WARSAW UNIVERSITY OF TECHNOLOGY
Faculty of Mathematics
and Information Science

Introduction to Image Processing and Computer Vision

Project 2:
Tree Species Recognition

Submitted by:
Amir Ali
317554

Master's in Data Science

15 January 2022

Table of Content

1. Abstract ……...03
2. Introduction ……...03
3. Literature Review ……04
4. Problem Statement …….. 04
5. System Architecture ……... 04
6. Results ……..… 12
7. Conclusion …….. 14

Reference …… 15

1. Abstract

Classification is one of the most important approaches to machine learning. Main task of machine learning is data analysis. Various algorithms are available for classification. In this project, we implement different machine learning classifiers to classify images based on their HOG features for Tree Species Recognition. Scikit tool is used for the implementation purpose. The dataset is too large and we take only 10 label class for implementation. And Support Vector Machine gave the best result.

2. Introduction

One of the most fundamental tasks in computer vision is image classification. Multi-label classification is a classification task where each image can contain more than one label, and some images can contain all the labels simultaneously. [1]

In this study, we will use Tree Species Dataset which has 185 label classes. And we use only randomly 10 labels classes for the classification task. First, we will do Feature Extraction by Implementing Histogram of Oriented Gradients (HOG) using scikit learn and then we implement Support Vector Machine to classify Tree Species Images.

There are many types of leaves in different shapes and colors. As you can see some sample below:

Figure 1.1: Sample of Input Images

3. Literature Review

The use of earth observation imagery as an alternative to the SDM-based approach for mapping tree species is an option. The identification of tree species is a well-known problem in forest optical remote sensing [2][3]. Many studies have already looked into this topic in various ways. Obtaining a precise tree species classification, however, remains a difficult undertaking. The spectral response of species is influenced by a variety of parameters including leaf biochemical characteristics, canopy structure, forest density and maturity, illumination, and acquisition conditions. As a result, spectral variability within species may be higher than variability between species at times.

In this paper [4], they proposed utilizing classification to identify iris flowers. They produced predictions based on data that was not used to train the model in this study. They demonstrated machine learning methods that accurately predict species features. They have worked on the machine learning model by training data sets, and they have also established a model for species prediction.

In this paper [5] worked on an effective neural fuzzy classification technique. The proposed technique is based on Iris informative indexes and divides the dataset into four categories in this research. In this case, the algorithm might select the most important features and eliminate a small but enough set of standards for the grouping assignment.

4. Problem Statement

The objective of image classification is to identify and portray, as a unique gray level (or color), the features occurring in an image in terms of the object or type of land cover these features actually represent on the ground.

In this study, we will use Leaf Dataset for Image Recognition [6]. The dataset currently covers all 185 tree species from the Northeastern United States. And we use only ten labels species data and built a SVM model to classify images based on their HOG features.

In this dataset two different subsets are included:

- 23147 Lab images, made up of high-resolution photos of pressed leaves from the Smithsonian's collection with many samples per species, these photos are available in controlled backlit and front-lit versions.
- 7719 Field images, comprising of "typical" photographs captured using mobile smartphones in natural settings. These photos have various levels of blur, noise, illumination patterns, shadows, and other effects.

5. System Architecture

We divide our solution into three parts in the first part we did data preprocessing after that we did Feature Extraction and then, in the end, we build model for classification of tree species.

5.1 Data Preprocessing

we create a function that reads, resizes, and saves the data in a dictionary that includes the images, labels (Tree Species), original filenames, and a description. Numpy arrays containing the RGB values of the images are used to store them. Using joblib, the dictionary is stored to a pickle file. The data structure is similar to that used in scikit-test learn's data sets.

```
def resize_all(src, pklname, include, width=150, height=None):

    height = height if height is not None else width

    data = dict()
    data['description'] = 'resized ({0}x{1})Tree Species images in
    rgb'.format(int(width), int(height))
    data['label'] = []
    data['filename'] = []
    data['data'] = []

    pklname = f"{pklname}_{width}x{height}px.pkl"

    # read all images in PATH, resize and write to DESTINATION_PATH
    for subdir in os.listdir(src):
        if subdir in include:
            print(subdir)
            current_path = os.path.join(src, subdir)

            for file in os.listdir(current_path):
                if file[-3:] in {'jpg', 'png'}:
                    im = imread(os.path.join(current_path, file))
                    im = resize(im, (width, height)) #[:,:,::-1]
                    data['label'].append(subdir[:-4])
                    data['filename'].append(file)
                    data['data'].append(im)

        joblib.dump(data, pklname)
```

After that we apply the resize function and take only ten Labels Tree Species Data. and we take only 10 Lablel Species

```
# Apply the resize function and take only ten Labels Tree Species Data
base_name = 'Tree_Species_Recognition'
width = 80

# I take only 10 Lablel Species
include = {'abies_nordmanniana',
 'acer_campestre',
 'acer_ginnala',
 'acer_pensylvanicum',
 'amelanchier_canadensis',
 'castanea_dentata',
 'cercidiphyllum_japonicum',
 'evodia_daniellii',
 'robinia_pseudo-acacia',
 'ulmus_pumila'}
```

```
resize_all(src=data_path, pklname=base_name, width=width, include =
include)
```

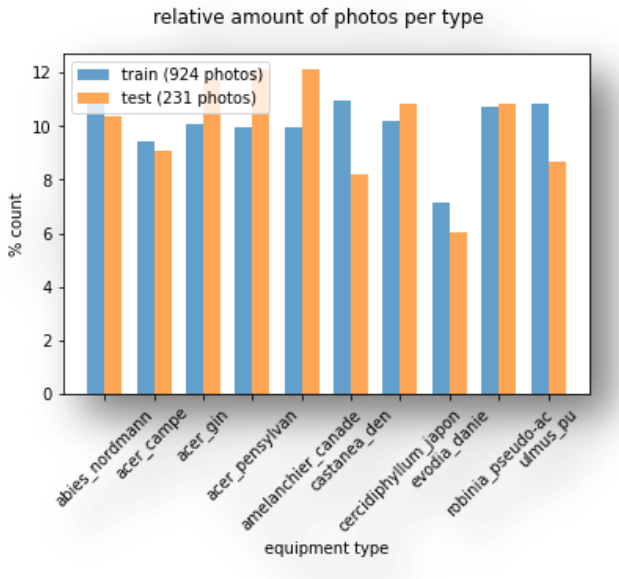
Here you can see the summary of our data

```
number of samples:  1155
keys:  ['description', 'label', 'filename', 'data']
description:  resized (80x80)Tree Species images in rgb
image shape:  (80, 80, 3)
labels: ['abies_nordmann' 'acer_campe' 'acer_gin' 'acer_pensylvan'
 'amelanchier_canade' 'castanea_den' 'cercidiphyllum_japon' 'evodia_danie'
 'robinia_pseudo-ac' 'ulmus_pu']

Counter({'abies_nordmann': 124,
         'acer_campe': 108,
         'acer_gin': 120,
         'acer_pensylvan': 120,
         'amelanchier_canade': 120,
         'castanea_den': 120,
         'cercidiphyllum_japon': 119,
         'evodia_danie': 80,
         'robinia_pseudo-ac': 124,
         'ulmus_pu': 120})
```

After that we plot each label class images as you can see

5.2 Split Data into Training and Testing

After that we split our data into training and testing. And the ratio for training and testing is 80% and 20%. Respectively



The distributions are not perfectly equal, but good enough for now.

5.3 Feature Extraction

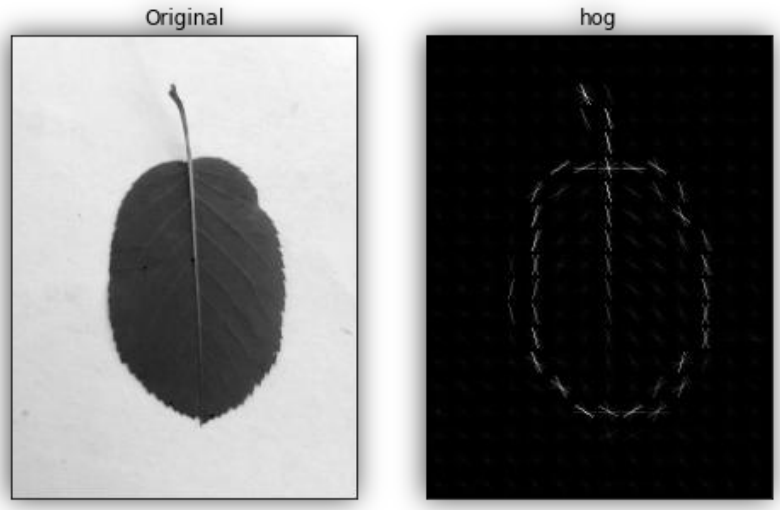
We used Histogram of Oriented Gradients (HOGs) for feature reduction, in other words: for lowering the complexity of the problem, while maintaining as much variation as possible.
To calculate a HOG, an image is divided into blocks, for example 8 by 8 pixels. For each of these blocks, the magnitude of the gradient in a given number of directions is calculated.[7]
We used a transformation when calculating our HOG. We use transformers to transform our entire data set. These are objects that accept an array of data, transform each item, and then return the transformed data.
First, convert color photos to grayscale, then calculate their HOGs before scaling the data. RGB2GrayTransformer, HOGTransformer, and StandardScaler are used. The output is an array containing a HOG for each image in the input.
Many built-in transformers are included in Scikit-learn, including a StandardScaler for scaling features and a Binarizer for mapping string features to numerical features. It also includes the BaseEstimator and TransformerMixin classes, which make creating your own Transformers easier.
By inheriting from these two classes and implementing a __init__, fit, and transform function, a custom transformer can be created. The fit transform method of the TransformerMixin class combines the fit and transform that we implemented.
The RGB2GrayTransformer and HOGTransformer are defined below.

```
('grayify', RGB2GrayTransformer()),
('hogify', HogTransformer(orientations=8)),
('scalify', StandardScaler()),
('classify', SVC(kernel='linear'))])
```

```
class RGB2GrayTransformer(BaseEstimator, TransformerMixin):
    """
    Convert an array of RGB images to grayscale
    """

    def __init__(self):
        pass

    def fit(self, X, y=None):
        """returns itself"""
        return self

    def transform(self, X, y=None):
        """perform the transformation and return an array"""
        return np.array([skimage.color.rgb2gray(img) for img in X])


class HogTransformer(BaseEstimator, TransformerMixin):
    """
    Expects an array of 2d arrays (1 channel images)
    Calculates hog features for each img
    """
```

```
def __init__(self, y=None, orientations=9,
             pixels_per_cell=(8, 8),
             cells_per_block=(3, 3), block_norm='L2-Hys'):
    self.y = y
    self.orientations = orientations
    self.pixels_per_cell = pixels_per_cell
    self.cells_per_block = cells_per_block
    self.block_norm = block_norm

def fit(self, X, y=None):
    return self

def transform(self, X, y=None):

    def local_hog(X):
        return hog(X,
                   orientations=self.orientations,
                   pixels_per_cell=self.pixels_per_cell,
                   cells_per_block=self.cells_per_block,
                   block_norm=self.block_norm)

    try: # parallel
        return np.array([local_hog(img) for img in X])
    except:
        return np.array([local_hog(img) for img in X])
```

Below you can see the example after applying HOG




```
number of pixels:  53133
number of hog features:  8424
```

The number of data points to process in our model has been reduced to ~15%, and with some imagination we can still recognize a Leaf Image in the HOG.

Note that for compatibility with scikit-learn, the fit and transform methods take both X and y as parameters, even though y is not used here.

With this, we are all set to preprocess our RGB images to scaled HOG features.

```
# create an instance of each transformer
grayify = RGB2GrayTransformer()
hogify = HogTransformer(
    pixels_per_cell=(14, 14),
    cells_per_block=(2,2),
    orientations=9,
    block_norm='L2-Hys'
)
scalify = StandardScaler()

# call fit_transform on each transform converting X_train step by step
X_train_gray = grayify.fit_transform(X_train)
X_train_hog = hogify.fit_transform(X_train_gray)
X_train_prepared = scalify.fit_transform(X_train_hog)

print(X_train_prepared.shape)

# tranform X_test as well
X_test_gray = grayify.transform(X_test)
X_test_hog = hogify.transform(X_test_gray)
X_test_prepared = scalify.transform(X_test_hog)
```

5.4 Experiments with Different Machine Learning Classifiers

Our Data is ready after feature extraction. Now it's time to build Machine Learning Classifier to predict the result. And, we experiments with five different machine learning classifier as you can see below:

5.4.1 Support Vector Machine

A Support Vector Machine (SVM) is a discriminative classifier that can be used for both classification and regression problems. But we use to classify for tree species recognition. The goal of SVM is to identify an optimal separating hyperplane which maximizes the margin between different classes of the training data. [8]

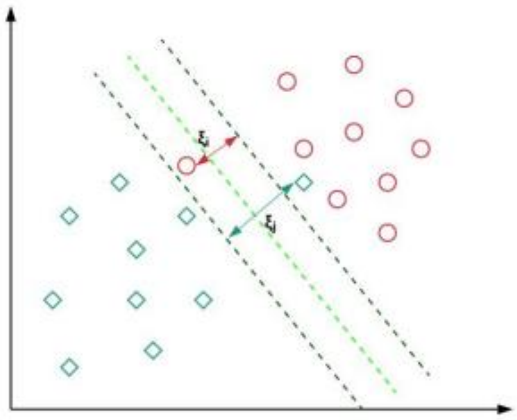


Figure 5.1: SVM with Kernel

```
# import svm model
from sklearn.svm import SVC

# init the model
svm = SVC(kernel = 'linear', random_state= 0)
```

```
# fit the data into model
svm.fit(X_train_prepared, y_train)

# predict the result
sym_y_pred = sgd_clf.predict(X_test_prepared)
```

5.4.2 Logistic Regression

It is a statistical method for analyzing a data set in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables. [9]

$$f(x) = \frac{1}{1+e^{-x}}$$

```
# import the model from sklean
from sklearn.linear_model import LogisticRegression

# initialize the model
lg = LogisticRegression(random_state= 0)

# fit the dataset into our classifier model for training
lg.fit(X_train_prepared, y_train)

# predict the result
lg_y_pred = lg.predict(X_test_prepared)
```

5.4.3 Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression, and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. [10]

```
# import the model from sklean
from sklearn.ensemble import RandomForestClassifier

# initialize the model
rf = RandomForestClassifier(n_estimators=11, criterion = 'entropy',
random_state=0)

# fit the dataset into our classifier model for training
rf.fit(X_train_prepared, y_train)

# predict the result
rf_y_pred = rf.predict(X_test_prepared)
```

5.4.4 Decision Tree

A decision tree is a tree whose internal nodes can be taken as tests (on input data patterns) and whose leaf nodes can be taken as categories (of these patterns). These tests are filtered down through the tree to get the right output to the input pattern. Decision Tree algorithms can be applied and used in various different fields. It can be used as a replacement for statistical procedures to find data, to extract text, to find missing data in a class, to improve search engines and it also finds various applications in medical fields. Many Decision tree algorithms have been formulated. They have different accuracy and cost-effectiveness. It is also very important for us to know which algorithm is best to use. The ID3 is one of the oldest Decision tree algorithms. It is very useful while making simple decision trees but as the complications increase its accuracy to make good Decision trees decreases. Hence IDA (intelligent decision tree algorithm) and C4.5 algorithms have been formulated.[11]

```
# import the model from sklean
from sklearn.tree import DecisionTreeClassifier

 # initialize the model
dt = DecisionTreeClassifier( criterion = 'entropy', random_state=0)

# fit the dataset into our classifier model for training
dt.fit(X_train_prepared, y_train)

# predict the result
dt_y_pred = dt.predict(X_test_prepared)
```

5.4.5 Naïve Bayes

It is a classification technique based on the Bayesian network with an assumption of Independence among predictors. In simple terms, a Naive Bayes classifier assumes that the Presence of a particular feature in a class is unrelated to the presence of any other feature. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability. NB is based on probability Estimations, called a posterior probability.[12]

$$P(a \mid b) = \frac{P(b \mid c) * P(a)}{P(b)}$$

```
# import the model from sklean
from sklearn.naive_bayes import GaussianNB

# initialize the model
nb = GaussianNB()

# fit the dataset into our classifier model for training
nb.fit(X_train_prepared, y_train)

# predict the result
nb_y_pred = nb.predict(X_test_prepared)
```

6. Result Evaluation

6.1 Accuracy Score

The proposed method is experimented with on Tree Species Images on randomly 10 labels from 184different species class. However, we implement different machine learning classifiers like Support Vector Machine, Random Forest, Decision Tree, Logistic Regression, and Naïve Bayes to classify images based on their HOG features. And you can see the result of different classifiers that we use in this project.

```
# Accuracy Score of  Classifiers that we implement
lg_acc = 100*np.sum(lg_y_pred == y_test)/len(y_test)
svm_acc = 100*np.sum(svm_y_pred == y_test)/len(y_test)
dt_acc = 100*np.sum(dt_y_pred == y_test)/len(y_test)
rf_acc = 100*np.sum(rf_y_pred == y_test)/len(y_test)
nb_acc = 100*np.sum(nb_y_pred == y_test)/len(y_test)
print("Logistic Regression", lg_acc)
print("Support Vector Machine", svm_acc)
print("Decision Tree", dt_acc)
print("Random Forrest", rf_acc)
print("Naive Bayes", nb_acc)
```

```
Logistic Regression 97.40259740259741
Support Vector Machine 98.7012987012987
Decision Tree 94.37229437229438
Random Forrest 94.37229437229438
Naive Bayes 92.20779220779221
```

Figure 6.1: Accuracy Score

Below you can see the visualization of classifiers accuracy score

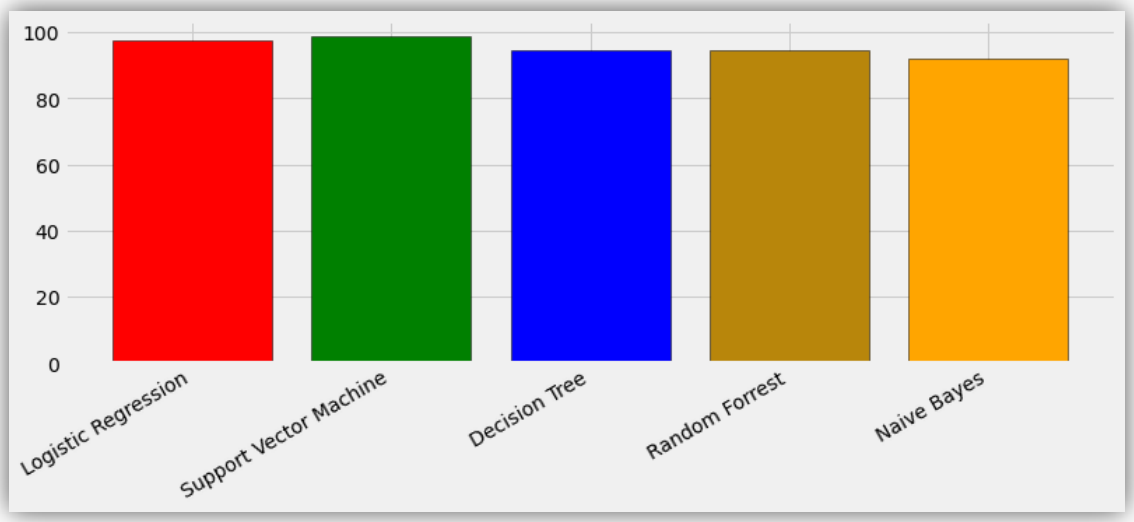


Figure 6.2: Results Evaluation

We experiment five different machine learning classifiers and as we can see SVM gave higher accuracy. So below you can see the confusion matrix result of SVM. Remember

we didn't use whole dataset we just use only 10 label tree species and this result is based on 10 label class that we use.

6.1 Confusion Matrix

During the training process, the confusion matrix was used to evaluate the classification models. The confusion matrix is a matrix that maps the predicted outputs across actual outputs. It is often used to describe the performance of a classification model on a set of test data.

Below you can see the confusion matrix result of SVM

```
confusion_matrix= confusion_matrix(y_test, svm_y_pred)
print(confusion matrix)
```

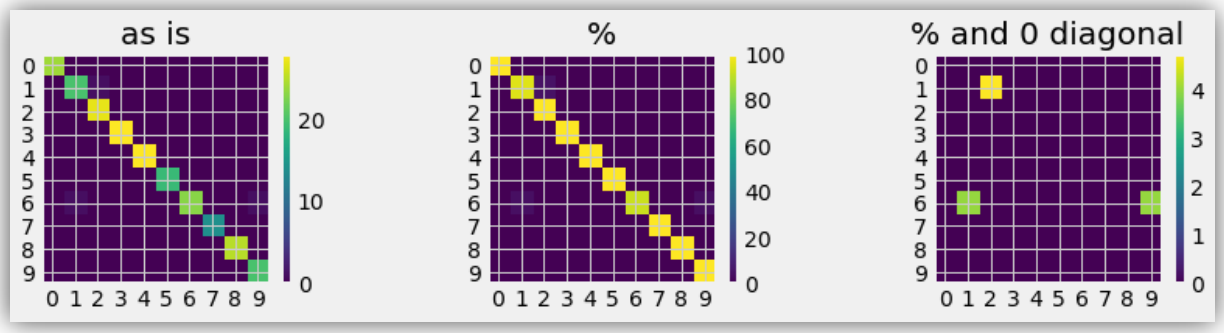
```
[[24  0  0  0  0  0  0  0  0  0]
 [ 0 20  1  0  0  0  0  0  0  0]
 [ 0  0 27  0  0  0  0  0  0  0]
 [ 0  0  0 28  0  0  0  0  0  0]
 [ 0  0  0  0 28  0  0  0  0  0]
 [ 0  0  0  0  0 19  0  0  0  0]
 [ 0  1  0  0  0  0 23  0  0  1]
 [ 0  0  0  0  0  0  0 14  0  0]
 [ 0  0  0  0  0  0  0  0 25  0]
 [ 0  0  0  0  0  0  0  0  0 20]]
```

Figure 6.3: Confusion Matrix of SVM

The largest values are on the diagonal, hence most of the predictions are correct. To visualize this more clearly as an image, we do two things. First, we normalize the matrix to 100, by dividing every value by the sum of its row (i.e. the number of actual items with a specific label). Second, we set the main diagonal to 0 in order to focus on the wrong predictions.

```
def plot_confusion_matrix(cmx, vmax1=None, vmax2=None, vmax3=None):
    cmx_norm = 100*cmx / cmx.sum(axis=1, keepdims=True)
    cmx_zero_diag = cmx_norm.copy()

    np.fill_diagonal(cmx_zero_diag, 0)

    fig, ax = plt.subplots(ncols=3)
    fig.set_size_inches(12, 3)
    [a.set_xticks(range(len(cmx)+1)) for a in ax]
    [a.set_yticks(range(len(cmx)+1)) for a in ax]

    im1 = ax[0].imshow(cmx, vmax=vmax1)
    ax[0].set_title('as is')
    im2 = ax[1].imshow(cmx_norm, vmax=vmax2)
    ax[1].set_title('%')
    im3 = ax[2].imshow(cmx_zero_diag, vmax=vmax3)
    ax[2].set_title('% and 0 diagonal')
```

```
dividers = [make_axes_locatable(a) for a in ax]
cax1, cax2, cax3 = [divider.append_axes("right", size="5%", pad=0.1)
                    for divider in dividers]

fig.colorbar(im1, cax=cax1)
fig.colorbar(im2, cax=cax2)
fig.colorbar(im3, cax=cax3)
fig.tight_layout()

plot_confusion_matrix(confusion_matrix)

# the types appear in this order
print('\n', sorted(np.unique(y_test)))
```



7. Conclusions

In this project, we classify tree species images. We take only 10 label class species images from 184 label species and built a different Machine Classifier models to classify images based on their HOG features. And Support Vector Machine gave best result as compare to other classifiers and achieved 0.98% accuracy. And most importantly, this methodology is generic and can be applied to all kinds of machine learning problems.

Remarks:

Thank you Dr. RAFAŁ JÓŹWIAK for assigning such a wonderful project on tree species recognition. The dataset is not so complicated to understand. And the biggest challenge in this project is to apply the Feature Extraction technique on all images. But it was very interesting for me to learn such technique like HOG and different ML classifiers. I really enjoyed it to finish it and I try my best to do it.

8. Reference

[1] https://www.v7labs.com/blog/image-classification-guide#what-is-image-classification

[2] Holmgren, P.; Thuressopn, T. Satellite remote sensing for forestry planning — A review. Scand. J. For. Res.1998, 13, 90–110.

[3] Boyd, D.S.; Danson, F.M. Satellite remote sensing of forest resources: Three decades of research development. Prog. Phys. Geogr. 2005, 29, 1–26.

[4] Cho, SungBae.and Dehuri, Satchidananda (2009) "A comprehensive survey on functional link neural network and an adaptive PSOBP learning for CFLNN, Neural Comput & Application"DOI10.1007/s00521-00902885.

[5] Vaishali Arya, R K Rathy, "An Efficient Neural-Fuzzy Approach For Classification of Dataset", International Conference on Reliability, Optimization and Information Technology Feb 2014.

[6] "Leafsnap: A Computer Vision System for Automatic Plant Species Identification," Neeraj Kumar, Peter N. Belhumeur, Arijit Biswas, David W. Jacobs, W. John Kress, Ida C. Lopez, João V. B. Soares,Proceedings of the 12th European Conference on Computer Vision (ECCV), October 2012.

[7] http://www.learnopencv.com/histogram-of-oriented-gradients/.

[8] Support Vector Machines — Soft Margin Formulation and Kernel Trick available at https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-andkernel-trick-4c9729dc8efe

[9]https://www.academia.edu/41037617/A_Briefly_Explanation_of_Logistic_Regression_with_Practical_Implementation_in_Scikit_Learn

[10]https://www.academia.edu/40999808/A_Deep_Level_Understanding_of_Random_Forest_with_Practical_Implementation_in_Scikit_Learn

[11]https://www.academia.edu/40971855/A_Briefly_Explanation_of_Decision_Tree_with_Practical_Implementation_in_Scikit_Learn

[12]https://www.academia.edu/40503407/An_Intuitive_Guide_of_Na%C3%AFve_Bayes_Classifier_with_Practical_Implementation_in_Scikit_Learn