TREE SPECIES RECOGNITION

INTRODUCTION TO IMAGE PROCESSING AND COMPUTER VISION

ADEDAYO TOLULOPE ADEDAYO - 24 January 2022

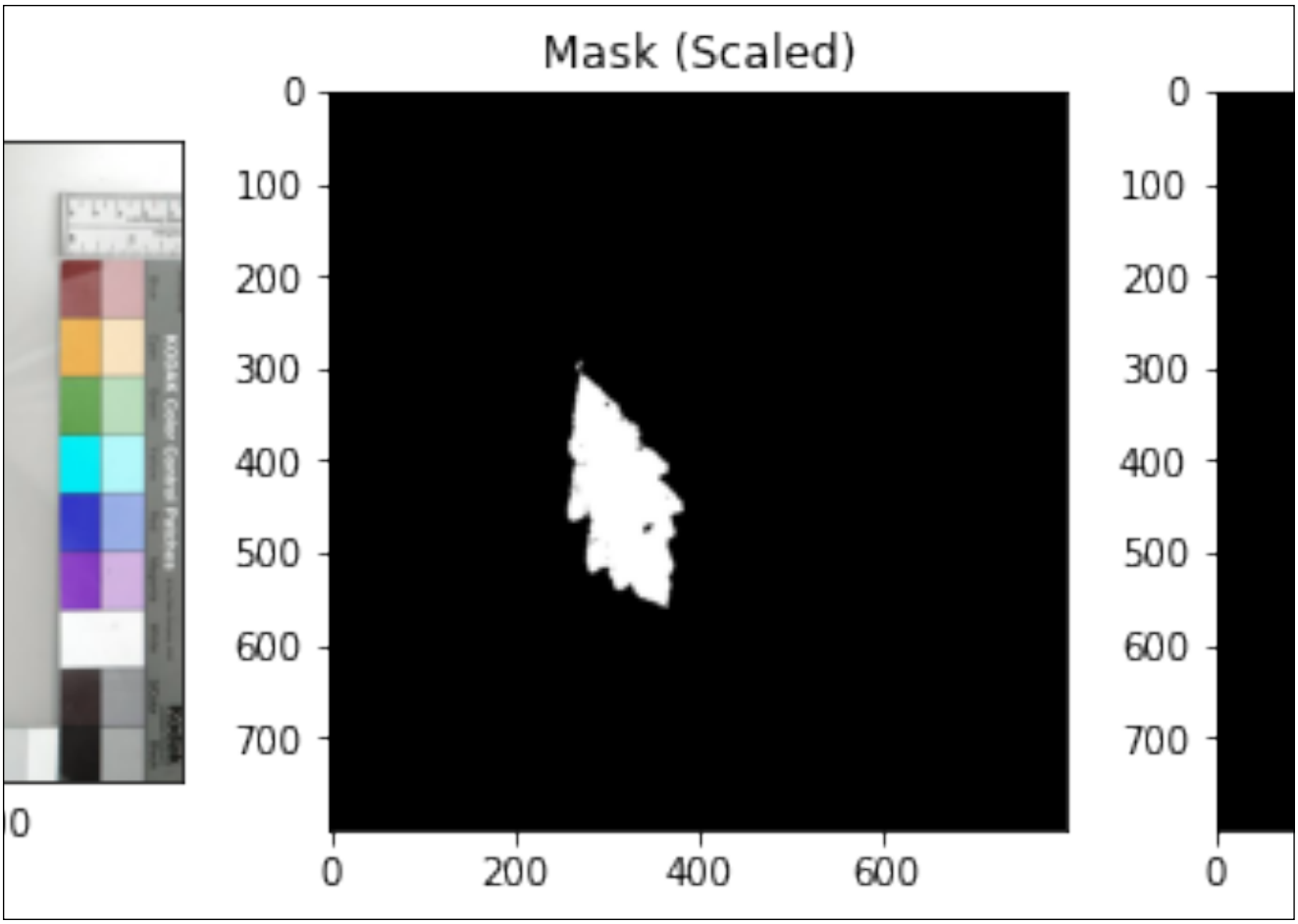


Table of Contents

1. Introduction

Image classification is a techniques used in detecting, classifying and predicting the class of an object in an image. Its main goal is the accurate identification of features in an image. A feature can be termed ad an individual measurable property or characteristic of a phenomenon being observed. Features are usually numeric, but structural features such as strings and graphs are used in syntactic pattern recognition.

Image classification is used in a lot of real life scenarios like medical imaging, object identification in images, traffic systems for control e.t.c. This makes it a very important aspect of computer vision.

In order to achieve the main goal of identifying features in an image, the features in the image needs to be extracted first. There are different techniques used in feature extraction including but not limited:

- Colour histogram : for extracting collar features
- Hu moments: for extracting shape features
- Haralick Texture: for extracting Texture features
- SIFT: for extracting local feature descriptors

1.1 Problem Definition

1.1.1 Task description

The aim of this project is to compare and assess the quality of different feature extraction methods for the classification of tree leaf species. The task is to classify each image according to their species. Below are sample images of the tree leaf species in the leaf snap dataset project. The expected input to this project is samples about 800 x 800 and

ACER_GINNALA

celtis_tenuifolia

zelkova_serrata

the expected output is samples class, classification accuracy and the features quality.

1.2 Dataset Overview

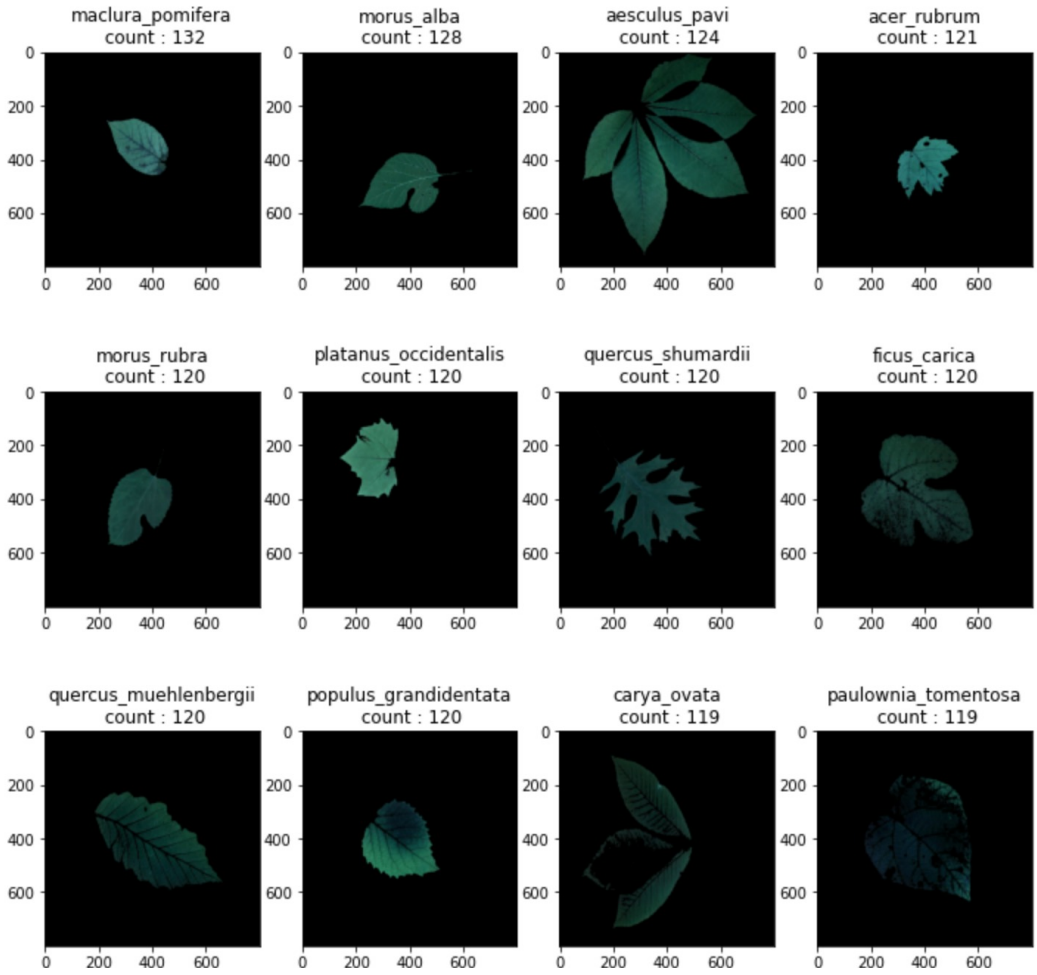
1.2.1 Dataset description

The dataset is a publicly available dataset: Leafsnap dataset. The dataset covers all 185 tree species from the Northeastern United States. It contains images taken from 2 different sources:

- "Lab" images, consisting of high-quality images taken of pressed leaves, from the Smithsonian collection. These images appear in controlled backlit and front-lit versions, with several samples per species.
- “Field" images, consisting of "typical" images taken by mobile devices (iPhones mostly) in outdoor environments. These images contain varying amounts of blur, noise, illumination patterns, shadows, etc..
- Segmented versions of all images, using the Leafsnap segmentation algorithm

1.2.2 Dataset preparation

For this project I will be using a subset of the Leafsnap “LAB" dataset. The classes chosen my dataset are and the number of images in each species:



Chosen Dataset

2. Proposed Solution

As discussed in the introduction, to correctly classify the leaf species accurately, feature identification and extraction is very important. The different types of features can be classified into global and local feature descriptors.

In this project, below are a list of the feature descriptors used:

1. Global feature descriptors :These feature descriptors takes in an entire image for processing and quantifies an image globally. Example of global feature descriptors used in this project:
   - Colour: Colour histogram which quantifies colour of the leafs
   - Shapes: Hu moments which quantifies the shape of the leaf
   - Texture: Haralick Texture and Local binary patterns both quantifies the texture of the leaf.
2. Local Features: These feature descriptors quantifies local regions of an image, they do not use a full image. The local feature used in this project is SIFT (Scale Invariant Feature Transform) and HOG(Histogram of Gradient). SIFT selects key points that are stable in scale space then, detects and describes the local features in the images. The HOG feature descriptor counts the occurrences of gradient orientation in localised portions of an image
3. Combination of features:
   - Combination of global feature descriptors: This is achieved by concatenating each global feature to from one global feature vector. Here I will be combining all the global features as one global feature vector. I will also be combining the colour histogram and Haralick feature to form one global feature vector.
   - Bags of Features (BOF/BOVW) : This is achieved by combining local feature descriptors or combining both global and local feature vectors.

The dataset is divided into 80% for training and 20% for testing. To create models for each feature, I will use multiple classifiers:Logistic Regression, Linear Discriminant Analysis, K-Nearest Neighbours, Decision Trees, Random Forests, Gaussian Naive Bayes and Support Vector Machine. The K-Fold Cross Validation techniques is also used to predict the ML model's accuracy and choose the best classifier. The best classifier is used to test unseen data.

2.1 Solution description

2.1.1. Preparing the data

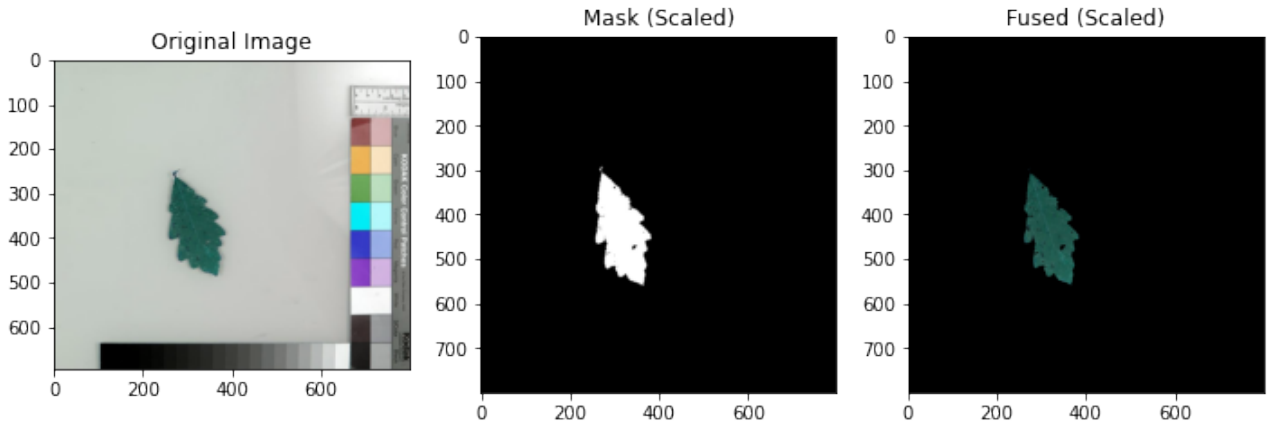
To prepare the images for feature extraction, the original image is fused with the masked image and rescaled to 800 x 800 using the defined "scale_and_apply_mask" function .

Then, using the HSV colour range mask to remove the noise on the side of the images (the colour palette). An example is seen in the image below.

```python
if not os.path.isdir("./clean_data"):
    os.mkdir("./clean_data")

for leaf_class_num, leaf_class in tqdm_notebook(enumerate(os.listdir('./dataset/images/lab'))):
    if not os.path.isdir(f"./clean_data/{leaf_class}"):
        os.mkdir(f"./clean_data/{leaf_class}")
    counter = 0
    for leaf_num , leaf_image_path in enumerate(glob.glob(f'./dataset/images/lab/{leaf_class}/*.jpg')):
        segmented_leaf_image_path = leaf_image_path.replace("images","segmented").replace("jpg", "png")
        leaf_image = cv2.imread(leaf_image_path)
        leaf_image_segmented = cv2.imread(segmented_leaf_image_path)
        if 255 in leaf_image_segmented:
            if leaf_class_num == 0 and leaf_num == 0:
                result = scale_and_apply_mask(leaf_image, leaf_image_segmented, True)
            else :
                result = scale_and_apply_mask(leaf_image, leaf_image_segmented, False)

            image_hsv = cv2.cvtColor(result, cv2.COLOR_BGR2HSV)

            mask_yellow_green = cv2.inRange(image_hsv, (10, 39, 30), (86, 255, 255))
            mask_brown = cv2.inRange(image_hsv, (8, 60, 10), (30, 255, 200))
            mask = cv2.bitwise_or(mask_yellow_green, mask_brown)
            result = cv2.bitwise_and(result, result, mask=mask)


            if  (result > 0).sum() > 24000:
                cv2.imwrite(f"./clean_data/{leaf_class}/{counter}.png", result)
                counter = counter +1
```

```python
def scale_and_apply_mask(image, mask, plot_example):
    original_image = cv2.resize(image, dsize=(800, 800))
    mask  = cv2.resize(mask, dsize=(800, 800))
    original_image[mask != 255] = 0
    if plot_example :
        _, ax = plt.subplots(nrows=1, ncols=3, figsize=(12,4))
        plt.sca(ax[0])
        plt.title("Original Image")
        plt.imshow(image)

        plt.sca(ax[1])
        plt.title("Mask (Scaled)")
        plt.imshow(mask)

        plt.sca(ax[2])
        plt.title("Fused (Scaled)")
        plt.imshow(original_image)

        plt.show()

    return original_image
```

Preparing data Code

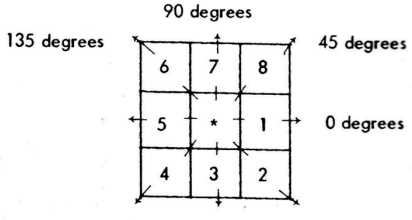


Sample output of processed image

The next step after processing the images is to extract the features.

2.1.2 Feature extractors

1. **Haralick Tecture** : Haralick Texture also known as GLCM (Gray level co-occurrence matrix) was suggested by Robert Haralick. It is based on the co-occurrence matrix of



Haralick Texture

joint probability of distribution of pixel pairs functions. This matrix contains counted pairs of pixels having the same distribution of gray level values.

To extract the Haralick feature, we convert the image to grayscale then the `mahotas.features.haralick()from the` mahotas library is used to get the feature descriptor.

```python
# 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
```

Haralick code

2. **Hu Moments:** Hu moment invariants are a set of 7 numbers calculated using central moments that are invariant to image transformations. The first 6 moments have been proved to be invariant to translation, scale, and rotation, and reflection. While the 7th moment's sign changes for image reflection.

$$h_0 = \eta_{20} + \eta_{02}$$
$$h_1 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$
$$h_2 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$
$$h_3 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$
$$h_4 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$
$$h_5 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})]$$
$$h_6 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

Hu moments

To extract Hu Moments features from the image, the image is first converted to grayscale. Then, the `cv2.HuMoments()` function provided by OpenCV takes in the flattened

```python
# 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
```

Hu moments code

cv2.moments as an arguments. The argument to this function is the flattened moments of the image `cv2.moments()`.

3. **Colour Histogram**: Colour histograms feature is extracted by usingTthe `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 normalise the histogram using `normalize()` function of OpenCV and return a

```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()
```

Colour Histogram code

flattened version of this normalised matrix using `flatten()`.

4. **Local Binary Pattern histogram**: LBP focuses on points surrounding a central point and tests if the surrounding points are greater than or less than the central point.

```python
# feature-descriptor-4: Local binary patterns histogram
def lbph_describe(image, eps=1e-7):
        # compute the Local Binary Pattern representation
        # of the image, and then use the LBP representation
        # to build the histogram of patterns
        lbp = feature.local_binary_pattern(image[:,:,0], 24, 8, method="uniform")
        (hist, _) = np.histogram(
            lbp.ravel(),
            bins=np.arange(0, 27),
            range=(0, 10),
        )
        # normalize the histogram
        hist = hist.astype("float")
        hist /= hist.sum() + eps
        # return the histogram of Local Binary Patterns
        return hist
```

Local Binary Pattern code

The parameters used in this code are : 24 number of circularly symmetric neighbour set points (quantisation of the angular space) and **8** Radius of circle (spatial resolution of the operator). The **Method = "uniform"** is used for improving the rotation invariance with uniform patterns and finer quantisation of the angular space which is grey scale and rotation invariant. Then, a numpy histogram based on the result is created and normalised.

5. **Histogram of Oriented Gradient features**: This feature descriptor focuses on the shape/structure of an object in the image. It counts the occurrences of gradient orientation in localised portions of an image and generates a **Histogram** for each of these regions separately.To get the HoG feature descriptor, we firstly resize the shape

```
# feature-descriptor-6: HOG FEATURES

# importing required libraries
from skimage.io import imread, imshow
from skimage.transform import resize
from skimage.feature import hog
from skimage import exposure

def fd_hog(img):

  #resize image
  resized_img = resize(img, (128,64))

  #generating HOG features
  fd, hog_image = hog(resized_img, orientations=9, pixels_per_cell=(8, 8),
  cells_per_block=(2, 2), visualize=True, multichannel=True)

  return fd
```

Histogram of Gradient Code

of the image to 64 x 128. Then, the skimage.feature HOG method which calculates the gradients and returns the feature matrix is used. Below are the parameter description of the hog():

- The *orientations* are the number of buckets we want to create. Here set it to 9
- *pixels_per_cell* defines the size of the cell for which we create the histograms. Here a 8 x 8 cells is used.
- Hyperparameter *cells_per_block* which is the size of the block over which we normalise the histogram. 2 x 2 cells per block is used here

6. **SIFT: Scale Invariant Feature transform** main steps are:
   - detection of Scale Space Extrema: Candidate keypoints are defined as maxima and minima of the DoG applied in scale space. Scale invariant
   - Accuracy key point localisation: a Taylor series is computed around the candidate keypoint. Then, discard low-contrast keypoints by checking the value of the second order Taylor expansion at the offset. The edge like key points are also discarded.
   - **Orientation assignment**: Compute gradient for each pixel in a region around the keypoint, and create a histogram of gradients (36 angle bins). Rotation invariant
   - Keypoints are described: This is illumination invariant.

To get the local feature using SIFT, the Cv2.SIFT_Create is used to create the sift key point extractor. Then the sift.detectandcompute() method is used on the created sift key point extractor to compute detect features from the image. Then a BAG OF FEATURES(BOF) function is used to assign a set of features that are similar to a specific cluster centre thereby forming a bag of word approach.

```python
# feature-descriptor-5: SIFT
import cv2

def fd_sift(img):
    # reading the image

    # create SIFT keypoint feature extractor
    # sift = cv2.xfeatures2d.SIFT_create()
    sift = cv2.SIFT_create()



    # detect features from the image

    keypoints, descriptors = sift.detectAndCompute(img, None)

    return descriptors

def bag_of_features(features, centres, k = 150):
    vec = np.zeros((1, k))
    for i in range(features.shape[0]):
        feat = features[i]
        diff = np.tile(feat, (k, 1)) - centres
        dist = pow(((pow(diff, 2)).sum(axis = 1)), 0.5)
        idx_dist = dist.argsort()
        idx = idx_dist[0]
        vec[0][idx] += 1
    return vec
```

SIFT CODE

7. Global Feature descriptors: Here a combination of global features are used.
    - The first one is the concatenation of colour histogram and haralick features
    - The second id the combination of all global features(colour histogram, haralick, hu moments, local binary pattern)

```python
###################################
        # Global Feature extraction
        ###################################

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

        ###################################
        # Concatenate global features
        ###################################
        global_feature_all = np.hstack([fv_histogram, fv_haralick, fv_hu_moments, fv_lbph_describe])
        global_feature = np.hstack([fv_histogram, fv_haralick])
```

Global Features Codes

2.1.3 Training

The training dataset is 80% of my chosen dataset and multiple classifiers were used to create models for each feature. The classifiers are: Logistic Regression, Linear Discriminant Analysis, K-Nearest Neighbours, Decision Trees, Random Forests, Gaussian Naive Bayes and Support Vector Machine.

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

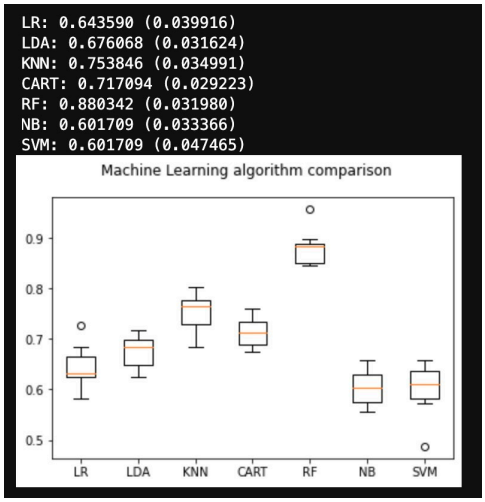
```
# split the training and testing data
(trainDataGlobal, testDataGlobal, trainLabelsGlobal, testLabelsGlobal) =
train_test_split(np.array(global_features),

 np.array(global_labels),

 test_size=test_size,

 random_state=seed)
```

```
# variables to hold the results and names
results = []
names   = []

# 10-fold cross validation for Local feature : HOG
for name, model in tqdm_notebook(models):
    kfold = KFold(n_splits=10, shuffle=True)
    cv_results = cross_val_score(model, trainDatahog, trainLabelshog, 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()
```

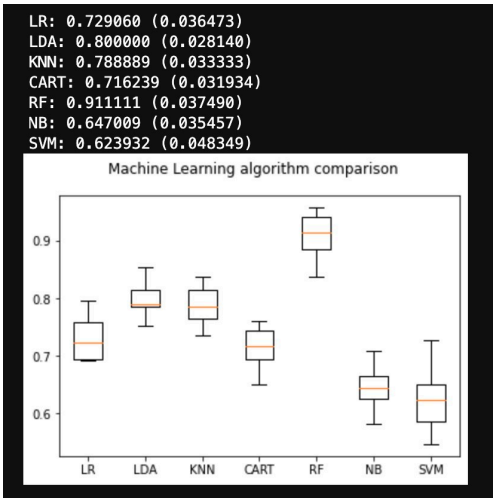
Training Code

The K-Fold Cross Validation techniques is also used to predict the ML model's accuracy and choose the best classifier. The best classifier is used to test unseen data.

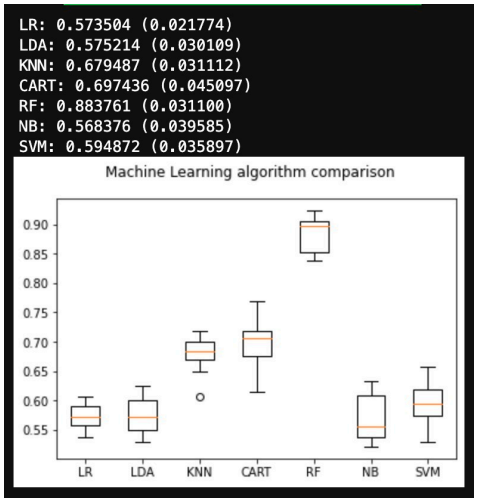
Results of the K-Fold cross validation:



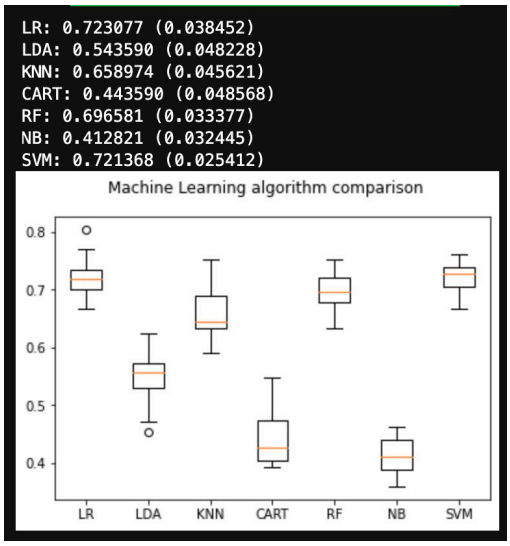
Global feature: haralick and colour histogram



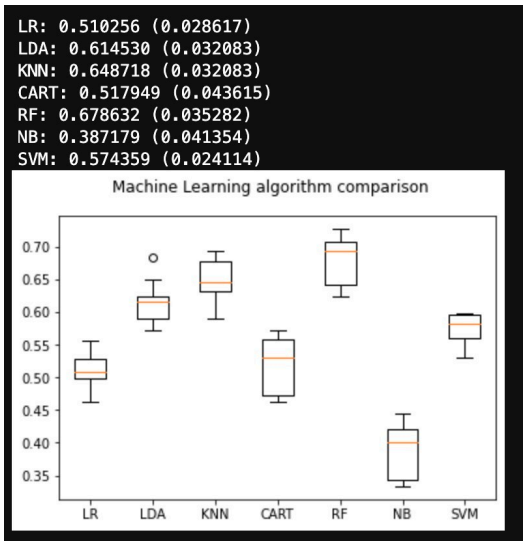
Global feature vector for all global features



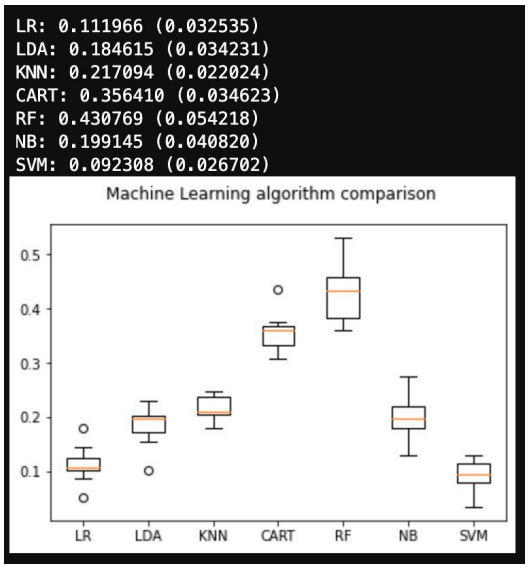
Colour Histogram



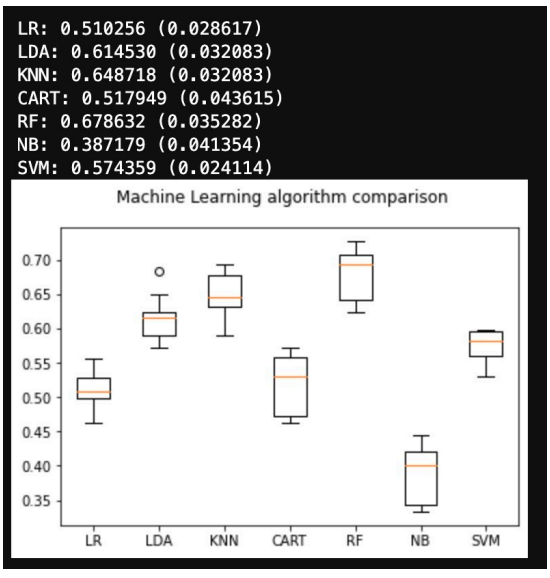
Histogram of Oriented Gradient



Local Binary Patterns



Hu Moments



Harralick

2.1.3 Testing

The testing dataset is 20% of my chosen dataset and is used in testing the Random Forests classifier as it is the best classifier on average during the K-FOLD cross validation.

```
clf = RandomForestClassifier(n_estimators=15, random_state=seed, max_depth = 20)
clf.fit(trainDatahog, trainLabelshog)
print("Random Forest : score on training set params: ", clf.score(trainDatahog, trainLabelshog))
print("Random Forest : score on testing set params: ", clf.score(testDatahog, testLabelshog)
```

Random Forest test code

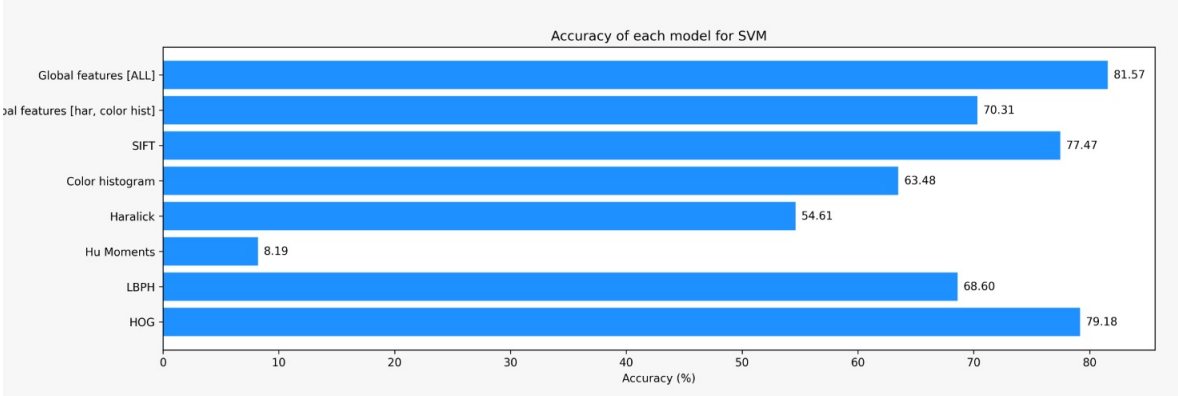
The test data was also used to test the SVM model.

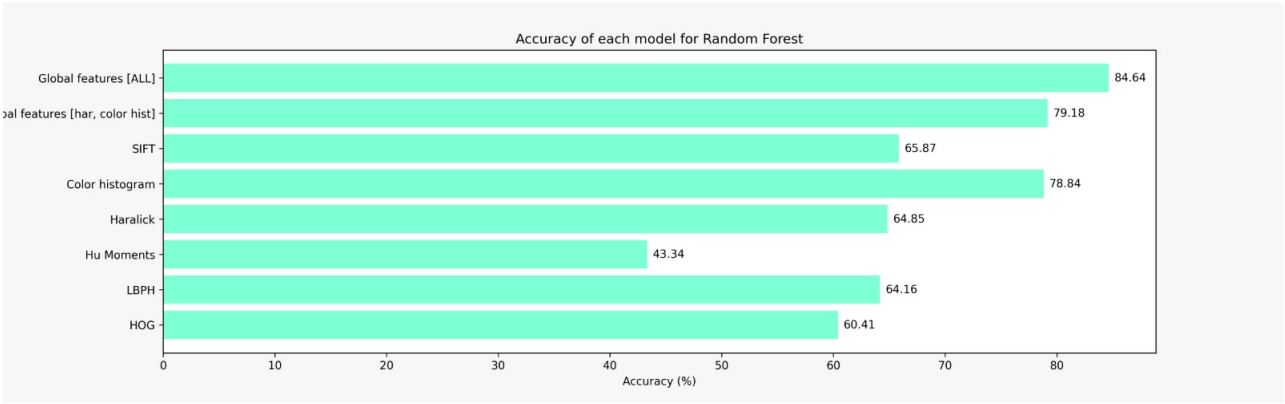
```
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import GridSearchCV
model_svm = SVC()
kernel = ["linear", "rbf", "sigmoid", "poly"]
tolerance = [1e-3, 1e-4, 1e-5, 1e-6]
C = [1, 1.5, 2, 2.5, 3]
grid = dict(kernel=kernel, tol=tolerance, C=C)

cvFold = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
gridSearch = GridSearchCV(estimator=model_svm, param_grid=grid, n_jobs=-1,
    cv=cvFold, scoring="neg_mean_squared_error")
searchResults = gridSearch.fit(trainDatahog, trainLabelshog)
# extract the best model and evaluate it

bestModel = searchResults.best_estimator_
print("SVM : score on training set params: ", bestModel.score(trainDatahog, trainLabelshog))
print("SVM : score on testing set params: ", bestModel.score(testDatahog, testLabelshog))
```

SVM test code

3. RESULTS

The following results were achieved in the project:
Above shows the accuracy of each Feature descriptor for SVM model





Above shows the accuracy of each feature descriptor using the Random Forest Model.

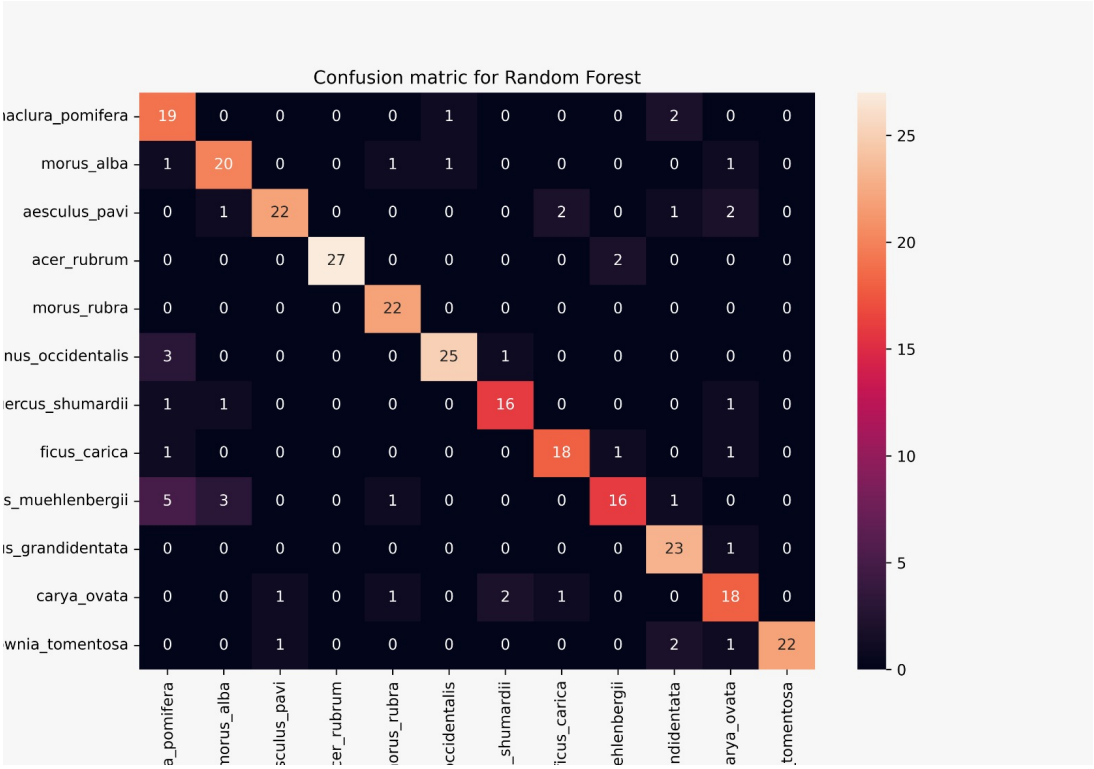
In conclusion, it is noticed that the Random Forest model works better with the Local feature descriptors than the SVM. Also the global feature descriptor comprising of all the global features works best in both the SVM and Random Forest model.

In order to improve the accuracy, the use of BOF(Bag of features) comprising of both global and local features can be used. Also more data for each class can also help in improving the accuracy of the classifier.

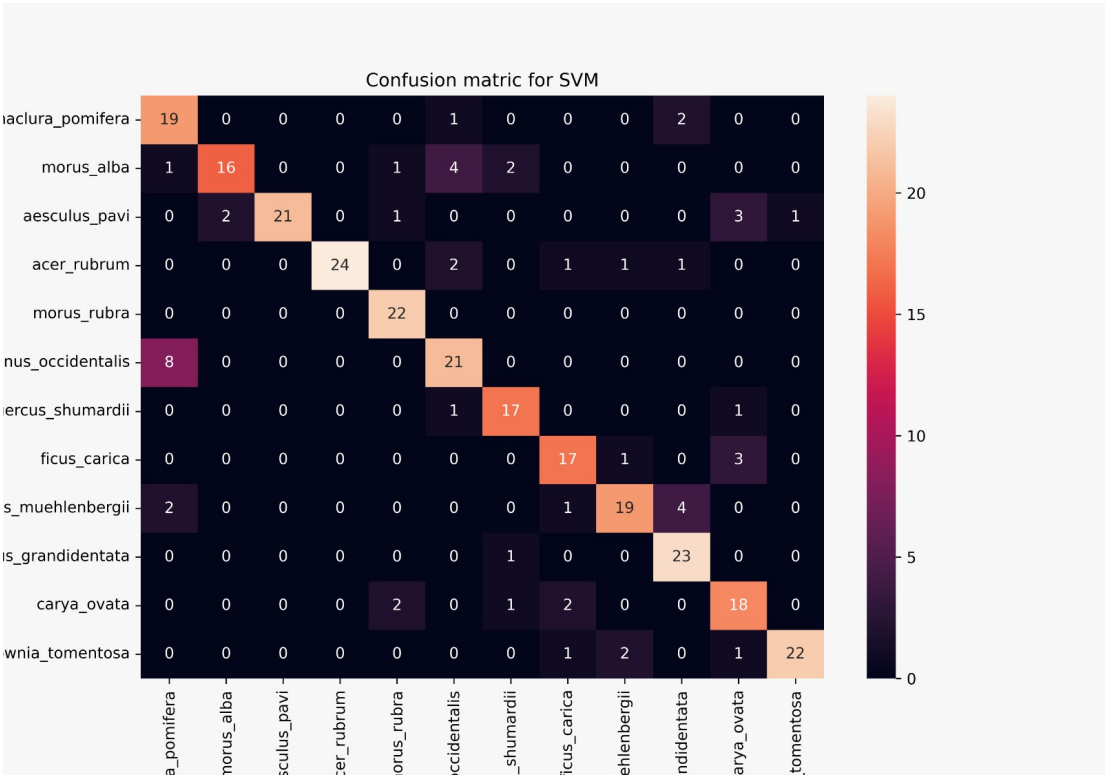
As it was observed during the experiments, using Hu moments alone is not great for the recognition of leaves as it is a shape feature extractor. Shape alone is not good for classifying leaves as they all have similar shape.

The LBPH and Haralick are both texture based feature descriptor. In the SVM classifier model, it was observed that LBPH was a bit more accurate than Haralick. However , in the Random Forest classifier they are almost on par.

Below is an image showing the performance of the random forest model on the different classes in my dataset. It shows how many numbers of the classes were correctly classified and the classes there were misclassified as. For example in the image below, the class “ficus_carica” was accurately classified 18 times out of 21. But it was mis classified as maclura_pomifera, quercus_ muehlenbergii and carya_ovata once respectively.



Below is an image showing the performance of the support Vector Machine model on the different classes in my dataset.



4.REFERENCES

1. https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/
2. https://gogul.dev/software/image-classification-python
3. https://github.com/Akhilesh64/Image-Classification-using-SIFT/blob/main/main.py
4. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/ py_table_of_contents_feature2d/py_table_of_contents_feature2d.html