GUJARAT UNIVERSITY

ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

DEPARTMENT OF COMPUTER CENTER

ROLLWALA COMPUTER CENTER

GUJARAT UNIVERSITY

FLOWER CLASSIFICATION

PROJECT BY:

Nishet Vyas

GUIDED BY:

Ms.Jigna Satani

Index

Acknowledgement……………………………………………………3
Introduction……………………………………………………………4
Project Detail…………………………………………………………5
Problem Definition………………………………………………....6
Purpose……………………………………………………………….…7
Application……………………………………………………………8
Classification Problem…………………………………………9
Implementation Environment……………………………………10
Scikit-Learn…………………………………………………………..11
Mahotas……………………………………………………………….11
H5py……………………………………………………………………..13
Feature Extraction…………………………………………………...14
Flower-17 Dataset………………………………………………...17
Folder Structure……………………………………………………...19
Project Flow…………………………………………………………..20
Function for Feature Descriptor……………………………………22
1.Hu moments………………………………………………………22
2.Haralick Textures…………………………………………………23
3.Color Histogram…………………………………………………23
Set Label………………………………………………………………...26
Training Classifiers………………………………………………….27
Testing the best classifier…………………………………………...34

Output……………………………………………………………36
Project Code…………………………………………………………41
References……………………………………………………………52

Acknowledgement

I would like to thank Ms. Jigna Ma’am for a great deal of support and patience, and also for inspiring to make this report.

My thanks also to Dr. Jyoti ma’am and R. Savita ma’am and to encourage me which was very helpful in the writing of this paper. I owe a great debt to my all staff of Rollwala computer center for their guidance and support.

I am thankful to Mr. Tapan Bhavsar for their guidance and suggestion during this project work.

I am also extremely grateful to my all classmates, for giving me their support.

INTRODUCTION

The main goal of this project is to identify the flower from flower image data**.** There are places where data are mainly based on flower. Some flower is easily identified by humans but many flowers with limited Botanical knowledge would not know the exact type of a flower just by looking at it.

This work proposed Flower classification that help recognizing a flower image in order to get further information about their species.

This system employs image classification based on existing database.

Still flower recognition is considered the hello world program or the beginning of learning machine learning approach. Different people have used different methods to extract and to classify the data. Now let’s have a look on our work.

Application

- Automating tasks in garden and farms helps in increasing productivity

- Applies recent technology such as IOT and ML in agriculture domain

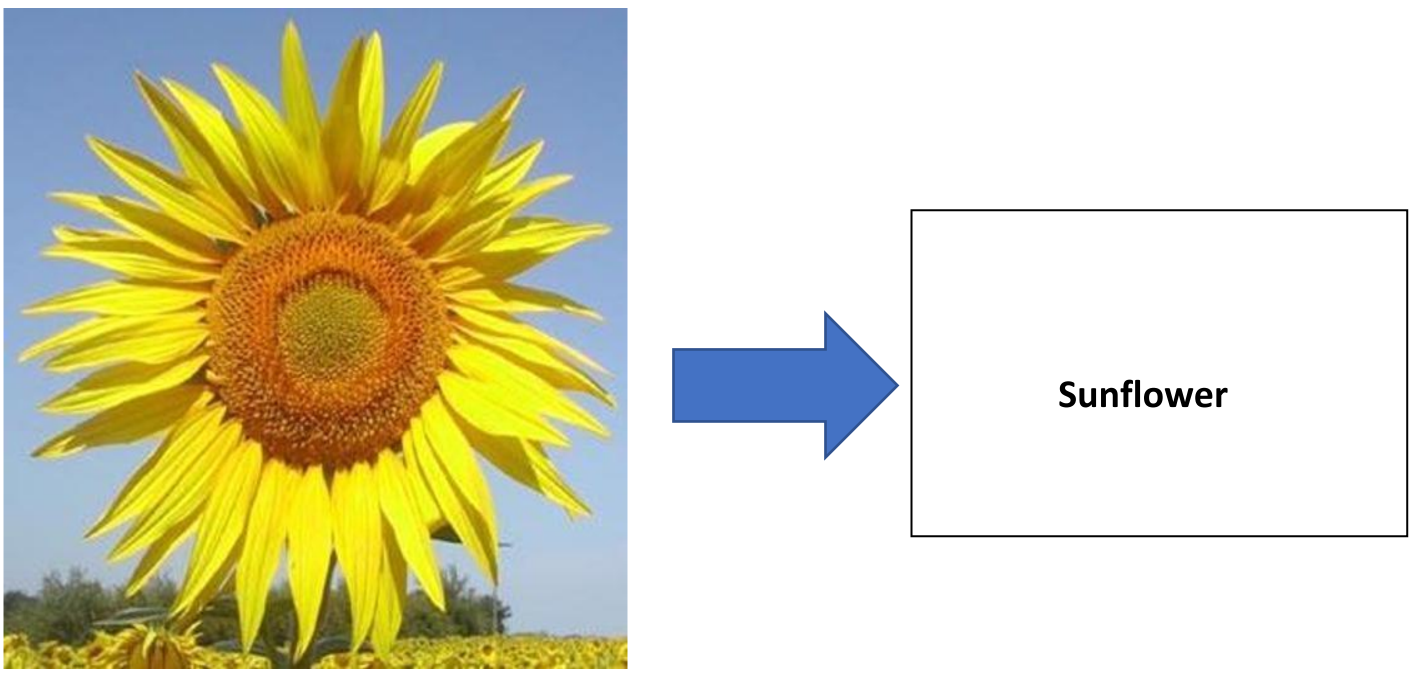
- A system for our home or our garden to monitor plants using raspberry pi

Project Details

Project Title	Flower classification
Project Description	This project is to identify the flower from flower image data. This system employs image classification based on existing database.
Goal	The main goal of this project is to identify the flower from flower image data.
Developed By	Nishet Vyas
Developed At	Department of Computer Science, Rollwala Computer Center, Gujrat University
Project Guide	Ms.Jigna Satani

Problem Definition

- Automatic classify Flower Name from Flower images



Purpose

- In general, there are many different types of flowers. Presently, new flower species are being discovered on an ongoing basis, even though they are very similar in color, shape, texture, petals and features.

- Ordinary persons with limited Botanical knowledge would not know the exact type of a flower just by looking at it.

- In order to classify a flower correctly, it is important to provide enough important information including the flower's name.

- This work proposed Flower classification that help recognizing a flower image in order to get further information about their species.

- This system employs image classification based on existing database.

Application

- predicts the label/class of the flower/plant using Computer Vision techniques and Machine Learning algorithms.

- intelligent system that was trained with massive dataset of flower/plant images.

- Automating tasks in garden and farms helps in increasing productivity

- Applies recent technology such as IOT and ML in agriculture domain

- A system for our home or our garden to monitor plants using raspberry pi

Classification Problem

Plant or Flower Species Classification is one of the most challenging and difficult problems in Computer Vision due to a variety of reasons.

1.Availability of plant/flower dataset

Collecting plant/flower dataset is a time-consuming task. Although training a machine with these dataset might help in some scenarios, there are still more problems to be solved.

2. Millions of plant/flower species around the world

we need to train our model with such large number of images with its labels. We are talking about 6 digit class labels here for which we need tremendous computing power (GPU farms).

3. High inter-class as well as intra-class variation
What we mean here is that “Sunflower” might be looking similar to a “Daffodil” in terms of color. This becomes an inter-class variation problem. Similarly, sometimes a single “Sunflower” image might have differences within it’s class itself, which boils down to intra-class variation problem.

4. Fine-grained classification problem
It means our model must not look into the image or video sequence and find *“Oh yes! there is a flower in this image”*. It means our model must tell *“Yeah! I found a flower in this image and I can tell you it’s a tulip”*.

5. Segmentation, View-point, Occlusion, Illumination and the list goes on..
Segmenting the plant/flower region from an image is a challenging task. This is because we might need to remove the unwanted background and take only the foreground object (plant/flower) which is again a difficult thing due to the shape of plant/flower.

Implementation Environment

- Software packages and libraries needed:

    1. Python
    2. OpenCV
    3. Scikit-learn
    4. Mahotas
    5. NumPy
    6. SciPy
    7. h5py

What is Scikit-Learn?

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.

It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

- **NumPy**: Base n-dimensional array package
- **SciPy**: Fundamental library for scientific computing
- **Matplotlib**: Comprehensive 2D/3D plotting
- **IPython**: Enhanced interactive console
- **Sympy**: Symbolic mathematics
- **Pandas**: Data structures and analysis

.

The vision for the library is a level of robustness and support required for use in production systems. This means a deep focus on concerns such as easy of use, code quality, collaboration, documentation and performance.

What are the features?

The library is focused on modeling data. It is not focused on loading, manipulating and summarizing data. For these features, refer to NumPy and Pandas

Some popular groups of models provided by scikit-learn include:

- **Clustering**: for grouping unlabeled data such as KMeans.
- **Cross Validation**: for estimating the performance of supervised models on unseen data.
- **Datasets**: for test datasets and for generating datasets with specific properties for investigating model behavior.
- **Dimensionality Reduction**: for reducing the number of attributes in data for summarization, visualization and feature selection such as Principal component analysis.
- **Ensemble methods**: for combining the predictions of multiple supervised models.

- **Feature extraction**: for defining attributes in image and text data.
- **Feature selection**: for identifying meaningful attributes from which to create supervised models.
- **Parameter Tuning**: for getting the most out of supervised models.
- **Manifold Learning**: For summarizing and depicting complex multi-dimensional data.
- **Supervised Models**: a vast array not limited to generalized linear models, discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.

What is Mahotas?

Mahotas is a computer vision and image processing library for Python.

It includes many algorithms implemented in C++ for speed while operating in numpy arrays and with a very clean Python interface.

Mahotas currently has over 100 functions for image processing and computer vision and it keeps growing. Some examples of mahotas functionality:

- watershed
- convex points calculations.
- hit & miss. thinning
- Zernike & Haralick, local binary patterns, and TAS features.
- morphological processing
- Speeded-Up Robust Features (SURF), a form of local features
- thresholding
- convolution.
- Sobel edge detection.

What is h5py?

The h5py package is a Pythonic interface to the HDF5 binary data format.

It lets you store huge amounts of numerical data, and easily manipulate that data from NumPy. For example, you can slice into multi-terabyte datasets stored on disk, as if they were real NumPy arrays. Thousands of datasets can be stored in a single file, categorized and tagged however you want.

H5py uses straightforward NumPy and Python metaphors, like dictionary and NumPy array syntax. For example, you can iterate over datasets in a file, or check out the .shape or .dtype attributes of datasets.

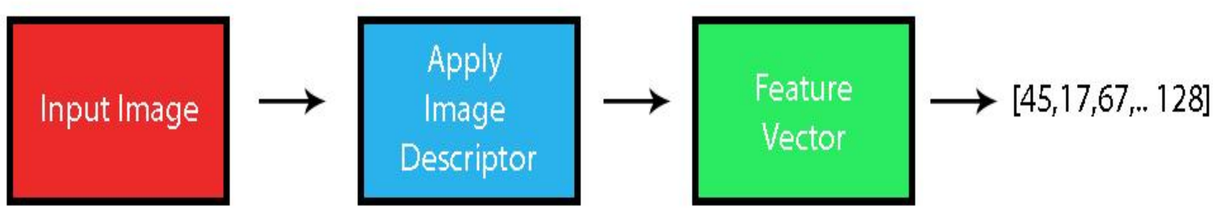
In addition to the easy-to-use high level interface, h5py rests on a object-oriented Cython wrapping of the HDF5 C API. Almost anything you can do from C in HDF5, you can do from h5py.

Feature Extraction

- Features are the information or list of numbers that are extracted from an image. These are real-valued numbers (integers, float or binary). There are a wider range of feature extraction algorithms in Computer Vision.

- When deciding about the features that could quantify plants and flowers, we could possibly think of Color, Texture and Shape as the primary ones. This is an obvious choice to globally quantify and represent the plant or flower image.

- But this approach is less likely to produce good results, if we choose only one feature vector, as these species have many attributes in common like **sunflower** will be similar to **daffodil** in terms of color and so on. So, we need to quantify the image by combining different feature descriptors so that it describes the image more **effectively**.



Global Feature Descriptors

- These are the feature descriptors that quantifies an image globally.

- Some of the commonly used global feature descriptors are:-

  Color - Color Channel Statistics (Mean, Standard Deviation) and Color Histogram

  Shape - Hu Moments, Zernike Moments

  Texture - Haralick Texture, Local Binary Patterns(LBP)

  Others - Histogram of Oriented Gradients (HOG), Threshold Adjacency Statistics (TAS)

Local Feature Descriptors

- These are the feature descriptors that quantifies local regions of an image.
- Some of the commonly used local feature descriptors are: -

  SIFT (Scale Invariant Feature Transform)

  SURF (Speeded Up Robust Features)

  ORB (Oriented Fast and Rotated BRIEF)

  BRIEF (Binary Robust Independed Elementary Features)

Combining Features

- For global feature vectors, we just concatenate each feature vector to form a single global feature vector.
- For local feature vectors as well as combination of global and local feature vectors.

Global Features to quantify a flower image.

Flower-17 Dataset

- In this project use the FLOWER17 dataset provided by the University of Oxford, Visual Geometry group.

- This dataset has 17 classes of flower species, each having 80 images. So, totally we have 1360 images to train our model.

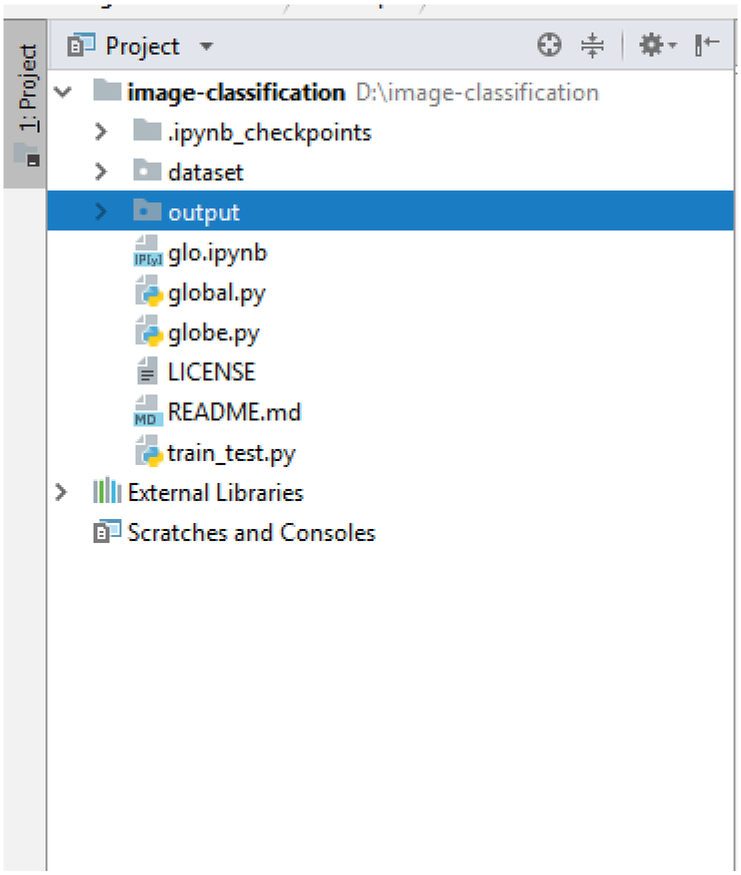
- The dataset is split into the training (40 images per class), validation (20 images per class), and test (20 images per class) sets.

- REFRENCES: - http://www.robots.ox.ac.uk/~vgg/data/flowers/17/


Buttercup
Colts'Foot
Daffodil
Daisy
Dandelion
Fritillary
Iris
Pansy
Sunflower
Windflower


Snowdrop
LilyValley
Bluebell
Crocus
Tigerlily
Tulip
Cowslip

Folder Structure



- In dataset folder it’s have 2 sub-folder test and training.
- That two subfolder have images with flower class-name folder.

Project Flow

- Main aim of this project is to classify the flower name from the image.
- Now that we have our dataset ready to use we implement our project in the following manner.

  - Load The dataset
  - Global Feature Extraction
  - Apply Function for Global Feature Descriptors and save feature into file
  - Apply Training Classifier
  - Testing the best classifier
  - Improve classifier Accuracy

train_test.py × globe.py × global.py ×

```
1
2  # global feature extractionn
3
4  # organize imports
5  from sklearn.preprocessing import LabelEncoder
6  from sklearn.preprocessing import MinMaxScaler
7  import numpy as np
8  import mahotas
9  import glob
10 import cv2
11 import os
12 import h5py
13
14 # fixed-sizes for image
15 fixed_size = tuple((500, 500))
16
17 # path to training data
18 #path = os.getcwd()
19 train_path = 'dataset/train'
20 #data_dir_list =os.listdir(train_path)
21
22 # no.of.trees for Random Forests
23 num_trees = 100
24
25 # bins for histogram
26 bins = 8
27
28 # train_test_split size
29 test_size = 0.10
30
31 # seed for reproducing same results
32 seed = 9
33 '''
   for training_name in train_labe
```

- Line 1 - 12 imports the necessary libraries we need to work with.
- Line 15 used to convert the input image to a fixed size of (500, 500).
- Line 19 is the path to our training dataset.
- Line 23 is the number of trees we need to initialize for Random Forests classifier.
- Line 26 is the number of bins for color histograms.
- Line 29 is the amount of training data and testing data split (0.10 means splitting the training dataset as 90% train data and 10% test data).

Functions for Global Feature Descriptors

1.Hu Moments

- To extract Hu Moments features from the image, we use cv2.moments()function provided by OpenCV.

- The argument to this function is the moments of the image Cv2.moments() flattened. It means we compute the moments of the image and convert it to a vector using flatten().

- Before doing that, we convert our color image into a grayscale image as moments expect images to be grayscale.

```
# feature-descriptor-1: Hu Moments
def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature
```

2. Haralick Textures

- To extract Haralick Texture features from the image, we make use of mahotas library.

- The function we will be using is mahotas.features.haralick().

- Before doing that, we convert our color image into a grayscale image as haralick feature descriptor expect images to be grayscale.

```
# feature-descriptor-2: Haralick Texture
def fd_haralick(image):
    # convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # compute the haralick texture feature vector
    haralick = mahotas.features.haralick(gray).mean(axis=0)
    # return the result
    return haralick
```

3. Color Histogram

- To extract Color Histogram features from the image, we use `cv2.calcHist()` function provided by OpenCV.

- The arguments it expects are the image, channels, mask, histSize (bins) and ranges for each channel [typically 0-256).

- We then normalize the histogram using `normalize()` function of OpenCV and return a flattened version of this normalized matrix using `flatten()`.

```python

# feature-descriptor-3: Color Histogram
def fd_histogram(image, mask=None):
    # convert the image to HSV color-space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # compute the color histogram
    hist  = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins], [0, 256, 0, 256, 0, 256])
    # normalize the histogram
    cv2.normalize(hist, hist)
    # return the histogram
    return hist.flatten()

```

After applying global features descriptor to get the list of training labels associated With each image, under our training path.

```python
# get the training labels
train_labels = os.listdir(train_path)

# sort the training labels
train_labels.sort()
print(train_labels)

# empty lists to hold feature vectors and labels
global_features = []
labels = []

i, j = 0, 0
k = 0

# num of images per class
images_per_class = 80

```

Output: -

['bluebell', 'buttercup', 'colts_foot', 'cowslip', 'crocus', 'daffodil', 'daisy', 'dandelion', 'fritillary', 'iris', 'lily_valley', 'pan3.jpg', 'pansy', 'snowdrop', 'sunflower', 'tigerlily', 'tulip', 'windflower']

For each of the training label name, we iterate through the corresponding folder to get all the images inside it.

For each image that we iterate, we first resize the image into a fixed size. Then, we extract the three global features and concatenate these three features using NumPy's np.stack() function.

We keep track of the feature with its label using those two lists we created above - labels and global features.

train_test.py global.py

```
# loop over the training data sub-folders
for training_name in train_labels:
    # join the training data path and each species training folder
    img_list=os.listdir(train_path+'/'+training_name)
    print('Loaded the images of dataset-' + '{}\n'.format(training_name))
    # get the current training label
    current_label = training_name
    k = 1
    # loop over the images in each sub-folder
    for x in img_list:
        # get the image file name
        file = train_path +'/'+training_name+'/'+x
        image_path = os.path.join(train_path, training_name, x)
        if os.path.exists(image_path):
            # read the image and resize it to a fixed-size
            image = cv2.imread(file)
            image = cv2.resize(image, fixed_size)
            img_data_list.append(image)

        # Global Feature extraction
        fv_hu_moments = fd_hu_moments(image)
        fv_haralick   = fd_haralick(image)
        fv_histogram  = fd_histogram(image)
        # Concatenate global features
        global_feature = np.hstack([fv_histogram, fv_haralick, fv_hu_moments])
        # update the list of labels and feature vectors
        labels.append(current_label)
        global_features.append(global_feature)
        i += 1
        k += 1
    print ("[STATUS] processed folder: {}".format(current_label))
    j += 1
print( "[STATUS] completed Global Feature Extraction...")
```

for training_name in train_labe...

Output: -

[STATUS] processed folder: tiger lily

[STATUS] processed folder: Tulip

[STATUS] processed folder: Windflower

[STATUS] completed Global Feature Extraction…

Set Label in Format

- After extracting features and concatenating it, we need to save this data locally. Before saving this data, we use something called LabelEncoder() to encode our labels in a proper format. This is to make sure that the labels are represented as unique numbers.

- As we have used different global features, one feature might dominate the other with respect to it's value.

- It is better to normalize everything within a range (say 0-1). Thus, we normalize the features using scikit-learn's MinMaxScaler() function.
  After doing these two steps, we use h5py to save our features and labels locally in .h5 file format.

train_test.py | global.py

```python

# get the overall feature vector size
print ("[STATUS] feature vector size {}".format(np.array(global_features).shape))

# get the overall training label size
print ("[STATUS] training Labels {}".format(np.array(labels).shape))

# encode the target labels
targetNames = np.unique(labels)
le = LabelEncoder()
target = le.fit_transform(labels)
print ("[STATUS] training labels encoded...")

# normalize the feature vector in the range (0-1)
scaler = MinMaxScaler(feature_range=(0, 1))
rescaled_features = scaler.fit_transform(global_features)
print ("[STATUS] feature vector normalized...")

print ("[STATUS] target labels: {}".format(target))
print ("[STATUS] target labels shape: {}".format(target.shape))

# save the feature vector using HDF5
h5f_data = h5py.File('output/data.h5', 'w')
h5f_data.create_dataset('dataset_1', data=np.array(rescaled_features))

h5f_label = h5py.File('output/labels.h5', 'w')
h5f_label.create_dataset('dataset_1', data=np.array(target))

h5f_data.close()
h5f_label.close()

print ("[STATUS] end of training..")
```

Output: -

[STATUS] features shape: (1190, 532)
[STATUS] labels shape: (1190,)

- there are 532 columns in the global feature vector which tells us that when we concatenate color histogram, haralick texture and hu moments, we get a single row with 532 columns.
- So, for 1190 images, we get a feature vector of size (1190, 532).
- The target labels are encoded as integer values in the range (0-16) denoting the 17 classes of flower species.

Training classifiers

- After extracting, concatenating and saving global features and labels from our training dataset, it’s time to train our system. To do that, we need to create our Machine Learning models. For creating our machine learning model’s, we take the help of scikit-learn.

- We will choose Logistic Regression, Linear Discriminant Analysis, K-Nearest Neighbors, Decision Trees, Random Forests, Gaussian Naive Bayes and Support Vector Machine as our machine learning models.

- we will use train_test_split() function provided by scikit-learn to split our training dataset into train_data and test_data. By this way, we train the models with the train_data and test the trained model with the unseen test_data. The split size is decided by the test_size parameter.

- We will also use a technique called K-Fold Cross Validation, a model-validation technique which is the best way to predict ML model’s accuracy. In short, if we choose K = 10, then we split the entire data into 9 parts for training and 1 part for testing uniquely over each round up to 10 times.

- We import all the necessary libraries to work with and create a models list. This list will have all our machine learning models that will get trained with our locally stored features. During import of our features from the locally saved ".h5" file-format, it is always a good practice to check its shape. To do that, we make use of np.array() function to convert the ".h5" data into a numpy array and then print its shape.

train_test.py × global.py ×

```
...
import h5py
import numpy as np
import os
import glob
import cv2
from matplotlib import pyplot
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.externals import joblib

num_trees = 100
test_size = 0.10
seed = 9
fixed_size = tuple((500, 500))
def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature

# feature-descriptor-2: Haralick Texture
def fd_haralick(image):
    # convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # compute the haralick texture feature vector
    haralick = mahotas.features.haralick(gray).mean(axis=0)
```

train_test.py global.py

```
    haralick = mahotas.features.haralick(gray).mean(axis=0)
    # return the result
    return haralick

# feature-descriptor-3: Color Histogram
def fd_histogram(image, mask=None):
    # convert the image to HSV color-space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # compute the color histogram
    hist  = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins], [0, 256, 0, 256, 0, 256])
    # normalize the histogram
    cv2.normalize(hist, hist)
    # return the histogram
    return hist.flatten()

# create all the machine learning models
models = []
models.append(('LR', LogisticRegression(random_state=9)))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(random_state=9)))
models.append(('RF', RandomForestClassifier(n_estimators=num_trees, random_state=9)))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(random_state=9)))

#fixed_size = tuple((500, 500))
# variables to hold the results and names
results = []
names = []
scoring = "accuracy"

# import the feature vector and trained labels
h5f_data = h5py.File('output/data.h5', 'r')
```

fd_haralick()

train_test.py   global.py

```python
# import the feature vector and trained labels
h5f_data = h5py.File('output/data.h5', 'r')
h5f_label = h5py.File('output/labels.h5', 'r')

global_features_string = h5f_data['dataset_1']
global_labels_string = h5f_label['dataset_1']

global_features = np.array(global_features_string)
global_labels = np.array(global_labels_string)

h5f_data.close()
h5f_label.close()

# verify the shape of the feature vector and labels
print ("[STATUS] features shape: {}".format(global_features.shape))
print ("[STATUS] labels shape: {}".format(global_labels.shape))

print ("[STATUS] training started...")

# split the training and testing data
```

Output: -

STATUS] features shape: (1190, 532)

[STATUS] labels shape: (1190,)

[STATUS] training started...

- we will be splitting our training dataset into train_data as well as test_data.train_test_split() function does that for us and it returns four variables.

```

# split the training and testing data
(trainDataGlobal, testDataGlobal, trainLabelsGlobal, testLabelsGlobal) = train_test_split(np.array(global_features),
                                                                                          np.array(global_labels),
                                                                                          test_size= test_size,
                                                                                          random_state= seed)

print ("[STATUS] splitted train and test data...")
print ("Train data  : {}".format(trainDataGlobal.shape))
print ("Test data   : {}".format(testDataGlobal.shape))
print ("Train labels: {}".format(trainLabelsGlobal.shape))
print ("Test labels : {}".format(testLabelsGlobal.shape))

# filter all the warnings
import warnings
```

Output: -

[STATUS] splitted train and test data...

Train data : (1071, 532)

Test data   : (119, 532)

Train labels: (1071,)

Test labels : (119,)

- Notice we have decent amount of train_data and less test_data. We always want to train our model with more data so that our model generalizes well. So, we keep test_size variable to be in the range (0.10 - 0.30).
- we train each of our machine learning model and check the cross-validation results. Here, we have used only our train_data.

train_test.py × global.py ×

```python
print ("Test data   : {}".format(testDataGlobal.shape))
print ("Train labels: {}".format(trainLabelsGlobal.shape))
print ("Test labels : {}".format(testLabelsGlobal.shape))

# filter all the warnings
import warnings
warnings.filterwarnings('ignore')

# 10-fold cross validation
for name, model in models:
    kfold = KFold(n_splits=10, random_state=7)
    cv_results = cross_val_score(model, trainDataGlobal, trainLabelsGlobal, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

# boxplot algorithm comparison
fig = pyplot.figure()
fig.suptitle('Machine Learning algorithm comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```

Output: -

LR: 0.499602 (0.075640)

LDA: 0.406205 (0.071290)

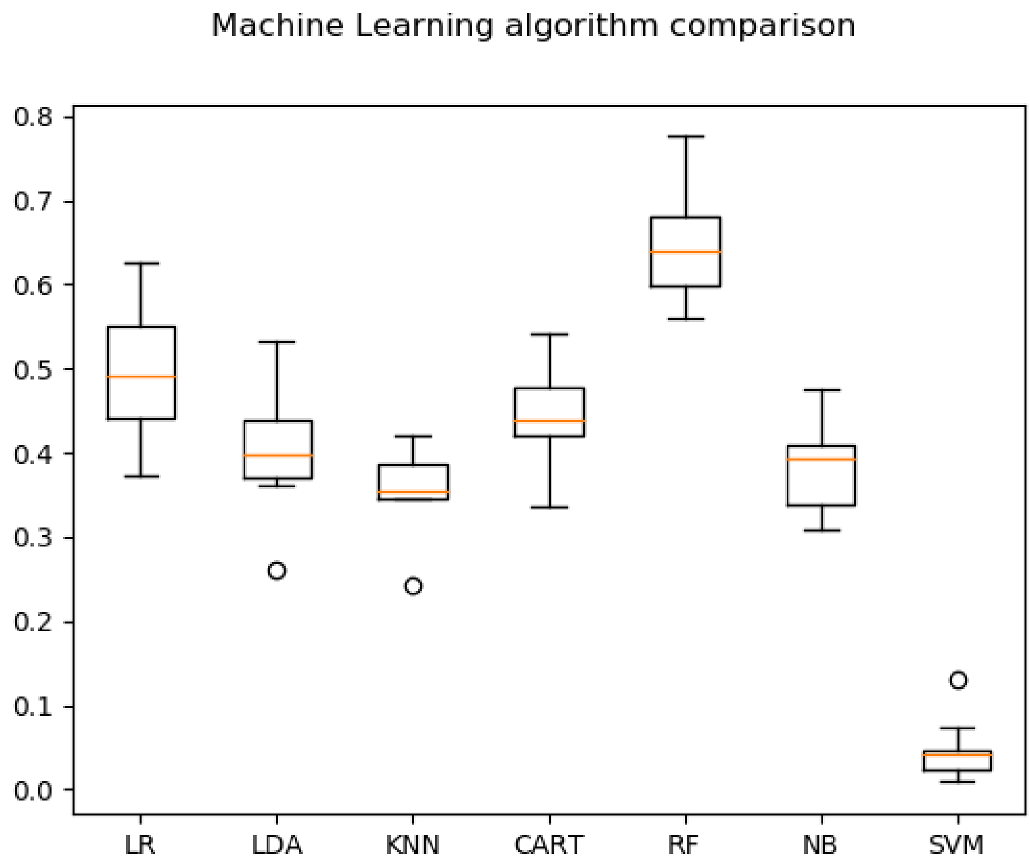
KNN: 0.358515 (0.046140)

CART: 0.448174 (0.057405)

RF: 0.649913 (0.063204)

NB: 0.385601 (0.052706)

SVM: 0.045786 (0.034080)



- As you can see, the accuracies are not so good. Random Forests (RF) gives the maximum accuracy of **64.21%**.
- This is mainly due to the number of images we use per class.
- We need large amounts of data to get better accuracy. For example, for a single class, we at least need around 500-1000 images which is indeed a time-consuming task.

Testing the best classifier

Let's quickly try to build a Random Forest model, train it with the training data and test it.

train_test.py global.py

```python

# to visualize results
import matplotlib.pyplot as plt

# create the model - Random Forests
clf  = RandomForestClassifier(n_estimators=100, random_state=9)

# fit the training data to the model
clf.fit(trainDataGlobal, trainLabelsGlobal)

# path to test data
test_path = 'dataset/test'
# path to test data
#test_path = "dataset/test"


train_labels = os.listdir(train_path)
train_labels.sort()
print(train_labels)
feature =[]
labels =[]
img_data_list =[]
for training_name in train_labels:
    img_list =os.listdir(train_path+'/'+training_name)
    print('Loaded the images of dataset-' + '{}\n'.format(training_name))
    # get the current training label
    current_label = training_name
    k = 1

    for image_path in img_list:
        path = train_path + '/' + training_name + '/'+image_path
        image_path = os.path.join(train_path, training_name, image_path)
        if os.path.exists(image_path):
```

for training_name in train_labe... › for image_path in img_list › if os.path.exists(image_path) train_test.py | global.py

```
    if os.path.exists(image_path):
        # read the image and resize it to a fixed-size
        image = cv2.imread(path)
        image = cv2.resize(image, fixed_size)
        img_data_list.append(image)


# Global Feature extraction

fv_hu_moments = fd_hu_moments(image)
fv_haralick   = fd_haralick(image)
fv_histogram  = fd_histogram(image)


# Concatenate global features

global_feature = np.hstack([fv_histogram, fv_haralick, fv_hu_moments])

# predict label of test image
prediction = clf.predict(global_feature.reshape(1,-1))[0]

# show predicted label on image
final =cv2.putText(image, train_labels[prediction], (20,30), cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,255,255), 3)

# display the output image
final = cv2.cvtColor(final,cv2.COLOR_BGR2RGB)
plt.imshow(final), plt.show()

key = cv2.waitKey(0) & 0xFF
if (key == ord('q')):
    cv2.destroyAllWindows()
```

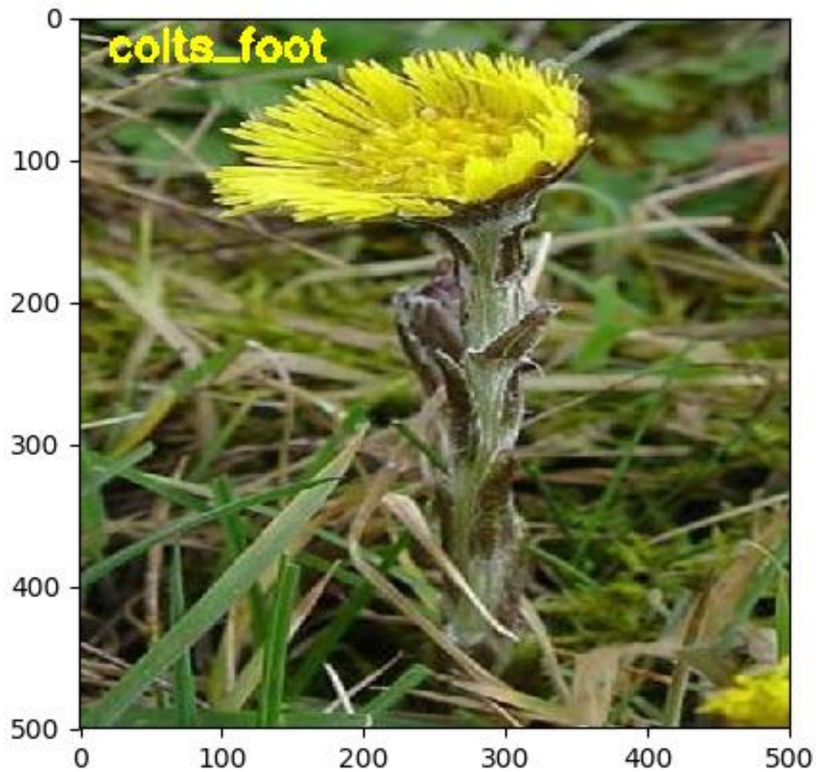
PEP 8: multiple spaces before operator

fv_haralick = fd_haralick(image)

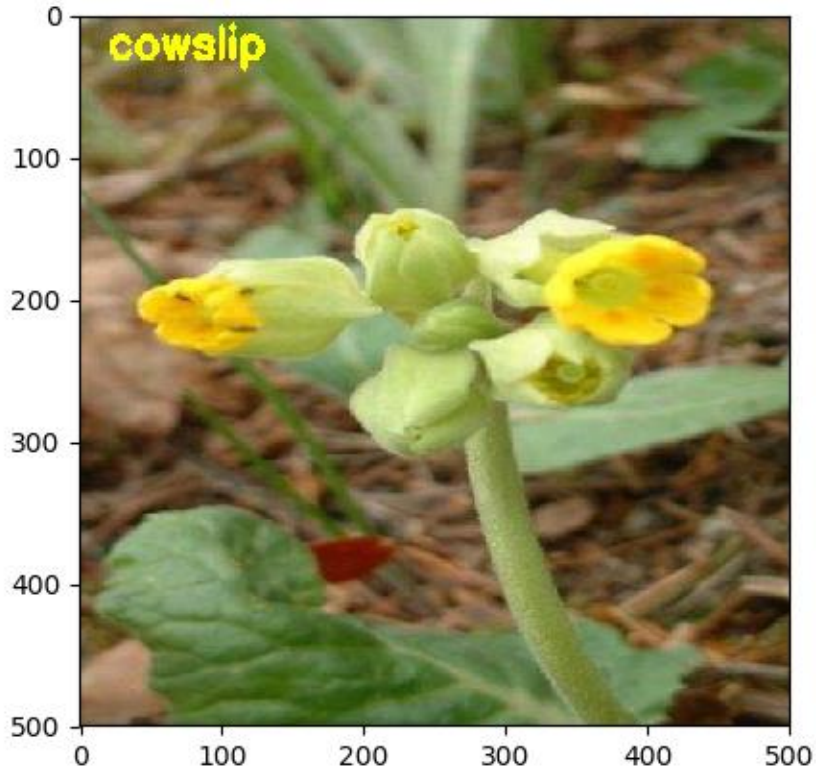
fv_histogram = fd_histogram(image)

fv_hu_moments = fd_hu_moments(image

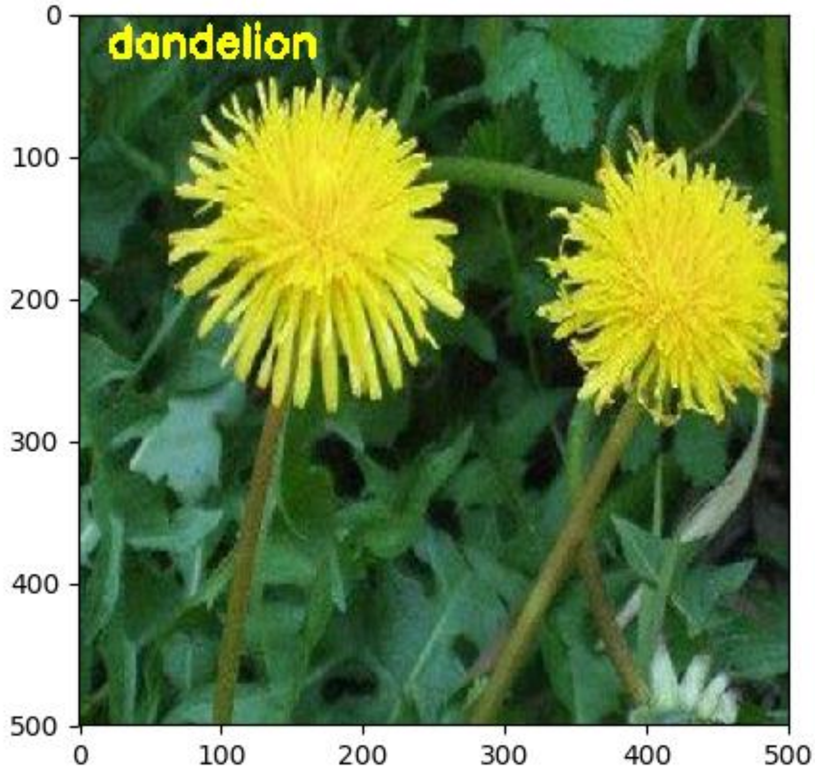
Output: -



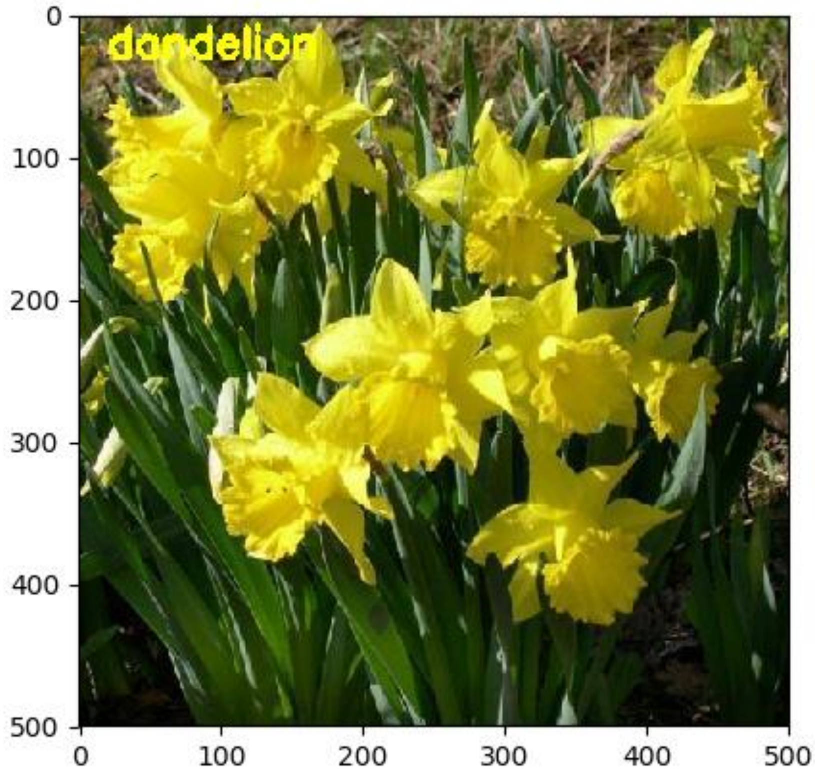
Prediction 1 – coltsfoot (correct)



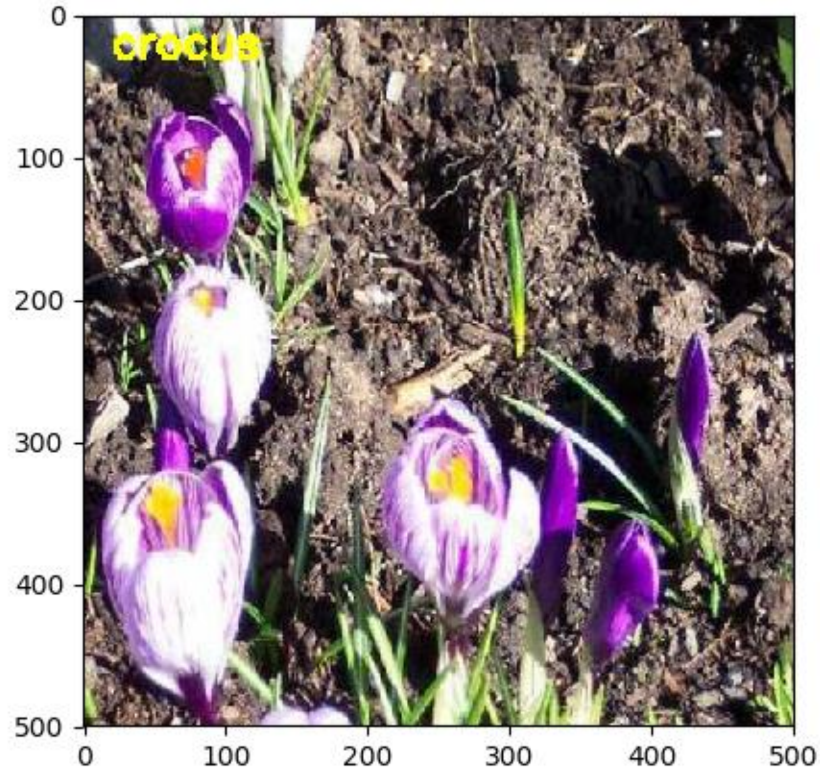
Prediction 2 – Cowslip (correct)



Prediction 3 – Dandelion (correct)



Prediction 4 – Dandelion (Wrong)



Prediction 5 – Dandelion (Correct)

- As we can see, our approach seems to do pretty good at recognizing flowers. But it also predicted wrong label like in prediction 4. Instead of daffodil, our model predicted dandelion.

Project Code

```
# global feature extractionn

# organize imports
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import mahotas
import glob
import cv2
import os
import h5py

# fixed-sizes for image
fixed_size = tuple((500, 500))

# path to training data
#path = os.getcwd()
train_path = 'dataset/train'
#data_dir_list =os.listdir(train_path)

# no.of.trees for Random Forests
num_trees = 100

# bins for histogram
bins = 8

# train_test_split size
test_size = 0.10

# seed for reproducing same results
seed = 9
```

```
# feature-descriptor-1: Hu Moments
def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature

# feature-descriptor-2: Haralick Texture
def fd_haralick(image):
    # convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # compute the haralick texture feature vector
    haralick = mahotas.features.haralick(gray).mean(axis=0)
    # return the result
    return haralick

# feature-descriptor-3: Color Histogram
def fd_histogram(image, mask=None):
    # convert the image to HSV color-space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # compute the color histogram
    hist  = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins],
[0, 256, 0, 256, 0, 256])
    # normalize the histogram
    cv2.normalize(hist, hist)
    # return the histogram
    return hist.flatten()

# get the training labels
train_labels = os.listdir(train_path)

# sort the training labels
train_labels.sort()
print(train_labels)

# empty lists to hold feature vectors and labels
```

```
global_features = []
labels = []

i, j = 0, 0
k = 0

# num of images per class
images_per_class = 80

img_data_list = []
# loop over the training data sub-folders
for training_name in train_labels:
    # join the training data path and each species training folder
    img_list=os.listdir(train_path+'/'+training_name)
    print('Loaded the images of dataset-' +
'{}\n'.format(training_name))
    # get the current training label
    current_label = training_name
    k = 1
    # loop over the images in each sub-folder
    for x in img_list:
        # get the image file name
        file = train_path +'/'+training_name+'/'+x
        image_path = os.path.join(train_path, training_name, x)
        if os.path.exists(image_path):
            # read the image and resize it to a fixed-size
            image = cv2.imread(file)
            image = cv2.resize(image, fixed_size)
            img_data_list.append(image)

        # Global Feature extraction
        fv_hu_moments = fd_hu_moments(image)
        fv_haralick   = fd_haralick(image)
        fv_histogram  = fd_histogram(image)
        # Concatenate global features
```

```
        global_feature = np.hstack([fv_histogram, fv_haralick,
fv_hu_moments])
        # update the list of labels and feature vectors
        labels.append(current_label)
        global_features.append(global_feature)
        i += 1
        k += 1
    print ("[STATUS] processed folder: {}".format(current_label))
    j += 1
print( "[STATUS] completed Global Feature Extraction...")

# get the overall feature vector size
print ("[STATUS] feature vector size
{}".format(np.array(global_features).shape))

# get the overall training label size
print ("[STATUS] training Labels {}".format(np.array(labels).shape))

# encode the target labels
targetNames = np.unique(labels)
le = LabelEncoder()
target = le.fit_transform(labels)
print ("[STATUS] training labels encoded...")

# normalize the feature vector in the range (0-1)
scaler = MinMaxScaler(feature_range=(0, 1))
rescaled_features = scaler.fit_transform(global_features)
print ("[STATUS] feature vector normalized...")

print ("[STATUS] target labels: {}".format(target))
print ("[STATUS] target labels shape: {}".format(target.shape))

# save the feature vector using HDF5
h5f_data = h5py.File('output/data.h5', 'w')
h5f_data.create_dataset('dataset_1', data=np.array(rescaled_features))
```

```
h5f_label = h5py.File('output/labels.h5', 'w')
h5f_label.create_dataset('dataset_1', data=np.array(target))

h5f_data.close()
h5f_label.close()

print ("[STATUS] end of training..")

# TRAINING OUR MODEL

# import the necessary packages
import h5py
import numpy as np
import os
import glob
import cv2
from matplotlib import pyplot
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import confusion_matrix, accuracy_score,
classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.externals import joblib

num_trees = 100
test_size = 0.10
seed = 9
fixed_size = tuple((500, 500))
```

```
def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature

# feature-descriptor-2: Haralick Texture
def fd_haralick(image):
    # convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # compute the haralick texture feature vector
    haralick = mahotas.features.haralick(gray).mean(axis=0)
    # return the result
    return haralick

# feature-descriptor-3: Color Histogram
def fd_histogram(image, mask=None):
    # convert the image to HSV color-space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # compute the color histogram
    hist  = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins],
[0, 256, 0, 256, 0, 256])
    # normalize the histogram
    cv2.normalize(hist, hist)
    # return the histogram
    return hist.flatten()

# create all the machine learning models
models = []
models.append(('LR', LogisticRegression(random_state=9)))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(random_state=9)))
models.append(('RF', RandomForestClassifier(n_estimators=num_trees,
random_state=9)))
models.append(('NB', GaussianNB()))
```

```
models.append(('SVM', SVC(random_state=9)))

#fixed_size = tuple((500, 500))
# variables to hold the results and names
results = []
names = []
scoring = "accuracy"

# import the feature vector and trained labels
h5f_data = h5py.File('output/data.h5', 'r')
h5f_label = h5py.File('output/labels.h5', 'r')

global_features_string = h5f_data['dataset_1']
global_labels_string = h5f_label['dataset_1']

global_features = np.array(global_features_string)
global_labels = np.array(global_labels_string)

h5f_data.close()
h5f_label.close()

# verify the shape of the feature vector and labels
print ("[STATUS] features shape: {}".format(global_features.shape))
print ("[STATUS] labels shape: {}".format(global_labels.shape))

print ("[STATUS] training started...")

# split the training and testing data
(trainDataGlobal, testDataGlobal, trainLabelsGlobal, testLabelsGlobal)
= train_test_split(np.array(global_features),

np.array(global_labels),

test_size= test_size,
```

```
random_state= seed)

print ("[STATUS] splitted train and test data...")
print ("Train data   : {}".format(trainDataGlobal.shape))
print ("Test data    : {}".format(testDataGlobal.shape))
print ("Train labels: {}".format(trainLabelsGlobal.shape))
print ("Test labels : {}".format(testLabelsGlobal.shape))

# filter all the warnings
import warnings
warnings.filterwarnings('ignore')

# 10-fold cross validation
for name, model in models:
    kfold = KFold(n_splits=10, random_state=7)
    cv_results = cross_val_score(model, trainDataGlobal,
trainLabelsGlobal, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

# boxplot algorithm comparison
fig = pyplot.figure()
fig.suptitle('Machine Learning algorithm comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()


# TESTING OUR MODEL

# to visualize results
import matplotlib.pyplot as plt
```

```
# create the model - Random Forests
clf  = RandomForestClassifier(n_estimators=100, random_state=9)

# fit the training data to the model
clf.fit(trainDataGlobal, trainLabelsGlobal)

# path to test data
test_path = 'dataset/test'
# path to test data
#test_path = "dataset/test"



train_labels = os.listdir(train_path)
train_labels.sort()
print(train_labels)
feature =[]
labels =[]
img_data_list =[]
for training_name in train_labels:
    img_list =os.listdir(train_path+'/'+training_name)
    print('Loaded the images of dataset-' +
'{}\n'.format(training_name))
    # get the current training label
    current_label = training_name
    k = 1

    for image_path in img_list:
        path = train_path + '/' + training_name + '/'+image_path
        image_path = os.path.join(train_path, training_name,
image_path)
        if os.path.exists(image_path):
            # read the image and resize it to a fixed-size
            image = cv2.imread(path)
            image = cv2.resize(image, fixed_size)
```

```
            img_data_list.append(image)


    # Global Feature extraction

    fv_hu_moments = fd_hu_moments(image)
    fv_haralick   = fd_haralick(image)
    fv_histogram  = fd_histogram(image)



    # Concatenate global features

    global_feature = np.hstack([fv_histogram, fv_haralick,
fv_hu_moments])

    # predict label of test image
    prediction = clf.predict(global_feature.reshape(1,-1))[0]

    # show predicted label on image
    final =cv2.putText(image, train_labels[prediction],(20,30)
cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,255,255), 3)

    # display the output image
    final = cv2.cvtColor(final,cv2.COLOR_BGR2RGB)
    plt.imshow(final), plt.show()

    key = cv2.waitKey(0) & 0xFF
    if (key == ord('q')):
        cv2.destroyAllWindows()
```

REFERENCES

- http://cs231n.stanford.edu/reports/2016/pdfs/253_Report.pdf
- https://stackoverflow.com/questions/21129020/how-to-fix-unicodedecodeerror-ascii-codec-cant-decode-byte
- https://www.iis.sinica.edu.tw/papers/song/19840-F.pdf
- https://gogul09.github.io/software/image-classification-python
- https://www.researchgate.net/publication/303014817_Flower_Identification_System_by_Image_Processing
- http://scikit-learn.org/stable/supervised_learning.html#supervised-learning
- http://www.robots.ox.ac.uk/~vgg/publications/papers/nilsback08.pdf