**Assignment -3**

Python Programming

| | |
|---|---|
| Assignment Date | 9 october 2022 |
| Student Name | RAKESH V |
| Student Roll Number | 110719106035 |
| Maximum Marks | 2 Marks |

**Question-1:**

## Download the Dataset

**Solution:**

```
from google.colab import drivedrive.mount('/content/drive')
```

```
#---------------------------------------#
#---------------------------------------#
```

**Download the Dataset**

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

**Question-2:**

## Image Augmentation

Solution :

**Image Augmentation**

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
import cv2
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import PIL
import random
import cv2
from tensorflow.keras import layers, models
import tensorflow as tf
import pandas as pd
from sklearn.model_selection import train_test_split
import seaborn as sns
import pickle
import zipfile
tf.__version__
```

Out[3]:
```
'2.8.2'
```

In [4]:
```
!ls
```

```
drive  sample_data
```

In [5]:
```
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Device:', tpu.master())
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
except:
    strategy = tf.distribute.get_strategy()
print('Number of replicas:', strategy.num_replicas_in_sync)
```

```
Number of replicas: 1
```

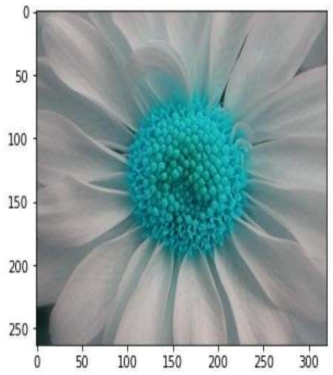
In [6]:
```
AUTOTUNE = tf.data.experimental.AUTOTUNE
batch_size = 32
IMAGE_SIZE = [128, 128]
EPOCHS = 25
```

In [7]:
```
image = cv2.imread(r'/content/drive/MyDrive/Flowers-Dataset/flowers/daisy/100080576_f52e8ee070_n.jpg')
```

In [8]:
```
print(image.shape)
```

```
(263, 320, 3)
```

In [9]:
```
imgplot = plt.imshow(image)
plt.show()
```




In [10]:
```
GCS_PATH = "/content/drive/MyDrive/Flowers-Dataset/flowers"

CLASS_NAMES = np.array([str(tf.strings.split(item, os.path.sep)[-1].numpy())[2:-1]
                        for item in tf.io.gfile.glob(str(GCS_PATH + "*/*"))])
CLASS_NAMES
```

Out[10]:
```
array(['daisy', 'rose', 'dandelion', 'sunflower', 'tulip'], dtype='<U9')
```

In [11]:
```
files_count = []
for i,f in enumerate(CLASS_NAMES):
    folder_path = os.path.join(GCS_PATH, f)
    for path in os.listdir(os.path.join(folder_path)):
        files_count.append(['{}/{}'.format(folder_path,path), f, i])
flowers_df = pd.DataFrame(files_count, columns=['filepath', 'class_name', 'label'])
flowers_df.head()
```

Out[11]:

| | filepath | class_name | label |
|---|---|---|---|
| 0 | /content/drive/MyDrive/Flowers-Dataset/flowers... | daisy | 0 |
| 1 | /content/drive/MyDrive/Flowers-Dataset/flowers... | daisy | 0 |
| 2 | /content/drive/MyDrive/Flowers-Dataset/flowers... | daisy | 0 |
| 3 | /content/drive/MyDrive/Flowers-Dataset/flowers... | daisy | 0 |
| 4 | /content/drive/MyDrive/Flowers-Dataset/flowers... | daisy | 0 |

In [12]:
```
flowers_df.class_name.value_counts()
```

Out[12]:
```
dandelion    1052
tulip         984
rose          784
daisy         764
sunflower     733
Name: class_name, dtype: int64
```

In [13]:
```
quantidade_por_class = 500
flowers_df = pd.concat([flowers_df[flowers_df['class_name']== i][:quantidade_por_class] for i in CLASS_NAMES])
```

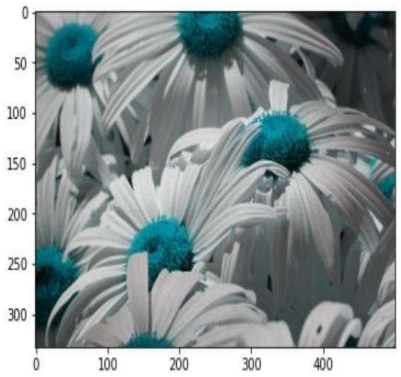
In [14]:
```
flowers_df.class_name.value_counts()
```

Out[14]:
```
daisy        500
rose         500
dandelion    500
sunflower    500
tulip        500
```

```
sunflower    500
tulip        500
Name: class_name, dtype: int64
```

In [15]:

```
image = cv2.imread(flowers_df.filepath[100])
imgplot = plt.imshow(image)
plt.show()
```




**Create Model**

In [16]:

```
X = flowers_df['filepath']
y = flowers_df['label']

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

In [17]:

**Question-**

**3:**

## Create Model

Solution :

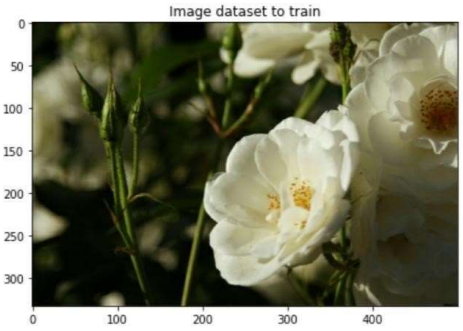
In [16]:
```
X = flowers_df['filepath']
y = flowers_df['label']

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

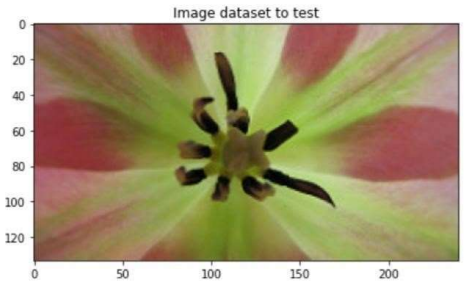
In [17]:
```
x_train_tensor = tf.convert_to_tensor(x_train.values, dtype=tf.string)
y_train_tensor = tf.convert_to_tensor(y_train.values)

x_test_tensor = tf.convert_to_tensor(x_test.values, dtype=tf.string)
y_test_tensor = tf.convert_to_tensor(y_test.values)
```

In [18]:
```
train_data = tf.data.Dataset.from_tensor_slices((x_train_tensor, y_train_tensor))
test_data = tf.data.Dataset.from_tensor_slices((x_test_tensor, y_test_tensor))
```

In [19]:
```
def map_fn(path, label):
    image = tf.image.decode_jpeg(tf.io.read_file(path))

    return image, label

#apply the function
train_data_img = train_data.map(map_fn)
test_data_img = test_data.map(map_fn)
```

In [20]:
```
fig, ax = plt.subplots(1,2, figsize = (15,5))
for i,l in train_data_img.take(1):
    ax[0].set_title('Image dataset to train');
    ax[0].imshow(i);
for i,l in test_data_img.take(1):
    ax[1].set_title('Image dataset to test');
    ax[1].imshow(i);
```



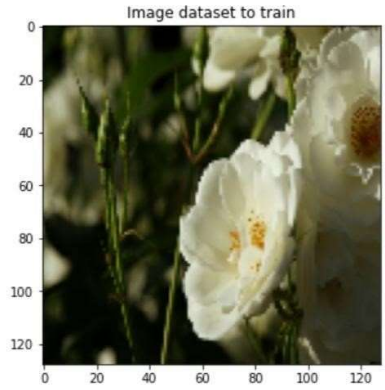



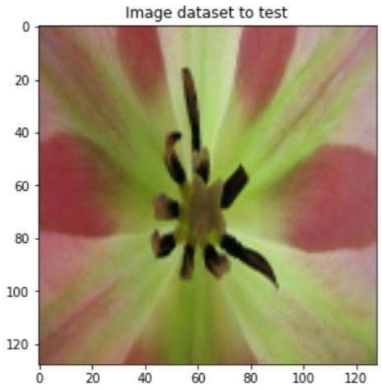

In [21]:

```
def preprocessing(image, label):
    """
    returns a image that is reshaped and normalized
    """
    image = tf.cast(image, tf.float32)
    image = image / 255.
    image = tf.image.resize(image, IMAGE_SIZE)

    return image, label


train_data_norm = train_data_img.map(preprocessing)
test_data_norm = test_data_img.map(preprocessing)
```

In [22]:

```
fig, ax = plt.subplots(1,2, figsize = (15,5))
for i,l in train_data_norm.take(1):
    ax[0].set_title('Image dataset to train');
    ax[0].imshow(i);
for i,l in test_data_norm.take(1):
    ax[1].set_title('Image dataset to test');
    ax[1].imshow(i);
```




In [23]:

```
train_batches = train_data_norm.batch(batch_size)
test_batches = test_data_norm.batch(batch_size)

for i, l in train_batches.take(1):
    print('Train Data Shape',i.shape)
for i, l in test_batches.take(1):
    print('Test Data Shape',i.shape)
```

```
Train Data Shape (32, 128, 128, 3)
Test Data Shape (32, 128, 128, 3)
```

**4 :**

**Question-**

## Add Layers (Convolution,MaxPooling,Flatten,Dense-(Hidden Layers),Output)

Solution :

**Add Layers (Convolution,MaxPooling,Flatten,Dense-(Hidden Layers),Output)**

In [24]:

```
LeNet = models.Sequential()
LeNet.add(layers.Conv2D(6, (5,5), activation = 'relu', input_shape = (128, 128, 3)))
LeNet.add(layers.MaxPooling2D())
LeNet.add(layers.Conv2D(16, (5,5), activation = 'relu'))
LeNet.add(layers.MaxPooling2D())
LeNet.add(layers.Flatten())
LeNet.add(layers.Dense(255, activation='relu'))
LeNet.add(layers.Dropout(0.2))
LeNet.add(layers.Dense(124, activation='relu'))
LeNet.add(layers.Dropout(0.2))
LeNet.add(layers.Dense(84, activation='relu'))
LeNet.add(layers.Dense(43, activation='sigmoid'))
LeNet.summary()
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 124, 124, 6)       456

 max_pooling2d (MaxPooling2D  (None, 62, 62, 6)        0
 )

 conv2d_1 (Conv2D)           (None, 58, 58, 16)        2416

 max_pooling2d_1 (MaxPooling  (None, 29, 29, 16)       0
 2D)

 flatten (Flatten)           (None, 13456)             0

 dense (Dense)               (None, 255)               3431535

 dropout (Dropout)           (None, 255)               0

 dense_1 (Dense)             (None, 124)               31744

 dropout_1 (Dropout)         (None, 124)               0

 dense_2 (Dense)             (None, 84)                10500

 dense_3 (Dense)             (None, 43)                3655

=================================================================
Total params: 3,480,306
Trainable params: 3,480,306
Non-trainable params: 0
_________________________________________________________________
```

**Question-5:**

**Compile The Model**

Solution :

**Compile The Model**

In [25]:
```
LeNet.compile(optimizer='Adam', loss='sparse_categorical_crossentropy',
              metrics = ['accuracy'])
```

**6: Fit**

**The Model**

Solution :

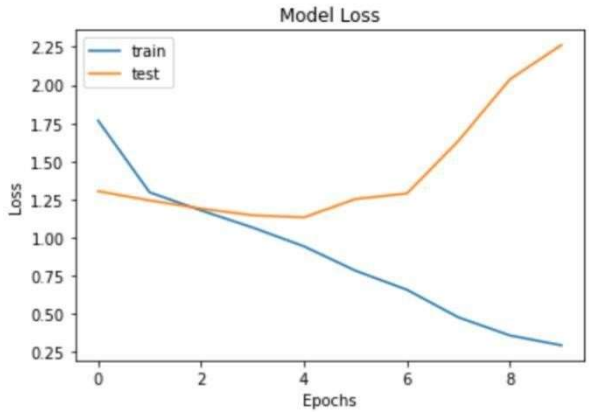
## Question-

**Fit The Model**

In [26]:
```
history = LeNet.fit(train_batches, epochs=10,batch_size = 16,validation_data=(test_batches))
```

```
Epoch 1/10
55/55 [==============================] - 130s 2s/step - loss: 1.7673 - accuracy: 0.2943 - val_loss: 1.3046 - val_accuracy: 0.4560
Epoch 2/10
55/55 [==============================] - 40s 724ms/step - loss: 1.2971 - accuracy: 0.4434 - val_loss: 1.2441 - val_accuracy: 0.4880
Epoch 3/10
55/55 [==============================] - 42s 752ms/step - loss: 1.1785 - accuracy: 0.5034 - val_loss: 1.1907 - val_accuracy: 0.5173
Epoch 4/10
55/55 [==============================] - 36s 650ms/step - loss: 1.0667 - accuracy: 0.5526 - val_loss: 1.1468 - val_accuracy: 0.5453
Epoch 5/10
55/55 [==============================] - 49s 889ms/step - loss: 0.9430 - accuracy: 0.6366 - val_loss: 1.1333 - val_accuracy: 0.5520
Epoch 6/10
55/55 [==============================] - 37s 673ms/step - loss: 0.7835 - accuracy: 0.7051 - val_loss: 1.2531 - val_accuracy: 0.5333
Epoch 7/10
55/55 [==============================] - 36s 648ms/step - loss: 0.6586 - accuracy: 0.7531 - val_loss: 1.2900 - val_accuracy: 0.5427
Epoch 8/10
55/55 [==============================] - 40s 719ms/step - loss: 0.4778 - accuracy: 0.8257 - val_loss: 1.6341 - val_accuracy: 0.5080
Epoch 9/10
55/55 [==============================] - 36s 647ms/step - loss: 0.3595 - accuracy: 0.8703 - val_loss: 2.0376 - val_accuracy: 0.4947
Epoch 10/10
55/55 [==============================] - 41s 744ms/step - loss: 0.2947 - accuracy: 0.9023 - val_loss: 2.2624 - val_accuracy: 0.4693
```

In [31]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```



**Question-7: Save the Model**

Solution :

**Save the Model**

In [32]:

```
from sklearn.neighbors import KNeighborsClassifier as KNN
import numpy as np

# Load dataset
from sklearn.datasets import load_iris
iris = load_iris()

X = iris.data
y = iris.target

# Split dataset into train and test
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.3,
                     random_state=2018)

# import KNeighborsClassifier model
knn = KNN(n_neighbors=3)

# train model
knn.fit(X_train, y_train)
```

Out[32]:

```
KNeighborsClassifier(n_neighbors=3)
```

In [30]:

```
import pickle
saved_model = pickle.dumps(knn)
knn_from_pickle = pickle.loads(saved_model)
knn_from_pickle.predict(X_test)
```

Out[30]:

```
array([0, 1, 1, 1, 0, 1, 2, 1, 2, 0, 0, 2, 2, 2, 0, 2, 2, 0, 1, 1, 1, 0,
       2, 0, 0, 2, 0, 0, 2, 1, 0, 2, 0, 1, 2, 0, 0, 0, 0, 1, 0, 2, 2, 2,
       1])
```

**Question-8:**

**Test The Model**

Solution :

**Test The Model**

In [27]:
```
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')
```

In [28]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```