfitting. Once the training and validation curves start to diverge, our model has started to memorize the training data and is unable to perform well on the validation data.

EXPERIMENT 5

AIM:

Image classifying using transfer learning for dogs and cats dataset

THEORY

A pre-trained model is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task. You either use the pretrained model as is or use transfer learning to customize this model to a given task.

The intuition behind transfer learning for image classification is that if a model is trained on a large and general enough dataset, this model will effectively serve as a generic model of the visual world. You can then take advantage of these learned feature maps without having to start from scratch by training a large model on a large dataset.

You do not need to (re)train the entire model. The base convolutional network already contains features that are generically useful for classifying pictures. However, the final, classification part of the pretrained model is specific to the original classification task, and subsequently specific to the set of classes on which the model was trained.

Importing Packages

```
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf
```

Data preprocessing

Data download

In this tutorial, you will use a dataset containing several thousand images of cats and dogs. Download and extract a zip file containing the images, then create a `tf.data.Dataset` for training and validation using the `tf.keras.utils.image_dataset_from_directory` utility. You can learn more about loading images in this tutorial.

```
_URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'
path_to_zip = tf.keras.utils.get_file('cats_and_dogs.zip', origin=_URL, extract=True)
PATH = os.path.join(os.path.dirname(path_to_zip), 'cats_and_dogs_filtered')

train_dir = os.path.join(PATH, 'train')
validation_dir = os.path.join(PATH, 'validation')

BATCH_SIZE = 32
IMG_SIZE = (160, 160)

train_dataset = tf.keras.utils.image_dataset_from_directory(train_dir,
                                                            shuffle=True,
                                                            batch_size=BATCH_SIZE,
                                                            image_size=IMG_SIZE)
```

```
Downloading data from https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
68606236/68606236 [==============================] - 5s 0us/step
Found 2000 files belonging to 2 classes.
```

```
validation_dataset = tf.keras.utils.image_dataset_from_directory(validation_dir,
                                                                 shuffle=True,
                                                                 batch_size=BATCH_SIZE,
                                                                 image_size=IMG_SIZE)
```

```
Found 1000 files belonging to 2 classes.
```

Show the first nine images and labels from the training set:

```
class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

As the original dataset doesn't contain a test set, you will create one. To do so, determine how many batches of data are available in the validation set using `tf.data.experimental.cardinality`, then move 20% of them to a test set.

```
val_batches = tf.data.experimental.cardinality(validation_dataset)
test_dataset = validation_dataset.take(val_batches // 5)
validation_dataset = validation_dataset.skip(val_batches // 5)
```

```
print('Number of validation batches: %d' % tf.data.experimental.cardinality(validation_dataset))
print('Number of test batches: %d' % tf.data.experimental.cardinality(test_dataset))
```

```
Number of validation batches: 26
Number of test batches: 6
```

Configure the dataset for performance

Use buffered prefetching to load images from disk without having I/O become blocking. To learn more about this method see the [data performance] guide.

```
AUTOTUNE = tf.data.AUTOTUNE
```

```
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

Use data augmentation

When you don't have a large image dataset, it's a good practice to artificially introduce sample diversity by applying random, yet realistic, transformations to the training images, such as rotation and horizontal flipping. This helps expose the model to different aspects of the training data and reduce overfitting. You can learn more about data augmentation in this tutorial.

```
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.2),
])
```

Note: These layers are active only during training, when you call `Model.fit`. They are inactive when the model is used in inference mode in `Model.evaluate` or `Model.fit`.

Let's repeatedly apply these layers to the same image and see the result.

```
for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tf.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```

Rescale pixel values

In a moment, you will download `tf.keras.applications.MobileNetV2` for use as your base model. This model expects pixel values in `[-1, 1]`, but at this point, the pixel values in your images are in `[0, 255]`. To rescale them, use the preprocessing method included with the model.

```
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
```

Note: Alternatively, you could rescale pixel values from `[0, 255]` to `[-1, 1]` using `tf.keras.layers.Rescaling`.

```
rescale = tf.keras.layers.Rescaling(1./127.5, offset=-1)
```

Note: If using other `tf.keras.applications`, be sure to check the API doc to determine if they expect pixels in `[-1, 1]` or `[0, 1]`, or use the included `preprocess_input` function.

Create the base model from the pre-trained convnets

You will create the base model from the **MobileNet V2** model developed at Google. This is pre-trained on the ImageNet dataset, a large dataset consisting of 1.4M images and 1000 classes. ImageNet is a research training dataset with a wide variety of categories like `jackfruit` and `syringe`. This base of knowledge will help us classify cats and dogs from our specific dataset.

First, you need to pick which layer of MobileNet V2 you will use for feature extraction. The very last classification layer (on "top", as most diagrams of machine learning models go from bottom to top) is not very useful. Instead, you will follow the common practice to depend on the very last layer before the flatten operation. This layer is called the "bottleneck layer". The bottleneck layer features retain more generality as compared to the final/top layer.

First, instantiate a MobileNet V2 model pre-loaded with weights trained on ImageNet. By specifying the **include_top=False** argument, you load a network that doesn't include the classification layers at the top, which is ideal for feature extraction.

```
# Create the base model from the pre-trained model MobileNet V2
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_orderin
9406464/9406464 [==============================] - 1s 0us/step
```

This feature extractor converts each `160x160x3` image into a `5x5x1280` block of features. Let's see what it does to an example batch of images:

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(32, 5, 5, 1280)
```

Feature extraction

In this step, you will freeze the convolutional base created from the previous step and to use as a feature extractor. Additionally, you add a classifier on top of it and train the top-level classifier.

Freeze the convolutional base

It is important to freeze the convolutional base before you compile and train the model. Freezing (by setting layer.trainable = False) prevents the weights in a given layer from being updated during training. MobileNet V2 has many layers, so setting the entire model's `trainable` flag to False will freeze all of them.

```
base_model.trainable = False
```

Important note about BatchNormalization layers

Many models contain `tf.keras.layers.BatchNormalization` layers. This layer is a special case and precautions should be taken in the context of fine-tuning, as shown later in this tutorial.

When you set `layer.trainable = False`, the `BatchNormalization` layer will run in inference mode, and will not update its mean and variance statistics.

When you unfreeze a model that contains BatchNormalization layers in order to do fine-tuning, you should keep the BatchNormalization layers in inference mode by passing `training = False` when calling the base model. Otherwise, the updates applied to the non-trainable weights will destroy what the model has learned.

For more details, see the Transfer learning guide.

```
# Let's take a look at the base model architecture
base_model.summary()
```

```
Model: "mobilenetv2_1.00_160"
_________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 160, 160, 3  0           []
                                )]

 Conv1 (Conv2D)                 (None, 80, 80, 32)   864         ['input_1[0][0]']

 bn_Conv1 (BatchNormalization)  (None, 80, 80, 32)   128         ['Conv1[0][0]']

 Conv1_relu (ReLU)              (None, 80, 80, 32)   0           ['bn_Conv1[0][0]']

 expanded_conv_depthwise (Depth  (None, 80, 80, 32)  288         ['Conv1_relu[0][0]']
 wiseConv2D)

 expanded_conv_depthwise_BN (Ba  (None, 80, 80, 32)  128         ['expanded_conv_depthwise[0][0]']
 tchNormalization)

 expanded_conv_depthwise_relu (  (None, 80, 80, 32)  0           ['expanded_conv_depthwise_BN[0][0
 ReLU)                                                           ]']

 expanded_conv_project (Conv2D)  (None, 80, 80, 16)  512         ['expanded_conv_depthwise_relu[0]
                                                                 [0]']

 expanded_conv_project_BN (Batc  (None, 80, 80, 16)  64          ['expanded_conv_project[0][0]']
 hNormalization)

 block_1_expand (Conv2D)        (None, 80, 80, 96)   1536        ['expanded_conv_project_BN[0][0]'
                                                                 ]

 block_1_expand_BN (BatchNormal  (None, 80, 80, 96)  384         ['block_1_expand[0][0]']
 ization)

 block_1_expand_relu (ReLU)     (None, 80, 80, 96)   0           ['block_1_expand_BN[0][0]']

 block_1_pad (ZeroPadding2D)    (None, 81, 81, 96)   0           ['block_1_expand_relu[0][0]']

 block_1_depthwise (DepthwiseCo  (None, 40, 40, 96)  864         ['block_1_pad[0][0]']
 nv2D)

 block_1_depthwise_BN (BatchNor  (None, 40, 40, 96)  384         ['block_1_depthwise[0][0]']
 malization)

 block_1_depthwise_relu (ReLU)  (None, 40, 40, 96)   0           ['block_1_depthwise_BN[0][0]']

 block_1_project (Conv2D)       (None, 40, 40, 24)   2304        ['block_1_depthwise_relu[0][0]']

 block_1_project_BN (BatchNorma  (None, 40, 40, 24)  96          ['block_1_project[0][0]']
 lization)

 block_2_expand (Conv2D)        (None, 40, 40, 144)  3456        ['block_1_project_BN[0][0]']

 block_2_expand_BN (BatchNormal  (None, 40, 40, 144)  576        ['block_2_expand[0][0]']
 ization)

 block_2_expand_relu (ReLU)     (None, 40, 40, 144)  0           ['block_2_expand_BN[0][0]']
```

Add a classification head

To generate predictions from the block of features, average over the spatial `5x5` spatial locations, using a `tf.keras.layers.GlobalAveragePooling2D` layer to convert the features to a single 1280-element vector per image.

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(32, 1280)
```

Apply a `tf.keras.layers.Dense` layer to convert these features into a single prediction per image. You don't need an activation function here because this prediction will be treated as a `logit`, or a raw prediction value. Positive numbers predict class 1, negative numbers predict class 0.

```
prediction_layer = tf.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(32, 1)
```

Build a model by chaining together the data augmentation, rescaling, `base_model` and feature extractor layers using the Keras Functional API. As previously mentioned, use `training=False` as our model contains a `BatchNormalization` layer.

```
inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

Compile the model

Compile the model before training it. Since there are two classes, use the `tf.keras.losses.BinaryCrossentropy` loss with `from_logits=True` since the model provides a linear output.

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 160, 160, 3)]     0

 sequential (Sequential)     (None, 160, 160, 3)       0

 tf.math.truediv (TFOpLambda  (None, 160, 160, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 160, 160, 3)      0
 a)

 mobilenetv2_1.00_160 (Funct  (None, 5, 5, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 1)                 1281

=================================================================
Total params: 2,259,265
Trainable params: 1,281
Non-trainable params: 2,257,984
_________________________________________________________________
```

The 2.5 million parameters in MobileNet are frozen, but there are 1.2 thousand *trainable* parameters in the Dense layer. These are divided between two `tf.Variable` objects, the weights and biases.

```
len(model.trainable_variables)
```

```
2
```

Train the model

After training for 10 epochs, you should see ~94% accuracy on the validation set.

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
26/26 [==============================] - 4s 47ms/step - loss: 0.5706 - accuracy: 0.6287
```

```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 0.57
initial accuracy: 0.63
```

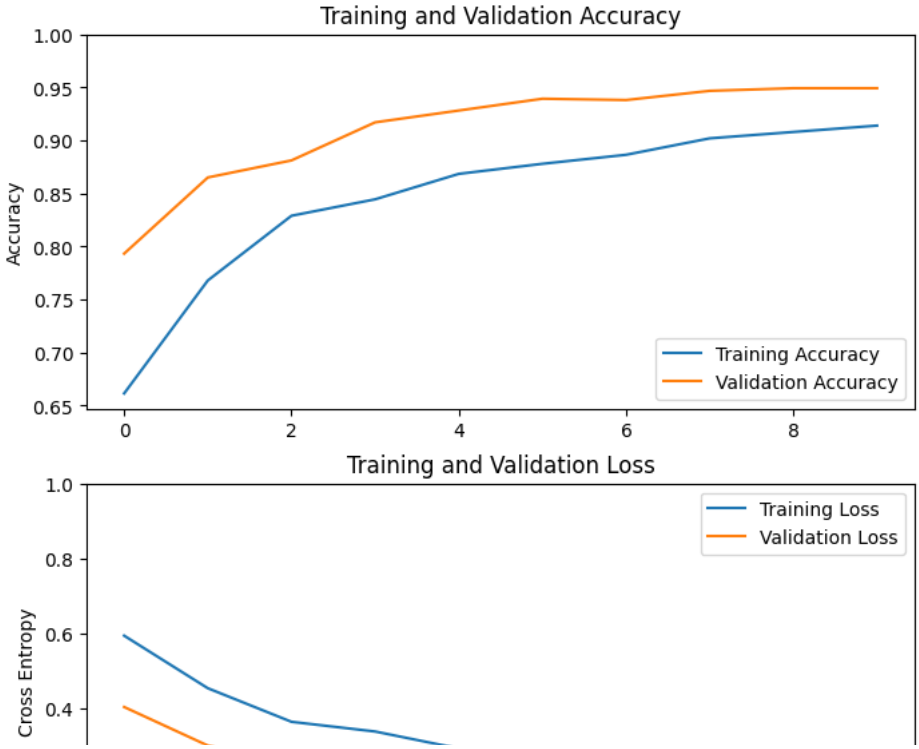
```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

```
Epoch 1/10
63/63 [==============================] - 9s 85ms/step - loss: 0.5934 - accuracy: 0.6615 - val_loss: 0.4028 - val_accuracy: 0.7933
Epoch 2/10
63/63 [==============================] - 4s 56ms/step - loss: 0.4530 - accuracy: 0.7680 - val_loss: 0.2998 - val_accuracy: 0.8651
Epoch 3/10
63/63 [==============================] - 4s 56ms/step - loss: 0.3634 - accuracy: 0.8290 - val_loss: 0.2488 - val_accuracy: 0.8812
Epoch 4/10
63/63 [==============================] - 6s 87ms/step - loss: 0.3374 - accuracy: 0.8445 - val_loss: 0.2011 - val_accuracy: 0.9171
Epoch 5/10
63/63 [==============================] - 4s 55ms/step - loss: 0.2920 - accuracy: 0.8685 - val_loss: 0.1775 - val_accuracy: 0.9282
Epoch 6/10
63/63 [==============================] - 4s 58ms/step - loss: 0.2640 - accuracy: 0.8780 - val_loss: 0.1518 - val_accuracy: 0.9394
Epoch 7/10
63/63 [==============================] - 4s 55ms/step - loss: 0.2552 - accuracy: 0.8865 - val_loss: 0.1443 - val_accuracy: 0.9381
Epoch 8/10
63/63 [==============================] - 4s 55ms/step - loss: 0.2316 - accuracy: 0.9020 - val_loss: 0.1261 - val_accuracy: 0.9468
Epoch 9/10
63/63 [==============================] - 5s 78ms/step - loss: 0.2141 - accuracy: 0.9080 - val_loss: 0.1136 - val_accuracy: 0.9493
Epoch 10/10
63/63 [==============================] - 4s 58ms/step - loss: 0.1991 - accuracy: 0.9140 - val_loss: 0.1128 - val_accuracy: 0.9493
```

Learning curves

Let's take a look at the learning curves of the training and validation accuracy/loss when using the MobileNetV2 base model as a fixed feature extractor.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Note: If you are wondering why the validation metrics are clearly better than the training metrics, the main factor is because layers like `tf.keras.layers.BatchNormalization` and `tf.keras.layers.Dropout` affect accuracy during training. They are turned off when calculating validation loss.

To a lesser extent, it is also because training metrics report the average for an epoch, while validation metrics are evaluated after the epoch, so validation metrics see a model that has trained slightly longer.

Summary

Using a pre-trained model for feature extraction: When working with a small dataset, it is a common practice to take advantage of features learned by a model trained on a larger dataset in the same domain. This is done by instantiating the pre-trained model and adding a fully-connected classifier on top. The pre-trained model is "frozen" and only the weights of the classifier get updated during training. In this case, the convolutional base extracted all the features associated with each image and you just trained a classifier that determines the image class given that set of extracted features.

EXPERIMENT 6

AIM:

Implementation of inflated 3D CNN for action recognition

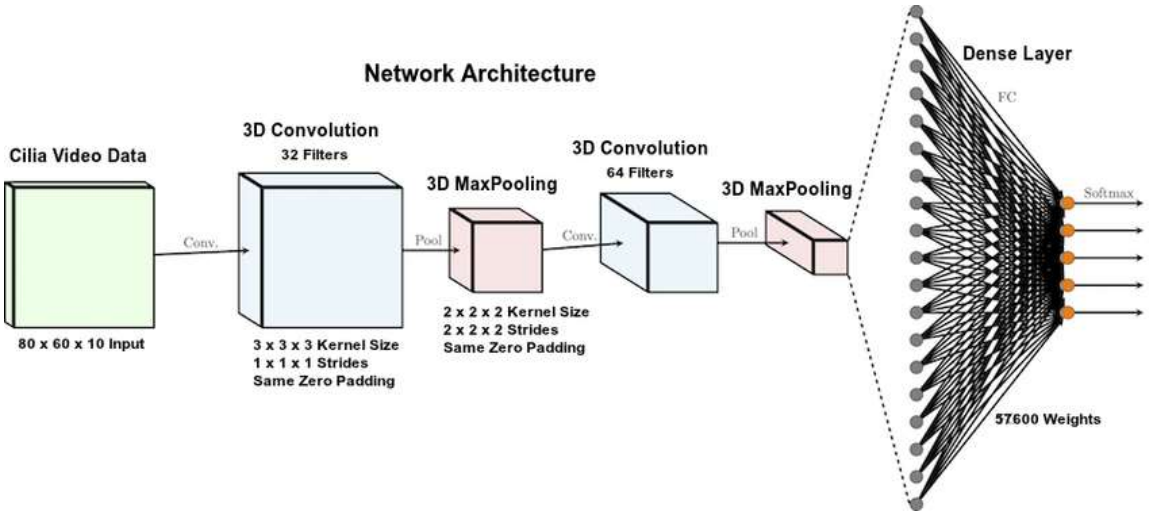
THEORY AND SOURCE CODE

Inflated 3D CNN (Convolutional Neural Network) is a deep learning architecture used for video recognition tasks. This model uses a pre-trained 2D CNN on large-scale image classification datasets such as ImageNet, and then "inflates" it to work with 3D convolutional layers to extract spatio-temporal features from video data.

The basic idea behind Inflated 3D CNN is that the first few layers of the 2D CNN can be directly reused in the 3D CNN, since the lower-level image features learned in the 2D CNN are also useful for video data. The 2D CNN is then "inflated" to 3D by replicating the learned 2D filters across the time dimension to create 3D filters.

This approach has several advantages. First, using pre-trained 2D CNN weights saves time and computational resources, since training a 3D CNN from scratch can be time-consuming and require a large amount of data. Second, inflating the 2D CNN to 3D allows the model to learn spatio-temporal features that are specific to video data, which can improve the accuracy of the model for video recognition tasks.

Inflated 3D CNN has been used in various applications such as action recognition, human pose estimation, and gesture recognition. However, it is worth noting that this approach may not be optimal for all video recognition tasks, as it assumes that the lower-level features learned in the 2D CNN are useful for the specific video dataset at hand. Therefore, careful consideration should be given to the choice of pre-trained 2D CNN and the specific video dataset being used.



In [ ]:
```
import imageio
import logging
from IPython import display
import os
import random
import re
```

In [ ]:
```
import tensorflow as tf
import tensorflow_hub as hub
from tensorflow_docs.vis import embed
import tempfile
import ssl
import numpy as np
```

In [ ]:
```
from IPython import display
from absl import logging
import imageio
import cv2
```

In [ ]:
```
from urllib import request
from urllib.request import *
from urllib.error import *
```

In [ ]:
```
UCF_URL = "https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/"
VIDEO_LIST = None
```

In [ ]:
```
CACHE_DIR = tempfile.mkdtemp()
unverified = ssl._create_unverified_context()
```

In [ ]:
```
def list_ucf_videos():
   global VIDEO_LIST
   if not VIDEO_LIST:
     index = request.urlopen(UCF_URL, context=unverified).read().decode("utf-8")
     videos = re.findall("(v_[\w_]+\.avi)", index)
     VIDEO_LIST = sorted(set(videos))
   return list(VIDEO_LIST)
```

In [ ]:
```
def fetch(video):
   '''
   condition for existing video
   '''
   path = os.path.join(CACHE_DIR, video)
   '''
   for a new video, define a ,
   path using requests, url
   '''
   if not os.path.exists(path):
     url_path = request.urljoin(UCF_URL, video)
     print("Fetching %s => %s" % (url_path, path))
     data = request.urlopen(url_path, context=unverified).read()
     '''
     writing all of this into the file
     '''
```

```
    open(path, "wb").write(data)
  return path
```

In [ ]:

```
def crop_center(frame):
  '''
  frame shape
  '''
  y, x = frame.shape[0:2]
  min_dimension = min(y, x)
  '''
  setting start points for
  both the dimensions
  '''
  starting_x = (x // 2) - (min_dimension // 2)
  starting_y = (y // 2) - (min_dimension // 2)
  '''
  returning limits to dimensions
  '''
  return frame[starting_y:starting_y+min_dimension,starting_x:starting_x+min_dimen

def load(path, max_frames=0, resize=(224, 224)):
  '''
  variable to capture paths
  '''
  cap = cv2.VideoCapture(path)
  frames = []
  try:
    while True:
      ret, frame = cap.read()
      if not ret:
        break
      '''
      applying all above mentioned functions
      video processing
      '''
      frame = crop_center(frame)
      frame = cv2.resize(frame, resize)
      frame = frame[:, :, [2, 1, 0]]
      frames.append(frame)
      if len(frames) == max_frames:
        break
  finally:
    cap.release()
  '''
  dividing by 255 to get values
  b/w 0-1
  '''
  return np.array(frames) / 255.0

def gif(images):
  '''
  cliping the images for gif
  '''
  converted = np.clip(images * 255, 0, 255).astype(np.uint8)
  '''
  save gif of 25 frames
```

```
  '''
  imageio.mimsave('./animation.gif', converted, fps=25)
  return embed.embed_file('./animation.gif')
```

In [ ]:
```
# Get the kinetics-400 action labels from the GitHub repository.
KINETICS_URL = "https://raw.githubusercontent.com/deepmind/kinetics-i3d/master/data
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.
```

In [ ]:
```
videos = list_ucf_videos()
'''
 empty dict. for storing
'''
categories = {}
'''
 running loop on videos
 from above function called
 '''
for video in videos:
  cat = video[2:-12]
  '''
  if not present add the video
  '''
  if cat not in categories:
    categories[cat] = []
  categories[cat].append(video)
  '''
  string formatting for showing output
  easily
  '''
print("Found %d videos in %d categories." % (len(videos), len(categories)))
for cat, seq in categories.items():
  '''
  join is used to remove the seperator and
  concatenate the objects
  '''
  summary = ", ".join(seq[:2])
  print("%-20s %4d videos (%s, ...)" % (categories, len(seq), summary))
```

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

In [ ]:
```
import urllib.request
import cv2
```

In [ ]:
```
path = fetch('v_CricketShot_g04_c02.avi')
sample_video = load(path)
```

```
Fetching https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/v_CricketShot_g04_c02.avi =
> C:\Users\Shivansh\AppData\Local\Temp\tmpt34slp3v\v_CricketShot_g04_c02.avi
```

In [ ]:
```
gif(sample_video)
```

Out[ ]:

In [ ]:

```
def pred(sample_video):
    '''
    Model input
    '''
    model_input = tf.constant(sample_video, dtype=tf.float32)[tf.newaxis, ...]
    '''
    saving logits and probabilities of each
    prediction
    '''
    log = model(model_input)['default'][0]
    prob = tf.nn.softmax(log)
    print("Printing Top 5 actions:")
    for i in np.argsort(prob)[::-1][:5]:
      print(f"  {labels[i]:22}: {prob[i] * 100:5.2f}%")
'''
 calling the function for
 prediction
'''
pred(sample_video)
```

```
Printing Top 5 actions:
  playing cricket       : 97.77%
  skateboarding         :  0.71%
  robot dancing         :  0.56%
  roller skating        :  0.56%
  golf putting          :  0.13%
```

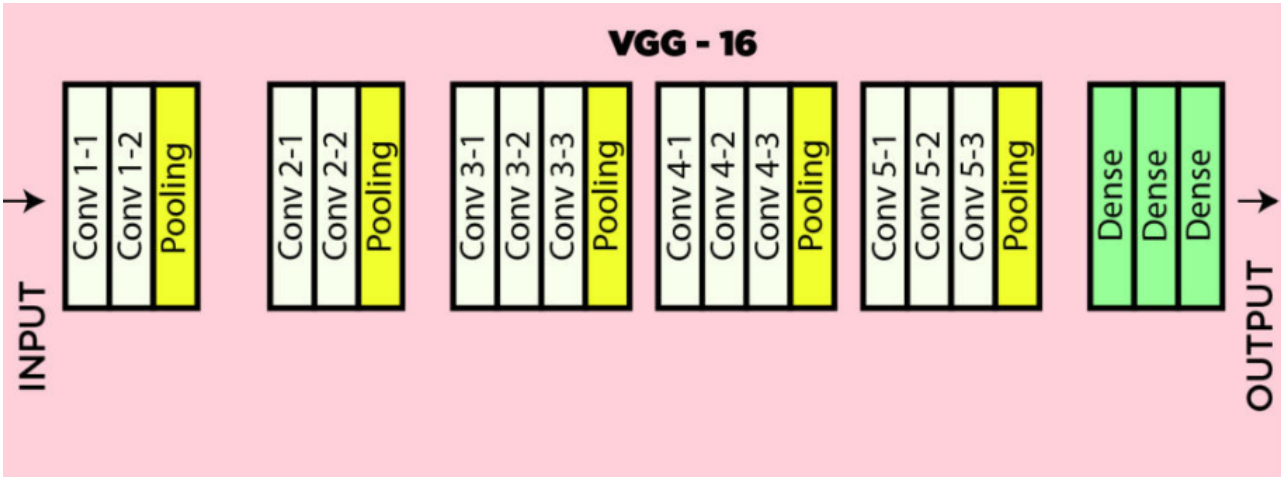
EXPERIMENT 7

AIM:

Object Recognition Using CNN

THEORY AND SOURCE CODE:

In this practical, the VGG16 model has been trained on pre-trained weights on ImageNet for the task of object detection.To perform localization,bounding box location coordinates are used. A bounding box location is represented by the 4-D vector (center coordinates(x,y), height, width).



In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call d
rive.mount("/content/drive", force_remount=True).
```

In [2]:
```
import os
base_path="/content/drive/MyDrive/SEM 6 (DEEP L)"
images=os.path.sep.join([base_path,'images'])
annotations=os.path.sep.join([base_path,'airplanes.csv'])
```

In [3]:
```
# Loading the Dataset

# Airplanes annotation is a Csv file thats why we can see through with rows
rows= open(annotations).read().strip().split("\n")

# Make three list where we save our exact bounding boxes
data=[]
targets=[]
filenames=[]
```

In [4]:
```
# split dataset according to images

# import libraries
import cv2
from tensorflow.keras.preprocessing.image import load_img
# we also save images into array format so, import img_array library too
from tensorflow.keras.preprocessing.image import img_to_array

for row in rows:
  row=row.split(",")
  # we always create rectangle with h + w so we have to know where exactly we sho
  (filename,startX,startY,endX,endY)=row

  imagepaths=os.path.sep.join([images,filename])
  image=cv2.imread(imagepaths)
  (h,w)=image.shape[:2]

  # initializing starting point

  # Take float values for the ease of conversion into array
  startX = float(startX) / w
  startY = float(startY) / h
  # initialize ending point
  endX = float(endX) / w
  endY = float(endY) / h

  #load image and give them default size
  image=load_img(imagepaths,target_size=(224,224))
  # see here if we cant take it into float then we face trouble
  image=img_to_array(image)

  # Lets append into data , targets ,filenames
  targets.append((startX,startY,endX,endY))
  filenames.append(filename)
  data.append(image)
```

In [5]:
```
# Normalizing Data
import numpy as np
data=np.array(data,dtype='float32') / 255.0
targets=np.array(targets,dtype='float32')
```

In [6]:
```
# we should seperate data into train and split so import sklearn library
from sklearn.model_selection import train_test_split
```

In [7]:
```
# split into testing and training
split=train_test_split(data,targets,filenames,test_size=0.10,random_state=42)
```

In [8]:
```
# lets split into steps
(train_images,test_images) = split[:2]
(train_targets,test_targets) = split[2:4]
(train_filenames,test_filenames) = split[4:]
```

In [9]:
```
# lets import pre trained VGG16 : A builtin model for computer vision
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Input
```

In [10]:
```
# so here we just want limited layers so included_top is set as false

vgg=VGG16(weights='imagenet',include_top=False,input_tensor=Input(shape=(224,224,
```

In [11]:
```
vgg.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

In [12]:
```
from tensorflow.keras.layers import Input,Flatten,Dense
```

In [13]:
```
# we use VGG16 as per requirement
vgg.trainable = False

flatten = vgg.output

flatten = Flatten()(flatten)
```

In [14]:
```
# Lets make bboxhead
bboxhead = Dense(128,activation="relu")(flatten)
bboxhead = Dense(64,activation="relu")(bboxhead)
bboxhead = Dense(32,activation="relu")(bboxhead)
bboxhead = Dense(4,activation="relu")(bboxhead)
```

In [15]:
```
# Lets import Model
from tensorflow.keras.models import Model
model = Model(inputs = vgg.input,outputs = bboxhead)
```

In [16]:
```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168
```

In [17]:
```
# Fit the model

# Optimization
from tensorflow.keras.optimizers import Adam

opt = Adam(1e-4)
```

In [18]:
```
model.compile(loss='mse',optimizer=opt)
```

In [28]:
```
history = model.fit(train_images,train_targets,validation_data=(test_images,test_
```

```
Epoch 1/50
23/23 [==============================] - 20s 257ms/step - loss: 0.0562 - val
_loss: 0.0081
Epoch 2/50
23/23 [==============================] - 3s 131ms/step - loss: 0.0053 - val_
loss: 0.0036
Epoch 3/50
23/23 [==============================] - 3s 132ms/step - loss: 0.0028 - val_
loss: 0.0031
Epoch 4/50
23/23 [==============================] - 3s 132ms/step - loss: 0.0019 - val_
loss: 0.0028
Epoch 5/50
23/23 [==============================] - 3s 136ms/step - loss: 0.0014 - val_
loss: 0.0026
Epoch 6/50
23/23 [==============================] - 3s 132ms/step - loss: 0.0011 - val_
loss: 0.0025
Epoch 7/50
```

In [33]:
```
# lets save model
model.save('/content/drive/MyDrive/SEM 6 (DEEP L)/detect_Planes.h5')
```

In [19]:
```
from tensorflow.keras.models import load_model
```

In [20]:
```
model=load_model('/content/drive/MyDrive/SEM 6 (DEEP L)/detect_Planes.h5')
```

In [21]:
```
imagepath='/content/drive/MyDrive/SEM 6 (DEEP L)/images/image_0700.jpg'
```

In [22]:
```
image = load_img(imagepath,
                 target_size=(224,224))
image = img_to_array(image) / 255.0
image = np.expand_dims(image,axis=0)
```

In [23]:
```
preds=model.predict(image)[0]
(startX,startY,endX,endY)=preds
```

```
1/1 [==============================] - 3s 3s/step
```

In [24]:
```
import imutils
```

In [25]:
```
image=cv2.imread(imagepaths)
image=imutils.resize(image,width=600)
```

In [26]:
```
(h,w)=image.shape[:2]
```

In [27]:
```
startX=int(startX * w)
startY=int(startY * h)

endX=int(endX * w)
endY=int(endY * h)
```

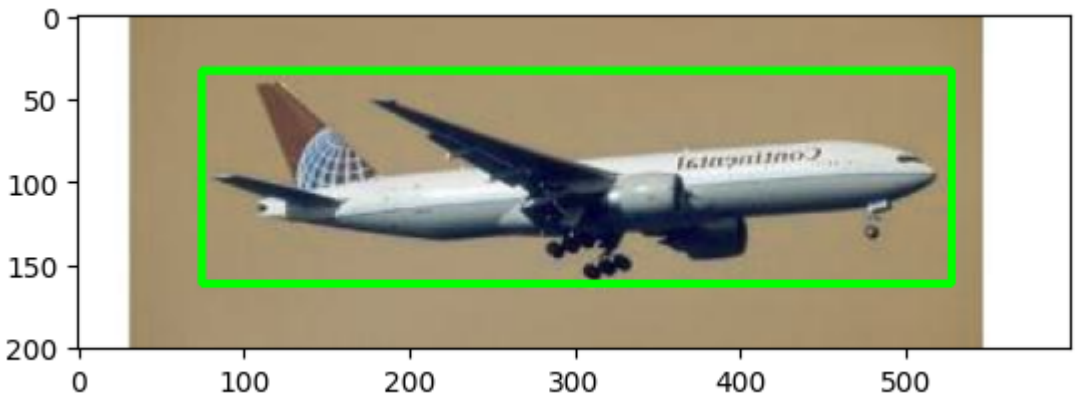
In [28]:
```
cv2.rectangle(image,(startX,startY),(endX,endY),(0,255,0),3)
```

Out[28]:
```
array([[[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
```

In [29]:
```
from google.colab.patches import cv2_imshow
```

In [30]:
```
import matplotlib.pyplot as plt
plt.imshow(image)
cv2.waitKey(0)
```

Out[30]:
```
-1
```



EXPERIMENT 8

AIM:

Genrating Images using BigGAN

THEORY AND SOURCE CODE

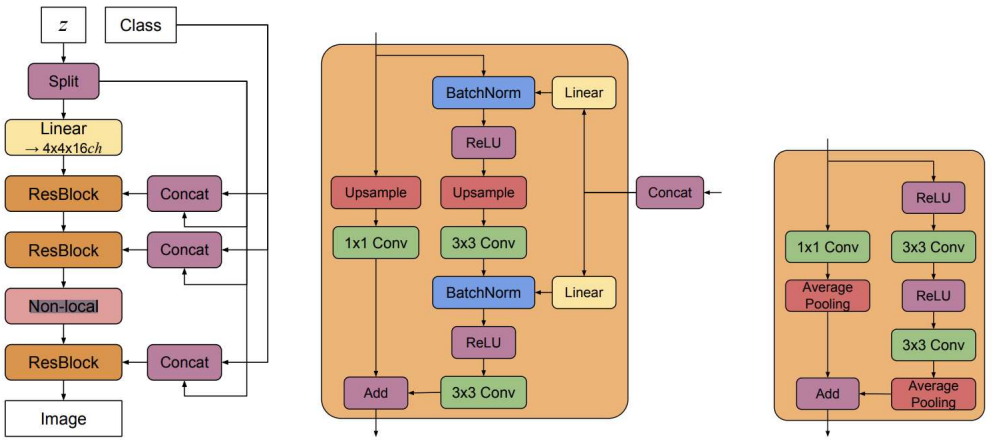
BigGAN is a generative adversarial network (GAN) architecture that was introduced in 2018 by researchers at Google Brain. It is considered one of the most powerful and high-resolution GAN models to date, capable of generating large and diverse images with high fidelity.

The key feature of BigGAN is its use of a large number of parameters, which allows it to capture fine-grained details and produce images at high resolution. The model has over 11 million parameters, which is much larger than previous GAN models such as DCGAN and ProGAN.

BigGAN also uses a novel technique called "class-conditional GAN," where the generator is conditioned on a specific class label in addition to random noise. This allows the model to generate images of specific objects or categories, such as different breeds of dogs, without having to train a separate model for each class.

To train BigGAN, the researchers used a distributed training approach, where multiple GPUs were used to accelerate the training process. This allowed them to train the model on large-scale image datasets such as ImageNet, which contains millions of images.

The results of BigGAN are impressive, with the model being able to generate high-quality images of various objects and scenes, including animals, landscapes, and human faces. The images produced by BigGAN are often indistinguishable from real photographs, and can even surpass the quality of images produced by professional photographers in some cases.



In [ ]:
```
# BigGAN-deep models
# module_path = 'https://tfhub.dev/deepmind/biggan-deep-128/1'  # 128x128 BigGAN-de
module_path = 'https://tfhub.dev/deepmind/biggan-deep-256/1'  # 256x256 BigGAN-deep
# module_path = 'https://tfhub.dev/deepmind/biggan-deep-512/1'  # 512x512 BigGAN-de

# BigGAN (original) models
# module_path = 'https://tfhub.dev/deepmind/biggan-128/2'  # 128x128 BigGAN
# module_path = 'https://tfhub.dev/deepmind/biggan-256/2'  # 256x256 BigGAN
# module_path = 'https://tfhub.dev/deepmind/biggan-512/2'  # 512x512 BigGAN
```

Setup

In [ ]:
```
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 c:\Users\Shivansh\AppData\Local\Programs\Python\Python311\Lib\site-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 future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

Load a BigGAN generator module from TF Hub

In [ ]:
```
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
INFO:tensorflow:Saver not created because there are no variables in the graph to restore
INFO:tensorflow:Saver not created because there are no variables in the graph to restore
```

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

Output: Tensor("module_apply_default/G_trunc_output:0", shape=(?, 256, 256, 3), dtyp
e=float32)
```

Define some functions for sampling and displaying BigGAN images

In [ ]:

```
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

def 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
```

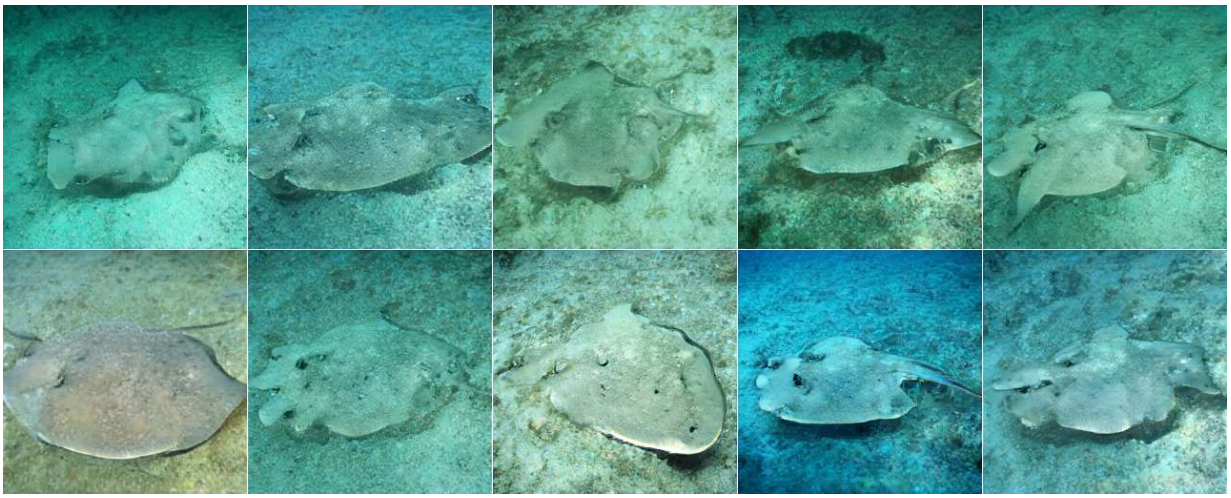
Create a TensorFlow session and initialize variables

In [ ]:
```
initializer = tf.global_variables_initializer()
sess = tf.Session()
sess.run(initializer)
```

Explore BigGAN samples of a particular category

Try varying the `truncation` value.

In [ ]:
```
#@title Category-conditional sampling { display-mode: "form", run: "auto" }

num_samples = 10 #@param {type:"slider", min:1, max:20, step:1}
truncation = 0.4 #@param {type:"slider", min:0.02, max:1, step:0.02}
noise_seed = 0 #@param {type:"slider", min:0, max:100, step:1}
category = "5) electric ray, crampfish, numbfish, torpedo" #@param ["0) tench, Tinc

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)))
```



Interpolate between BigGAN samples

Try setting different `category`s with the same `noise_seed`s, or the same `category`s with different `noise_seed`s. Or go wild and set both any way you like!

In [ ]:
```
#@title Interpolation { display-mode: "form", run: "auto" }

num_samples = 2 #@param {type:"slider", min:1, max:5, step:1}
num_interps = 5 #@param {type:"slider", min:2, max:10, step:1}
truncation = 0.2 #@param {type:"slider", min:0.02, max:1, step:0.02}
noise_seed_A = 0 #@param {type:"slider", min:0, max:100, step:1}
```

```
category_A = "14) indigo bunting, indigo finch, indigo bird, Passerina cyanea" #@pa
noise_seed_B = 0 #@param {type:"slider", min:0, max:100, step:1}
category_B = "16) bulbul" #@param ["0) tench, Tinca tinca", "1) goldfish, Carassius

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))
```

EXPERIMENT 9

AIM:

Implement Transformer Network for translating language

THEORY AND SOURCE CODE:

In this practical, a sequence-to-sequence Transformer model is created and trained to translate Portuguese into English.

Transformers are deep neural networks that replace CNNs and RNNs with self-attention. Self attention allows Transformers to easily transmit information across the input sequences.Neural networks for machine translation typically contain an encoder reading the input sentence and generating a representation of it. A decoder then generates the output sentence word by word while consulting the representation generated by the encoder.

The Transformer starts by generating initial representations, or embeddings, for each word, Then, using self-attention, it aggregates information from all of the other words, generating a new representation per word informed by the entire context. This step is then repeated multiple times in parallel for all words, successively generating new representations.

Significance of Transformers

- Transformers excel at modeling sequential data, such as natural language.
- Unlike the recurrent neural networks (RNNs), Transformers are parallelizable. This makes them efficient on hardware like GPUs and TPUs. The main reasons is that Transformers replaced recurrence with attention, and computations can happen simultaneously. Layer outputs can be computed in parallel, instead of a series like an RNN.
- Unlike RNNs(like seq2seq) or convolutional neural networks (CNNs),for example, ByteNet), Transformers are able to capture distant or long-range contexts and dependencies in the data between distant positions in the input or output sequences. Thus, longer connections can be learned. Attention allows each location to have access to the entire input at each layer, while in RNNs and CNNs, the information needs to pass through many processing steps to move a long distance, which makes it harder to learn.
- Transformers make no assumptions about the temporal/spatial relationships across the data. This is ideal for processing a set of objects (for example, StarCraft units).

Setup

In [2]:

```
# Install the most re version of TensorFlow to use the improved
# masking support for `tf.keras.layers.MultiHeadAttention`.
!apt install --allow-change-held-packages libcudnn8=8.1.0.77-1+cuda11.2
!pip uninstall -y -q tensorflow keras tensorflow-estimator tensorflow-text
!pip install protobuf~=3.20.3
!pip install -q tensorflow_datasets
!pip install -q -U tensorflow-text tensorflow
```

```
E: Could not open lock file /var/lib/dpkg/lock-frontend - open (13: Permis
sion denied)
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend),
are you root?
WARNING: Skipping tensorflow-text as it is not installed.
Requirement already satisfied: protobuf~=3.20.3 in /tmpfs/src/tf_docs_env/
lib/python3.9/site-packages (3.20.3)
```

In [3]:

```
import logging
import time

import numpy as np
import matplotlib.pyplot as plt

import tensorflow_datasets as tfds
import tensorflow as tf

import tensorflow_text
```

Data handling

Download the dataset

In [4]:

```
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 [5]:

```
for pt_examples, en_examples in train_examples.batch(3).take(1):
  print('> Examples in Portuguese:')
  for pt in pt_examples.numpy():
    print(pt.decode('utf-8'))
  print()

  print('> Examples in English:')
  for en in en_examples.numpy():
    print(en.decode('utf-8'))
```

```
> Examples in Portuguese:
e quando melhoramos a procura , tiramos a única vantagem da impressão , qu
e é a serendipidade .
mas e se estes fatores fossem ativos ?
mas eles não tinham a curiosidade de me testar .

> Examples in English:
and when you improve searchability , you actually take away the one advant
age of print , which is serendipity .
but what if it were active ?
but they did n't test for curiosity .
```

Set up the tokenizer

Now that you have loaded the dataset, you need to tokenize the text, so that each element is represented as a token or token ID (a numeric representation).

Tokenization is the process of breaking up text, into "tokens". Depending on the tokenizer, these tokens can represent sentence-pieces, words, subwords, or characters.

In [6]:

```
model_name = 'ted_hrlr_translate_pt_en_converter'
tf.keras.utils.get_file(
    f'{model_name}.zip',
    f'https://storage.googleapis.com/download.tensorflow.org/models/{model_name}.zip',
    cache_dir='.', cache_subdir='', extract=True
)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.o
rg/models/ted_hrlr_translate_pt_en_converter.zip (https://storage.googleap
is.com/download.tensorflow.org/models/ted_hrlr_translate_pt_en_converter.z
ip)
184801/184801 [==============================] - 0s 0us/step
```

Out[6]:

```
'./ted_hrlr_translate_pt_en_converter.zip'
```

In [7]:

```
tokenizers = tf.saved_model.load(model_name)
```

In [8]:

```
[item for item in dir(tokenizers.en) if not item.startswith('_')]
```

Out[8]:

```
['detokenize',
 'get_reserved_tokens',
 'get_vocab_path',
 'get_vocab_size',
 'lookup',
 'tokenize',
 'tokenizer',
 'vocab']
```

In [9]:

```
print('> This is a batch of strings:')
for en in en_examples.numpy():
  print(en.decode('utf-8'))
```

```
> This is a batch of strings:
and when you improve searchability , you actually take away the one advant
age of print , which is serendipity .
but what if it were active ?
but they did n't test for curiosity .
```

In [10]:

```
encoded = tokenizers.en.tokenize(en_examples)

print('> This is a padded-batch of token IDs:')
for row in encoded.to_list():
  print(row)
```

```
> This is a padded-batch of token IDs:
[2, 72, 117, 79, 1259, 1491, 2362, 13, 79, 150, 184, 311, 71, 103, 2308, 7
4, 2679, 13, 148, 80, 55, 4840, 1434, 2423, 540, 15, 3]
[2, 87, 90, 107, 76, 129, 1852, 30, 3]
[2, 87, 83, 149, 50, 9, 56, 664, 85, 2512, 15, 3]
```

In [11]:

```
round_trip = tokenizers.en.detokenize(encoded)

print('> This is human-readable text:')
for line in round_trip.numpy():
  print(line.decode('utf-8'))
```

```
> This is human-readable text:
and when you improve searchability , you actually take away the one advant
age of print , which is serendipity .
but what if it were active ?
but they did n ' t test for curiosity .
```

In [12]:

```
print('> This is the text split into tokens:')
tokens = tokenizers.en.lookup(encoded)
tokens
```

```
> This is the text split into tokens:
```

Out[12]:

```
<tf.RaggedTensor [[b'[START]', b'and', b'when', b'you', b'improve', b'sear
ch', b'##ability',
  b',', b'you', b'actually', b'take', b'away', b'the', b'one', b'advantag
e',
  b'of', b'print', b',', b'which', b'is', b's', b'##ere', b'##nd', b'##i
p',
  b'##ity', b'.', b'[END]']
,
 [b'[START]', b'but', b'what', b'if', b'it', b'were', b'active', b'?',
  b'[END]']                                                          ,
 [b'[START]', b'but', b'they', b'did', b'n', b"'", b't', b'test', b'for',
  b'curiosity', b'.', b'[END]']                                       ]
>
```

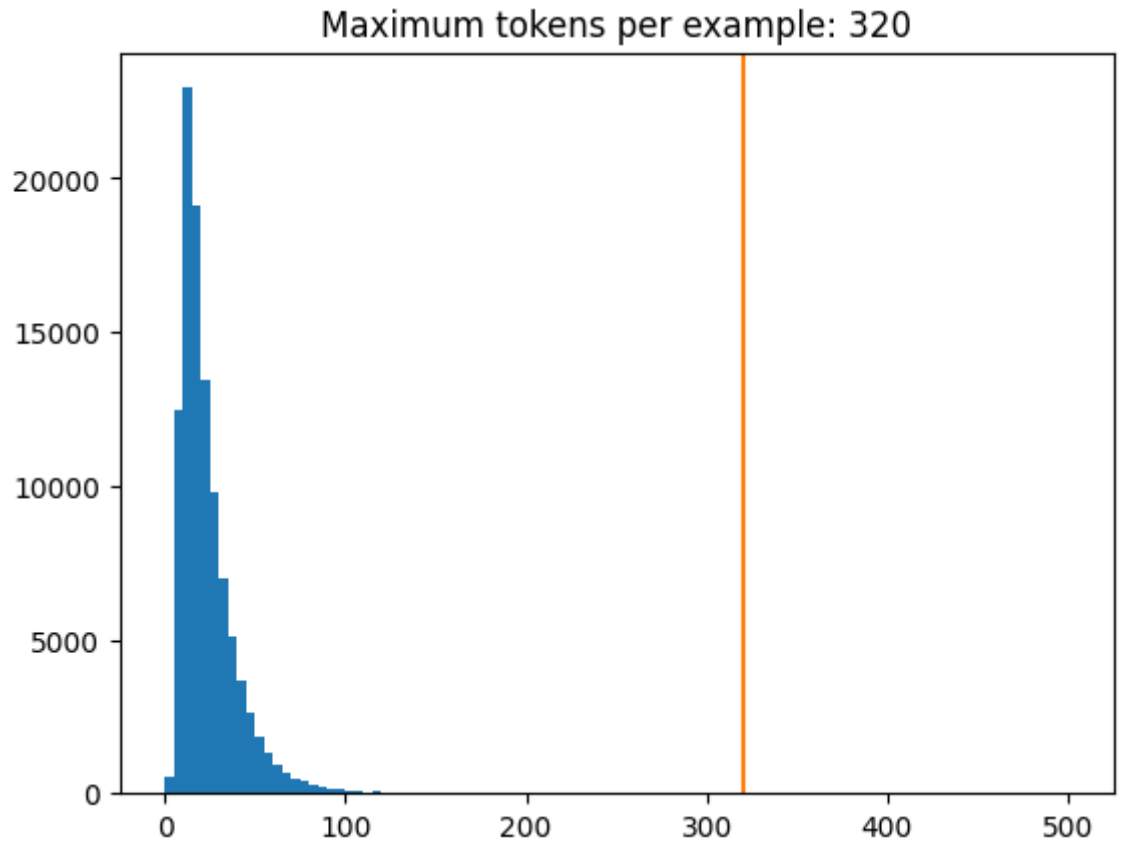
In [13]:

```
lengths = []

for pt_examples, en_examples in train_examples.batch(1024):
  pt_tokens = tokenizers.pt.tokenize(pt_examples)
  lengths.append(pt_tokens.row_lengths())

  en_tokens = tokenizers.en.tokenize(en_examples)
  lengths.append(en_tokens.row_lengths())
  print('.', end='', flush=True)
```

```
..................................................
```

In [14]:

```
all_lengths = np.concatenate(lengths)

plt.hist(all_lengths, np.linspace(0, 500, 101))
plt.ylim(plt.ylim())
max_length = max(all_lengths)
plt.plot([max_length, max_length], plt.ylim())
plt.title(f'Maximum tokens per example: {max_length}');
```



Set up a data pipeline with `tf.data`

In [15]:

```
MAX_TOKENS=128
def prepare_batch(pt, en):
    pt = tokenizers.pt.tokenize(pt)      # Output is ragged.
    pt = pt[:, :MAX_TOKENS]    # Trim to MAX_TOKENS.
    pt = pt.to_tensor()  # Convert to 0-padded dense Tensor

    en = tokenizers.en.tokenize(en)
    en = en[:, :(MAX_TOKENS+1)]
    en_inputs = en[:, :-1].to_tensor()  # Drop the [END] tokens
    en_labels = en[:, 1:].to_tensor()   # Drop the [START] tokens

    return (pt, en_inputs), en_labels
```

In [16]:

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

In [17]:

```
def make_batches(ds):
  return (
      ds
      .shuffle(BUFFER_SIZE)
      .batch(BATCH_SIZE)
      .map(prepare_batch, tf.data.AUTOTUNE)
      .prefetch(buffer_size=tf.data.AUTOTUNE))
```

Test the Dataset

In [18]:

```
# Create training and validation set batches.
train_batches = make_batches(train_examples)
val_batches = make_batches(val_examples)
```

In [19]:

```
for (pt, en), en_labels in train_batches.take(1):
  break

print(pt.shape)
print(en.shape)
print(en_labels.shape)
```

```
(64, 86)
(64, 81)
(64, 81)
```

In [20]:

```
print(en[0][:10])
print(en_labels[0][:10])
```

```
tf.Tensor([   2  476 2569 2626 6010   52 2564 1915  188   15], shape=(1
0,), dtype=int64)
tf.Tensor([ 476 2569 2626 6010   52 2564 1915  188   15    3], shape=(1
0,), dtype=int64)
```

Define the components

The embedding and positional encoding layer

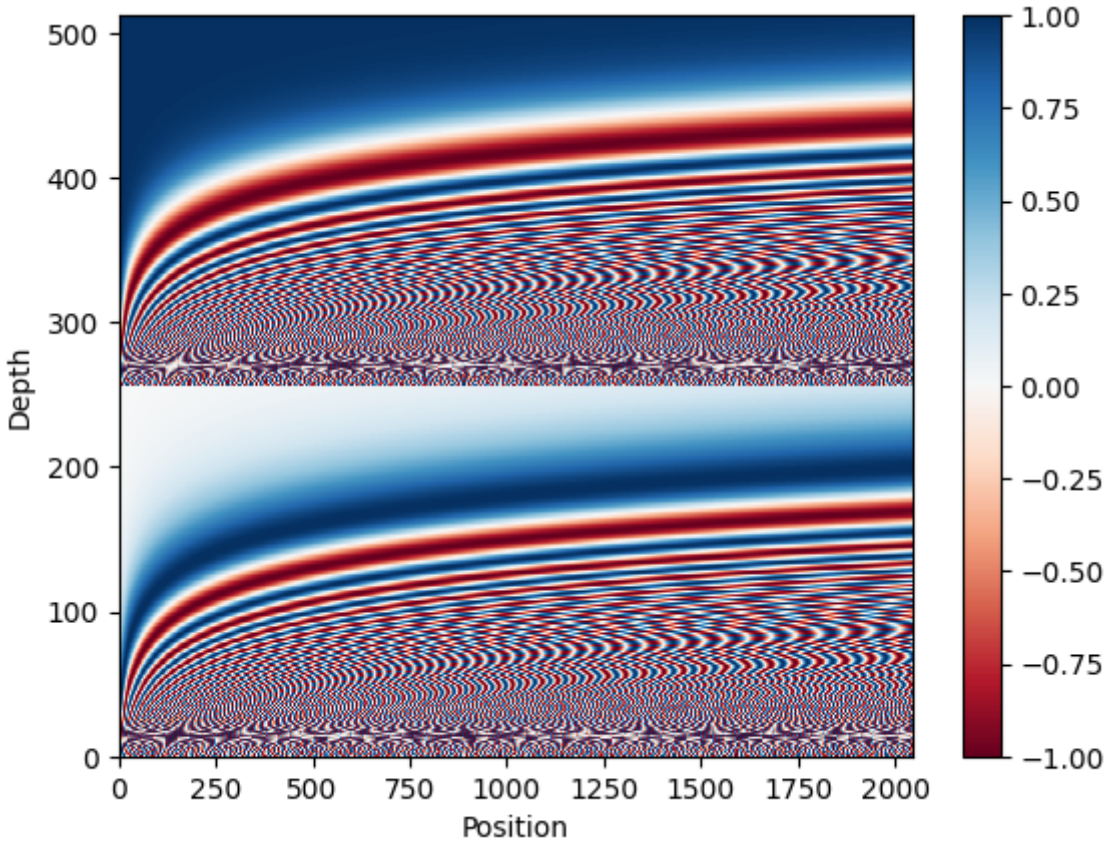
In [21]:

```
def positional_encoding(length, depth):
  depth = depth/2

  positions = np.arange(length)[:, np.newaxis]     # (seq, 1)
  depths = np.arange(depth)[np.newaxis, :]/depth   # (1, depth)

  angle_rates = 1 / (10000**depths)         # (1, depth)
  angle_rads = positions * angle_rates      # (pos, depth)

  pos_encoding = np.concatenate(
      [np.sin(angle_rads), np.cos(angle_rads)],
      axis=-1)

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

In [22]:

```
#@title
pos_encoding = positional_encoding(length=2048, depth=512)

# Check the shape.
print(pos_encoding.shape)

# Plot the dimensions.
plt.pcolormesh(pos_encoding.numpy().T, cmap='RdBu')
plt.ylabel('Depth')
plt.xlabel('Position')
plt.colorbar()
plt.show()
```

```
(2048, 512)
```

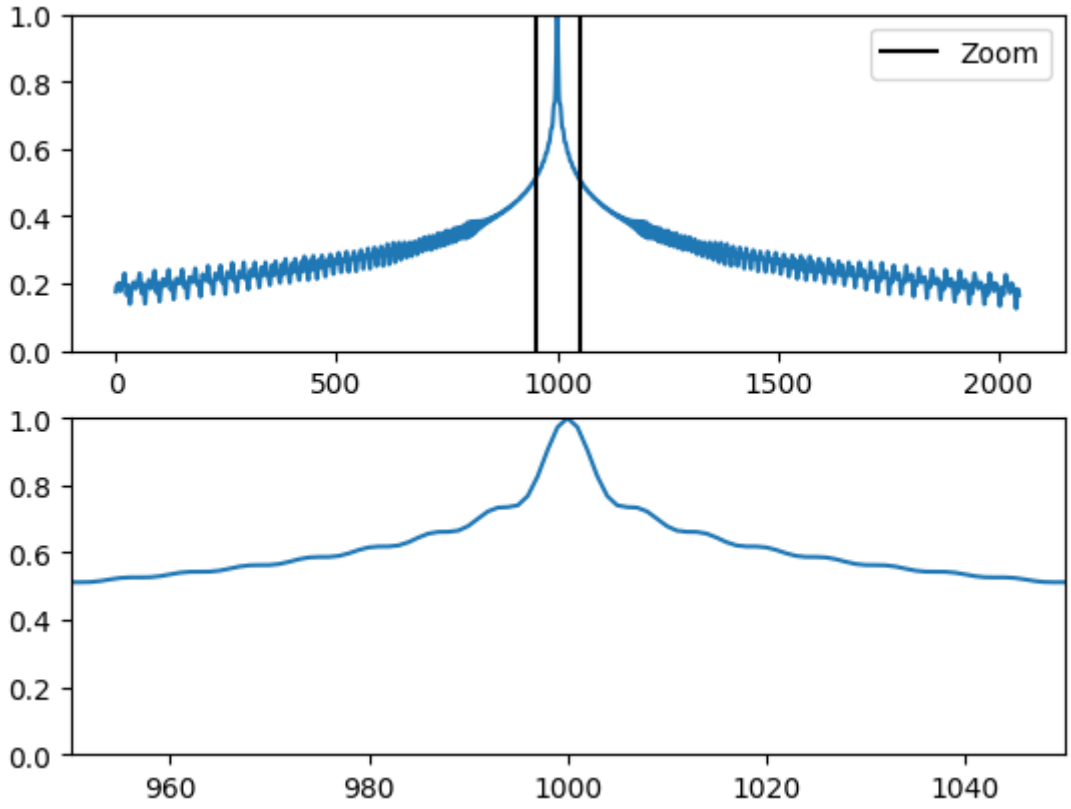


In [23]:

```
#@title
pos_encoding/=tf.norm(pos_encoding, axis=1, keepdims=True)
p = pos_encoding[1000]
dots = tf.einsum('pd,d -> p', pos_encoding, p)
plt.subplot(2,1,1)
plt.plot(dots)
plt.ylim([0,1])
plt.plot([950, 950, float('nan'), 1050, 1050],
         [0,1,float('nan'),0,1], color='k', label='Zoom')
plt.legend()
plt.subplot(2,1,2)
plt.plot(dots)
plt.xlim([950, 1050])
plt.ylim([0,1])
```

Out[23]:

```
(0.0, 1.0)
```



In [24]:

```
class PositionalEmbedding(tf.keras.layers.Layer):
  def __init__(self, vocab_size, d_model):
    super().__init__()
    self.d_model = d_model
    self.embedding = tf.keras.layers.Embedding(vocab_size, d_model, mask_zero=True)
    self.pos_encoding = positional_encoding(length=2048, depth=d_model)

  def compute_mask(self, *args, **kwargs):
    return self.embedding.compute_mask(*args, **kwargs)

  def call(self, x):
    length = tf.shape(x)[1]
    x = self.embedding(x)
    # This factor sets the relative scale of the embedding and positonal_encoding.
    x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
    x = x + self.pos_encoding[tf.newaxis, :length, :]
    return x
```

In [25]:

```
embed_pt = PositionalEmbedding(vocab_size=tokenizers.pt.get_vocab_size(), d_model=512)
embed_en = PositionalEmbedding(vocab_size=tokenizers.en.get_vocab_size(), d_model=512)

pt_emb = embed_pt(pt)
en_emb = embed_en(en)
```

In [26]:

```
en_emb._keras_mask
```

Out[26]:

```
<tf.Tensor: shape=(64, 81), dtype=bool, numpy=
array([[ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False],
       ...,
       [ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False]])>
```

Add and normalize

The base attention layer

In [27]:

```
class BaseAttention(tf.keras.layers.Layer):
  def __init__(self, **kwargs):
    super().__init__()
    self.mha = tf.keras.layers.MultiHeadAttention(**kwargs)
    self.layernorm = tf.keras.layers.LayerNormalization()
    self.add = tf.keras.layers.Add()
```

The cross attention layer

In [28]:

```
class CrossAttention(BaseAttention):
  def call(self, x, context):
    attn_output, attn_scores = self.mha(
        query=x,
        key=context,
        value=context,
        return_attention_scores=True)

    # Cache the attention scores for plotting later.
    self.last_attn_scores = attn_scores

    x = self.add([x, attn_output])
    x = self.layernorm(x)

    return x
```

In [29]:

```
sample_ca = CrossAttention(num_heads=2, key_dim=512)

print(pt_emb.shape)
print(en_emb.shape)
print(sample_ca(en_emb, pt_emb).shape)
```

```
(64, 86, 512)
(64, 81, 512)
(64, 81, 512)
```

The global self attention layer

In [30]:

```
class GlobalSelfAttention(BaseAttention):
  def call(self, x):
    attn_output = self.mha(
        query=x,
        value=x,
        key=x)
    x = self.add([x, attn_output])
    x = self.layernorm(x)
    return x
```

In [31]:

```
sample_gsa = GlobalSelfAttention(num_heads=2, key_dim=512)

print(pt_emb.shape)
print(sample_gsa(pt_emb).shape)
```

```
(64, 86, 512)
(64, 86, 512)
```

The causal self attention layer

In [32]:

```
class CausalSelfAttention(BaseAttention):
  def call(self, x):
    attn_output = self.mha(
        query=x,
        value=x,
        key=x,
        use_causal_mask = True)
    x = self.add([x, attn_output])
    x = self.layernorm(x)
    return x
```

In [33]:

```
sample_csa = CausalSelfAttention(num_heads=2, key_dim=512)

print(en_emb.shape)
print(sample_csa(en_emb).shape)
```

```
(64, 81, 512)
(64, 81, 512)
```

In [34]:

```
out1 = sample_csa(embed_en(en[:, :3]))
out2 = sample_csa(embed_en(en))[:, :3]

tf.reduce_max(abs(out1 - out2)).numpy()
```

Out[34]:

```
4.7683716e-07
```

The feed forward network

In [35]:

```
class FeedForward(tf.keras.layers.Layer):
  def __init__(self, d_model, dff, dropout_rate=0.1):
    super().__init__()
    self.seq = tf.keras.Sequential([
      tf.keras.layers.Dense(dff, activation='relu'),
      tf.keras.layers.Dense(d_model),
      tf.keras.layers.Dropout(dropout_rate)
    ])
    self.add = tf.keras.layers.Add()
    self.layer_norm = tf.keras.layers.LayerNormalization()

  def call(self, x):
    x = self.add([x, self.seq(x)])
    x = self.layer_norm(x)
    return x
```

In [36]:

```
sample_ffn = FeedForward(512, 2048)

print(en_emb.shape)
print(sample_ffn(en_emb).shape)
```

```
(64, 81, 512)
(64, 81, 512)
```

The encoder layer

In [37]:

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

    self.self_attention = GlobalSelfAttention(
        num_heads=num_heads,
        key_dim=d_model,
        dropout=dropout_rate)

    self.ffn = FeedForward(d_model, dff)

  def call(self, x):
    x = self.self_attention(x)
    x = self.ffn(x)
    return x
```

In [38]:

```
sample_encoder_layer = EncoderLayer(d_model=512, num_heads=8, dff=2048)

print(pt_emb.shape)
print(sample_encoder_layer(pt_emb).shape)
```

```
(64, 86, 512)
(64, 86, 512)
```

The encoder

In [39]:

```
class Encoder(tf.keras.layers.Layer):
  def __init__(self, *, num_layers, d_model, num_heads,
               dff, vocab_size, dropout_rate=0.1):
    super().__init__()

    self.d_model = d_model
    self.num_layers = num_layers

    self.pos_embedding = PositionalEmbedding(
        vocab_size=vocab_size, d_model=d_model)

    self.enc_layers = [
        EncoderLayer(d_model=d_model,
                     num_heads=num_heads,
                     dff=dff,
                     dropout_rate=dropout_rate)
        for _ in range(num_layers)]
    self.dropout = tf.keras.layers.Dropout(dropout_rate)

  def call(self, x):
    # `x` is token-IDs shape: (batch, seq_len)
    x = self.pos_embedding(x)  # Shape `(batch_size, seq_len, d_model)`.

    # Add dropout.
    x = self.dropout(x)

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

    return x  # Shape `(batch_size, seq_len, d_model)`.
```

In [40]:

```
# Instantiate the encoder.
sample_encoder = Encoder(num_layers=4,
                         d_model=512,
                         num_heads=8,
                         dff=2048,
                         vocab_size=8500)

sample_encoder_output = sample_encoder(pt, training=False)

# Print the shape.
print(pt.shape)
print(sample_encoder_output.shape)  # Shape `(batch_size, input_seq_len, d_model)`.
```

```
(64, 86)
(64, 86, 512)
```

The decoder layer

In [41]:

```
class DecoderLayer(tf.keras.layers.Layer):
  def __init__(self,
               *,
               d_model,
               num_heads,
               dff,
               dropout_rate=0.1):
    super(DecoderLayer, self).__init__()

    self.causal_self_attention = CausalSelfAttention(
        num_heads=num_heads,
        key_dim=d_model,
        dropout=dropout_rate)

    self.cross_attention = CrossAttention(
        num_heads=num_heads,
        key_dim=d_model,
        dropout=dropout_rate)

    self.ffn = FeedForward(d_model, dff)

  def call(self, x, context):
    x = self.causal_self_attention(x=x)
    x = self.cross_attention(x=x, context=context)

    # Cache the last attention scores for plotting later
    self.last_attn_scores = self.cross_attention.last_attn_scores

    x = self.ffn(x)  # Shape `(batch_size, seq_len, d_model)`.
    return x
```

In [42]:

```
sample_decoder_layer = DecoderLayer(d_model=512, num_heads=8, dff=2048)

sample_decoder_layer_output = sample_decoder_layer(
    x=en_emb, context=pt_emb)

print(en_emb.shape)
print(pt_emb.shape)
print(sample_decoder_layer_output.shape)  # `(batch_size, seq_len, d_model)`
```

```
(64, 81, 512)
(64, 86, 512)
(64, 81, 512)
```

The decoder

In [43]:

```
class Decoder(tf.keras.layers.Layer):
  def __init__(self, *, num_layers, d_model, num_heads, dff, vocab_size,
               dropout_rate=0.1):
    super(Decoder, self).__init__()

    self.d_model = d_model
    self.num_layers = num_layers

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

    self.last_attn_scores = None

  def call(self, x, context):
    # `x` is token-IDs shape (batch, target_seq_len)
    x = self.pos_embedding(x)  # (batch_size, target_seq_len, d_model)

    x = self.dropout(x)

    for i in range(self.num_layers):
      x  = self.dec_layers[i](x, context)

    self.last_attn_scores = self.dec_layers[-1].last_attn_scores

    # The shape of x is (batch_size, target_seq_len, d_model).
    return x
```

In [44]:

```
# Instantiate the decoder.
sample_decoder = Decoder(num_layers=4,
                         d_model=512,
                         num_heads=8,
                         dff=2048,
                         vocab_size=8000)

output = sample_decoder(
    x=en,
    context=pt_emb)

# Print the shapes.
print(en.shape)
print(pt_emb.shape)
print(output.shape)
```

```
(64, 81)
(64, 86, 512)
(64, 81, 512)
```

In [45]:

```
sample_decoder.last_attn_scores.shape  # (batch, heads, target_seq, input_seq)
```

Out[45]:

```
TensorShape([64, 8, 81, 86])
```

The Transformer

In [46]:

```
class Transformer(tf.keras.Model):
  def __init__(self, *, num_layers, d_model, num_heads, dff,
               input_vocab_size, target_vocab_size, dropout_rate=0.1):
    super().__init__()
    self.encoder = Encoder(num_layers=num_layers, d_model=d_model,
                           num_heads=num_heads, dff=dff,
                           vocab_size=input_vocab_size,
                           dropout_rate=dropout_rate)

    self.decoder = Decoder(num_layers=num_layers, d_model=d_model,
                           num_heads=num_heads, dff=dff,
                           vocab_size=target_vocab_size,
                           dropout_rate=dropout_rate)

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

  def call(self, inputs):
    # To use a Keras model with `.fit` you must pass all your inputs in the
    # first argument.
    context, x  = inputs

    context = self.encoder(context)  # (batch_size, context_len, d_model)

    x = self.decoder(x, context)  # (batch_size, target_len, d_model)

    # Final linear layer output.
    logits = self.final_layer(x)  # (batch_size, target_len, target_vocab_size)

    try:
      # Drop the keras mask, so it doesn't scale the losses/metrics.
      # b/250038731
      del logits._keras_mask
    except AttributeError:
      pass

    # Return the final output and the attention weights.
    return logits
```

Hyperparameters

In [47]:

```
num_layers = 4
d_model = 128
dff = 512
num_heads = 8
dropout_rate = 0.1
```

Try it out

In [48]:

```
transformer = Transformer(
    num_layers=num_layers,
    d_model=d_model,
    num_heads=num_heads,
    dff=dff,
    input_vocab_size=tokenizers.pt.get_vocab_size().numpy(),
    target_vocab_size=tokenizers.en.get_vocab_size().numpy(),
    dropout_rate=dropout_rate)
```

In [49]:

```
output = transformer((pt, en))

print(en.shape)
print(pt.shape)
print(output.shape)
```

```
(64, 81)
(64, 86)
(64, 81, 7010)
```

In [50]:

```
attn_scores = transformer.decoder.dec_layers[-1].last_attn_scores
print(attn_scores.shape)  # (batch, heads, target_seq, input_seq)
```

```
(64, 8, 81, 86)
```

In [51]:

```
transformer.summary()
```

```
Model: "transformer"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder_1 (Encoder)         multiple                  3632768

 decoder_1 (Decoder)         multiple                  5647104

 dense_38 (Dense)            multiple                  904290

=================================================================
Total params: 10,184,162
Trainable params: 10,184,162
Non-trainable params: 0
_________________________________________________________________
```

Training

Set up the optimizer

In [52]:

```
class CustomSchedule(tf.keras.optimizers.schedules.LearningRateSchedule):
  def __init__(self, d_model, warmup_steps=4000):
    super().__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):
    step = tf.cast(step, dtype=tf.float32)
    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 [53]:

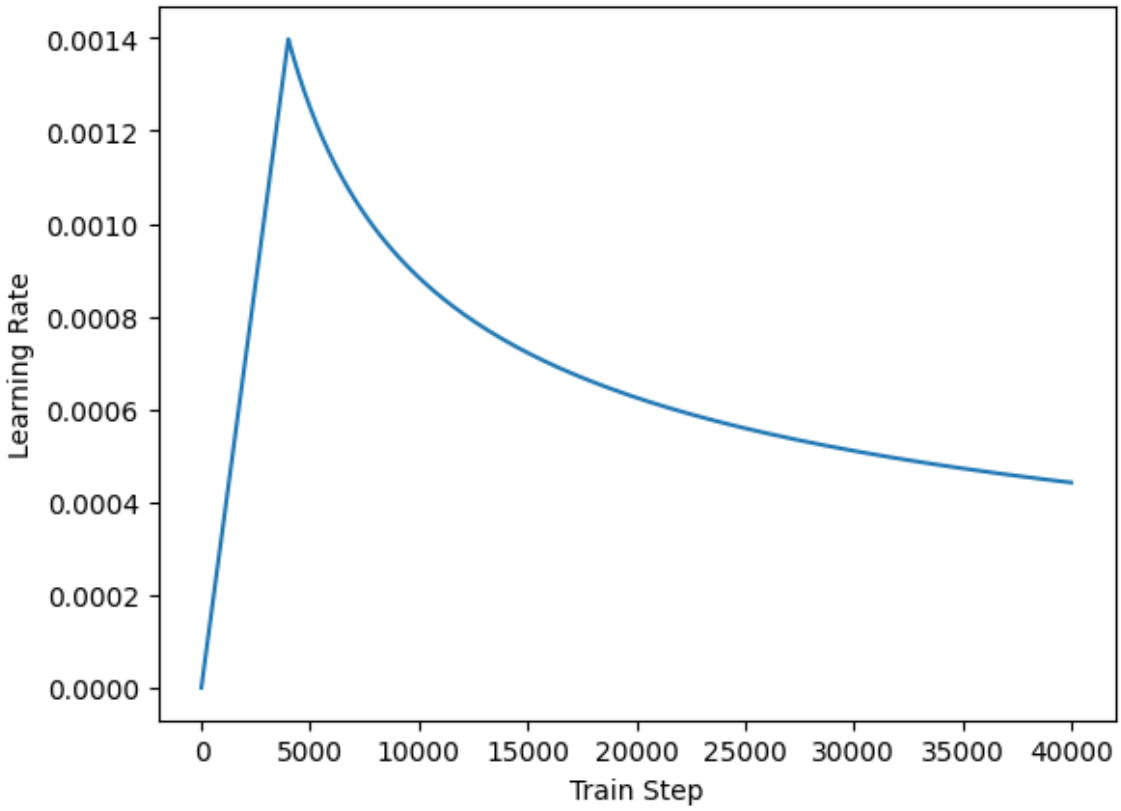
```
learning_rate = CustomSchedule(d_model)

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

In [54]:

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

Out[54]:

```
Text(0.5, 0, 'Train Step')
```



Set up the loss and metrics

In [55]:

```
def masked_loss(label, pred):
  mask = label != 0
  loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')
  loss = loss_object(label, pred)

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

  loss = tf.reduce_sum(loss)/tf.reduce_sum(mask)
  return loss


def masked_accuracy(label, pred):
  pred = tf.argmax(pred, axis=2)
  label = tf.cast(label, pred.dtype)
  match = label == pred

  mask = label != 0

  match = match & mask

  match = tf.cast(match, dtype=tf.float32)
  mask = tf.cast(mask, dtype=tf.float32)
  return tf.reduce_sum(match)/tf.reduce_sum(mask)
```

Train the model

In [56]:

```
transformer.compile(
    loss=masked_loss,
    optimizer=optimizer,
    metrics=[masked_accuracy])
```

In [57]:

```
transformer.fit(train_batches,
                epochs=20,
                validation_data=val_batches)
```

```
Epoch 1/20
810/810 [==============================] - 221s 235ms/step - loss: 6.60
41 - masked_accuracy: 0.1364 - val_loss: 5.0234 - val_masked_accuracy:
0.2487
Epoch 2/20
810/810 [==============================] - 143s 176ms/step - loss: 4.57
21 - masked_accuracy: 0.2972 - val_loss: 4.0413 - val_masked_accuracy:
0.3598
Epoch 3/20
810/810 [==============================] - 142s 174ms/step - loss: 3.82
24 - masked_accuracy: 0.3801 - val_loss: 3.4644 - val_masked_accuracy:
0.4299
Epoch 4/20
810/810 [==============================] - 138s 170ms/step - loss: 3.28
67 - masked_accuracy: 0.4391 - val_loss: 3.0112 - val_masked_accuracy:
0.4856
Epoch 5/20
810/810 [==============================] - 138s 169ms/step - loss: 2.89
53 - masked_accuracy: 0.4836 - val_loss: 2.7537 - val_masked_accuracy:
```

Run inference

In [58]:

```
class Translator(tf.Module):
  def __init__(self, tokenizers, transformer):
    self.tokenizers = tokenizers
    self.transformer = transformer

  def __call__(self, sentence, max_length=MAX_TOKENS):
    # The input sentence is Portuguese, hence adding the `[START]` and `[END]` tokens.
    assert isinstance(sentence, tf.Tensor)
    if len(sentence.shape) == 0:
      sentence = sentence[tf.newaxis]

    sentence = self.tokenizers.pt.tokenize(sentence).to_tensor()

    encoder_input = sentence

    # As the output language is English, initialize the output with the
    # English `[START]` token.
    start_end = self.tokenizers.en.tokenize([''])[0]
    start = start_end[0][tf.newaxis]
    end = start_end[1][tf.newaxis]

    # `tf.TensorArray` is required here (instead of a Python list), so that the
    # dynamic-loop can be traced by `tf.function`.
    output_array = tf.TensorArray(dtype=tf.int64, size=0, dynamic_size=True)
    output_array = output_array.write(0, start)

    for i in tf.range(max_length):
      output = tf.transpose(output_array.stack())
      predictions = self.transformer([encoder_input, output], training=False)

      # Select the last token from the `seq_len` dimension.
      predictions = predictions[:, -1:, :]  # Shape `(batch_size, 1, vocab_size)`.

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

      # Concatenate the `predicted_id` to the output which is given to the
      # decoder as its input.
      output_array = output_array.write(i+1, predicted_id[0])

      if predicted_id == end:
        break

    output = tf.transpose(output_array.stack())
    # The output shape is `(1, tokens)`.
    text = tokenizers.en.detokenize(output)[0]  # Shape: `()`.

    tokens = tokenizers.en.lookup(output)[0]

    # `tf.function` prevents us from using the attention_weights that were
    # calculated on the last iteration of the loop.
    # So, recalculate them outside the loop.
    self.transformer([encoder_input, output[:,:-1]], training=False)
    attention_weights = self.transformer.decoder.last_attn_scores

    return text, tokens, attention_weights
```

In [59]:

```
translator = Translator(tokenizers, transformer)
```

In [60]:

```
def print_translation(sentence, tokens, ground_truth):
  print(f'{"Input:":15s}: {sentence}')
  print(f'{"Prediction":15s}: {tokens.numpy().decode("utf-8")}')
  print(f'{"Ground truth":15s}: {ground_truth}')
```

Example 1:

In [61]:

```
sentence = 'este é um problema que temos que resolver.'
ground_truth = 'this is a problem we have to solve .'

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:         : este é um problema que temos que resolver.
Prediction     : this is a problem that we have to solve .
Ground truth   : this is a problem we have to solve .
```

Example 2:

In [62]:

```
sentence = 'os meus vizinhos ouviram sobre esta ideia.'
ground_truth = 'and my neighboring homes heard about this idea .'

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:         : os meus vizinhos ouviram sobre esta ideia.
Prediction     : my neighbors have heard this idea .
Ground truth   : and my neighboring homes heard about this idea .
```

Example 3:

In [63]:

```
sentence = 'vou então muito rapidamente partilhar convosco algumas histórias de algumas
ground_truth = "so i'll just share with you some stories very quickly of some magical th

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:         : vou então muito rapidamente partilhar convosco algumas hi
stórias de algumas coisas mágicas que aconteceram.
Prediction     : so i ' m going to share with you some magic stories that
will happen to happen .
Ground truth   : so i'll just share with you some stories very quickly of
some magical things that have happened.
```

Create attention plots

In [64]:

```
sentence = 'este é o primeiro livro que eu fiz.'
ground_truth = "this is the first book i've ever done."

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:         : este é o primeiro livro que eu fiz.
Prediction     : this is the first book i did .
Ground truth   : this is the first book i've ever done.
```

In [65]:

```
def plot_attention_head(in_tokens, translated_tokens, attention):
  # The model didn't generate `<START>` in the output. Skip it.
  translated_tokens = translated_tokens[1:]

  ax = plt.gca()
  ax.matshow(attention)
  ax.set_xticks(range(len(in_tokens)))
  ax.set_yticks(range(len(translated_tokens)))

  labels = [label.decode('utf-8') for label in in_tokens.numpy()]
  ax.set_xticklabels(
      labels, rotation=90)

  labels = [label.decode('utf-8') for label in translated_tokens.numpy()]
  ax.set_yticklabels(labels)
```

In [66]:

```
head = 0
# Shape: `(batch=1, num_heads, seq_len_q, seq_len_k)`.
attention_heads = tf.squeeze(attention_weights, 0)
attention = attention_heads[head]
attention.shape
```

Out[66]:

```
TensorShape([9, 11])
```

In [67]:

```
in_tokens = tf.convert_to_tensor([sentence])
in_tokens = tokenizers.pt.tokenize(in_tokens).to_tensor()
in_tokens = tokenizers.pt.lookup(in_tokens)[0]
in_tokens
```

Out[67]:

```
<tf.Tensor: shape=(11,), dtype=string, numpy=
array([b'[START]', b'este', b'e', b'o', b'primeiro', b'livro', b'que',
       b'eu', b'fiz', b'.', b'[END]'], dtype=object)>
```

In [68]:

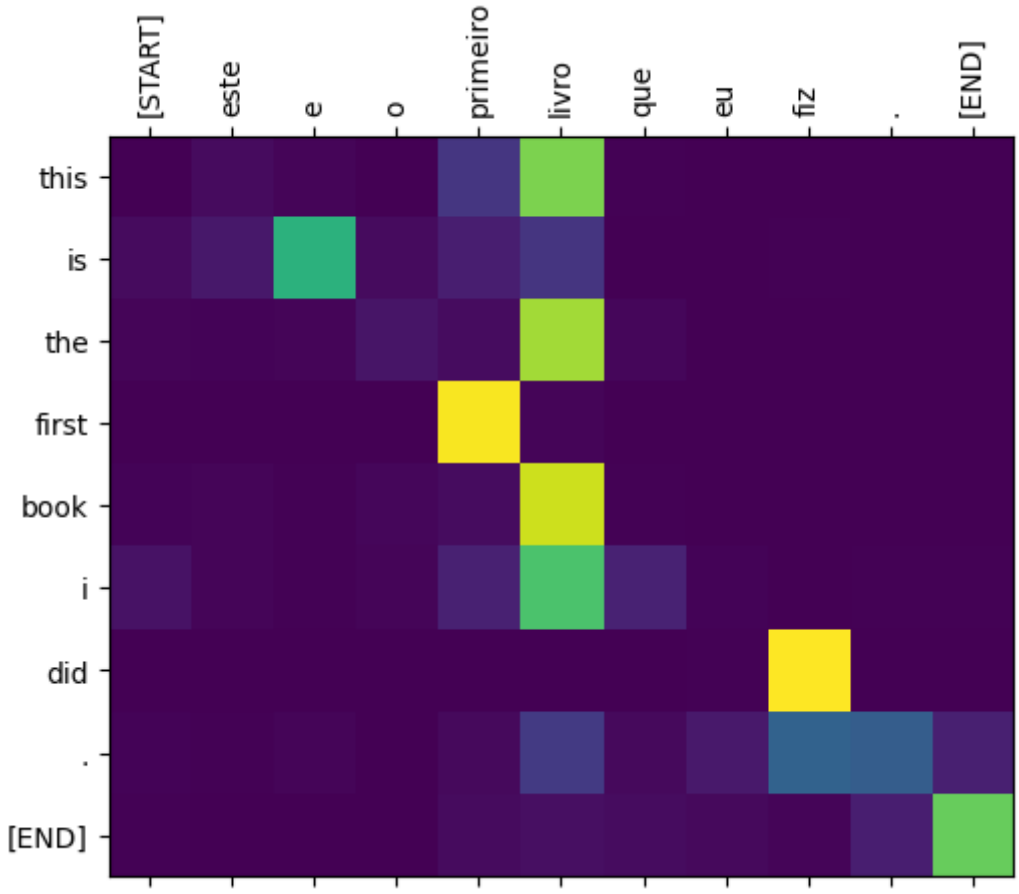
```
translated_tokens
```

Out[68]:

```
<tf.Tensor: shape=(10,), dtype=string, numpy=
array([b'[START]', b'this', b'is', b'the', b'first', b'book', b'i',
       b'did', b'.', b'[END]'], dtype=object)>
```

In [69]:

```
plot_attention_head(in_tokens, translated_tokens, attention)
```

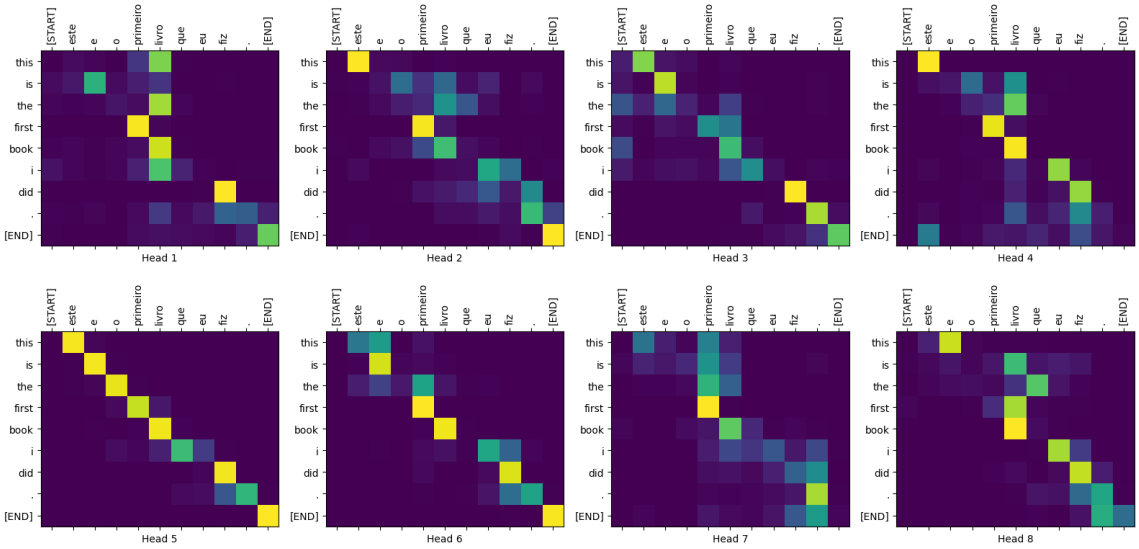


In [70]:

```
def plot_attention_weights(sentence, translated_tokens, attention_heads):
  in_tokens = tf.convert_to_tensor([sentence])
  in_tokens = tokenizers.pt.tokenize(in_tokens).to_tensor()
  in_tokens = tokenizers.pt.lookup(in_tokens)[0]

  fig = plt.figure(figsize=(16, 8))

  for h, head in enumerate(attention_heads):
    ax = fig.add_subplot(2, 4, h+1)

    plot_attention_head(in_tokens, translated_tokens, head)

    ax.set_xlabel(f'Head {h+1}')

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

In [71]:

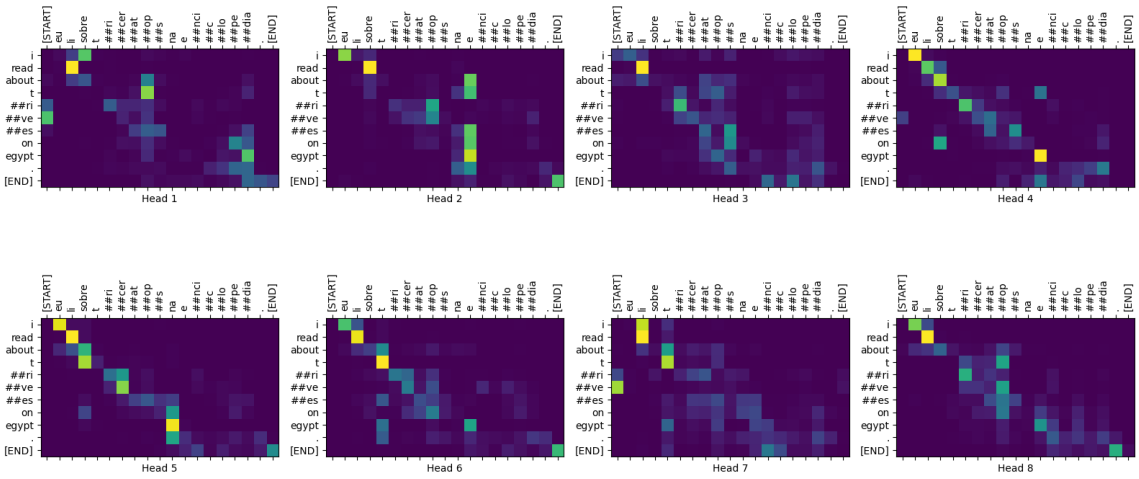
```
plot_attention_weights(sentence,
                       translated_tokens,
                       attention_weights[0])
```



In [72]:

```
sentence = 'Eu li sobre triceratops na enciclopédia.'
ground_truth = 'I read about triceratops in the encyclopedia.'

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)

plot_attention_weights(sentence, translated_tokens, attention_weights[0])
```

```
Input:          : Eu li sobre triceratops na enciclopédia.
Prediction      : i read about trivees on egypt .
Ground truth    : I read about triceratops in the encyclopedia.
```



Export the model

In [73]:

```
class ExportTranslator(tf.Module):
  def __init__(self, translator):
    self.translator = translator

  @tf.function(input_signature=[tf.TensorSpec(shape=[], dtype=tf.string)])
  def __call__(self, sentence):
    (result,
     tokens,
     attention_weights) = self.translator(sentence, max_length=MAX_TOKENS)

    return result
```

In [74]:

```
translator = ExportTranslator(translator)
```

In [75]:

```
translator('este é o primeiro livro que eu fiz.').numpy()
```

Out[75]:

```
b'this is the first book i did .'
```

In [76]:

```
tf.saved_model.save(translator, export_dir='translator')
```

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable
objects. Please implement a `get_config()` method in the subclassed Model
for proper saving and loading. Defaulting to empty config.

WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable
objects. Please implement a `get_config()` method in the subclassed Model
for proper saving and loading. Defaulting to empty config.

WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable
objects. Please implement a `get_config()` method in the subclassed Model
for proper saving and loading. Defaulting to empty config.

WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable
objects. Please implement a `get_config()` method in the subclassed Model
for proper saving and loading. Defaulting to empty config.
WARNING:absl:Found untraced functions such as positional_embedding_4_layer
_call_fn, positional_embedding_4_layer_call_and_return_conditional_losses,
dropout_35_layer_call_fn, dropout_35_layer_call_and_return_conditional_los
ses, positional_embedding_5_layer_call_fn while saving (showing 5 of 316).
These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: translator/assets

INFO:tensorflow:Assets written to: translator/assets
```

In [77]:

```
reloaded = tf.saved_model.load('translator')
```

In [78]:

```
reloaded('este é o primeiro livro que eu fiz.').numpy()
```

Out[78]:

```
b'this is the first book i did .'
```

EXPERIMENT-10

AIM:

Implementing MLOps techniques for predicting Bangalore Housing Prices: An experiment using ML models on real Estate data

THEORY AND SOURCE CODE:

The main goal of this project is to nd the price of the Bangalore house using their features.

1. Import Libraries

Importing numpy for array processing, pandas for csv operations and matplotlib,seaborn for plotting maps etc.

Next step would be to load the dataset which contains information such as number of rooms, square feet, and other details

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

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the
  import pandas.util.testing as tm
```

2. Load dataset

Load csv file from google drive

Main Source: https://www.kaggle.com/amitabhajoy/bengaluru-house-price-data

```
path = "https://drive.google.com/uc?export=download&id=13mP8FeMX09L3utbPcCDp-U2fXnf53gwx"
df_raw = pd.read_csv(path)
df_raw.shape
```

```
(13320, 9)
```

```
df_raw.head()
```

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

```
df_raw.tail()
```

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
13315	Built-up Area	Ready To Move	Whitefield	5 Bedroom	ArsiaEx	3453	4.0	0.0	231.0
13316	Super built-up Area	Ready To Move	Richards Town	4 BHK	NaN	3600	5.0	NaN	400.0
13317	Built-up Area	Ready To Move	Raja Rajeshwari Nagar	2 BHK	Mahla T	1141	2.0	1.0	60.0
13318	Super built-								

3. Exploratory Data Analysis

```
df = df_raw.copy() # get the copy of raw data
```

```
# get the information of data
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
```

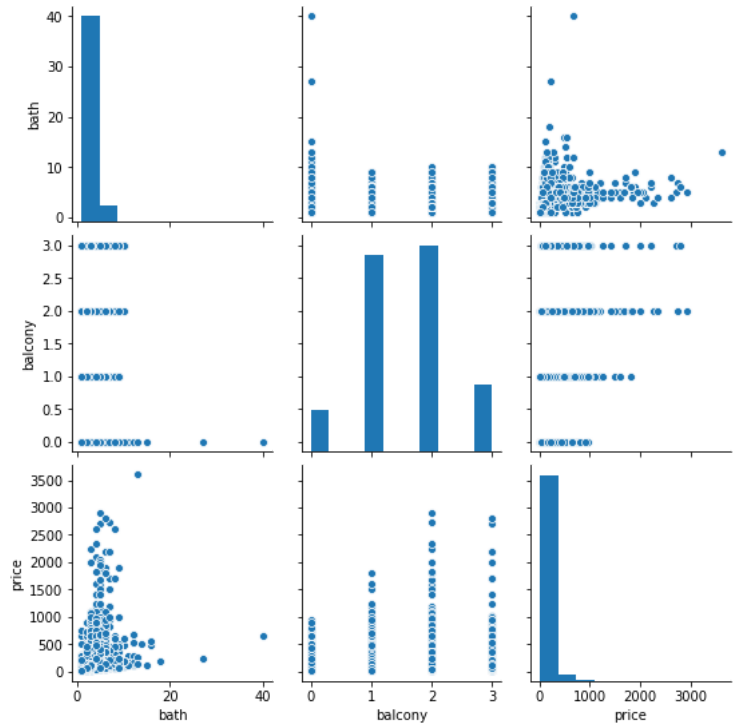
```
# We have only 3 neumerical features - bath, balcony and price
# 6 categorical features - area type, availability, size, society, and total_srft
# Target Feature =======>>>>>> price >>>>>>
# Price in lakh
```

```
df.describe()
#observe 75% and max value it shows huge diff
```

	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)

# bath and price have slightly linear correlation with some outliers
```

```
<seaborn.axisgrid.PairGrid at 0x7f49346ed320>
```




```
# value count of each feature
def value_count(df):
```

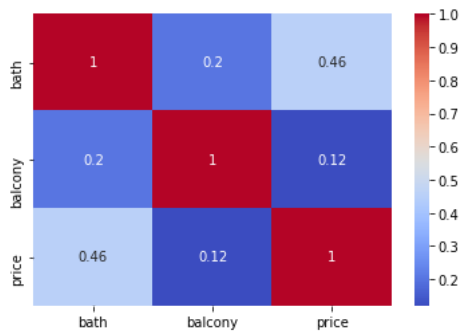
```
  for var in df.columns:
    print(df[var].value_counts())
    print("---------------------------------")
```

```
value_count(df)
```

```
1 Bedroom      105
8 Bedroom       84
7 Bedroom       83
5 BHK           59
9 Bedroom       46
6 BHK           30
7 BHK           17
1 RK            13
10 Bedroom      12
9 BHK            8
8 BHK            5
10 BHK           2
11 Bedroom       2
11 BHK           2
12 Bedroom       1
18 Bedroom       1
13 BHK           1
14 BHK           1
43 Bedroom       1
16 BHK           1
27 BHK           1
19 BHK           1
Name: size, dtype: int64
---------------------------------
GrrvaGr    80
PrarePa    76
Sryalan    59
Prtates    59
GMown E    56
           ..
CalanGo     1
J axyGa     1
Seamaha     1
VentsAp     1
Teire G     1
Name: society, Length: 2688, dtype: int64
---------------------------------
1200    843
1100    221
1500    205
2400    196
600     180
       ...
445       1
911       1
2251      1
1793      1
4041      1
Name: total_sqft, Length: 2117, dtype: int64
---------------------------------
2.0     6908
3.0     3286
4.0     1226
1.0      788
5.0      524
6.0      273
7.0      102
8.0       64
9.0       43
```

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

# correlation of bath is greater than a balcony with price
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4931a38a58>
```



4. Preare Data for Machine Learning Model

Data cleaning

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

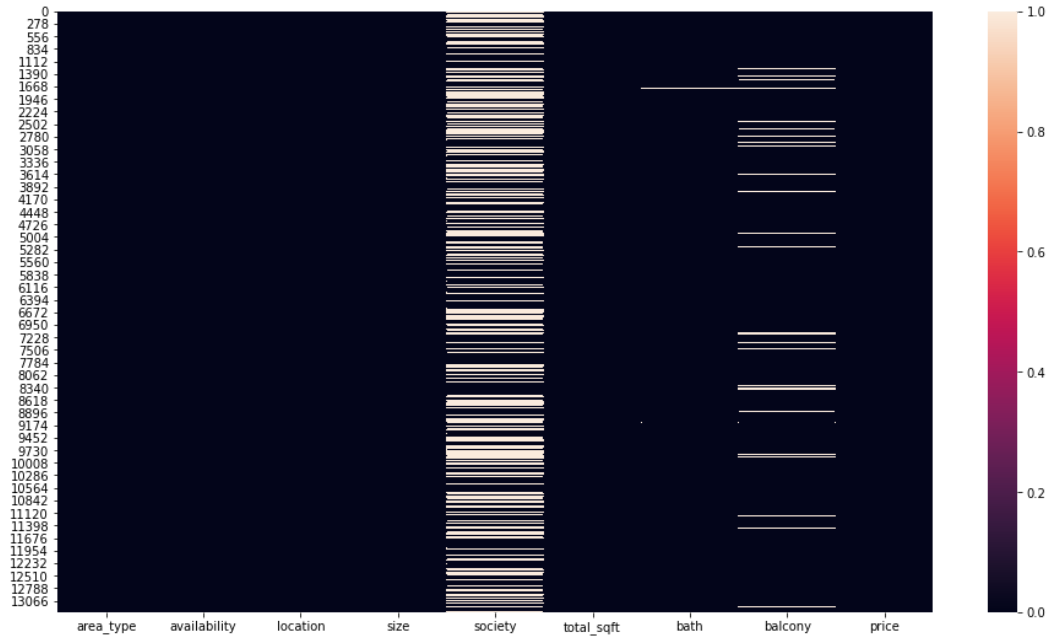
```
area_type          0
availability       0
location           1
size              16
society         5502
total_sqft         0
bath              73
balcony          609
price              0
dtype: int64
```

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

#society has 41.3% missing value (need to drop)
```

```
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())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4930355d68>
```




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

```
(13320, 8)
```

```
# fill mean value in --------> balcony feature
# because it contain 4.5% missing value
```

```
df2['balcony'] = df2['balcony'].fillna(df2['balcony'].mean())
df2.isnull().sum()
```

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

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

```
(13246, 8)
```

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

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

```
df3.head()
```

	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

Feature Engineering

```
# to show all th ecolumns and rows
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", None)
```

Converting 'total_sqft' cat feature in numeric

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

# here we observe that 'total_sqft' contain string value in diff format
#float, int like value 1689.28,817
# range value: 540 - 740
# number and string: 142.84Sq. Meter, 117Sq. Yards, 1Grounds

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

```
1200                843
1100                221
1500                204
2400                195
600                 180
1000                172
1350                132
1050                123
1300                117
1250                114
900                 112
1400                108
1800                104
1150                101
1600                100
1140                 91
```

```
2000                82
1450                70
1650                69
800                 67
1075                66
3000                66
1020                63
2500                62
1125                60
1160                60
1550                60
950                 59
1700                58
1180                58
1260                57
1255                56
1080                55
1220                55
1070                53
700                 52
750                 52
4000                48
1175                48
1225                48
2100                46
1240                46
1320                46
1060                45
1230                45
1210                44
850                 43
1280                42
1185                41
1270                41
1410                40
1170                40
1190                40
1750                39
1025                38
1330                38
1850                37
```

```
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 conver in float
  except:
    try:
      temp = []
      temp = str_val.split('-')
      total_sqft_int.append((float(temp[0])+float(temp[-1]))/2) # '123 - 534' this str value split and take mean
    except:
      total_sqft_int.append(np.nan) # if value not contain in above format then consider as nan
```

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

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

	area_type	availability	location	size	total_sqft	bath	balcony	price	total_sqft_int
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	1056	2.0	1.0	39.07	1056.0
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	2600	5.0	3.0	120.00	2600.0
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	1440	2.0	3.0	62.00	1440.0

```
df5.tail()
```

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

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

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

```
(13200, 9)
```

```
df6.info()
```

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

# size feature shows the number of rooms
```

```
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
8 Bedroom       83
7 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
10 BHK           2
11 Bedroom       2
11 BHK           2
12 Bedroom       1
18 Bedroom       1
13 BHK           1
14 BHK           1
43 Bedroom       1
16 BHK           1
27 BHK           1
19 BHK           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)
```

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

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

```
(13200, 10)
```

```
df7.tail()
```

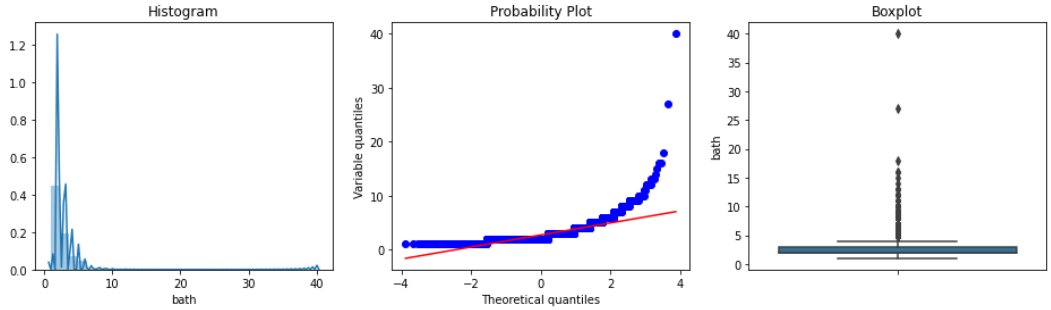
	area_type	availability	location	size	total_sqft	bath	balcony	price	total_sqft
13195	Built-up Area	Ready To Move	Whitefield	5 Bedroom	3453	4.0	0.000000	231.0	3
13196	Super built-up Area	Ready To Move	Richards Town	4 BHK	3600	5.0	1.584376	400.0	3
13197	Built-up Area	Ready To Move	Raja Rajeshwari Nagar	2 BHK	1141	2.0	1.000000	60.0	1

Finding Outlier and Removing

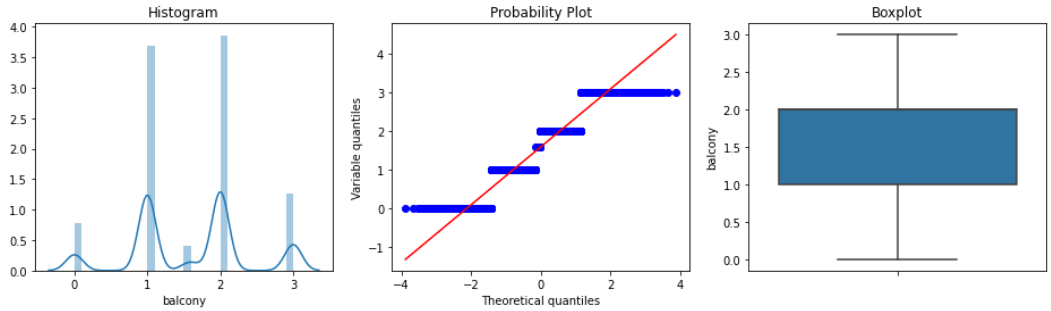
```
# function to create histogram, Q-Q plot and boxplot

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

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()
```

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

  # here we observe outlier using histogram,, qq plot and boxplot
```

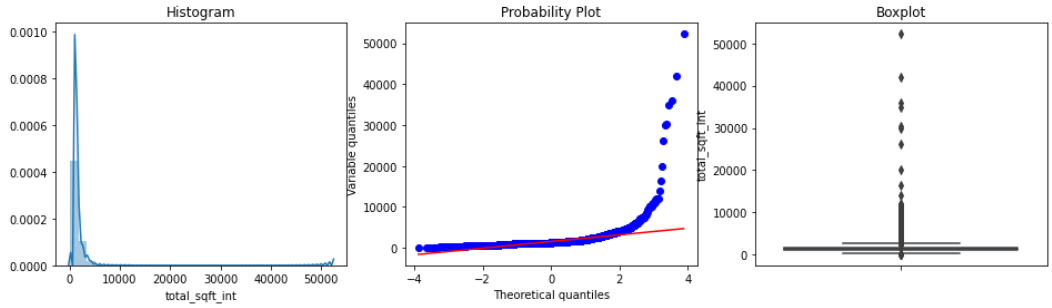
```
******* bath *******
```



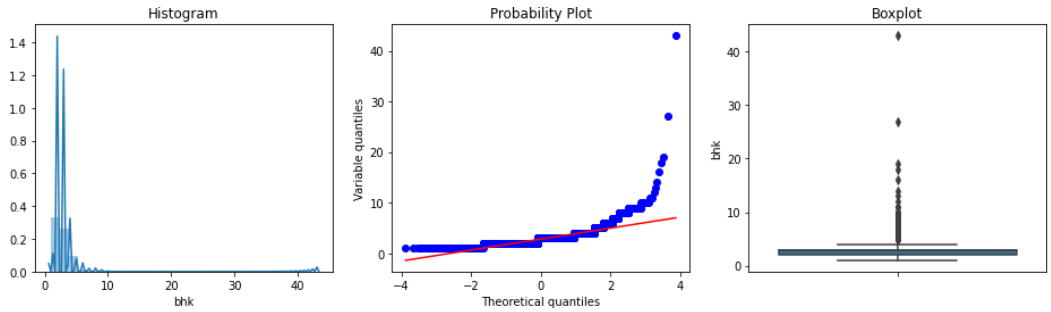

```
******* balcony *******
```



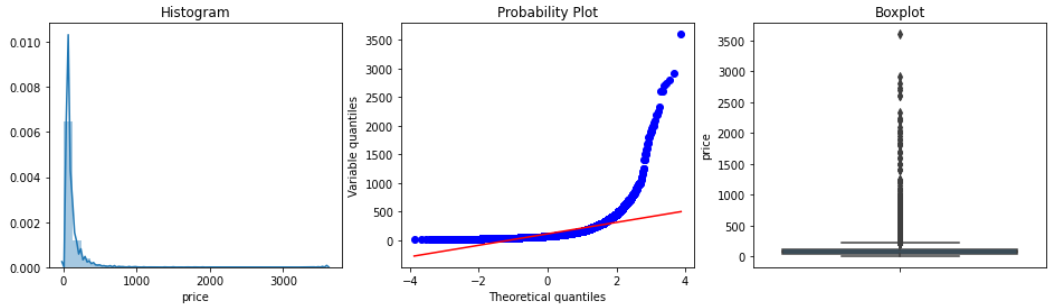

```
******* total_sqft_int *******
```




```
******* bhk *******
```




```
******* price *******
```




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

# no we found outliers
```

	area_type	availability	location	size	total_sqft	bath	balcony	price	total_sqft_int
9	Plot Area	Ready To Move	Gandhi Bazar	6 Bedroom	1020	6.0	1.584376	370.0	1020.0

Super

```
# 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
```

```
(12106, 10)
```

```
# 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()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py: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,
  """
```

	area_type	availability	location	size	total_sqft	bath	balcony	price	total_sqft_int
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	1056	2.0	1.0	39.07	1056.0
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	2600	5.0	3.0	120.00	2600.0
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	1440	2.0	3.0	62.00	1440.0

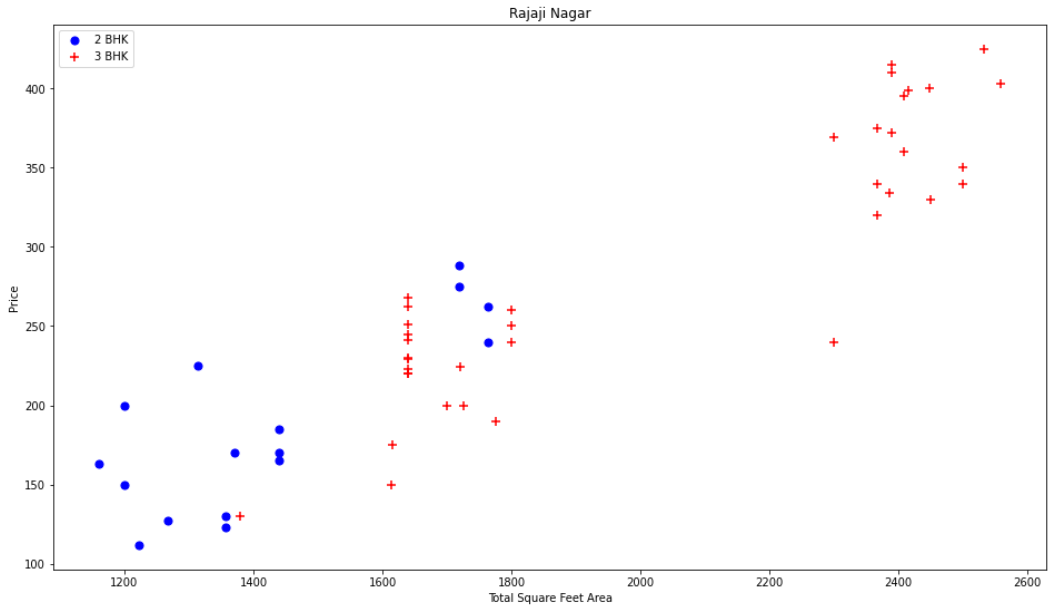
```
df8.price_per_sqft.describe()

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

```
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
```

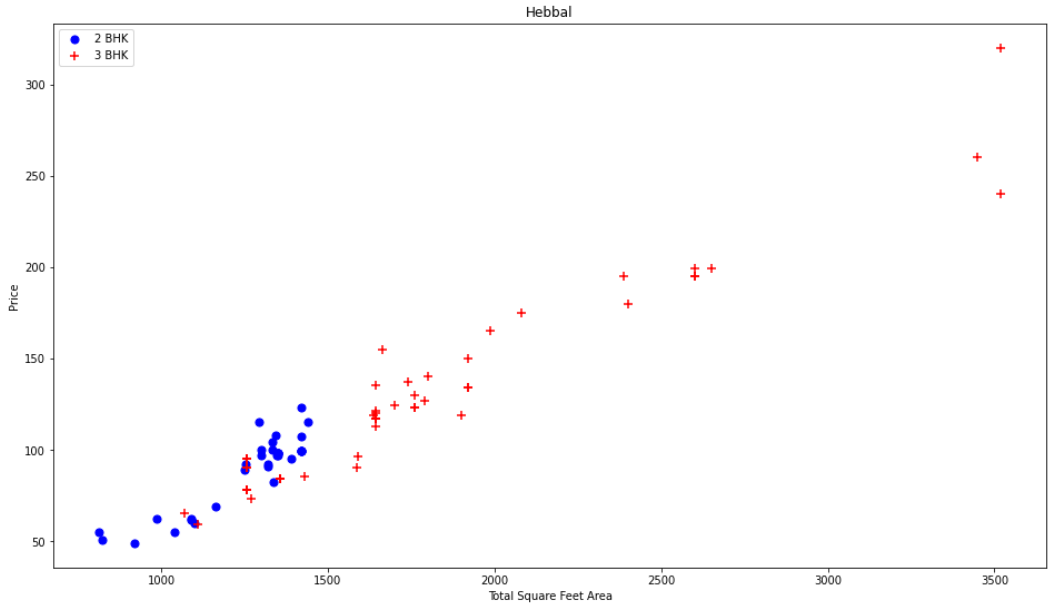
```
# Removing outliers using help of 'price per sqrt'  taking std and mean per location
def remove_pps_outliers(df):
  df_out = pd.DataFrame()
  for key, subdf in df.groupby('location'):
    m=np.mean(subdf.price_per_sqft)
    st=np.std(subdf.price_per_sqft)
    reduced_df = subdf[(subdf.price_per_sqft>(m-st))&(subdf.price_per_sqft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

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

```
(8888, 11)
```

```
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 below scatterplot we observe that at same location price of
# 2 bhk house is greater than 3 bhk so it is outlier
```



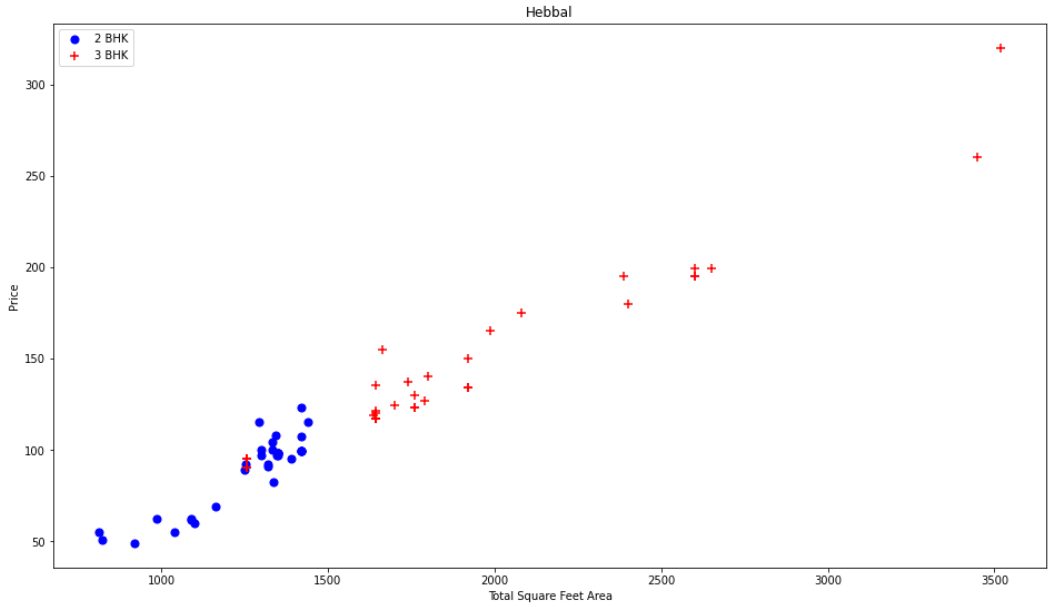

```
plot_scatter_chart(df9, "Hebbal")
# in below scatterplot we observe that at same location price of
# 3 bhk house is less than 2 bhk so it is outlier
```



```
# 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['mean'])].index.values)
  return df.drop(exclude_indices, axis='index')

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

```
(7194, 11)
```

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



Remove outliers using the help of 'bath' feature

```
df10.bath.unique()
```

```
array([ 3.,  2.,  1.,  4.,  5.,  8.,  9.,  6.,  7., 12.])
```

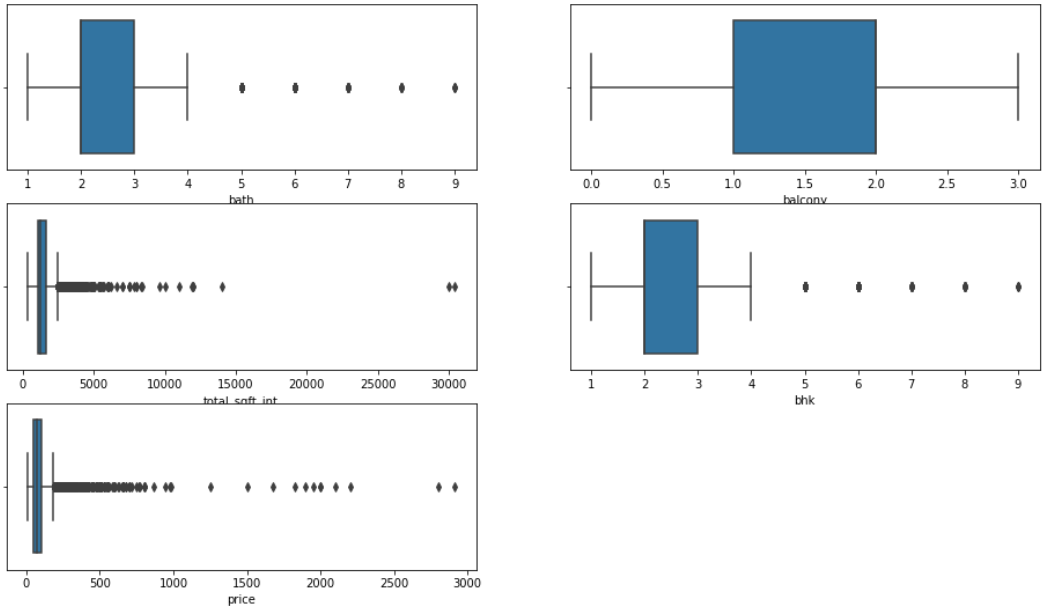
```
df10[df10.bath > df10.bhk+2]
```

	area_type	availability	location	size	total_sqft	bath	balcony	price	total_sqft_int
1861	Built-up Area	Ready To Move	Chikkabanavar	4 Bedroom	2460	7.0	2.000000	80.0	2460.(
5836	Built-up Area	Ready To Move	Nagasandra	4 Bedroom	7000	8.0	1.584376	450.0	7000.(
7098	Super built-up	Ready To	Sathya Sai	6 BHK	11338	9.0	1.000000	1000.0	11338.(

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

```
(7120, 11)
```

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




```
df11.head()
```

	area_type	availability	location	size	total_sqft	bath	balcony	price	total_sqft_int
0	Super built-up Area	Ready To Move	Devarabeesana Halli	3 BHK	1672	3.0	2.0	150.0	1672.(
1	Built-up Area	Ready To Move	Devarabeesana Halli	3 BHK	1750	3.0	3.0	149.0	1750.(
2	Super built-up Area	Ready To Move	Devarabeesana Halli	3 BHK	1750	3.0	2.0	150.0	1750.(

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

	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

```
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
```

Categorical Variable Encoding

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

area_type	availability	location	bath	balcony	price	total_sqft_int	bhk	price_per_sqft

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

```
(7120, 820)
```

```
df14.head()
```

	bath	balcony	price	total_sqft_int	bhk	price_per_sqft	area_type_Carpet Area	area_type_Plot Area	area_ty built
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	

```
df14.to_csv('oh_encoded_data.csv', index=False) # test ml model on this data
```

In ['area_type','availability','location'] contain multiple classe and if we convert them into OHE so it increase the size of DF so try to use those classes which are *frequently* present in the car var

Working on <<<<<< area_type >>>>> feature

```
df13['area_type'].value_counts()
```

```
Super built-up  Area    5345
Built-up  Area          1298
Plot  Area               441
Carpet  Area              36
Name: area_type, dtype: int64
```

```
df15 = df13.copy()
# 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
```

```
(7120, 12)
```

```
df15.head(2)
```

	area_type	availability	location	bath	balcony	price	total_sqft_int	bhk	price_per_sqft	ar b
0	Super built-up Area	Ready To Move	Devarabeesana Halli	3.0	2.0	150.0	1672.0	3	8971.291866	
1	Built-up Area	Ready To Move	Devarabeesana Halli	3.0	3.0	149.0	1750.0	3	8514.285714	

Working with <<<<< availability >>>>> Feature

```
df15["availability"].value_counts()
```

```
Ready To Move    5644
18-Dec            159
18-May            156
18-Apr            154
18-Aug            118
19-Dec            101
18-Jul             76
18-Mar             72
21-Dec             55
20-Dec             54
18-Jun             51
19-Mar             48
```

```
18-Feb            36
18-Nov            26
19-Jun            24
18-Oct            24
19-Jan            19
20-Jan            18
18-Sep            18
18-Jan            17
17-Dec            14
21-Mar            13
21-Jun            13
17-Oct            13
19-Sep            12
19-Jul            11
19-Aug            11
17-Jul            10
21-Jan            10
17-Jun             9
19-Apr             9
22-Dec             7
20-Jun             7
19-Oct             7
21-Jul             6
20-Aug             6
20-Oct             6
17-Sep             6
17-May             6
20-Sep             5
21-Feb             5
20-Nov             4
17-Nov             4
22-Jan             4
21-Oct             4
17-Apr             4
17-Aug             3
22-Mar             3
16-Dec             3
20-May             3
17-Mar             3
19-May             3
21-Sep             3
20-Mar             2
21-May             2
22-May             2
22-Nov             2
19-Feb             2
```

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

```
(7120, 13)
```

```
df15.tail()
```

	area_type	availability	location	bath	balcony	price	total_sqft_int	bhk	price_per_sqft	ar bu
8883	Super built-up Area	Ready To Move	frazertown	3.0	2.0	325.00	2900.0	3	11206.896552	
8884	Super built-up Area	18-Nov	manyata park	3.0	1.0	84.83	1780.0	3	4765.730337	
8885	Plot Area	Ready To Move	tc.palya	2.0	1.0	48.00	880.0	2	5454.545455	
8886	Plot Area	18-Apr	tc.palya	2.0	1.0	55.00	1000.0	2	5500.000000	

Working on <<<< Location >>>> feature

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

```
Whitefield                    234
Sarjapur  Road                183
Electronic City               158
Raja Rajeshwari Nagar         116
Marathahalli                  116
Haralur Road                  116
Hennur Road                   108
Bannerghatta Road             108
```

```
Uttarahalli                          106
Thanisandra                          103
Electronic City Phase II              89
Hebbal                                87
7th Phase JP Nagar                    86
Yelahanka                             86
Kanakpura Road                        85
KR Puram                              61
Sarjapur                              56
Rajaji Nagar                          55
Bellandur                             53
Kasavanhalli                          53
Begur Road                            51
Kothanur                              49
Banashankari                          49
Hormavu                               47
Harlur                                44
Akshaya Nagar                         43
Electronics City Phase 1              42
Jakkur                                42
Varthur                               41
HSR Layout                            39
Chandapura                            39
Ramamurthy Nagar                      39
Hennur                                39
Kundalahalli                          38
Ramagondanahalli                      38
Kaggadasapura                         38
Koramangala                           38
Budigere                              37
Hoodi                                 37
Hulimavu                              37
Malleshwaram                          35
Hegde Nagar                           33
Gottigere                             33
Yeshwanthpur                          33
8th Phase JP Nagar                    33
JP Nagar                              33
Bisuvanahalli                         32
Channasandra                          32
Indira Nagar                          31
Vittasandra                           31
Kengeri                               29
Vijayanagar                           29
Brookefield                           29
Hosa Road                             29
Sahakara Nagar                        29
Old Airport Road                      29
Green Glen Layout                     28
```

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

```
Index(['Whitefield', 'Sarjapur  Road', 'Electronic City',
       'Raja Rajeshwari Nagar', 'Marathahalli', 'Haralur Road', 'Hennur Road',
       'Bannerghatta Road', 'Uttarahalli', 'Thanisandra',
       'Electronic City Phase II', 'Hebbal', '7th Phase JP Nagar', 'Yelahanka',
       'Kanakpura Road', 'KR Puram', 'Sarjapur', 'Rajaji Nagar', 'Bellandur',
       'Kasavanhalli', 'Begur Road', 'Kothanur', 'Banashankari', 'Hormavu',
       'Harlur', 'Akshaya Nagar', 'Electronics City Phase 1', 'Jakkur',
       'Varthur', 'HSR Layout', 'Chandapura', 'Ramamurthy Nagar', 'Hennur',
       'Kundalahalli', 'Ramagondanahalli', 'Kaggadasapura', 'Koramangala',
       'Budigere', 'Hoodi', 'Hulimavu', 'Malleshwaram', 'Hegde Nagar',
       'Gottigere', 'Yeshwanthpur', '8th Phase JP Nagar', 'JP Nagar',
       'Bisuvanahalli', 'Channasandra', 'Indira Nagar', 'Vittasandra',
       'Kengeri', 'Vijayanagar', 'Brookefield', 'Hosa Road', 'Sahakara Nagar',
       'Old Airport Road', 'Green Glen Layout', 'Balagere', 'Bommasandra',
       'Old Madras Road', 'Panathur', 'Rachenahalli', 'Kudlu Gate',
       'Thigalarapalya', 'Jigani', 'Talaghattapura', 'Mysore Road',
       'Yelahanka New Town', 'Ambedkar Nagar', 'Kadugodi', 'Attibele',
       'Devanahalli', 'Kanakapura', 'Dodda Nekkundi', 'Frazer Town',
       'Ananth Nagar', 'Nagarbhavi', 'TC Palaya', 'Lakshminarayana Pura',
       'Anekal', '5th Phase JP Nagar', 'Kudlu', 'Kengeri Satellite Town',
       'CV Raman Nagar', 'Jalahalli', 'Horamavu Agara', 'Bhoganhalli',
       'Kalena Agrahara', 'Subramanyapura', 'Doddathoguru', 'BTM 2nd Stage',
       'Vidyaranyapura', 'Hebbal Kempapura', 'Hosur Road', 'Domlur',
       'Mahadevpura', 'Tumkur Road', 'Horamavu Banaswadi'],
      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 encoding)
df16 = df15.copy()
for cat_var in location_gert_20:
  df16['location_'+cat_var]=np.where(df16['location']==cat_var, 1,0)
df16.shape
```

```
(7120, 111)
```

```
df16.head()
```

	area_type	availability	location	bath	balcony	price	total_sqft_int	bhk	price_per_sqft
0	Super built-up Area	Ready To Move	Devarabeesana Halli	3.0	2.0	150.0	1672.0	3	8971.291866
1	Built-up Area	Ready To Move	Devarabeesana Halli	3.0	3.0	149.0	1750.0	3	8514.285714
2	Super built-up Area	Ready To Move	Devarabeesana Halli	3.0	2.0	150.0	1750.0	3	8571.428571
4	Super built-up Area	Ready To Move	Devarachikkanahalli	2.0	2.0	40.0	1250.0	2	3200.000000
5	Plot Area	Ready To Move	Devarachikkanahalli	2.0	2.0	83.0	1200.0	2	6916.666667

Drop categorical variable

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

```
(7120, 108)
```

```
df17.head()
```

	bath	balcony	price	total_sqft_int	bhk	price_per_sqft	area_typeSuper built-up Area	area_typeBuilt-up Area	area_typ
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	
4	2.0	2.0	40.0	1250.0	2	3200.000000	1	0	
5	2.0	2.0	83.0	1200.0	2	6916.666667	0	0	

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

Bangalore House Price Prediction - Outlier Detection

This notebook only train ML model on different ml algorithms

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

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the
  import pandas.util.testing as tm
```

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

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

"from google.colab import files\nfiles=files.upload()\ndf = pd.read_csv('oh_encoded_data.csv')"

```
# Get clean data
path = r"https://drive.google.com/uc?export=download&id=1P49POlAk27uRzWKXoR2WaEfb1lyyfiRJ" # oh_encoded_data.csv from drive

# This file contain [area_type  availability  location  bath  balcony price total_sqft_int  bhk price_per_sqft]
# and ['area_type','availability','location'] this are cat var
# We encoded few classes from above car var in OHE

df = pd.read_csv(path)
df.shape
```

```
(7120, 109)
```

```
df.shape
```

```
(7120, 109)
```

```
df.head()
```

	Unnamed: 0	bath	balcony	price	total_sqft_int	bhk	price_per_sqft	area_typeSuper built-up Area	area_typeBuilt-up Area
0	0	3.0	2.0	150.0	1672.0	3	8971.291866	1	0
1	1	3.0	3.0	149.0	1750.0	3	8514.285714	0	1
2	2	3.0	2.0	150.0	1750.0	3	8571.428571	1	0
3	4	2.0	2.0	40.0	1250.0	2	3200.000000	1	0
4	5	2.0	2.0	83.0	1200.0	2	6916.666667	0	0

```
df = df.drop(['Unnamed: 0'], axis=1)
df.head()
```

	bath	balcony	price	total_sqft_int	bhk	price_per_sqft	area_typeSuper built-up Area	area_typeBuilt-up Area	area_typ
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.shape
```

```
(7120, 108)
```

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,)
```

```
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 = 51)
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)
```

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()
```

```
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
```

```
(0.7903837092682254, 64.89843531105606)
```

```
# 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
```

```
(0.803637294021051, 62.81324273988926)
```

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
```

```
(0.20638035840828173, 126.27806378079055)
```

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
```

```
(0.8896232895502098, 47.09344200463701)
```

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
```

```
[09:48:22] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
(0.8759391907485039, 49.92740744112327)
```

```
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   62.813243
2  Support Vector Machine  0.206380  126.278064
3           Random Forest  0.889623   47.093442
4                 XGBoost  0.875939   49.927407
```

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)'''
```

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

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

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

```
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)'''
```

```
(array([0.99494408, 0.96682912, 0.99720454, 0.96433211, 0.96151867,
        0.94774651, 0.94212832, 0.91069009, 0.99610078, 0.98860838]),
 0.9670102612461828)
```

Hyper Parmeter Tuning

```
from sklearn.model_selection import GridSearchCV
from xgboost.sklearn import XGBRegressor
'''
# Various hyper-parameters to tune
xgb1 = XGBRegressor()
```

```
parameters = {'learning_rate': [0.1,0.03, 0.05, 0.07], #so called `eta` value, # [default=0.3] Analogous to learning rate in GBM
              'min_child_weight': [1,3,5], #[default=1] Defines the minimum sum of weights of all observations required in a child.
              '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 required to make a split.
              'subsample': [0.7,1,1.5], #Denotes the fraction of observations to be randomly samples for each tree.
              'colsample_bytree': [0.7,1,1.5], #Denotes the fraction of columns to be randomly samples for each tree.
              'objective':['reg:linear'], #This defines the loss function to be minimized.

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

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

xgb_grid.fit(X_train, y_train)

print(xgb_grid.best_score_) # 0.9397345161940295
print(xgb_grid.best_params_)'''
```

```
"\n# Various hyper-parameters to tune\nxgb1 = XGBRegressor()\nparameters = {'learning_rate': [0.1,0.03, 0.05, 0.07], #so called
`eta` value, # [default=0.3] Analogous to learning rate in GBM\n              'min_child_weight': [1,3,5], #[default=1] Defines
the minimum sum of weights of all observations required in a child.\n              'max_depth': [4, 6, 8], #[default=6] The
maximum depth of a tree,\n              'gamma':[0,0.1,0.001,0.2], #Gamma specifies the minimum loss reduction required to make a
split.\n              'subsample': [0.7,1,1.5], #Denotes the fraction of observations to be randomly samples for each tree.\n
'colsample_bytree': [0.7,1,1.5], #Denotes the fraction of columns to be randomly samples for each tree.\n
'objective':['reg:linear'], #This defines the loss function to be minimized.\n\n              'n_estimators':
[100,300,500]}\n\nxgb_grid = GridSearchCV(xgb1,\n                         parameters,\n                         cv = 2,\n
n_jobs = -1,\n                         verbose=True)\n\nxgb_grid.fit(X_train, y_train)\n\nprint(xgb_grid.best_score_) #
0.9397345161940295\nprint(xgb_grid.best_params_)"
```

```
'''xgb_tune = xgb_grid.estimator

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

```
'xgb_tune = xgb_grid.estimator\n\nxgb_tune.fit(X_train,y_train) # 0.9117591385438816\nxgb_tune.score(X_test,y_test)'
```

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

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

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

```
#xgb_grid.estimator
```

```
xgb_tune2 =  XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=0.6, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.25, max_delta_step=0,
             max_depth=4, min_child_weight=1, missing=None, n_estimators=400,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
xgb_tune2.fit(X_train,y_train) # 0.9412851220926807
xgb_tune2.score(X_test,y_test)
```

```
[09:48:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
0.8945812461922704
```

```
'''parameters = {'learning_rate': [0.1,0.03, 0.05, 0.07], #so called `eta` value, # [default=0.3] Analogous to learning rate in GBM
              'min_child_weight': [1,3,5], #[default=1] Defines the minimum sum of weights of all observations required in a child.
              '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 required to make a split.
              'subsample': [0.7,1,1.5], #Denotes the fraction of observations to be randomly samples for each tree.
              'colsample_bytree': [0.7,1,1.5], #Denotes the fraction of columns to be randomly samples for each tree.
              'objective':['reg:linear'], #This defines the loss function to be minimized.
              'n_estimators': [100,300,500]}'''

xgb_tune2 =  XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=0.9, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.05, max_delta_step=0,
             max_depth=4, min_child_weight=5, missing=None, n_estimators=100,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

```
xgb_tune2.fit(X_train,y_train) # 0.9412851220926807
xgb_tune2.score(X_test,y_test)
```

```
[09:48:57] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
0.8840983866988691
```

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

```
[09:48:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[09:49:00] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[09:49:01] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[09:49:02] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[09:49:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
(array([0.97924577, 0.98376376, 0.97530216, 0.90127522, 0.96273069]),
 0.9604635172361338)
```

```
np.sqrt(mean_squared_error(y_test, xgb_tune2.predict(X_test)))
```

```
48.25768129834004
```

Test Model

```
list(X.columns)
```

```
['bath',
 'balcony',
 'total_sqft_int',
 'bhk',
 'price_per_sqft',
 'area_typeSuper built-up  Area',
 'area_typeBuilt-up  Area',
 'area_typePlot  Area',
 'availability_Ready To Move',
 'location_Whitefield',
 'location_Sarjapur  Road',
 'location_Electronic City',
 'location_Raja Rajeshwari Nagar',
 'location_Marathahalli',
 'location_Haralur Road',
 'location_Hennur Road',
 'location_Bannerghatta Road',
 'location_Uttarahalli',
 'location_Thanisandra',
 'location_Electronic City Phase II',
 'location_Hebbal',
 'location_Yelahanka',
 'location_7th Phase JP Nagar',
 'location_Kanakpura Road',
 'location_KR Puram',
 'location_Sarjapur',
 'location_Rajaji Nagar',
 'location_Bellandur',
 'location_Kasavanhalli',
 'location_Begur Road',
 'location_Kothanur',
 'location_Banashankari',
 'location_Hormavu',
 'location_Harlur',
 'location_Akshaya Nagar',
 'location_Jakkur',
 'location_Electronics City Phase 1',
 'location_Varthur',
 'location_HSR Layout',
 'location_Chandapura',
 'location_Ramamurthy Nagar',
 'location_Hennur',
 'location_Kundalahalli',
 'location_Ramagondanahalli',
 'location_Kaggadasapura',
 'location_Koramangala',
 'location_Hulimavu',
 'location_Budigere',
 'location_Hoodi',
 'location_Malleshwaram',
 'location_JP Nagar',
 'location_Hegde Nagar',
 'location_Yeshwanthpur',
 'location_8th Phase JP Nagar',
 'location_Gottigere',
 'location_Channasandra',
```

```
      'location_Bisuvanahalli',
      'location_Vittasandra'
```

```
# 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,availability,location):

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

  # 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 array
  #print(x)

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

```
predict_house_price(model=xgb_tune2, bath=3,balcony=2,total_sqft_int=1672,bhk=3,price_per_sqft=8971.291866,area_type="Plot  Area",availab
```

```
     145.91656
```

```
##test sample
#area_type  availability    location    bath    balcony price   total_sqft_int  bhk price_per_sqft
#2  Super built-up Area Ready To Move   Devarabeesana Halli 3.0 2.0 150.0   1750.0  3   8571.428571

predict_house_price(model=xgb_tune2, bath=3,balcony=2,total_sqft_int=1750,bhk=3,price_per_sqft=8571.428571,area_type="Super built-up",ava
```

```
     143.71669
```

```
##test sample
#area_type  availability    location    bath    balcony price   total_sqft_int  bhk price_per_sqft
#1  Built-up Area   Ready To Move   Devarabeesana Halli 3.0 3.0 149.0   1750.0  3   8514.285714
predict_house_price(model=xgb_tune2,bath=3,balcony=3,total_sqft_int=1750,bhk=3,price_per_sqft=8514.285714,area_type="Built-up Area",avail
```

```
     143.71669
```

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')
```

```
     ['bangalore_house_price_prediction_rfr_model.pkl']
```

```
# load model
bangalore_house_price_prediction_model = joblib.load("bangalore_house_price_prediction_model.pkl")
```

```
     [09:49:05] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```
# predict house price
predict_house_price(bangalore_house_price_prediction_model,bath=3,balcony=3,total_sqft_int=150,bhk=3,price_per_sqft=8514.285714,area_type
```

```
     65.78738
```

MLOps:

Filename: model.py

```
#Import Libraries
import numpy as np
import pandas as pd
import joblib

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

#load data
df = pd.read_csv("data/ohe_data_reduce_cat_class.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('bangalore_house_price_prediction_rfr_model.pkl')


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

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

  # 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

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

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

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

Filename: app.py

```
#Import Libraries

from flask import Flask, request, render_template
import model #load model.py
app = Flask(__name__)
# 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()
```

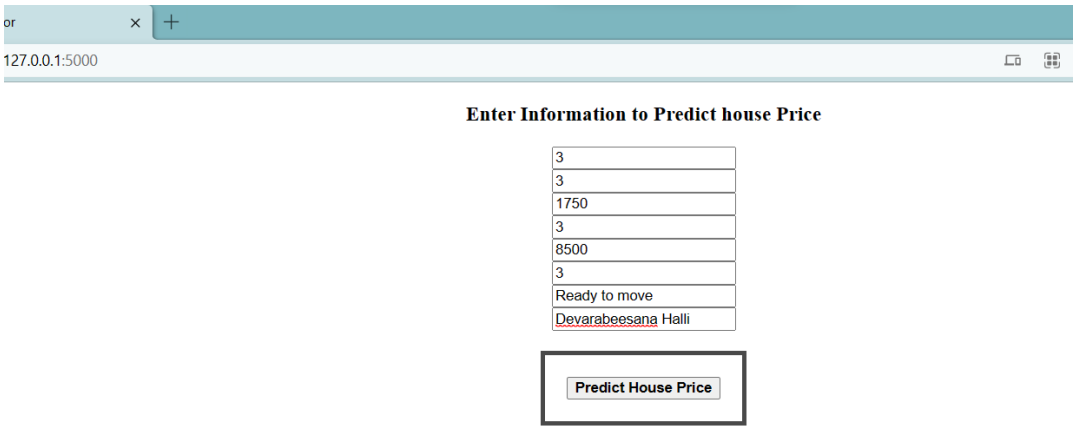
```
Filename: index.html
<!DOCTYPE html>
<html >
<head>
  <meta charset="UTF-8">
  <title>Price Predictor</title>
  <style>

      body {
      background-repeat: no-repeat;
      background-attachment: fixed;
      background-size: cover;
    }

    h1   {color: brown;}  /* CSS code for heading h1 */
    p   {color: green;}  /* 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: #203864;
  color: white;
  text-align: center;

  /* unvisited link */
  a:link { color: White; }
  /* visited link */
  a:visited { color: green; }
```

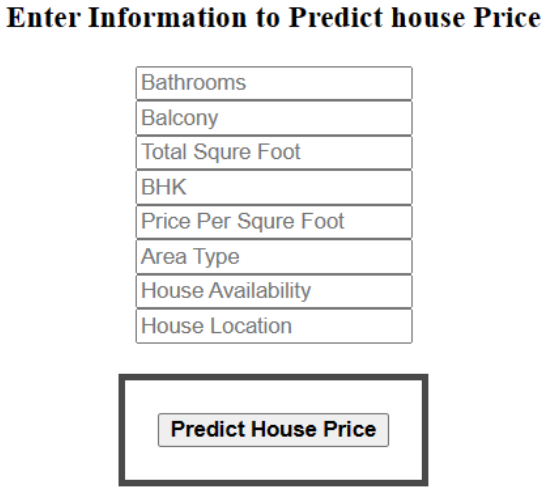
```
}
  </style>

</head>

<body>
 <div class="login">

    <!-- Form Get input to predict Marks-->
    <center>
    <form action="{{ url_for('predict')}}"method="post">
        <h3>Enter Information to Predict house Price</h3>

        <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="House
Availability" required="required" /><br>
        <input type="text" name="location" placeholder="House
Location" required="required" />
        <br>
        <br>
        <!-- Show button -->
        <div class="button_cont" align="center"><a
class="button_css" target="_blank" rel="nofollow noopener">
            <button type="submit" class="btn btn-primary btn-block
btn-large"><strong>Predict House Price</strong></button></a>
        </div>
    </form>
    </center>
   <!-- Show predicted output using ML model -->
   <div>
       <center>
   <h2>{{ prediction_text }}</h2>
       </center>
```

```
    </div>
  </div>
</div>
</body>
</html>
```

Output:

```
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [17/Apr/2023 11:04:22] "GET / HTTP/1.1" 200 -
C:\Users\adity\AppData\Roaming\Python\Python311\site-packages\sklearn\base.py:439: UserWarning: X does not have valid featur
e names, but StandardScaler was fitted with feature names
  warnings.warn(
127.0.0.1 - - [17/Apr/2023 11:05:57] "POST /predict HTTP/1.1" 200 -
```





Predicted Price of Bangalore House is 149.63