In [1]:

```
import pandas as pd
import numpy as np
import os
import cv2
import matplotlib.pyplot as plt
```

In [2]:

```
from zipfile import ZipFile
ZipFile("flowers-new.zip").extractall("C:/Users/Acer")
```

In [3]:

```
X =[]
y =[]
```

In [4]:

```
def readingFiles(folderpath,foldername):
#os.listdir - get names of files inside a folder
    paths = os.path.join(folderpath, foldername)


    for img in os.listdir(paths):
        #read first file name
        filepath = os.path.join(paths, img)
        #read image
        img_read = cv2.imread(filepath, cv2.IMREAD_COLOR)
        #resize image
        img_resized = cv2.resize(img_read, (256,256))
        #covert to numpy
        img_np = np.array(img_resized)
        #add to X
        X.append(img_np)
        #add foldername as label to y
        y.append(foldername)
```

In [5]:

```
folderpath='flowers/'

flowername = os.listdir('flowers/')
```

In [6]:

```
for i in flowername:
    readingFiles(folderpath, i)
```

In [7]:

```
X
```

Out[7]:

```
[array([[[133, 135, 135],
         [138, 139, 139],
         [144, 144, 144],
         ...,
         [152, 154, 154],
         [155, 155, 155],
         [149, 149, 149]],

        [[132, 134, 134],
         [136, 138, 138],
         [142, 143, 143],
         ...,
         [152, 154, 154],
         [155, 155, 155],
         [149, 149, 149]],

        [[131, 133, 133],
         [136, 138, 138],
```

In [8]:

```
y
```

Out[8]:

```
['daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'dandelion',
 'dandelion',
 'dandelion',
 'dandelion',
 'dandelion',
```

In [9]:

```
len(X)
```

Out[9]:

```
117
```

In [10]:

```
 from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

enc = LabelEncoder()
y_le = enc.fit_transform(y)

```

In [11]:

```
enc
```

Out[11]:

```
▾ LabelEncoder
LabelEncoder()
```

In [12]:

```
y_le
```

Out[12]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4], dtype=int64)
```

In [13]:

```
y_ohe = to_categorical(y_le, num_classes=5)
X_np = np.array(X)
y_np = np.array(y_ohe)

```

In [14]:

```
y_ohe
```

Out[14]:

```
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
```

In [15]:

```
y_ohe.shape
```

Out[15]:

```
(117, 5)
```

In [16]:

```
X_np
```

Out[16]:

```
array([[[[133, 135, 135],
         [138, 139, 139],
         [144, 144, 144],
         ...,
         [152, 154, 154],
         [155, 155, 155],
         [149, 149, 149]],

        [[132, 134, 134],
         [136, 138, 138],
         [142, 143, 143],
         ...,
         [152, 154, 154],
         [155, 155, 155],
         [149, 149, 149]],

        [[131, 133, 133],
         [136, 138, 138],
```

In [17]:

```
X_np.shape
```

Out[17]:

```
(117, 256, 256, 3)
```

In [18]:

```
X_scaled = X_np / 255.0

```

In [19]:

```
X_scaled
```

Out[19]:

```
array([[[[0.52156863, 0.52941176, 0.52941176],
         [0.54117647, 0.54509804, 0.54509804],
         [0.56470588, 0.56470588, 0.56470588],
         ...,
         [0.59607843, 0.60392157, 0.60392157],
         [0.60784314, 0.60784314, 0.60784314],
         [0.58431373, 0.58431373, 0.58431373]],

        [[0.51764706, 0.5254902 , 0.5254902 ],
         [0.53333333, 0.54117647, 0.54117647],
         [0.55686275, 0.56078431, 0.56078431],
         ...,
         [0.59607843, 0.60392157, 0.60392157],
         [0.60784314, 0.60784314, 0.60784314],
         [0.58431373, 0.58431373, 0.58431373]],

        [[0.51372549, 0.52156863, 0.52156863],
         [0.53333333, 0.54117647, 0.54117647],
```

In [20]:

```
X_scaled.shape

```

Out[20]:

```
(117, 256, 256, 3)
```

In [21]:

```
y_np.shape
```

Out[21]:

```
(117, 5)
```

In [22]:

```
test_size=0.2
random_state=34
shuffle=True
stratify=y_np
```

In [23]:

```
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X_scaled, y_np, test_size=0.2, random_state=34, shuffle=True, stratify=y_np)

```

In [24]:

```
Xtrain, Xval, ytrain, yval = train_test_split(Xtrain, ytrain, test_size=0.2, random_state=34, shuffle=True, stratify=ytrain)
```

In [25]:

```
Xtrain.shape
```

Out[25]:

```
(74, 256, 256, 3)
```

In [26]:

```
Xval.shape
```

Out[26]:

```
(19, 256, 256, 3)
```

In [27]:

```
ytrain.shape
```

Out[27]:

```
(74, 5)
```

In [28]:

```
Xtest.shape
```

Out[28]:

```
(24, 256, 256, 3)
```

In [29]:

```
yval.shape
```

Out[29]:

```
(19, 5)
```

ANN#

In [30]:

```
from keras.models import Sequential
from keras.layers import Dense, Flatten
```

In [31]:

```
flowerANN = Sequential()
```

In [32]:

```
flowerANN.add(Flatten())
```

In [33]:

```
#256*256*3 - input dimensions
flowerANN.add(Dense(units=1024, activation='relu'))
#final layer - classification problem with 5 classes
flowerANN.add(Dense(units=5, activation='softmax'))
```

In [34]:

```
flowerANN.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
```

In [35]:

```
from keras.callbacks import ModelCheckpoint

mc = ModelCheckpoint(filepath='D:\\Ashwini\\IMAGE\\bestModel.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)
```

In [36]:

```
history = flowerANN.fit(Xtrain, ytrain, epochs=10, validation_data=(Xval, yval), callbacks=[mc])
```

```
Epoch 1/10
3/3 [==============================] - ETA: 0s - loss: 267.8847 - accuracy: 0.1757
Epoch 1: val_accuracy improved from -inf to 0.21053, saving model to D:\Ashwini\IMAGE\bestModel.h5
3/3 [==============================] - 21s 5s/step - loss: 267.8847 - accuracy: 0.1757 - val_loss: 475.1231 - val_accuracy: 
0.2105
Epoch 2/10
3/3 [==============================] - ETA: 0s - loss: 392.5522 - accuracy: 0.2703
Epoch 2: val_accuracy improved from 0.21053 to 0.31579, saving model to D:\Ashwini\IMAGE\bestModel.h5
3/3 [==============================] - 11s 4s/step - loss: 392.5522 - accuracy: 0.2703 - val_loss: 207.6998 - val_accuracy: 
0.3158
Epoch 3/10
3/3 [==============================] - ETA: 0s - loss: 241.7541 - accuracy: 0.3243
Epoch 3: val_accuracy did not improve from 0.31579
3/3 [==============================] - 9s 3s/step - loss: 241.7541 - accuracy: 0.3243 - val_loss: 222.8740 - val_accuracy: 
0.3158
Epoch 4/10
3/3 [==============================] - ETA: 0s - loss: 213.7950 - accuracy: 0.4595
Epoch 4: val_accuracy improved from 0.31579 to 0.36842, saving model to D:\Ashwini\IMAGE\bestModel.h5
3/3 [==============================] - 11s 4s/step - loss: 213.7950 - accuracy: 0.4595 - val_loss: 112.8398 - val_accuracy: 
0.3684
Epoch 5/10
3/3 [==============================] - ETA: 0s - loss: 117.1641 - accuracy: 0.5270
Epoch 5: val_accuracy did not improve from 0.36842
3/3 [==============================] - 7s 2s/step - loss: 117.1641 - accuracy: 0.5270 - val_loss: 100.9803 - val_accuracy: 
0.3158
Epoch 6/10
3/3 [==============================] - ETA: 0s - loss: 85.4432 - accuracy: 0.5270
Epoch 6: val_accuracy did not improve from 0.36842
3/3 [==============================] - 7s 2s/step - loss: 85.4432 - accuracy: 0.5270 - val_loss: 59.0970 - val_accuracy: 0.
3684
Epoch 7/10
3/3 [==============================] - ETA: 0s - loss: 39.6178 - accuracy: 0.5000
Epoch 7: val_accuracy did not improve from 0.36842
3/3 [==============================] - 8s 3s/step - loss: 39.6178 - accuracy: 0.5000 - val_loss: 36.6176 - val_accuracy: 0.
3684
Epoch 8/10
3/3 [==============================] - ETA: 0s - loss: 28.7426 - accuracy: 0.5541
Epoch 8: val_accuracy improved from 0.36842 to 0.42105, saving model to D:\Ashwini\IMAGE\bestModel.h5
3/3 [==============================] - 13s 5s/step - loss: 28.7426 - accuracy: 0.5541 - val_loss: 43.0471 - val_accuracy: 
0.4211
Epoch 9/10
3/3 [==============================] - ETA: 0s - loss: 25.9535 - accuracy: 0.6351
Epoch 9: val_accuracy improved from 0.42105 to 0.52632, saving model to D:\Ashwini\IMAGE\bestModel.h5
3/3 [==============================] - 11s 4s/step - loss: 25.9535 - accuracy: 0.6351 - val_loss: 23.2375 - val_accuracy: 
0.5263
Epoch 10/10
3/3 [==============================] - ETA: 0s - loss: 6.9328 - accuracy: 0.8378
Epoch 10: val_accuracy did not improve from 0.52632
3/3 [==============================] - 8s 2s/step - loss: 6.9328 - accuracy: 0.8378 - val_loss: 38.5553 - val_accuracy: 0.4
211
```

In [37]:

```
#ANN architecture

flowerANN.summary()
```

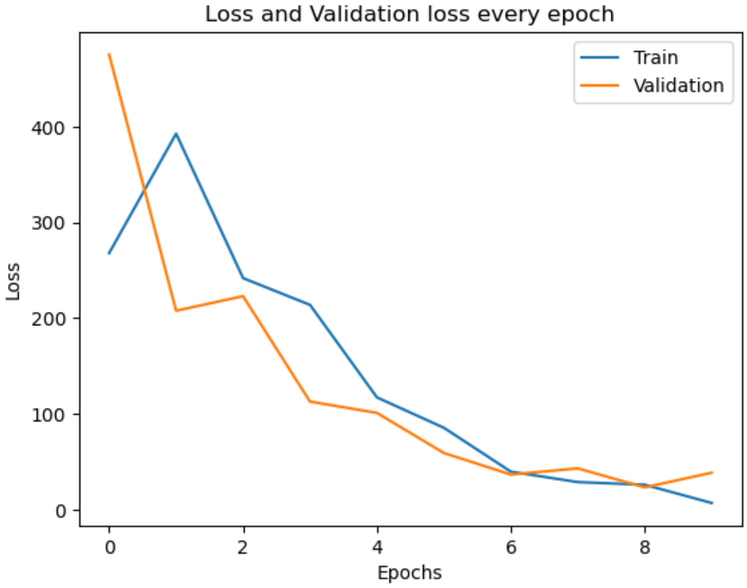
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 196608)            0         
                                                                 
 dense (Dense)               (None, 1024)              201327616 
                                                                 
 dense_1 (Dense)             (None, 5)                 5125      
                                                                 
=================================================================
Total params: 201,332,741
Trainable params: 201,332,741
Non-trainable params: 0
_________________________________________________________________
```

In [38]:

```
import matplotlib.pyplot as plt
```

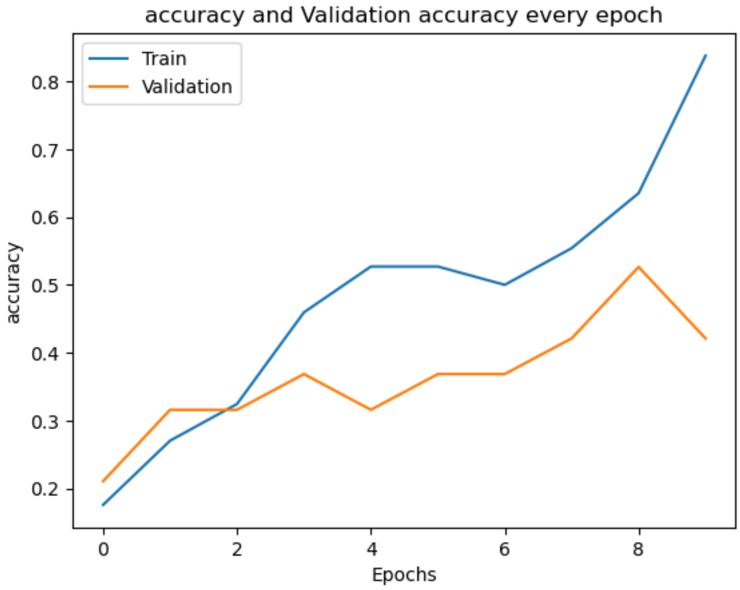
In [39]:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.title('Loss and Validation loss every epoch')
plt.show()
```



In [40]:

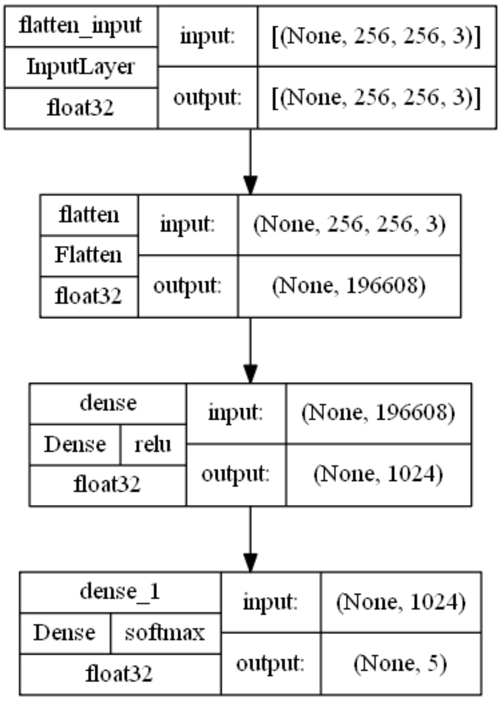
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('Epochs')
plt.ylabel('accuracy')
plt.legend(['Train', 'Validation'])
plt.title('accuracy and Validation accuracy every epoch')
plt.show()
```



In [41]:

```
from tensorflow.keras.utils import plot_model
plot_model(flowerANN, show_shapes=True, show_dtype=True, show_layer_activations=True, show_layer_names=True)
```

Out[41]:



In [42]:

```
ypred = flowerANN.predict(Xtest)
```

```
1/1 [==============================] - 0s 341ms/step
```

In [43]:

```
Xtest[0].shape
```

Out[43]:

```
(256, 256, 3)
```

In [44]:

```
sampleimage = np.reshape(Xtest[0],(1,256,256,3))
```

In [45]:

```
sampleimage.shape
```

Out[45]:

```
(1, 256, 256, 3)
```

In [46]:

```
ypred_first = flowerANN.predict(sampleimage)
```

```
1/1 [==============================] - 0s 73ms/step
```

In [47]:

```
ypred_first
```

Out[47]:

```
array([[0.0000000e+00, 0.0000000e+00, 1.0000000e+00, 3.2733997e-15,
        0.0000000e+00]], dtype=float32)
```

In [48]:

```
ypredclasses_first = np.argmax(ypred_first, axis=-1)
ypredclasses_first
```

Out[48]:

```
array([2], dtype=int64)
```

In [49]:

```
import numpy as np
ypredclasses = np.argmax(ypred, axis=-1)
```

In [50]:

```
ypredclasses
```

Out[50]:

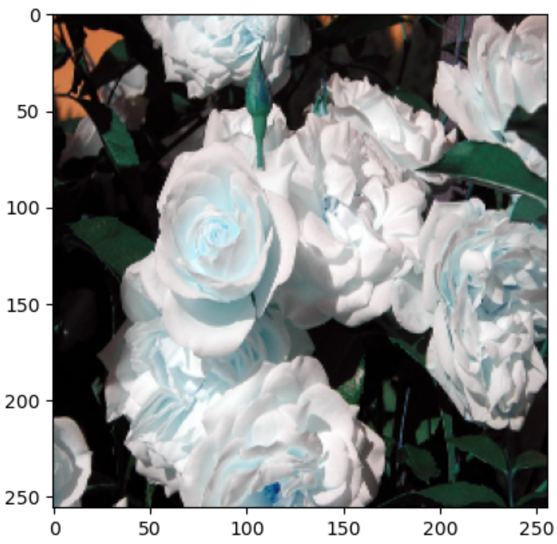
```
array([2, 2, 2, 3, 3, 2, 2, 3, 2, 0, 2, 2, 3, 3, 3, 2, 2, 2, 2, 3, 2, 2,
       3, 2], dtype=int64)
```

In [51]:

```
plt.imshow(Xtest[9])
```

Out[51]:

```
<matplotlib.image.AxesImage at 0x1b31aa55760>
```



In [52]:

```
ytest[1]
```

Out[52]:

```
array([0., 1., 0., 0., 0.], dtype=float32)
```

In [53]:

```
ypredclasses[1]
```

Out[53]:

```
2
```

In [54]:

```
##/content//bestmodel.h5   #having error
```

In [55]:

```
flowerANN.evaluate(Xtest, ytest)
```

```
1/1 [==============================] - 0s 125ms/step - loss: 62.6970 - accuracy: 0.3750
```

Out[55]:

```
[62.697017669677734, 0.375]
```

In [56]:

```
from keras.models import load_model
bestmodel = load_model("D:\\Ashwini\\IMAGE\\bestModel.h5")
```

In [57]:

```
bestmodel.evaluate(Xtest, ytest)
```

```
1/1 [==============================] - 1s 975ms/step - loss: 30.5506 - accuracy: 0.4167
```

Out[57]:

```
[30.550600051879883, 0.4166666567325592]
```

In [ ]:

```

```