Sajedeh Sepehrnia, Yahya Maleki Joobani

Pay attention: Following our talk with Mr. Ghasemi, I am sending this project to you once again. In this updated version, we added the feature vector histograms to our code, something that we had forgotten in the previous version. All other aspects of the project have remained the same. Therefore, please do not count the duration between our project's previous version and this one, towards our 7-day allowed delay. Thank you.

load_dataset() -> loads data from the database

classify() -> performs classification using Random Forest algorithm using the feature vectors that we are going to extract with clustering

arrayToRGB() -> turns the 1d array of pixels into a 3-channel array as a RGB image

nthImageRGBToHSV() -> turns a RGB image into a HSV one

featureExtractorFromPixels() -> takes in an input image and creates a feature vector for each pixel of it. In this function, we can give a ***weight*** of 3 to H-S-V features by concatenating H-S-V 3 times in the feature vector of each pixel, while using x and y of each pixel only once.

Ex: feature vector of a pixel = (x,y,h,s,v,h,s,v,h,s,v) -> len = 11

Standardize() -> standardizes a feature vector, so that our distance computation is not biased towards those features that have a wider range

pixelsToRegions() -> Performs a clustering algorithm on the pixels of a certain image and assigns each pixel to a cluster, resulting in different regions

regionBasedFeatureVectorGenerator() -> now we use feature vector of the pixels within each region to create a region-level feature vector. In other words, from now on, each of our regions will have a feature vector for themselves. We used the following features for each region

- averageXYColumns -> avg x, avg y of pixels in this region
- stdXYColumns -> std x, std y of pixels in this region
- numberOfPixelsInThisRegion -> number of pixels in this region
- averageHSVColumns -> avg h, avg s, avg v of pixels in this region
- stdHSVColumns -> std h, std s, std v of pixels in this region

now we will have a loop in which we use the above functions to create region-level feature vectors for each image.

- Since the first part of the project takes a long time, we store the extracted feature vectors in two files, namely 2d_array.npy and 3d_array.npy.

2d_array.npy -> has feature vectors of all images in it

3d_array.npy -> has feature vectors of all images in it in order of the images

- Assuming we have 400 images, 5 regions in each image, and 11 features for each region
  The shape of 2d_array.npy will be (400*5,11)
  The shape of 3d_array.npy will be (400,5,11)

in the rest of the code, we will select the best clustering algorithm to perform Bag-Of-Words

we will turn all regions of all images into T clusters. This leads to T types of regions. Then, we will check and see

- how many type i regions we have in the image k   for i from 0 to T

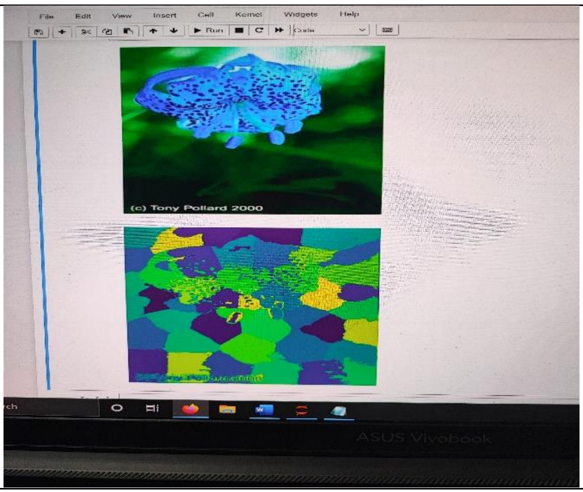
we create a vector of length T for each image, where the Rth index of this vector counts "how many type R regions exist in this image"

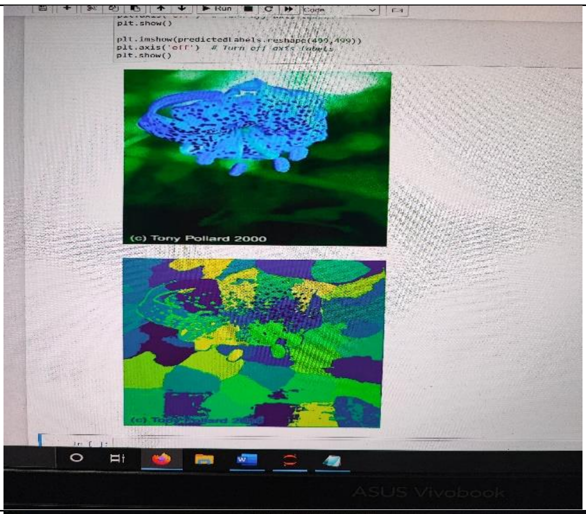
Now that we created the final feature vectors of all images, we have a P*T matrix, where P is the number of images and T is the number of region types.

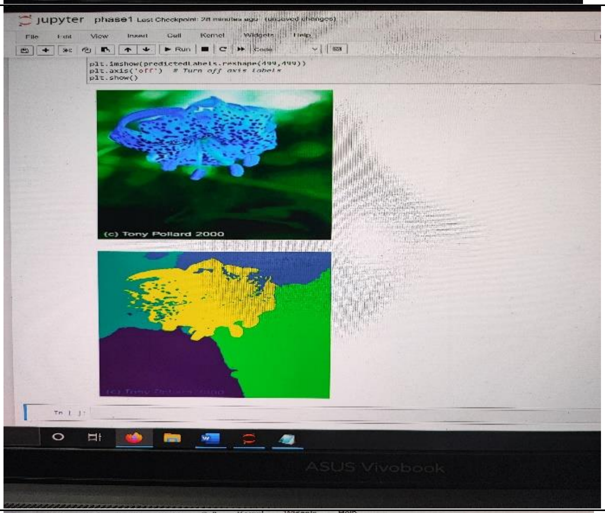
We now feed this matrix as our data points (P data point) to the Random Forest Classifier

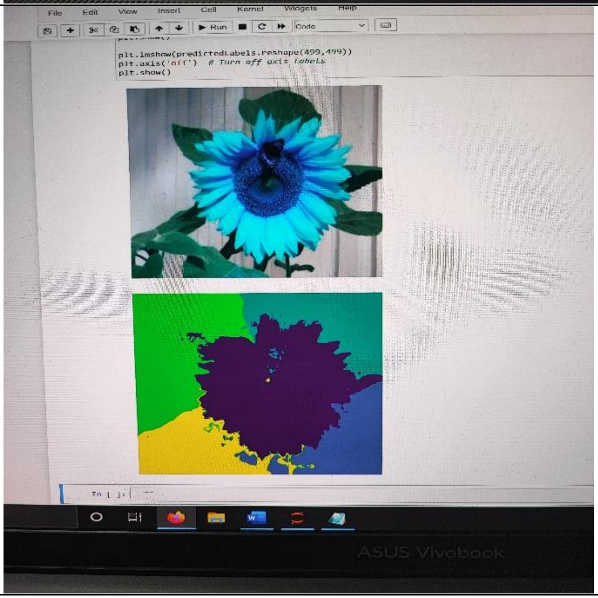
- then we compute Precision, Recall, F1-Score, and Confusion Matrix

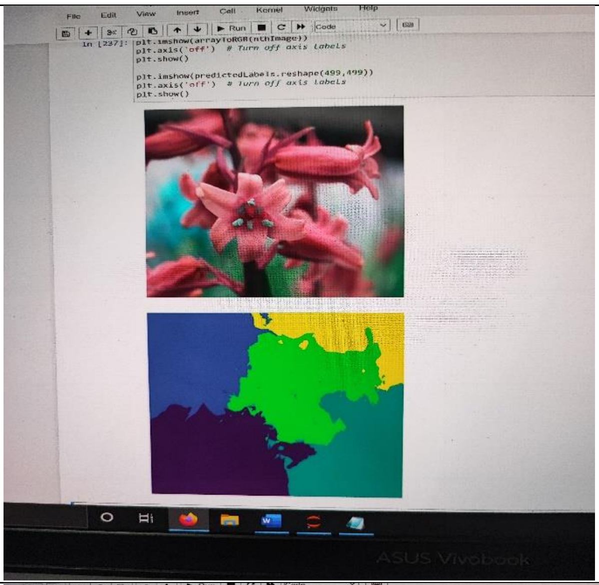
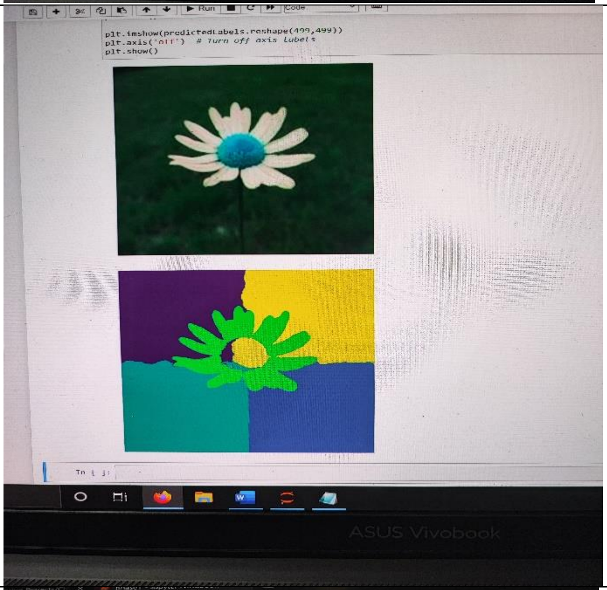
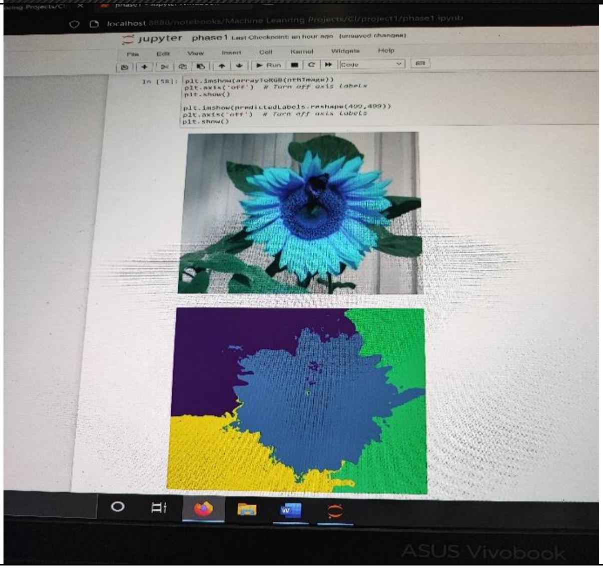
now we check how xy and hsv features' weights change our region extraction

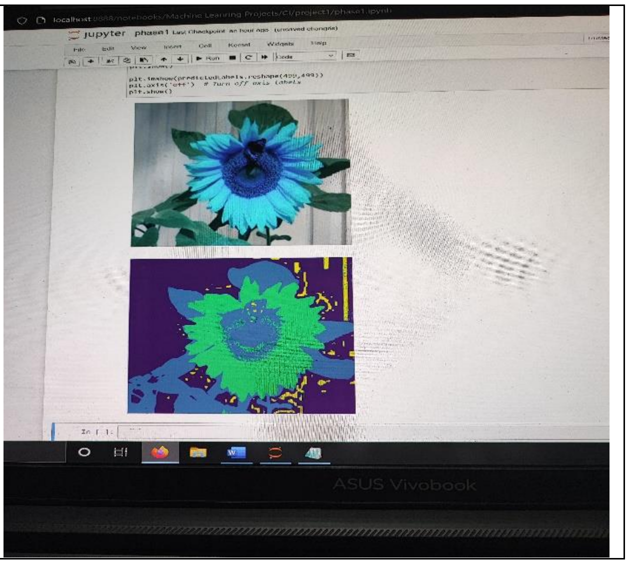
1: xy and hsv same weight. 50 clusters. kmeans.	

2: xy=1 and hsv=2 weights. 50 clusters. kmeans.	 `plt.show()`
	
	`plt.imshow(predictedLabels.reshape(499,499))`
	`plt.axis('off')  # Turn off axis labels`
	`plt.show()`
	 (c) Tony Pollard 2000
	(c) Tony Pollard 2000
	Run
	Code
	In [ ]:
	ASUS Vivobook

3: xy=1 and hsv=3 weights. 5 clusters. kmeans.	 `plt.imshow(predictedLabels.reshape(499,499))`
	`plt.axis('off')  # Turn off axis labels`
	`plt.show()`
	 jupyter phase1 Last Checkpoint: 28 minutes ago (unsaved changes)
	File Edit View Insert Cell Kernel Widgets Help
	Run
	Code
	(c) Tony Pollard 2000
	In [ ]:
	ASUS Vivobook

4: xy=1 and hsv=3 weights. 5 clusters. kmeans.	 `plt.imshow(predictedLabels.reshape(499,499))`
	`plt.axis('off')  # Turn off axis labels`
	`plt.show()`
	 File Edit View Insert Cell Kernel Widgets Help
	Run
	Code
	In [ ]:
	ASUS Vivobook

5: xy=1 and hsv=3 weights. 5 clusters. kmeans.	 `In [237]: plt.imshow(arrayToRGB(nthImage))` `plt.axis('off')  # Turn off axis labels` `plt.show()` `plt.imshow(predictedLabels.reshape(499,499))` `plt.axis('off')  # Turn off axis labels` `plt.show()`
6: xy=1 and hsv=3 weights. 5 clusters. kmeans.	 `plt.imshow(predictedLabels.reshape(499,499))` `plt.axis('off')  # Turn off axis labels` `plt.show()` `In [ ]:`
7: before standardization xy=1 and hsv=3 weights. 4 clusters. kmeans.	 localhost:8888/notebooks/Machine Learning Projects/CI/project1/phase1.ipynb jupyter phase1 Last Checkpoint: an hour ago (unsaved changes) `In [58]: plt.imshow(arrayToRGB(nthImage))` `plt.axis('off')  # Turn off axis labels` `plt.show()` `plt.imshow(predictedLabels.reshape(499,499))` `plt.axis('off')  # Turn off axis labels` `plt.show()`

8: after standardization xy=1 and hsv=3 weights. 4 clusters. kmeans.	

```
plt.imshow(predictedLabels.reshape(499,499))
plt.axis('off')  # Turn off axis labels
plt.show()
```

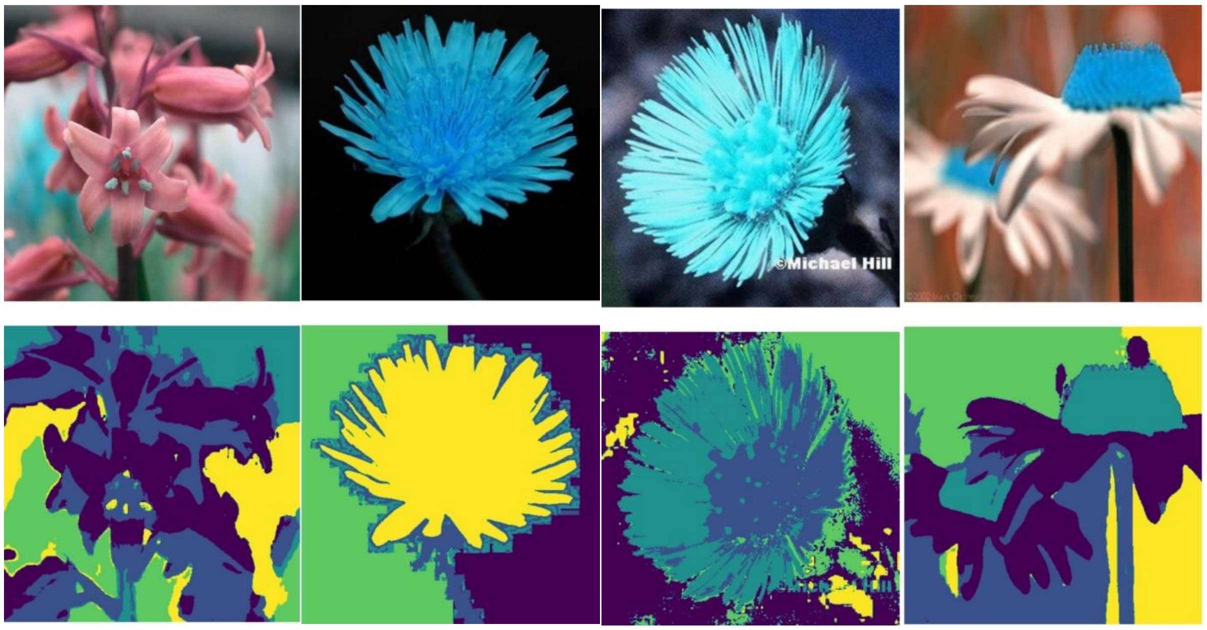
- ***Silhouette Score*** is **a metric used to calculate the goodness of a clustering technique**. Its value ranges from -1 to 1. 1: Means clusters are well apart from each other and clearly distinguished.

- ***Calinski-Harabasz (CH) Index*** is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation)

- ***Davies–Bouldin index*** is calculated as the average of the maximum ratio of the within-cluster distance and the between-cluster distance for each cluster.

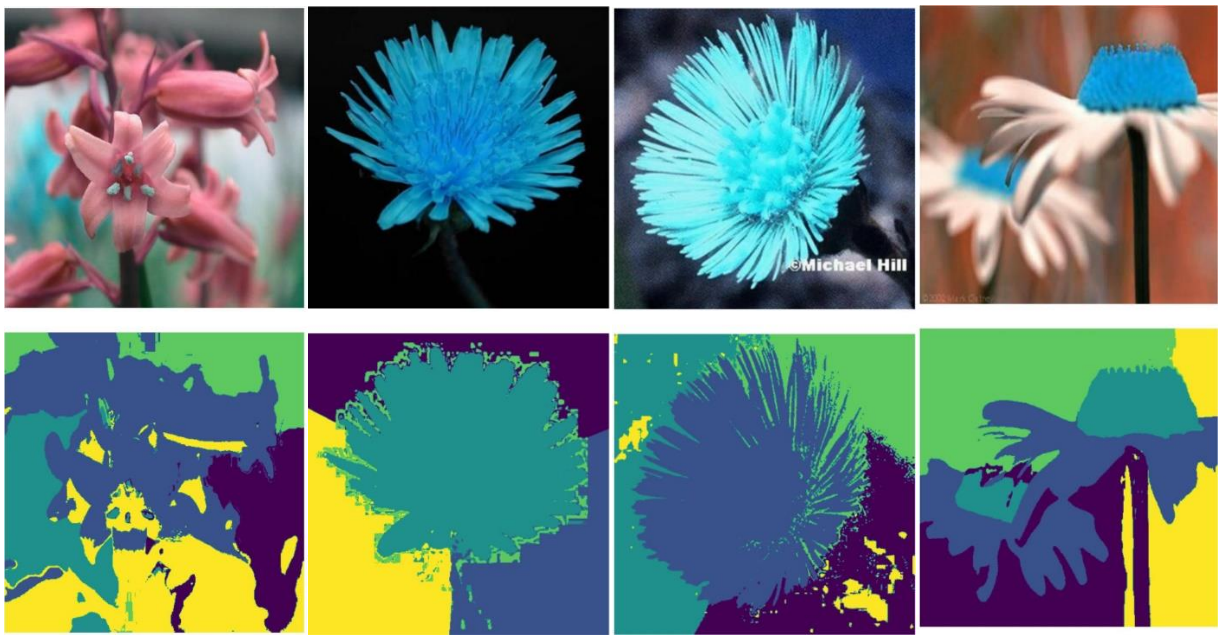
```
For our model with k=5, HSVweight = 3, XYweight = 1

confusion_matrix
 [[ 9  1  0  1  6  1  1]
 [ 1 11  0  0  0  6  3]
 [ 0  3  5  1  0  6  1]
 [ 1  0  1 11  0  0  0]
 [ 2  0  1  3  9  1  0]
 [ 0  0  1  0  0  7  1]
 [ 2  1  1  1  0  4  9]]
Accuracy: 54.46%
Precision: 56.72
Recall: 57.09
F1_Score: 54.22
```



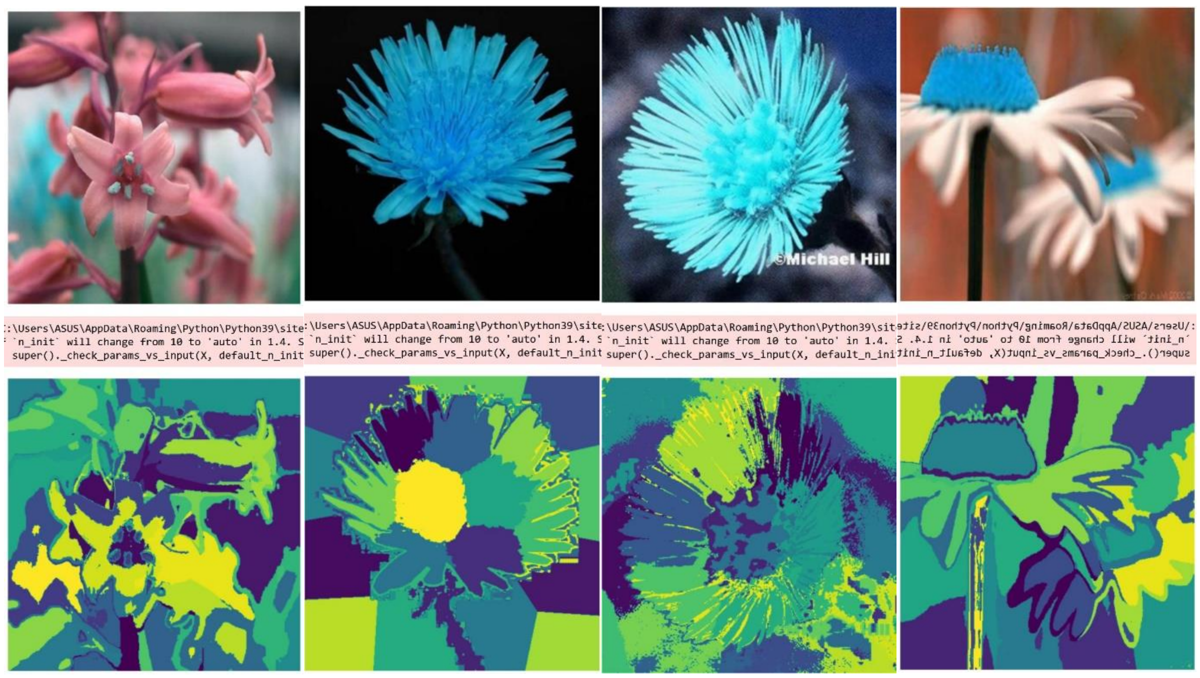
```
For our model with k=5, HSVweight = 1, XYweight = 1
```

```
confusion_matrix
 [[16  0  0  0  1  0  2]
 [ 0  9  2  1  1  2  6]
 [ 0  2  9  0  1  1  3]
 [ 1  0  0 12  0  0  0]
 [ 2  0  0  2 10  0  2]
 [ 0  0  2  0  0  7  0]
 [ 0  2  2  1  0  1 12]]
Accuracy: 66.96%
Precision: 68.14
Recall: 68.94
F1_Score: 67.54
```



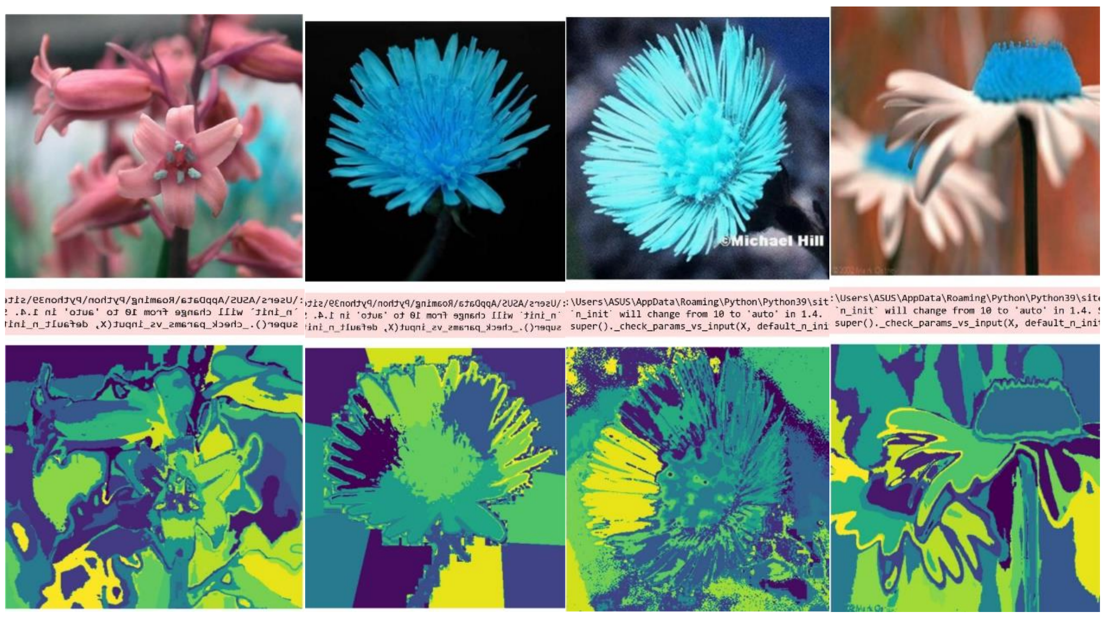
Best Model : For our model with k=30, HSVweight = 1, XYweight = 1 average ***Calinski-Harabasz(CH) Index*** was 148655.62 and average ***Davies-Bouldin index*** was 1.13.

```
confusion_matrix
 [[24  0  0  1  0  0  1]
 [ 0 17  4  1  0  3  3]
 [ 0  0 20  0  0  3  1]
 [ 0  0  0 21  0  0  0]
 [ 0  0  1  2 21  0  1]
 [ 0  2  2  0  0 14  0]
 [ 3  2  1  0  1  3 16]]
Accuracy: 79.17%
Precision: 79.19
Recall: 79.95
F1_Score: 78.93
```



```
:\Users\ASUS\AppData\Roaming\Python\Python39\site
`n_init` will change from 10 to 'auto' in 1.4. S
 super()._check_params_vs_input(X, default_n_init
```

```
For our model with k=30, HSVweight = 3, XYweight = 1
confusion_matrix
 [[12  0  0  0  1  0  0]
 [ 0  9  4  1  0  6  0]
 [ 0  0 19  0  0  0  0]
 [ 0  0  0 11  0  0  0]
 [ 4  0  0  2  8  0  0]
 [ 0  0  0  0  0 19  0]
 [ 0  1  1  1  3  1  9]]
Accuracy: 77.68%
Precision: 79.61
Recall: 78.67
F1_Score: 76.25
```



```
:\Users\ASUS\AppData\Roaming\Python\Python39\site
`n_init` will change from 10 to 'auto' in 1.4. S
 super()._check_params_vs_input(X, default_n_init
```

In order to improve our results we can add more images to our dataset in order to overcome overfitting of Random Forest.

Also, we can add more complicated features to our feature vectors.

Due to lack of time, we could not fine-tune the hyper-parameters perfectly.

Some evaluation matrices like ***`Silhouette Score`*** take a lot of time and memory to compute, and some algorithms like Mean-Shift are also time-consuming.