Mini Project Report

On

Prediction of Same Color Flowers from a Garden

of

Tools and Techniques Laboratory(CS3096)

Submitted by
Ayantika Bhaumik(1928021)
Ayush Saha(1928022)
Bibhu Prasanna Dash(1928023)
Bishop Bhaumik(1928024)
Dharani Chandra(1928026)
of

School of Computer Engineering

To
B.Tech Program in Computer Science Engineering
Kalinga Institute of Industrial Technology, Deemed to be University
Bhubaneswar, India
April 2022

INTRODUCTION:

Flower is a very important part of nature. Mostly we identify a plant through its flower.

Experienced botanists do this identification of flower but a naive person will have to consult flower guidebooks or browse any relevant web pages on the Internet through keywords searching.

Everyday we see a huge number of flower species in our house, parks, roadsides, in farms, on our rooftop but we have no knowledge of that flower species or their origin. Even we have no idea about its name. There are several guidebooks for flowers knowledge but it becomes quite difficult to find the name when have thepicture. Even the Internet sometimes is not useful. But it is quite difficult for human brain to memorize all the species they see. Even some flower is similar to look at.Our project works on the prediction of same colours of flowers using Machine Learning.

ABOUT THE PROJECT:

Our project topic is to predict the same color flower in a garden. In this project we will extract colors from an image using KMeans algorithm and then use this to search images based on colors.

We have used different libraries for this project. We have imported basic libraries like ***matplotlib.pyplot*** and ***numpy***.To use OpenCV which is a library to solve computer vision problems, we used ***cv2***. ***KMeans algorithm*** is used when we have unlabeled data and is a part of the ***sklearn's cluster*** subpackage.To compare colors we first convert them to lab using ***rgb2lab*** and then calculate similarity using ***deltaE_cie76.*** ***Counter*** is used to extract the count from the collection library. For combining paths while reading files from directory ***os*** is imported .
KMeans clustering tries to group similar kinds of items in the form of cluster.KMeans algorithm creates clusters based on the supplied count of clusters. In our case, it will form clusters of colors and these clusters will be our top colors. We then ***fit*** and ***predict*** on the same image to extract the prediction into the variable ***labels***. We will use ***Counter*** to count all the labels. Firstly we have extracted the color in ***RGB*** format.We use the method ***rgb2lab*** to convert the selected color to a format we can compare. This is how matching images with color works.

A function is defined which will iterate over all images and will filter them based on the color and displays on the screen using ***imshow.*** This is how the selected images are shown.
Results are filtered according to the selected color like green , blue yellow and will be shown respectively.

FLOWCHART /ALGORITHM:

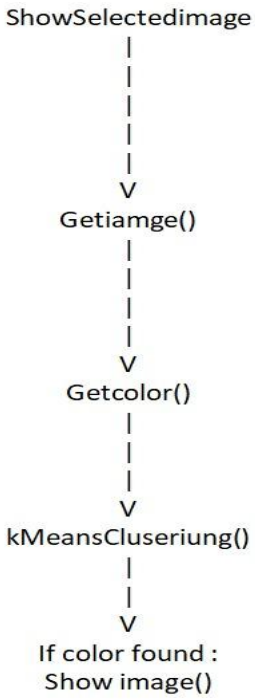


Fig1:Flowchart of project

Algorithm for K-Means:-

Algorithm 1 k-means algorithm

1: Specify the number k of clusters to assign.
2: Randomly initialize k centroids.
3: **repeat**
4: **expectation:** Assign each point to its closest centroid.
5: **maximization:** Compute the new centroid (mean) of each cluster.
6: **until** The centroid positions do not change.

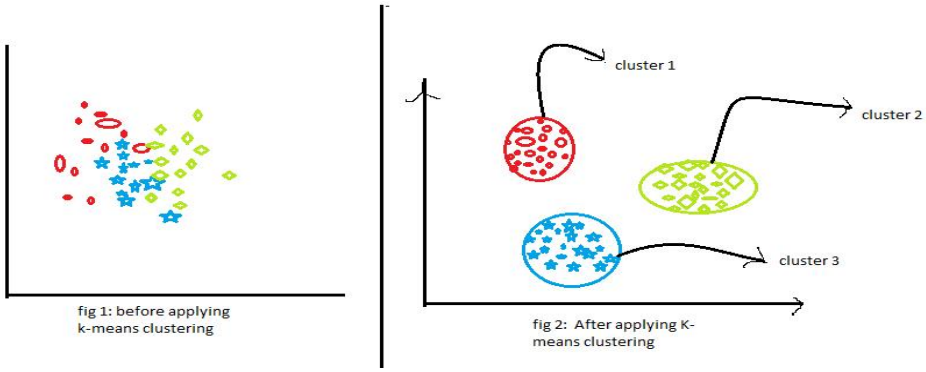


Fig2: Working of K-Means Clustering

RESULTS:

This project uses machine learning to extract a set of numbers of colors from an image.

By default ***OpenCV*** reads the images in the color order BLUE GREEN RED i.e. BGR. Thus, we need to convert it into REG GREEN BLUE i.e. RGB.

We extracted the major colors. This create the opportunity to search for images based on certain colors. We selected a color and if it's hex matches or is close to the hex of the major colors of the image, we say it's a match.

We kept a threshold value such that if the difference between the chosen color and any of the selected colors is less than that threshold, we declare it as a match.

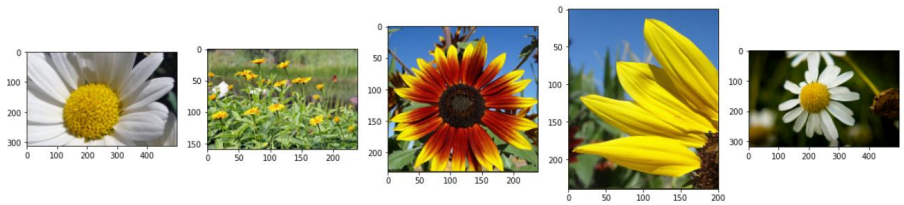
For the test data we have taken 5 sample images for classification. Upon testing the test using the algorithm that we have implemented the accuracy was found out to be 80% for the images with yellow color, white color and blue color.

This how we got the result.

Yellow Colour colour flower sample 5 flowers

In [14]:

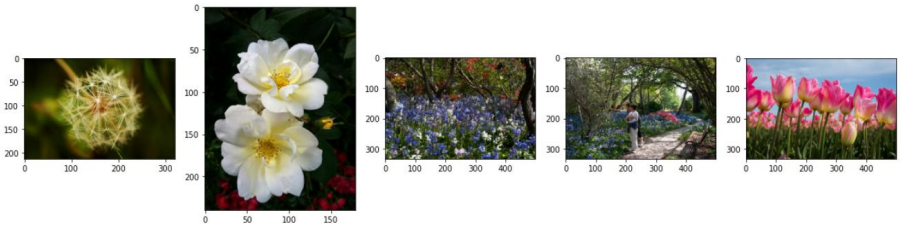
```
plt.figure(figsize = (20, 10))
show_selected_images(images,COLORS['YELLOW'], 60, 5)
```



white colour sample

In [19]:

```
plt.figure(figsize = (20, 10))
show_selected_images(images,COLORS['WHITE'], 60, 5)
```



In [20]:

```
plt.figure(figsize = (20, 10))
show_selected_images(images,COLORS['BLUE'], 60, 5)
```

RESULT DISCUSSION:

In this project we have used the K Means algorithm to extract the color from the images and hence classify them accordingly. One limitation of this model is that the major portion of the image should have the flower or else it can be classified in a faulty way. This is because according to our model we have extracted the colors from the image and then classified the image based on the majority color. Thus, if the majority part of the image has the flower itself then it will be classified correctly or else there will be inaccuracy.

In [1]:

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
import cv2
from collections import Counter
from skimage.color import rgb2lab, deltaE_cie76
import os

%matplotlib inline
```

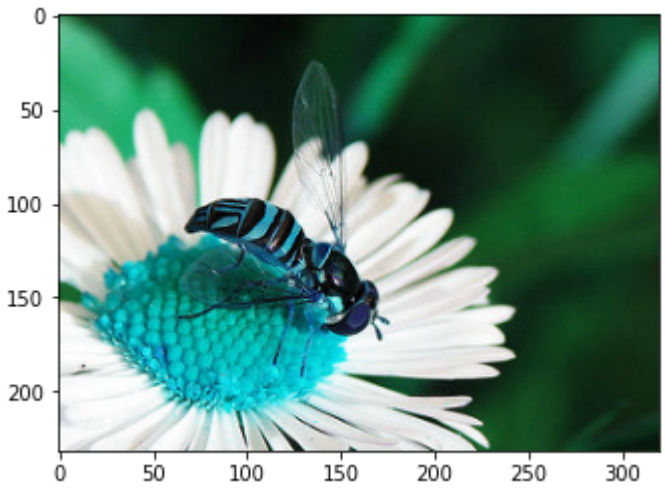
In [2]:

```
image = cv2.imread('images/sample.jpg')
print("The type of this input is {}".format(type(image)))
print("Shape: {}".format(image.shape))
plt.imshow(image)
```

```
The type of this input is <class 'numpy.ndarray'>
Shape: (232, 320, 3)
```

Out[2]:

```
<matplotlib.image.AxesImage at 0x1febb99e250>
```

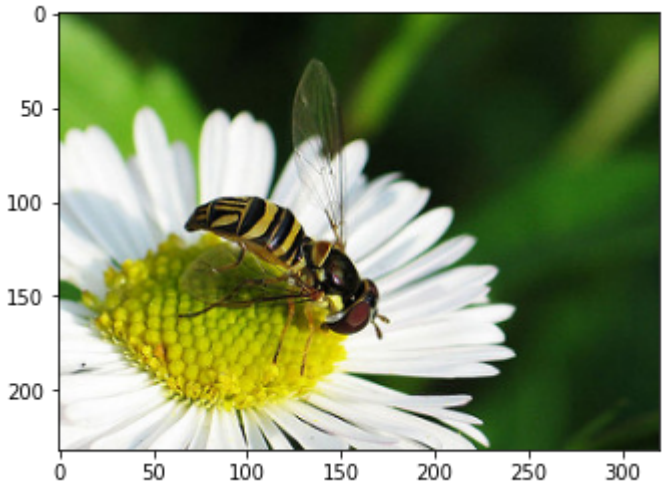


In [3]:

```
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image)
```

Out[3]:

```
<matplotlib.image.AxesImage at 0x1febdbcf2e0>
```

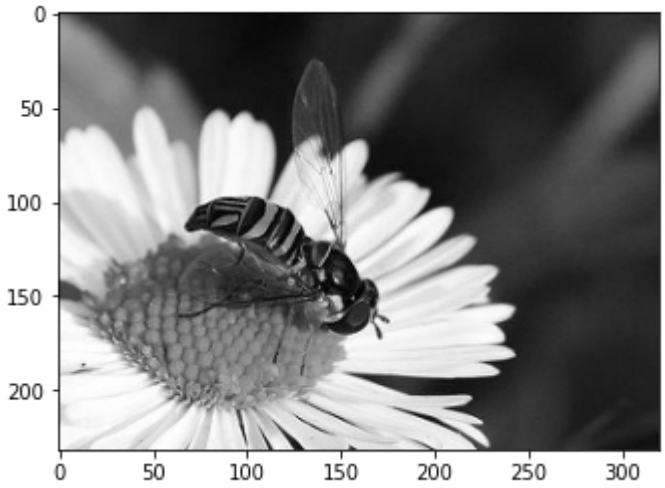


In [4]:

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
plt.imshow(gray_image, cmap='gray')
```

Out[4]:

```
<matplotlib.image.AxesImage at 0x1febdc25a90>
```



In [5]:

```
resized_image = cv2.resize(image, (1200, 600))
plt.imshow(resized_image)
```

Out[5]:

```
<matplotlib.image.AxesImage at 0x1febdc8bca0>
```

In [6]:

```
def RGB2HEX(color):
    return "#{:02x}{:02x}{:02x}".format(int(color[0]), int(color[1]), int(color[2]))
```

In [7]:

```
def get_image(image_path):
    image = cv2.imread(image_path)
    image =cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    return image
```

In [8]:

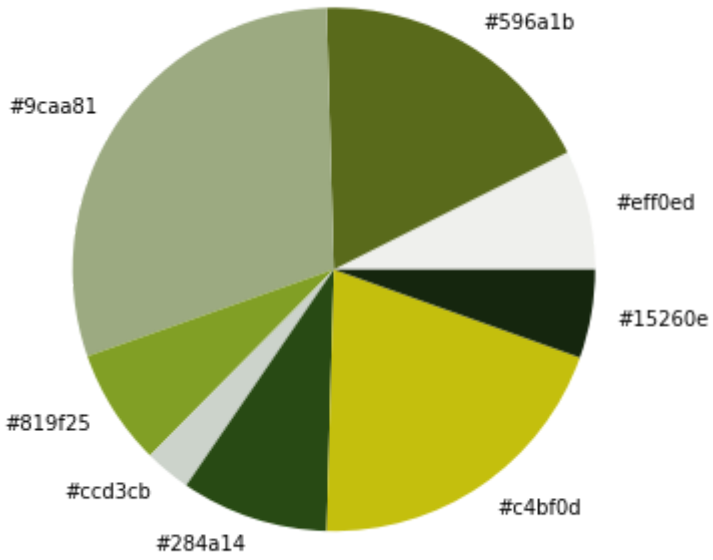
```
def get_colors(image,number_of_colors,show_chart):
    modified_image = cv2.resize(image, (600, 400), interpolation = cv2.INTER_AREA)
    modified_image = modified_image.reshape(modified_image.shape[0]*modified_image.shape[1]
    clf = KMeans(n_clusters =number_of_colors )
    labels = clf.fit_predict(modified_image)
    counts = Counter(labels)

    center_colors = clf.cluster_centers_
    # We get ordered colors by iterating through the keys
    ordered_colors = [center_colors[i] for i in counts.keys()]
    hex_colors = [RGB2HEX(ordered_colors[i]) for i in counts.keys()]
    rgb_colors = [ordered_colors[i] for i in counts.keys()]

    if (show_chart):
        plt.figure(figsize = (8, 6))
        plt.pie(counts.values(), labels = hex_colors, colors = hex_colors)

    return rgb_colors
```

In [9]:

```
get_colors(get_image('images/sample.jpg'), 8, True)
```

Out[9]:

```
[array([239.74681269, 240.38299962, 237.63127534]),
 array([ 89.19512058, 106.78820149,  27.26121253]),
 array([156.04903804, 170.28815276, 129.16215825]),
 array([129.19699074, 159.83373843,  37.87314815]),
 array([204.62332207, 211.49589079, 203.8350379 ]),
 array([40.26116458, 74.84137645, 20.37542985]),
 array([196.50692964, 191.34145599,  13.89788303]),
 array([21.71942937, 38.43931802, 14.34946416])]
```

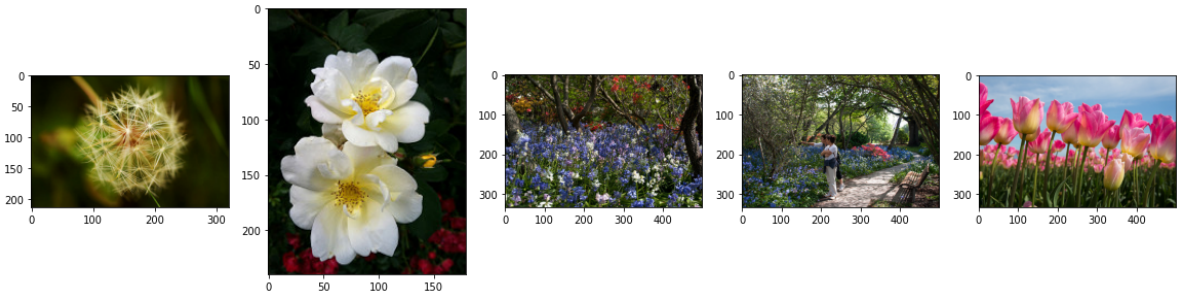


In [10]:

```
IMAGE_DIRECTORY = 'images'
COLORS = {
    'GREEN': [0, 128, 0],
    'BLUE': [0, 0, 128],
    'YELLOW': [255, 255, 0],
    'WHITE' : [255,255,255]
}
images = []

for file in os.listdir(IMAGE_DIRECTORY):
    if not file.startswith('.'):
        images.append(get_image(os.path.join(IMAGE_DIRECTORY, file)))
```

In [11]:

```
plt.figure(figsize=(20, 10))
for i in range(5):
    plt.subplot(1,5, i+1)
    plt.imshow(images[i])
```



In [12]:

```
def match_image_by_color(image, color, threshold = 60, number_of_colors = 10):

    image_colors = get_colors(image, number_of_colors, False)
    selected_color = rgb2lab(np.uint8(np.asarray([[color]])))

    select_image = False
    for i in range(number_of_colors):
        curr_color = rgb2lab(np.uint8(np.asarray([[image_colors[i]]])))
        diff = deltaE_cie76(selected_color, curr_color)
        if (diff < threshold):
            select_image = True

    return select_image
```

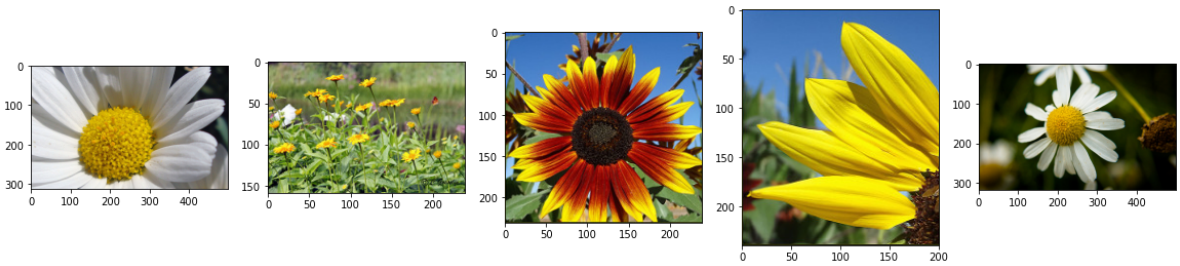
In [13]:

```
def show_selected_images(images, color, threshold, colors_to_match):
    index = 1
    count=0
    dis=[]
    for i in range(len(images)):
        selected = match_image_by_color(images[i],
                                        color,
                                        threshold,
                                        colors_to_match)
#         ims = np.random.randn(5, 224, 224)
        if (selected):
            plt.subplot(1, 5, index)
            plt.imshow(images[i])
            index+=1
            count+=1
        if count ==5:
            break
```

Yellow Colour colour flower sample 5 flowers

In [14]:

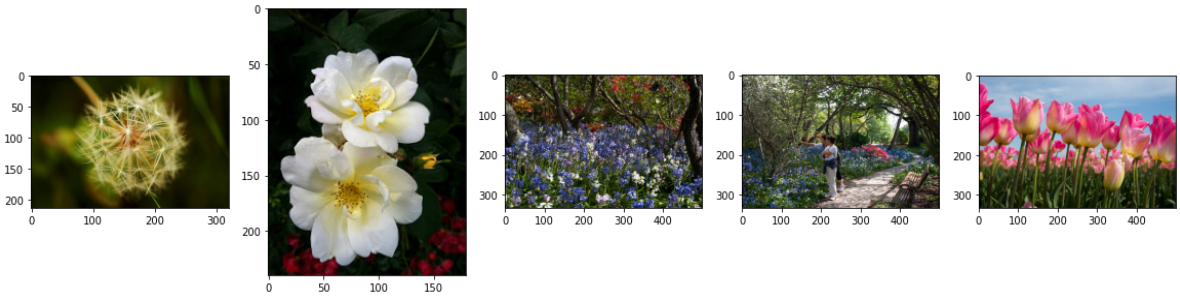
```
plt.figure(figsize = (20, 10))
show_selected_images(images,COLORS['YELLOW'], 60, 5)
```



white colour sample

In [19]:

```
plt.figure(figsize = (20, 10))
show_selected_images(images,COLORS['WHITE'], 60, 5)
```



In [20]:

```
plt.figure(figsize = (20, 10))
show_selected_images(images,COLORS['BLUE'], 60, 5)
```