Flower Detection and Recognition using Convolution Neural Network

Project

Submitted To: **Ma'am Mehreen Sirshar**

Class: **BSE-6A**

Submitted By:

Aatiqa Ejaz **(2017-BSE-001)**

Hamna Baqai **(2017-BSE-009)**

Introduction:

Identifying and recognizing the type of flower is the basic requirement for researches done in the field of Botany as well as other areas. Flowers identification is done by color, shape, texture or any patterns it contain.

Working with CNN:

An image of flower is input, that we're going to train out model using CNN. Convolutional Neural Network (CNN), also known as convents was proposed by Yann LeCun in 1988, it is used to identify any object from any image. Convolutional neural network has different architecture than other neural networks. Input given to the CNN algorithm is basically image to be classified and output generated by algorithm is set of features. Features of image means that the characteristics of image for instance flower's image has features like sepals, petals, stem and leaves. CNN is classified into four components named as Convolution, Non Linearity (ReLU), Sub-Sampling or Pooling and Fully Connected Layer.

Convolution (Convolutional Layer) is responsible for identifying and recognizing the low level features of objects such as edges, corners etc. from an image. It means the whole network will learn a specific pattern and will identify objects according to the pattern in any image. Convolution is basically the input image using a filter (usually a 3x3 matrix) or a kernel so for convolution the image is scanned from top left corner to the right moving covering the whole width of the image and then repeating the same process until we are done with the whole image. The filter multiplies it's all values with corresponding values of input image and add all of them making only value to output image unit till the entire image is traversed. The element-wise multiplication generates the output commonly known as feature map. In case of image having multiple channels matrix multiplication is performed in between all the images with three different filters (red, green and blue) and all the results are then summed up to give final output pixel value. Each neuron in output value has overlapping receipted field. After passing through Convolutional Layer the size of image is reduced. Inorder to regain the size same as input image padding is done. Furthermore the Convolutional Layer is organized by three parameters Depth (defines the total number of filters applied during the process of Convolution), Stride (defines the number of pixels to be jumped i.e. 1 pixel, 2 pixels or more) and Zero-Padding (process of adding rows and columns on each side of the feature map).

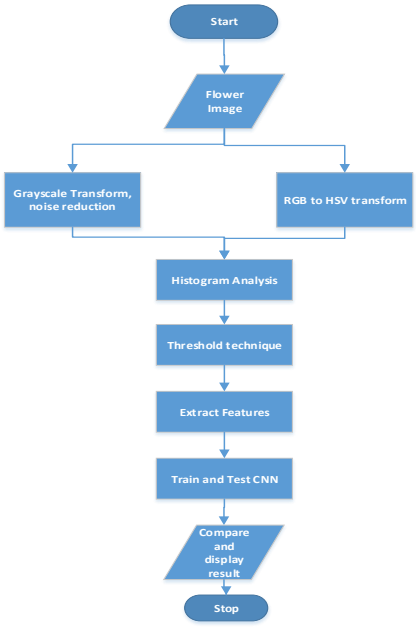
After passing the input image through filter the resulting image is passed through a function called activation function, usually activation function known as **ReLU (Rectified Linear Unit)** is used which transforms all the negative pixels value to 0 and other positive pixels values remains the similar.

After passing the input image from the convolutional layer we get the feature map then **Pooling or Sub-Sampling Layer** works on image. Similarly like convolutional layer the responsibility of pooling layer is to reducing the spatial size of the features. It is used to extract dominant features which are rotational and positional variants. Commonly Pooling is of two types one of it is max pooling and the other one is average pooling. **Max pooling** returns the max value from the slice

of the image convolved by the filter, it also work as noise sub percent it discards the noise from an image. **Average pooling** returns the average value from the slice of the image convolved by the filter.

The Fully Connected Layer is a traditional Multi-Layer Perceptron. And basically is an activation function known as classifier. In Fully connected Layer each neuron present in the previous layer is connected to every neuron present in the next layer, this layer uses the output generated from previous layer to identify image.

Project Flow Chart:



Dataset:

The dataset had 18540 images which we used for training, 2009 images for testing, similarly separate excel files for training as well as testing. We will be classifying images in 102 categories based on their groups they fall in. For instance image number 0, 10, 16, 45 and many more and fall in category 77. Data was divided 60 40 for training and testing but then we also tried with 80 20 and 70 30. Our CNN has 4 layers and has a training accuracy of 92%.

Coding:

Following are the libraries used:

In [1]:

```
import warnings
warnings.filterwarnings("ignore")
import os, cv2, random
import numpy as np
import pandas as pd
%pylab inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from matplotlib import ticker
import seaborn as sns
%matplotlib inline

from keras.models import Sequential
from keras.optimizers import RMSprop
from keras.layers import Input, Dropout, Flatten, Convolution2D, MaxPooling2D, Dense, Activation
from keras.utils import np_utils, to_categorical
from keras.callbacks import ModelCheckpoint, Callback, EarlyStopping

```

```
Populating the interactive namespace from numpy and matplotlib
```

Initially to store path of file, simply copy the destination of folder which contains the dataset and paste it in a variable.

Reading images from train file and storing them in a variable. Displayed only few of them

In [2]:

```
#Path to reach files
path = r'C:\Users\hamna\Desktop\flowers\files'
#check to see path is correct or not
print("Path is ",path)

#read excel file using panda
label_as_train = pd.read_csv(path + '\\train.csv')
#.head returns first 5 names by default

print("first 5 of Train file")

label_as_train.head()

```

```
Path is  C:\Users\hamna\Desktop\flowers\files
first 5 of Train file
```

Out[2]:

	image_id	category
0	0	77
1	1	81
2	2	52
3	3	72
4	4	58

Counting the number of images in train as well as test files

In [3]:

```
#Read Number of rows from train excel file
nRowsRead = None
#None will read the whole file
df1 = pd.read_csv(r'C:\Users\hamna\Desktop\flowers\files\train.csv', delimiter=',', nrows = nRowsRead)
df1.dataframeName = 'train.csv'
nRow, nCol = df1.shape
print(f'There are {nRow} rows and {nCol} columns in Train File')
```

```
There are 18540 rows and 2 columns in Train File
```

In [4]:

```
#Read Number of rows from test excel file
nRowsRead =None
df2 = pd.read_csv(r'C:\Users\hamna\Desktop\flowers\files\test.csv', delimiter=',', nrows = nRowsRead)
df2.dataframeName = 'test.csv'
nRow, nCol = df2.shape
print(f'There are {nRow} rows and {nCol} columns')

```

```
There are 2009 rows and 2 columns
```

We have 102 category images. Now to uniquely find out the image categories,

In [6]:

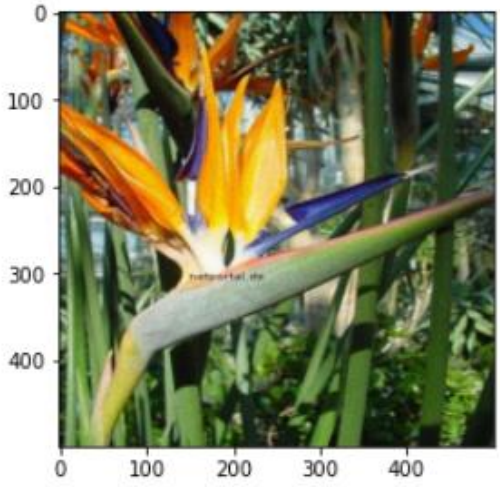
```
#obtain the number of catgories available using unique method from numpy
unique_label_as_train = np.unique(label_as_train['category'].values)
print(unique_label_as_train)
```

```
[  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
  91  92  93  94  95  96  97  98  99 100 101 102]
```

To view that images are correctly loading

In [8]:

```
# View a single mage to see
img=mpimg.imread(path + '/train/3260.jpg')
imgplot = plt.imshow(img)
plt.show()
```



This can also be done for more images. To store path of all images in a variable and checking first ten stored using slicing method

```
In [9]:
# train image names along with paths
train_image_name = [path+'\\train\\'+str(each)+'.jpg' for each in label_as_train['image_id'].values.tolist()]
train_image_name[:10]
```

```
Out[9]: ['C:\\Users\\hamna\\Desktop\\flowers\\files\\train\\0.jpg',
 'C:\\Users\\hamna\\Desktop\\flowers\\files\\train\\1.jpg',
 'C:\\Users\\hamna\\Desktop\\flowers\\files\\train\\2.jpg',
 'C:\\Users\\hamna\\Desktop\\flowers\\files\\train\\3.jpg',
 'C:\\Users\\hamna\\Desktop\\flowers\\files\\train\\4.jpg',
 'C:\\Users\\hamna\\Desktop\\flowers\\files\\train\\5.jpg',
 'C:\\Users\\hamna\\Desktop\\flowers\\files\\train\\6.jpg',
 'C:\\Users\\hamna\\Desktop\\flowers\\files\\train\\7.jpg',
 'C:\\Users\\hamna\\Desktop\\flowers\\files\\train\\8.jpg',
 'C:\\Users\\hamna\\Desktop\\flowers\\files\\train\\9.jpg']
```

```
In [10]:
test_label = pd.read_csv(path + '\\test.csv')
test_label[:5]
```

Out[10]:

	image_id	category
0	18540	NaN
1	18541	NaN
2	18542	NaN
3	18543	NaN
4	18544	NaN

Similarly for testing images

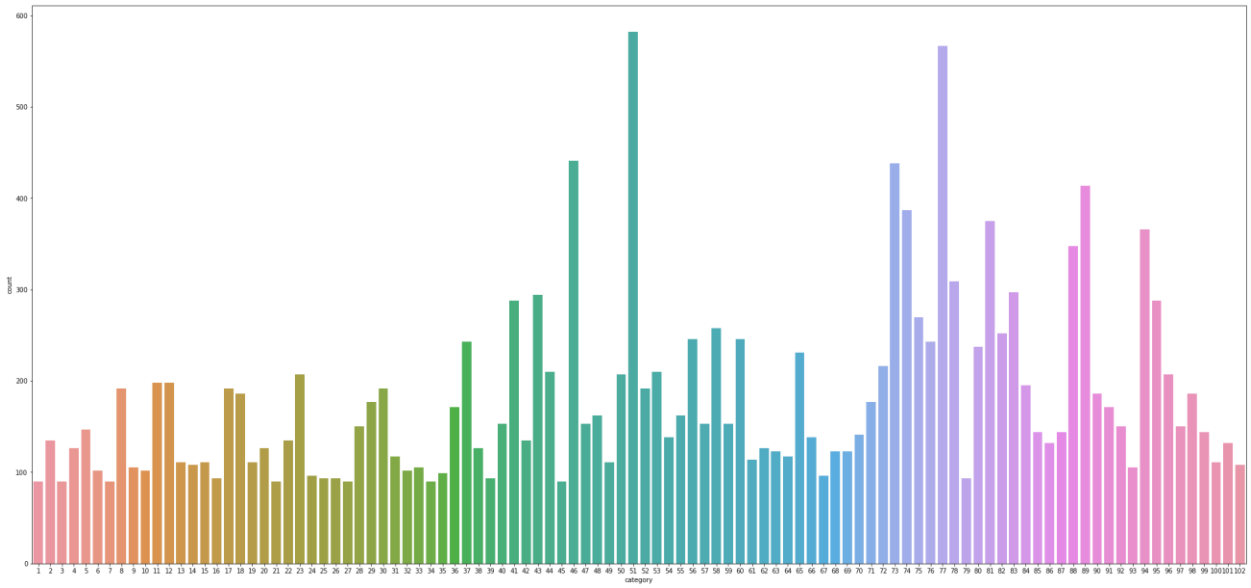
```
In [11]:
# test image names along with paths
test_image_name = [path+'\\test\\'+str(each)+'.jpg' for each in test_label['image_id'].values.tolist()]
test_image_name[:5]
```

```
Out[11]: ['C:\\Users\\hamna\\Desktop\\flowers\\files\\test\\18540.jpg',
 'C:\\Users\\hamna\\Desktop\\flowers\\files\\test\\18541.jpg',
 'C:\\Users\\hamna\\Desktop\\flowers\\files\\test\\18542.jpg',
 'C:\\Users\\hamna\\Desktop\\flowers\\files\\test\\18543.jpg',
 'C:\\Users\\hamna\\Desktop\\flowers\\files\\test\\18544.jpg']
```

To create Histogram as mentioned in the flow chart

```
In [7]:
#view graph using matploit using figure method
plt.figure(figsize=(34, 16))
#using seaborn to view statistical data
sns.countplot(label_as_train['category'])
plt.show()
```

Following graph shows images according to the categories. For instance an image appeared 10 times in a particular category, so the number of times is shown on y axes and x axes show categories.



Processing data i.e images.

In [30]:

```
# processing  and preparing data using opencv
ROWS = 40
COLS = 40
CHANNELS = 1
#reading images
def read_image(file_path):
    img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE) #cv2.IMREAD_GRAYSCALE
    return cv2.resize(img, (ROWS, COLS), interpolation=cv2.INTER_CUBIC)
#preparing images
def prep_data(images):
    count = len(images)
    data = np.ndarray((count, CHANNELS, ROWS, COLS), dtype=np.uint8)

    for i, image_file in enumerate(images):
        image = read_image(image_file)
        data[i] = image.T
        if i%2000 == 0: print('Processed {} of {}'.format(i, count))
    return data
#print(train)
train = prep_data(train_image_name)
test = prep_data(test_image_name)
```

```
Processed 0 of 18540
Processed 2000 of 18540
Processed 4000 of 18540
Processed 6000 of 18540
Processed 8000 of 18540
Processed 10000 of 18540
Processed 12000 of 18540
Processed 14000 of 18540
Processed 16000 of 18540
Processed 18000 of 18540
Processed 0 of 2009
Processed 2000 of 2009
```

Creating CNN Model

In [25]:

```
optimizer = RMSprop(lr=1e-4)
objective = 'categorical_crossentropy'

def flower_recognition():
    our_model = Sequential()
#1st argument, set to 32 is output filter value, 2nd argument is the kernal size ,
#3rd argument has batch, height, width, 4th argument is same to have padding results of output like that of input

# 1: Convolution layer with 32 filters
    our_model.add(Convolution2D(32, 3, 3, border_mode='same', input_shape=(1, ROWS, COLS), activation='relu'))
    our_model.add(Convolution2D(32, 3, 3, border_mode='same', activation='relu'))
    our_model.add(MaxPooling2D(pool_size=(2, 2), dim_ordering="th"))
# 2 Convolution layer with 64 filters
    our_model.add(Convolution2D(64, 3, 3, border_mode='same', activation='relu'))
    our_model.add(Convolution2D(64, 3, 3, border_mode='same', activation='relu'))
    our_model.add(MaxPooling2D(pool_size=(2, 2), dim_ordering="th"))
#3 Convolution layer with 128 filters
    our_model.add(Convolution2D(128, 3, 3, border_mode='same', activation='relu'))
    our_model.add(Convolution2D(128, 3, 3, border_mode='same', activation='relu'))
    our_model.add(MaxPooling2D(pool_size=(2, 2), dim_ordering="th"))
#4 Convolution layer with 256 filters
    our_model.add(Convolution2D(256, 3, 3, border_mode='same', activation='relu'))
    our_model.add(Convolution2D(256, 3, 3, border_mode='same', activation='relu'))
    our_model.add(Convolution2D(256, 3, 3, border_mode='same', activation='relu'))
    our_model.add(MaxPooling2D(pool_size=(2, 2), dim_ordering="th"))
    our_model.add(Flatten())
    our_model.add(Dense(256, activation='relu'))
    our_model.add(Dense(103, activation='softmax'))
    #our_model.add(Activation('softmax'))

    our_model.compile(loss=objective, optimizer='adam', metrics=['accuracy'])
    return our_model
our_model = flower_recognition()
```

In [16]:

```
our_model.fit(train, labs, batch_size=batch_size, epochs=nb_epoch,
              validation_split=0.25, verbose=1, shuffle=True, callbacks=[history, early_stopping])
```

```
Train on 13905 samples, validate on 4635 samples
Epoch 1/10
13905/13905 [==============================] - 138s 10ms/step - loss: 4.5046 - accuracy: 0.0308 - val_loss: 4.4220 - val_acc
uracy: 0.0317
Epoch 2/10
13905/13905 [==============================] - 139s 10ms/step - loss: 4.3458 - accuracy: 0.0398 - val_loss: 4.3148 - val_acc
uracy: 0.0462
Epoch 3/10
13905/13905 [==============================] - 140s 10ms/step - loss: 4.2452 - accuracy: 0.0513 - val_loss: 4.2004 - val_acc
uracy: 0.0537
Epoch 4/10
13905/13905 [==============================] - 134s 10ms/step - loss: 4.1667 - accuracy: 0.0606 - val_loss: 4.1963 - val_acc
uracy: 0.0589
Epoch 5/10
13905/13905 [==============================] - 122s 9ms/step - loss: 4.0975 - accuracy: 0.0718 - val_loss: 4.0936 - val_accu
racy: 0.0749
Epoch 6/10
13905/13905 [==============================] - 121s 9ms/step - loss: 4.0321 - accuracy: 0.0772 - val_loss: 4.1476 - val_accu
racy: 0.0634
Epoch 7/10
13905/13905 [==============================] - 121s 9ms/step - loss: 3.9721 - accuracy: 0.0870 - val_loss: 4.1073 - val_accu
racy: 0.0706
Epoch 8/10
13905/13905 [==============================] - 121s 9ms/step - loss: 3.9195 - accuracy: 0.0900 - val_loss: 4.0007 - val_accu
```

```
Epoch 7/10
13905/13905 [==============================] - 121s 9ms/step - loss: 3.9721 - accuracy: 0.0870 - val_loss: 4.1073 - val_accu
racy: 0.0706
Epoch 8/10
13905/13905 [==============================] - 121s 9ms/step - loss: 3.9195 - accuracy: 0.0900 - val_loss: 4.0007 - val_accu
racy: 0.0775
Epoch 9/10
13905/13905 [==============================] - 1776s 128ms/step - loss: 3.8636 - accuracy: 0.1005 - val_loss: 4.0087 - val_a
ccuracy: 0.0820
Epoch 10/10
13905/13905 [==============================] - 139s 10ms/step - loss: 3.7942 - accuracy: 0.1106 - val_loss: 3.9358 - val_acc
uracy: 0.0848
```

Out[16]: <keras.callbacks.callbacks.History at 0x1fd3765dec8>

```
In [17]:
1 predictions = our_model.predict(test)

In [18]:
1 max(predictions[0])

Out[18]: 0.06105118
```

```
In [33]:
1 predicted = []
2 for i in range((len(predictions))):
3     predicted.append(np.argmax(predictions[i]))

In [34]:
1 test_label['category'] = predicted
2 test_label.head()
```

Out[34]:

	image_id	category
0	18540	95
1	18541	74
2	18542	97
3	18543	46
4	18544	78

But when we repeatedly apply CNN to our model, increase epochs, change validation size, our accuracy of training increases but validation accuracy decreases.

```
In [58]:
1 model.fit(train, labs, batch_size=batch_size, epochs=nb_epoch,
2           validation_split=0.30, verbose=1, shuffle=True, callbacks=[history, early_stopping])
3 model.save('FlowerRecognitionCNN.h5')

Train on 12978 samples, validate on 5562 samples
Epoch 1/30
12978/12978 [==============================] - 185s 14ms/step - loss: 0.1981 - accuracy: 0.9409 - val_loss: 11.8490 - val_ac
curacy: 0.2159
Epoch 2/30
12978/12978 [==============================] - 183s 14ms/step - loss: 0.1740 - accuracy: 0.9470 - val_loss: 11.8847 - val_ac
curacy: 0.2120
Epoch 3/30
12978/12978 [==============================] - 183s 14ms/step - loss: 0.1778 - accuracy: 0.9447 - val_loss: 11.8090 - val_ac
curacy: 0.2208
Epoch 4/30
12978/12978 [==============================] - 184s 14ms/step - loss: 0.2329 - accuracy: 0.9325 - val_loss: 11.3870 - val_ac
curacy: 0.2129
Epoch 5/30
12978/12978 [==============================] - 120s 9ms/step - loss: 0.1032 - accuracy: 0.9679 - val_loss: 12.2817 - val_acc
uracy: 0.2175
Epoch 6/30
12978/12978 [==============================] - 118s 9ms/step - loss: 0.1139 - accuracy: 0.9636 - val_loss: 12.7304 - val_acc
uracy: 0.2102
Epoch 7/30
12978/12978 [==============================] - 118s 9ms/step - loss: 0.2749 - accuracy: 0.9236 - val_loss: 11.5637 - val_acc
uracy: 0.2068
Epoch 00007: early stopping
```

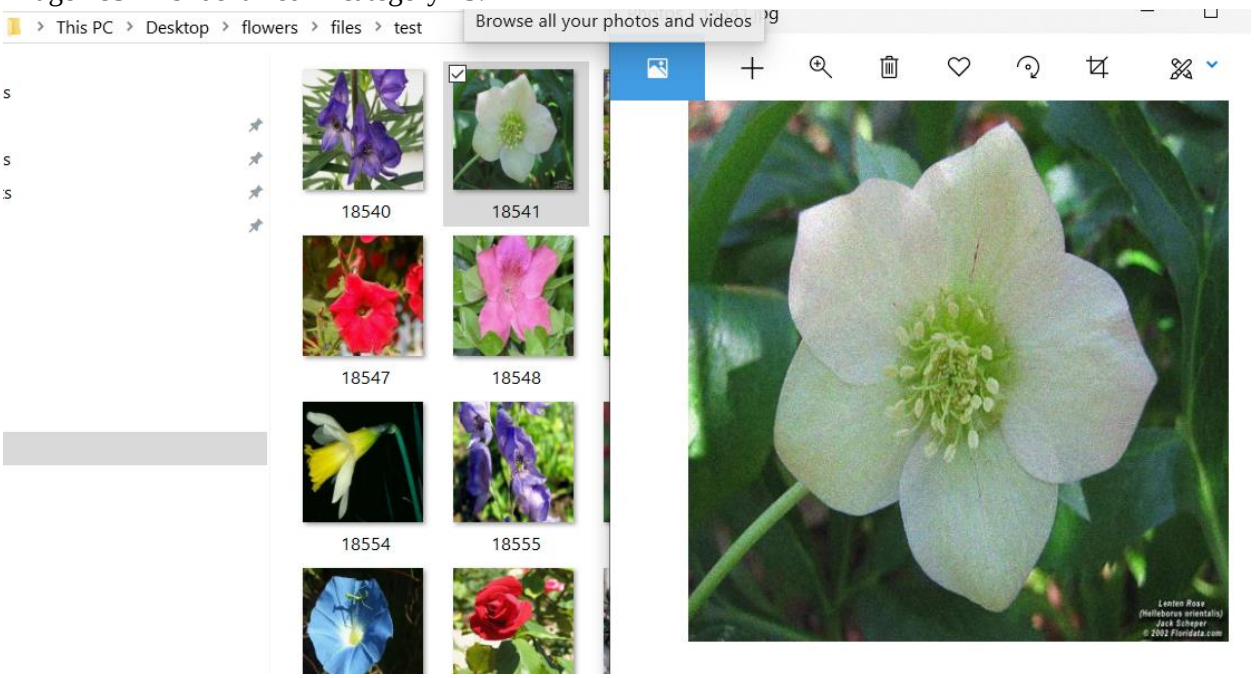
When changing batch to 128 we got training accuracy up to 100%.

```
In [166]:
1 our_model.fit(train, labs, batch_size=batch_size, epochs=nb_epoch,
2               validation_split=0.11, verbose=1, shuffle=True, callbacks=[history, early_stopping])
3 our_model.save('FlowerRecognitionCNN.h5')

Train on 16500 samples, validate on 2040 samples
Epoch 1/40
16500/16500 [==============================] - 130s 8ms/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 13.6152 - val_acc
uracy: 0.1603
Epoch 2/40
16500/16500 [==============================] - 122s 7ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 13.8372 - val_acc
uracy: 0.1618
Epoch 3/40
16500/16500 [==============================] - 120s 7ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 14.0373 - val_acc
uracy: 0.1637
Epoch 4/40
16500/16500 [==============================] - 120s 7ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 14.1927 - val_acc
uracy: 0.1647
Epoch 00004: early stopping
```

Checking for Images: Incorrect vs Correct Images prediction

Image 18541 is identified in category 75.



Other images of this category in training set include, which do not match, so CNN is not predicting the exact flower from this category. This is because the validation accuracy is varying from 15 to 20 %.

It is working for same cases like the flower in above image is in category 5 as predicted in test and also in category 5 as in trained images.

In [156]:
```
#Code to see images for a particular category as predicted by model
n=0
df2 = pd.read_csv(r'C:\Users\hamna\Desktop\flowers\files\test.csv', delimiter=',', nrows = nRowsRead)
path=r'C:\Users\hamna\Desktop\flowers\files'
test_label['category'] = predicted
#test_label.head()
x=int(input("Enter a category to check images"))
while n < len(test_label):
    if test_label.iloc[n]['category'] == x:
        y=test_label.iloc[n]['image_id']
        imag=mpimg.imread(path +'\\test\\'+str(y)+'.jpg')
        imgplot = plt.imshow(imag)
        plt.show()
    n += 1
```

In [*]:
```
#Shows all images which belong to category eneterd by user in trained file
n=0
x=""
cate=int(input("Enter a category from 1-102 "))
df1 = pd.read_csv(r'C:\Users\hamna\Desktop\flowers\files\train.csv', delimiter=',', nrows = nRowsRead)
path=r'C:\Users\hamna\Desktop\flowers\files'
#print(df1.loc[(df1[cate])].index[0])
while n < len(df1.index):
    if df1.iloc[n]['category'] == cate:
        x=df1.iloc[n]['image_id']
        imag=mpimg.imread(path +'\\train\\'+str(x)+'.jpg')
        imgplot = plt.imshow(imag)
        plt.show()
    n += 1

```

Conclusion:

Accuracy of training increased to 100 percent but validation accuracy remained at 20%. Due to the large number of categories in dataset, it has become very difficult for the network to correctly identify images into those categories. The training accuracy is far larger than the validation or performance accuracy. It is only because categories are in a large number. But in spite of the fact, some images are predicted in correct category. To obtain more better validation accuracy, we can have more training images and more layers in CNN.