Institute of Engineering & Technology

ऋते ज्ञानान्न मुक्तिः

GLA UNIVERSITY

ESTD. 2010

GLA UNIVERSITY

MATHURA

Established vide U.P. Act 21 of 2010.

Mini Project Report

On

Leaf Detection

Submitted by

Mudit Singhania
171500052

Sazal Goyal
171500307

Sidharth
171500195

Department of Computer Engineering & Applications
Institute of Engineering & Technology

GLA University, Mathura- 281406, INDIA

Submitted to:Manoj Varshney

Department of computer Engineering and Applications
GLA University, Mathura
17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha,
Mathura – 281406

Declaration

I hereby declare that the work which is being presented in the Mini Project Titled: **"Leaf Detection",** in partial fulfillment of the requirements for Mini-Project LAB, is an authentic record of our own work carried under the supervision of **Dr. Manoj Vashney, Technical Trainer, GLA University, Mathura**.

Signature of Candidate:

Name of Candidate: Sazal Goyal,Mudit Singhania,Siddharth

Roll. No. : 171500307,171500195,171500052

Course: Btech

Year: 3rd

Semester: VIth

Department of computer Engineering and Applications
GLA University, Mathura
17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha,
Mathura – 281406

CERTIFICATE

This is to certify that the project entitled **“Leaf Dtection”** carried out in Mini Project – II Lab is a bonafide work done by **Sazal Goyal(171500307), Mudit Singhania(17150052),Sidharth(171500195)** and is submitted in partial fulfillment of the requirements for the award of the degree Bachelor of Technology (Computer Science & Engineering)

.

Signature of Supervisor:

Name of Supervisor: Dr.Manoj Vashney

Date: 16/04/2020

ABSTRACT

The main objective is to develop for helping the farmers.Automated systems for plant recognition can be used to classify plants into appropriate taxonomies. Such information can be useful for botanists, industrialists, food engineers and physicians. In this work, a recognition system capable of identifying plants by using the images of their leaves has been developed. A mobile application was also developed to allow a user to take pictures of leaves and upload them to a server. The server runs pre-processing and feature extraction techniques on the image before a pattern matcher compares the information from this image with the ones in the database in order to get potential matches.

The different features that are extracted are the length and width of the leaf, the area of the leaf, the perimeter of the leaf, the hull area, the hull perimeter, a distance map along the vertical and horizontal axes, a colour histogram and a centroid-based radial distance map. A k-Nearest Neighbour classifier was implemented and tested on 640 leaves belonging to 32 different species of plants. An accuracy of 83.5% was obtained. The system was further enhanced by using information obtained from a colour histogram which increased the recognition accuracy to 87.3%. Furthermore, our system is simple to use, fast and highly scalabl

ACKNOWLEDGEMENT

It gives us a great sense of pleasure to present the report of the B. Tech Mini Project undertaken during B. Tech. Third Year. This project in itself is an acknowledgement to the inspiration, drive and technical assistance contributed to it by many individuals. This project would never have seen the light of the day without the help and guidance that we have received.

Our heartiest thanks to **Dr. (Prof). Anand Singh Jalal,** Head of Dept., Department of CEA for providing us with an encouraging platform to develop this project, which thus helped us in shaping our abilities towards a constructive goal.

We owe special debt of gratitude to **Dr. Manoj Vashney ,** Technical Trainer, Department of CEA, for his constant support and guidance throughout the course of our work. His sincerity, thoroughness and perseverance have been a constant source of inspiration for us. He has showered us with all his extensively experienced ideas and insightful comments at virtually all stages of the project & has also taught us about the latest industry-oriented technologies.

We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind guidance and cooperation during the development of our project. Last but not the least, we acknowledge our friends for their contribution in the completion of the project

Student Information

Name: Mudit Singhania, Sazal goyal, Siddharth	University Roll. No.- 171500052,171500307,171500195
Mobile:8570074019	Email:sazal.goyal_cs17@gla.ac.in, siddharth.gla1_cs17@gla.ac.in mudit.singhanaia_cs17@gla.ac.in,

Information about Industry/Organization:

Organization	GLA university
Contact Person	Dr.Manoj Vashney 9754435581

Project Information:

Title Of Project	Leaf Detection
Role & Responsibility	
Technical Details Project Implementation Details	Hardware Requirements: Main Processor Core I3 Hard-disk Capacity 1 G.B RAM 2 GB Clock Speed 2.8 Hz Keyboard 104 Key Software Requirements: Operating System Windows 10 Language Python,Machine Learning Algorithms techniques Fully Implemented

-

CONTENT

CHAPTER 1

INTRODUCTION

Introduction: When leaving a town and entering the suburbs, we may encounter many kinds of trees. We may be able to identify those trees that often grow on urban streets, however most of the trees and plants found in city suburbs will be unknown to the majority of us. There are approximately 100,000 species of trees on earth, which account for about 25% of all plants. Many of the trees are in tropical regions, and because only limited botanical research has been carried out in these areas, it is believed that there are many undiscovered species [1]. It is clear that identifying large numbers of such trees is a complex proces

Motivation: An example of the complexity of tree identification can be seen with plums and apricots. These are very similar in leaf shape, the shape of the tree, and even in the shape of the young fruit. The flower shape is also very similar, and the tree type can only be identified by determining whether the calyx is attached, or inverted relative to the petal. Additionally, some trees are not easily distinguishable except at particular times; for example, when they bloom or bear fruit. To identify trees like these, considerable information is required, including leaf shape, shape of the leaves that are directly attached to branches, branch shape, shape of the whole tree, tree size, flower shape, flowering time, and fruit.

Objective: When using branches of biology such as cell biology, molecular biology, phytochemistry, or morphologic anatomy, it may be possible to distinguish plants without time constraints. However, it is unrealistic for the general public to identify the names of trees or plants using these methods when, for example, they are walking in a woodland.

1.1 Scope Of Project : It will give increase the cost of barber and decrease the work of Farmers to identify the leaf .

CHAPTER 2

Technology Used

2.1 Dataset

In this project we will apply K-Means on a small dataset of 1600 binary leaf images with different shapes and try to get a feel for the distribution of leaf images using different visualizations that clarify different aspects about how one can interpret K-Means results.

Examples:

2.2 Machine Learning

Machine learning is a growing technology which enables computers to learn automatically from past data. Machine learning uses various algorithms for building mathematical models and making predictions using historical data or information. Currently, it is being used for various tasks such as image recognition, speech recognition, email filtering, Facebook auto-tagging, recommender system, and many more.

This machine learning tutorial gives you an introduction to machine learning along with the wide range of machine learning techniques such as Supervised, Unsupervised, and Reinforcement learning. You will learn about regression and classification models, clustering methods, hidden Markov models, and various sequential models.

Machine learning is a buzzword for today's technology, and it is growing very rapidly day by day. We are using machine learning in our daily life even without knowing it such as Google Maps, Google assistant, Alexa, etc. Below are some most trending real-world applications of Machine Learning.

2.3 Python

Python is a widely used high-level programming language for general-purpose programming. Apart from being open source programming language, python is a great object-oriented, interpreted, and interactive programming language. Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing.

Python 3.0 was released in 2008. Although this version is supposed to be backward incompatibles, later on many of its important features have been backported to be compatible with version 2.7.This tutorial gives enough understanding on Python 3 version programming language. Please refer to this link for our Python 2 tutorial.

Python is Interactive − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

2.4 K-Means Clustering

k-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed apriori. The main idea is to define k centers, one for each cluster. These centers should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed and an early group age is done. At this point we need to re-calculate k new centroids as barycenter of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new center. A loop has been generated. As a result of this loop we may notice that the k centers change their location step by step until no more changes are done or in other words centers do not move any more.

Visualizing K-Means with Leaf Dataset

This script is about perhaps the simplest and most popular **unsupervised learning algorithm** out there: the K-Means clustering algorithm.

In this script we will apply K-Means on a small dataset of 1600 binary leaf images with different shapes and try to get a feel for the distribution of leaf images using different visualizations that clarify different aspects about how one can interpret K-Means results.

We will then continue to see if the K-Means features (distances from cluster centers) are informative in terms of classifying leafs and determine what is the optimal K (number of clusters) for the sake of leaf type classification.

CHAPTER 3

Softwares

3.1 Jupyter

Project Jupyter is a nonprofit organization created to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". Spun-off from IPython in 2014 by Fernando Pérez, Project Jupyter supports execution environments in several dozen languages. Project Jupyter's name is a reference to the three core programming languages supported by Jupyter, which are Julia, Python and R, and also a homage to Galileo's notebooks recording the discovery of the moons of Jupiter. Project Jupyter has developed and supported the interactive computing products Jupyter Notebook, JupyterHub, and JupyterLab, the next-generation version of Jupyter Notebook.

3.2 Pycharm

PyCharm is an integrated development environment used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains.

3.3 Import Library

```
import pandas as pd

import numpy as np

import matplotlib

import matplotlib.image as mpimg

import matplotlib.pyplot as plt

import matplotlib.gridspec as gridspec


from sklearn import model_selection

from sklearn import decomposition

from sklearn import linear_model

from sklearn import cluster
```

```
from sklearn import ensemble

from sklearn import neighbors

from sklearn.preprocessing import LabelEncoder

from sklearn.neighbors import KernelDensity

from sklearn.manifold import TSNE

from sklearn.metrics import accuracy_score

from skimage.transform import rescale

from scipy import ndimage as ndi

matplotlib.style.use('fivethirtyeight')
```

CHAPTER 4

Project Implementation

4.1 Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from sklearn import model_selection
from sklearn import decomposition
from sklearn import linear_model
from sklearn import cluster
from sklearn import ensemble
from sklearn import neighbors
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KernelDensity
from sklearn.manifold import TSNE
from sklearn.metrics import accuracy_score

from skimage.transform import rescale
from scipy import ndimage as ndi

matplotlib.style.use('fivethirtyeight')
```

4.2 Store Dataset

```
trainData = pd.read_csv('train.csv')

classEncoder = preprocessing.LabelEncoder()

trainLabels  = classEncoder.fit_transform(trainData.loc[:,'species'])

trainIDs     = np.array(trainData.loc[:,'id'])

# show some random images

plt.figure(figsize=(14,12))

dataDir='../demo/'

plt.suptitle('Original Images (with variable image sizes)', fontsize=22)

for k in range(28):

    randTrainInd = np.random.randint(80)

    randomID = trainIDs[randTrainInd]

    path=dataDir+'images/'+str(randomID)+'.jpg'

    imageFilename =path

    plt.subplot(4,7,k+1)

    plt.imshow(mpimg.imread(imageFilename), cmap='gray')

    plt.title(classEncoder.classes_[trainLabels[randTrainInd]], fontsize=10)

    plt.axis('off')
```

4.3 Store Data into 2-D Array And Train Data

```
numImages = 80

dataDir='../demo/'

shapesMatrix = np.zeros((2,numImages))

listOfImages = []

for k in range(numImages):

    imageFilename = dataDir + 'images/' + str(k+1) + '.jpg'

    currImage = mpimg.imread(imageFilename)

    shapesMatrix[:,k] = np.shape(currImage)

    listOfImages.append(currImage)


# calculate the shape of an image that will contain all original images within it

maxShapeSize = shapesMatrix.max(axis=1)

for k in range(len(maxShapeSize)):

    if maxShapeSize[k] % 2 == 0:

        maxShapeSize[k] += 311

    else:

        maxShapeSize[k] += 310



# place all original images at the center of the large reference frame

fullImageMatrix3D = np.zeros(np.hstack((maxShapeSize,
np.shape(shapesMatrix[1]))).astype(int),dtype=np.dtype('u1'))

destXc = (maxShapeSize[1]+1)/2; destYc = (maxShapeSize[0]+1)/2

for k, currImage in enumerate(listOfImages):

    Yc, Xc = ndi.center_of_mass(currImage)

    Xd = destXc - Xc; Yd = destYc - Yc

    rowIndLims = (int(round(Yd)),int(round(Yd)+np.shape(currImage)[0]))
```

```
    colIndLims = (int(round(Xd)),int(round(Xd)+np.shape(currImage)[1]))

    fullImageMatrix3D[rowIndLims[0]:rowIndLims[1],colIndLims[0]:colIndLims[1],k] = currImage


'''

# make sure nothing was ruined in the process

plt.figure(figsize=(14,7))

plt.suptitle('Processed Images (fixed size)', fontsize=22)

for k in range(28):

    randInd = np.random.randint(np.shape(fullImageMatrix3D)[2])

    plt.subplot(4,7,k+1); plt.imshow(fullImageMatrix3D[:,:,randInd], cmap='gray'); plt.axis('off')

'''


# re crop according to rows and columns that don't have zeros in them in any image

xValid = fullImageMatrix3D.mean(axis=2).sum(axis=0) > 0

yValid = fullImageMatrix3D.mean(axis=2).sum(axis=1) > 0

xLims = (np.nonzero(xValid)[0][0],np.nonzero(xValid)[0][-1])

yLims = (np.nonzero(yValid)[0][0],np.nonzero(yValid)[0][-1])

fullImageMatrix3D = fullImageMatrix3D[yLims[0]:yLims[1],xLims[0]:xLims[1],:]


# make sure nothing was ruined in the process

plt.figure(figsize=(14,7))

plt.suptitle('Final Processed Images (with fixed image size)', fontsize=22)

for k in range(28):

    randInd = np.random.randint(np.shape(fullImageMatrix3D)[2])

    plt.subplot(4,7,k+1); plt.imshow(fullImageMatrix3D[:,:,randInd], cmap='gray'); plt.axis('off')

    if randInd < len(trainLabels):

        plt.title(classEncoder.classes_[trainLabels[randInd]], fontsize=10)

    else:
```

```
    plt.title('test data sample', fontsize=10)


# scale down all images to be in normal size
rescaleFactor = 0.15
scaledDownImage = rescale(fullImageMatrix3D[:,:,0],rescaleFactor)
scaledDownImages = np.zeros(np.hstack((np.shape(scaledDownImage),
                              np.shape(fullImageMatrix3D)[2])),dtype=np.dtype('f4'))
for imInd in range(np.shape(fullImageMatrix3D)[2]):
  scaledDownImages[:,:,imInd] = rescale(fullImageMatrix3D[:,:,imInd],rescaleFactor
del fullImageMatrix3D
```

4.3 K-Means Clustering

```
class KmeansModel:
  def _init_(self, X, numClusters=10, objectPixels=None):
    '''
    inputs:
      X                    - numSamples x numDimentions matrix
      numClusters          - number of clusters to use
      objectPixels (optional) - an binnary mask image used for presentation
                           will be used as Im[objectPixels] = dataSample
                           must satisfy objectPixels.ravel().sum() = X.shape[1]
    '''
    numDataSamples = X.shape[0]
    self.numClusters = numClusters
    if objectPixels is None:
      self.objectPixels = np.ones((1,X.shape[1]),dtype=np.bool)
    else:
      self.objectPixels = objectPixels
    assert(self.objectPixels.ravel().sum() == X.shape[1])
```

```
        KmeansModel = cluster.KMeans(n_clusters=numClusters, n_init=5)
        self.dataRepresentation = KmeansModel.fit_transform(X)
        self.KmeansModel = KmeansModel

        # calculate cluster frequency
        clusterInds = KmeansModel.labels_
        clusterFrequency = []
        for clusterInd in range(numClusters):
            clusterFrequency.append((clusterInds == clusterInd).sum()/float(numDataSamples))
        self.clusterFrequency = np.array(clusterFrequency)
        self.sortedTemplatesByFrequency = np.flipud(np.argsort(clusterFrequency))

    def RepresentUsingModel(self, X, representationMethod='distFromAllClusters'):

        if representationMethod == 'distFromAllClusters':
            return self.KmeansModel.transform(X)
        if representationMethod == 'clusterIndex':
            return self.KmeansModel.predict(X)
        if representationMethod == 'oneHotClusterIndex':
            clusterAssignment = self.KmeansModel.predict(X)
            X_transformed = np.zeros((X.shape[0],self.numClusters))
            for sample in range(X.shape[0]):
                X_transformed[sample,clusterAssignment[sample]] = 1
            return X_transformed

    def ReconstructUsingModel(self, X_transformed, representationMethod='distFromAllClusters'):
```

```
        if representationMethod == 'clusterIndex':
            clusterAssignment = X_transformed
        if representationMethod == 'oneHotClusterIndex':
            clusterAssignment = np.argmax(X_transformed,axis=1)
        if representationMethod == 'distFromAllClusters':
            clusterAssignment = np.argmin(X_transformed,axis=1)

        X_reconstructed = np.zeros((X_transformed.shape[0],self.KmeansModel.cluster_centers_.shape[1]))
        for sample in range(X_transformed.shape[0]):
            X_reconstructed[sample,:] = self.KmeansModel.cluster_centers_[clusterAssignment[sample],:]

        return X_reconstructed

    def InterpretUsingModel(self, X, representationMethod='clusterIndex'):
        return self.ReconstructUsingModel(\
                        self.RepresentUsingModel(X,representationMethod),representationMethod)

    # shows the cluster centers
    def ShowTemplates(self, numTemplatesToShow=16):
        numTemplatesToShow = min(numTemplatesToShow, self.numClusters)

        numFigRows = np.ceil(np.sqrt(numTemplatesToShow));
        numFigCols = np.ceil(np.sqrt(numTemplatesToShow));
        numTemplatesPerFigure = int(numFigRows*numFigCols)
        numFigures = int(np.ceil(float(numTemplatesToShow)/numTemplatesPerFigure))

        for figureInd in range(numFigures):
            plt.figure()
```

```
        for plotInd in range(numTemplatesPerFigure):
            templateInd = self.sortedTemplatesByFrequency[numTemplatesPerFigure*figureInd + plotInd]
            if templateInd >= self.numClusters:
                break
            templateImage = np.zeros(np.shape(self.objectPixels))
            templateImage[self.objectPixels] = \
                    self.KmeansModel.cluster_centers_[templateInd,:].ravel()

            plt.subplot(numFigRows,numFigCols,plotInd+1)
            if np.shape(self.objectPixels)[0] == 1:
                plt.plot(templateImage)
            else:
                plt.imshow(templateImage,cmap='hot'); plt.axis('off')
            plt.title(str(100*self.clusterFrequency[templateInd])[:4] + '% frequency');
        plt.tight_layout()

    # shows several random model reconstructions
    def ShowReconstructions(self, X, numReconstructions=6):
        assert(np.shape(X)[1] == self.objectPixels.ravel().sum())
        numSamples = np.shape(X)[0]
        numReconstructions = min(numReconstructions, numSamples)

        originalImage      = np.zeros(np.shape(self.objectPixels))
        reconstructedImage = np.zeros(np.shape(self.objectPixels))

        numReconstructionsPerFigure = min(6, numReconstructions)
        numFigures = int(np.ceil(float(numReconstructions)/numReconstructionsPerFigure))
```

```
    for figureInd in range(numFigures):
        plt.figure()
        for plotCol in range(numReconstructionsPerFigure):
            dataSampleInd = np.random.randint(numSamples)
            originalImage[self.objectPixels] = X[dataSampleInd,:].ravel()
            reconstructedImage[self.objectPixels] = \
                    self.InterpretUsingModel(np.reshape(X[dataSampleInd,:],[1,-1])).ravel()
            diffImage = abs(originalImage - reconstructedImage)

            # original image
            plt.subplot(3,numReconstructionsPerFigure,0*numReconstructionsPerFigure+plotCol+1)
            if np.shape(self.objectPixels)[0] == 1:
                plt.plot(originalImage); plt.title('original signal')
            else:
                plt.imshow(originalImage, cmap='gray');
                plt.title('original image'); plt.axis('off')

            # reconstred image
            plt.subplot(3,numReconstructionsPerFigure,1*numReconstructionsPerFigure+plotCol+1)
            if np.shape(self.objectPixels)[0] == 1:
                plt.plot(reconstructedImage); plt.title('reconstructed signal')
            else:
                plt.imshow(reconstructedImage, cmap='gray');
                plt.title('reconstructed image'); plt.axis('off')

            # diff image
            plt.subplot(3,numReconstructionsPerFigure,2*numReconstructionsPerFigure+plotCol+1)
            if np.shape(self.objectPixels)[0] == 1:
```

```
            plt.plot(diffImage); plt.title('abs difference signal')
        else:
            plt.imshow(diffImage, cmap='gray');
            plt.title('abs difference image'); plt.axis('off')
    plt.tight_layout()



# shows distrbution along the distance from a particular cluster and several examples for that distance
def ShowSingleTemplateDistances(self, X, listOfTemplates=[0,1]):

    showAsTraces = (np.shape(self.objectPixels)[0] == 1)
    assert(all([(x in range(self.numClusters)) for x in listOfTemplates]))

    X_rep = self.RepresentUsingModel(X, representationMethod='distFromAllClusters')

    percentilesToShow = [1,5,10,30,60,99]
    numReadDataSamplePerPercentile = 4
    representationPercentiles = []
    for percentile in percentilesToShow:
        representationPercentiles.append(np.percentile(self.dataRepresentation, percentile, axis=0))
    medianRepVec =  np.percentile(self.dataRepresentation, 50, axis=0)

    for templateInd in listOfTemplates:
        plt.figure(); gs = gridspec.GridSpec(numReadDataSamplePerPercentile+2,
                                             len(percentilesToShow))

        # calculate the Gaussian smoothed distribution of values along the eignevector direction
        sigmaOfKDE = (representationPercentiles[-1][templateInd] -
```

```
            representationPercentiles[1][templateInd])/100.0

      pdfStart   = representationPercentiles[1][templateInd]  - 15*sigmaOfKDE
      pdfStop    = representationPercentiles[-1][templateInd] + 15*sigmaOfKDE
      xAxis = np.linspace(pdfStart,pdfStop,200)
      PDF_Model = KernelDensity(kernel='gaussian', \
                  bandwidth=sigmaOfKDE).fit(self.dataRepresentation[:,templateInd].reshape(-1,1))
      logPDF = PDF_Model.score_samples(xAxis.reshape(-1,1))
      percentileValuesToShow = \
        [representationPercentiles[x][templateInd] for x in range(len(representationPercentiles))]
      percentilesToShowLogPDF = \
        PDF_Model.score_samples(np.array(percentileValuesToShow).reshape(-1,1))

      # show distribution of distance from current template and red dots at the list of precentiles to show
      plt.subplot(gs[0,:])
      plt.fill(xAxis, np.exp(logPDF), fc='b', alpha=0.9);
      plt.scatter(percentileValuesToShow, np.exp(percentilesToShowLogPDF), c='r',s=40);
      plt.title(str(100*self.clusterFrequency[templateInd])[:4] + '% assignment frequency');

      for plotCol, currPrecentile in enumerate(percentilesToShow):
        currPrecentileRepVec              = medianRepVec.copy()
        currPrecentileRepVec[templateInd] = representationPercentiles[plotCol][templateInd]

        currPrecentileImage = np.zeros(np.shape(self.objectPixels))
        currPrecentileRepVec = currPrecentileRepVec[:,np.newaxis].T
        currPrecentileImage[self.objectPixels] = \
                  self.ReconstructUsingModel(currPrecentileRepVec).ravel()

        # show the median image with current precentile as activation of the curr image
```

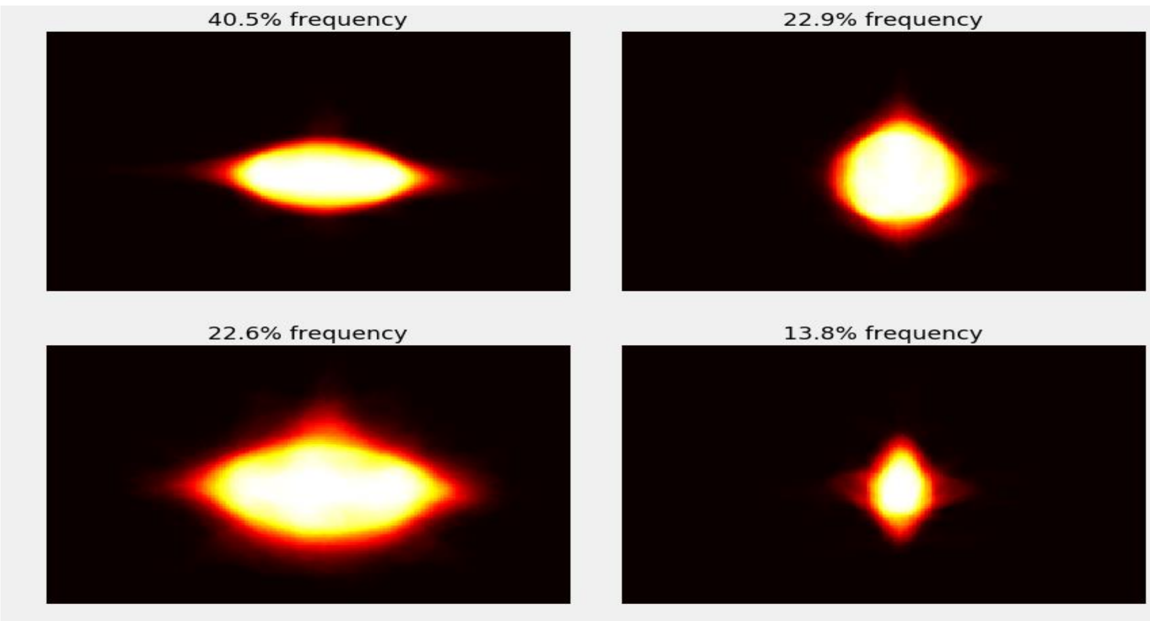
```
            plt.subplot(gs[1,plotCol]);
            if showAsTraces:
                plt.plot(currPrecentileImage);
                plt.title('precentile: ' + str(percentilesToShow[plotCol]) + '%')
            else:
                plt.imshow(currPrecentileImage, cmap='hot');
                plt.title('precentile: ' + str(percentilesToShow[plotCol]) + '%'); plt.axis('off')

            # find the most suitible candidates in X for current precentile
            distFromPercentile = abs(X_rep[:,templateInd] -
                                     representationPercentiles[plotCol][templateInd])
            X_inds = np.argpartition(distFromPercentile, \
                                     numReadDataSamplePerPercentile)[:numReadDataSamplePerPercentile]
            for k, X_ind in enumerate(X_inds):
                currNearestPrecentileImage = np.zeros(np.shape(self.objectPixels))
                currNearestPrecentileImage[self.objectPixels]  = X[X_ind,:].ravel()

                plt.subplot(gs[2+k,plotCol]);
                if showAsTraces:
                    plt.plot(currNearestPrecentileImage);
                    plt.title('NN with closest percentile');
                else:
                    plt.imshow(currNearestPrecentileImage, cmap='gray');
                    plt.title('NN with closest percentile'); plt.axis('off')
        plt.tight_layout()


    def ShowDataScatterPlotsWithTSNE(self, X=None, y=None, tSNE_perplexity=30.0, colorMap='Paired'):
```

```
        # show the distance from 2 most frequent clusters and the tSNE of the entire "distance form template" space


        if X is None:
            X_rep = self.dataRepresentation
        else:
            X_rep = self.RepresentUsingModel(X)

        if y is None:
            y = np.ones(X_rep.shape[0])

        tSNE_KmeansModel = TSNE(n_components=2, perplexity=tSNE_perplexity, random_state=0)
        X_rep_tSNE = tSNE_KmeansModel.fit_transform(X_rep)

        # take the two most frequent patterns
        mostFrequent = self.sortedTemplatesByFrequency[:2]

        plt.figure()
        plt.subplot(1,2,1);
        plt.scatter(X_rep[:,mostFrequent[0]], \
                    X_rep[:,mostFrequent[1]],c=y,cmap=colorMap,s=10,alpha=0.9)
        plt.title('"distance form template" representation');
        plt.xlabel('distance from template 1'); plt.ylabel('distance from template 2')
        plt.subplot(1,2,2);
        plt.scatter(X_rep_tSNE[:,0],X_rep_tSNE[:,1],c=y,cmap=colorMap,s=15,alpha=0.9)
        plt.title('t-SNE of Kmeans representation'); plt.xlabel('t-SNE axis1'); plt.ylabel('t-SNE axis2')


    def ShowTemplatesInPCASpace(self, X, y=None, tSNE_perplexity=30.0, colorMap='Paired'):
```

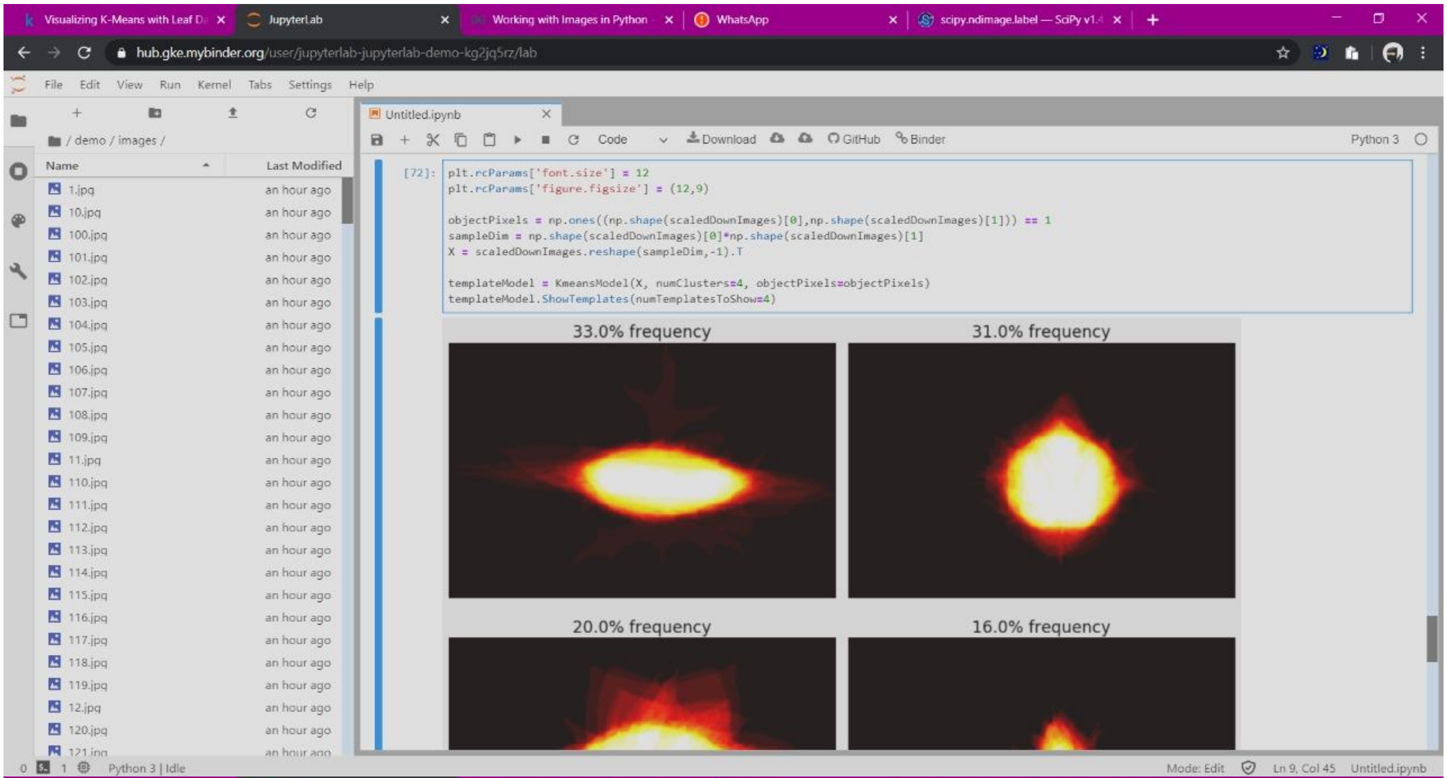
```
# show the templates in the 2PC space and the tSNE of the entire PCA space

# build PCA model and project the data onto the PCA space
PCAModel = decomposition.PCA(n_components=60, whiten=False)
X_rep = PCAModel.fit_transform(X)

# project the Kmeans templates onto the PCA space
templates_rep = PCAModel.transform(templateModel.KmeansModel.cluster_centers_)

if y is None:
    y = self.RepresentUsingModel(X, representationMethod='clusterIndex')

tSNE_PCAModel = TSNE(n_components=2, perplexity=tSNE_perplexity, random_state=0)
X_rep_tSNE = tSNE_PCAModel.fit_transform(np.vstack((X_rep,templates_rep)))

plt.figure()
plt.subplot(1,2,1); plt.scatter(X_rep[:,0],X_rep[:,1],c=y,cmap=colorMap,s=15,alpha=0.9)
plt.scatter(templates_rep[:,0],templates_rep[:,1],c='k',cmap=colorMap,s=50)
plt.title('PCA representation'); plt.xlabel('PC1 coeff'); plt.ylabel('PC2 coeff')

nC = templates_rep.shape[0]
plt.subplot(1,2,2);
plt.scatter(X_rep_tSNE[:-nC,0],\
            X_rep_tSNE[:-nC,1],c=y,cmap=colorMap,s=15,alpha=0.9)
plt.scatter(X_rep_tSNE[-nC:,0],\
            X_rep_tSNE[-nC:,1],c='k',cmap=colorMap,s=50)
plt.title('t-SNE of PCA representation'); plt.xlabel('t-SNE axis1'); plt.ylabel('t-SNE axis2')
```

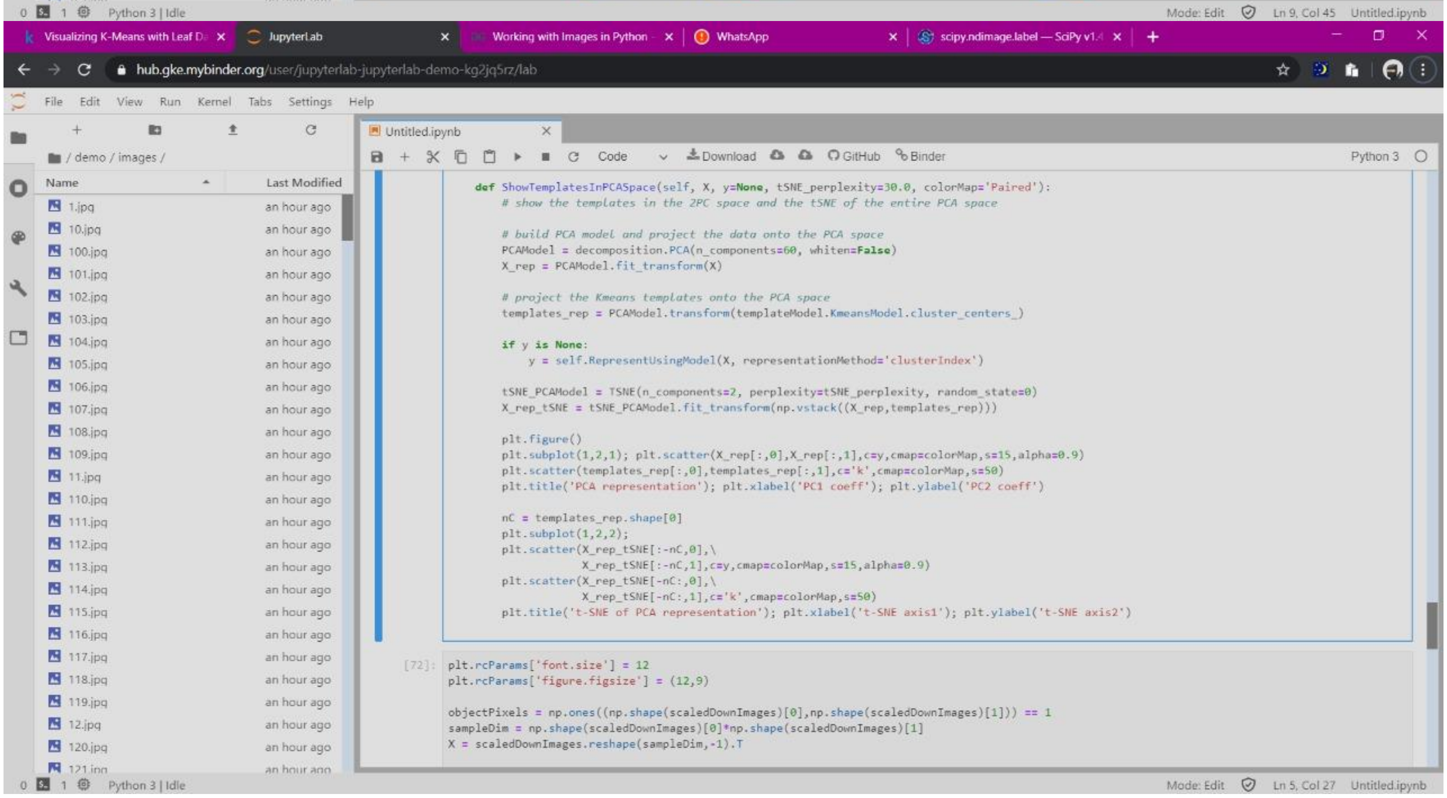
4 Effeciency

```
matplotlib.rcParams['font.size'] = 12

matplotlib.rcParams['figure.figsize'] = (12,9)

objectPixels = np.ones((np.shape(scaledDownImages)[0],np.shape(scaledDownImages)[1])) == 1

sampleDim = np.shape(scaledDownImages)[0]*np.shape(scaledDownImages)[1]

X = scaledDownImages.reshape(sampleDim,-1).T

templateModel = KmeansModel(X, numClusters=4, objectPixels=objectPixels)

templateModel.ShowTemplates(numTemplatesToShow=4)
```

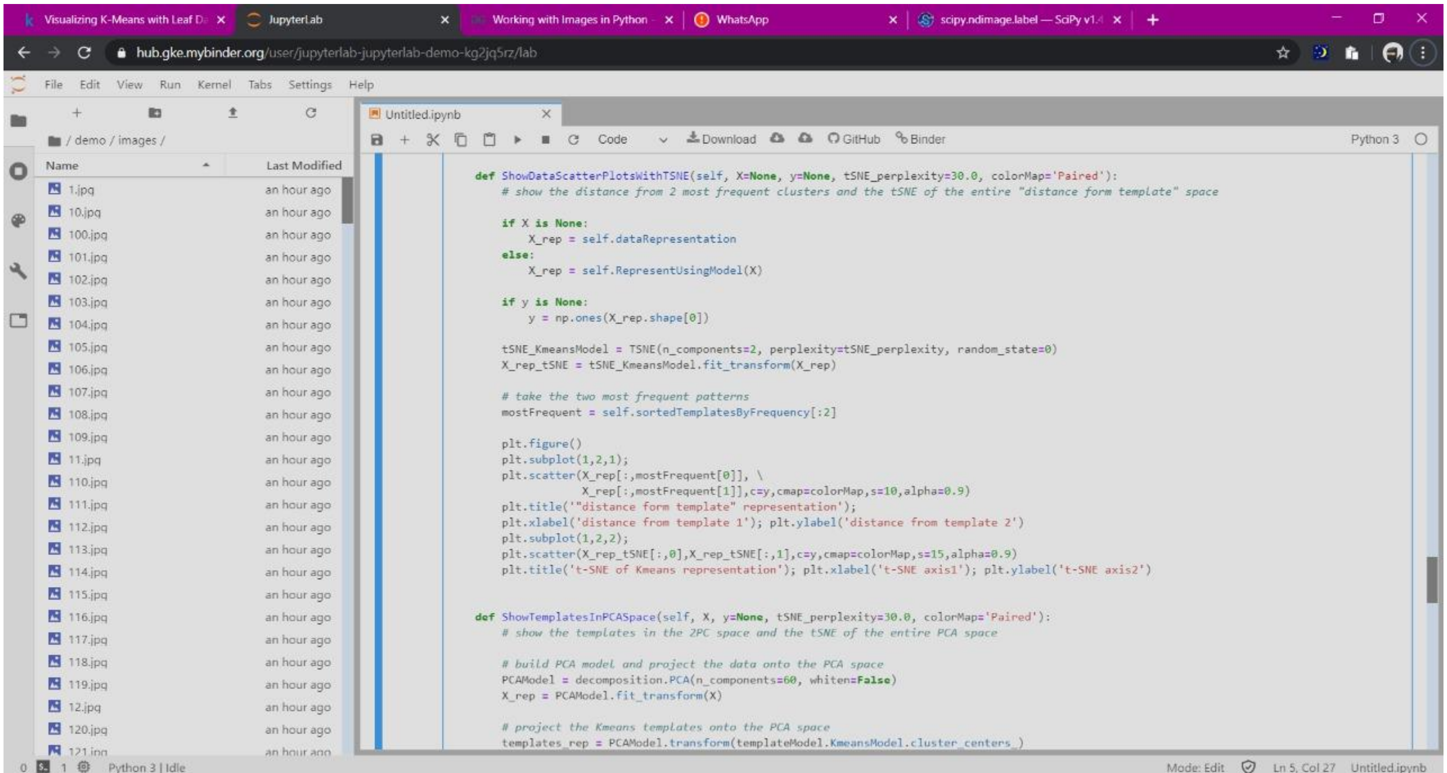


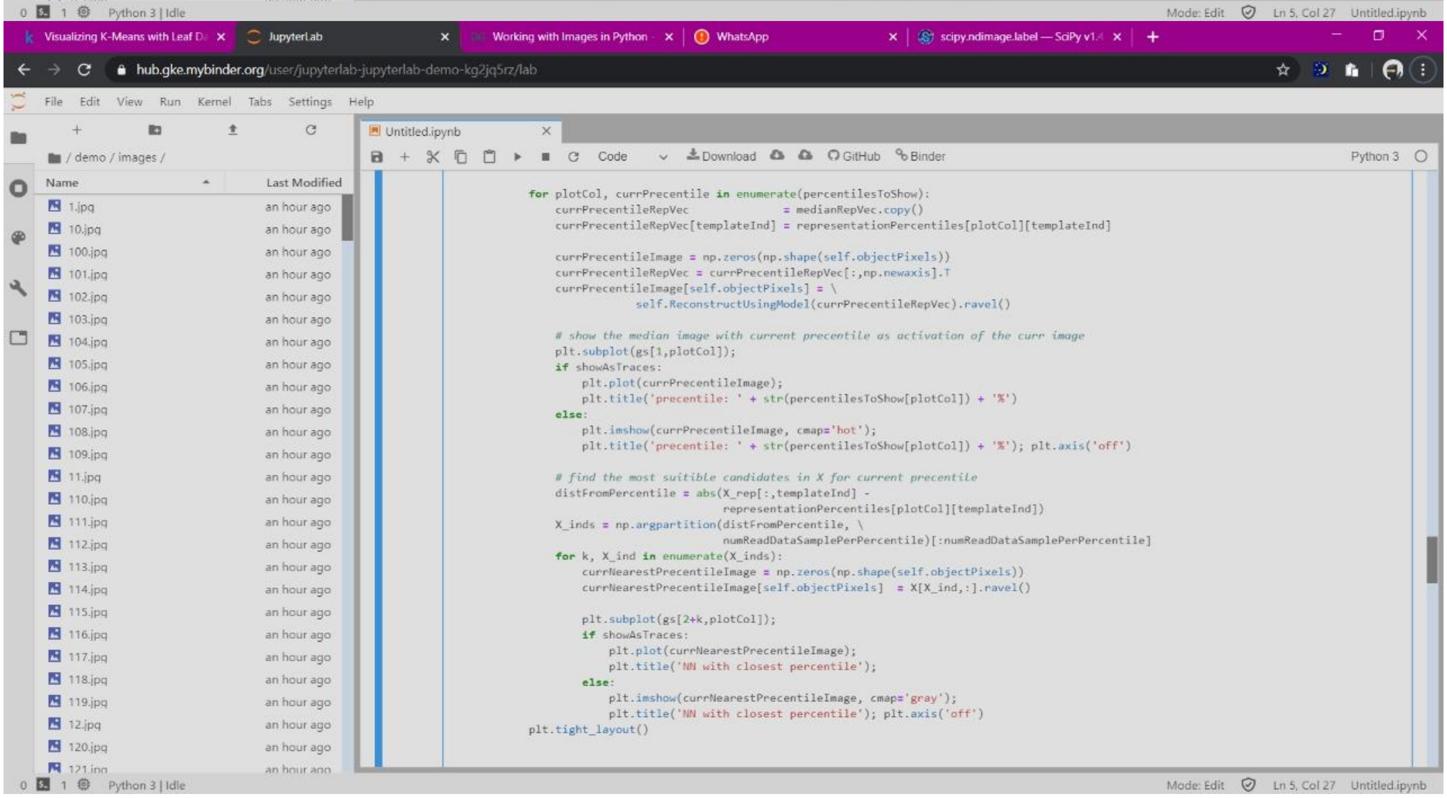
Screen Shot And Layouts



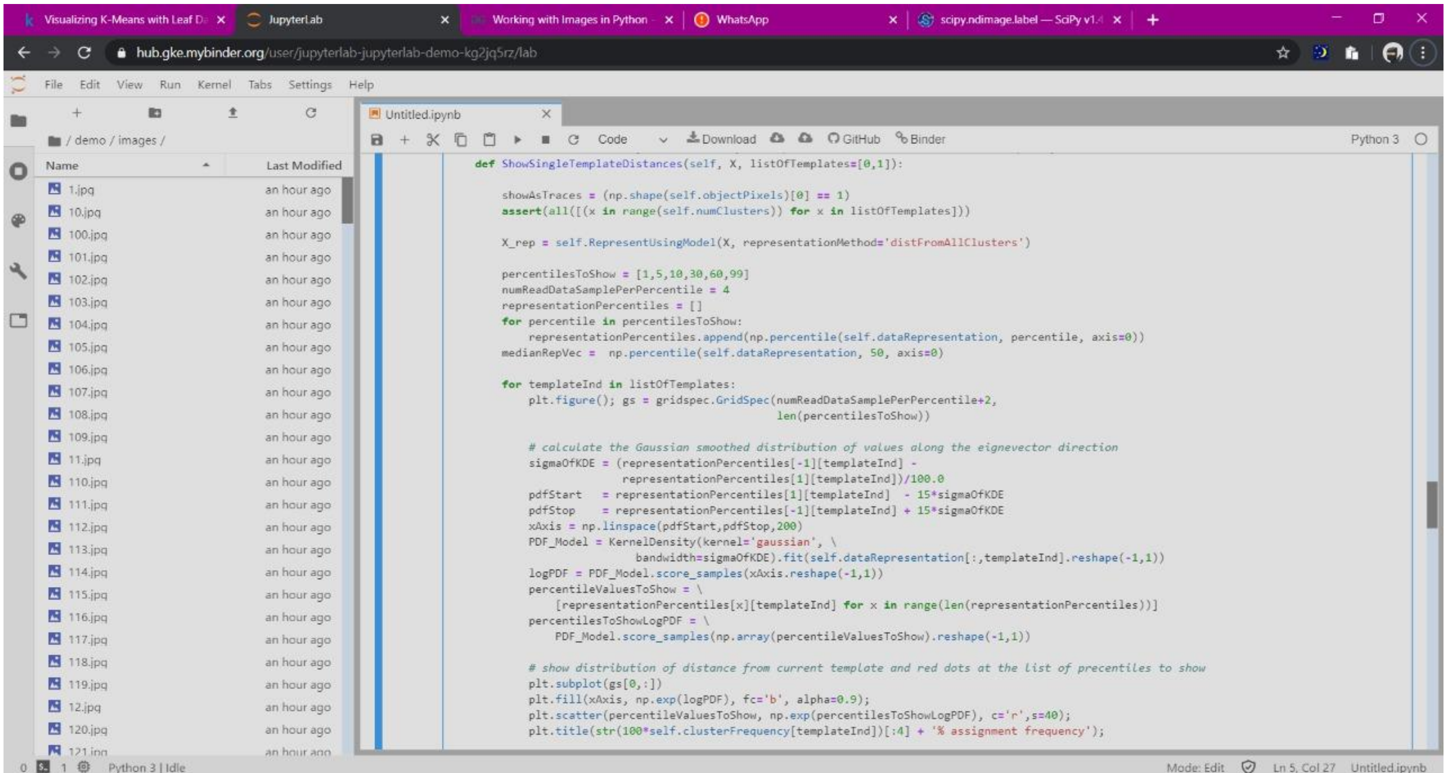


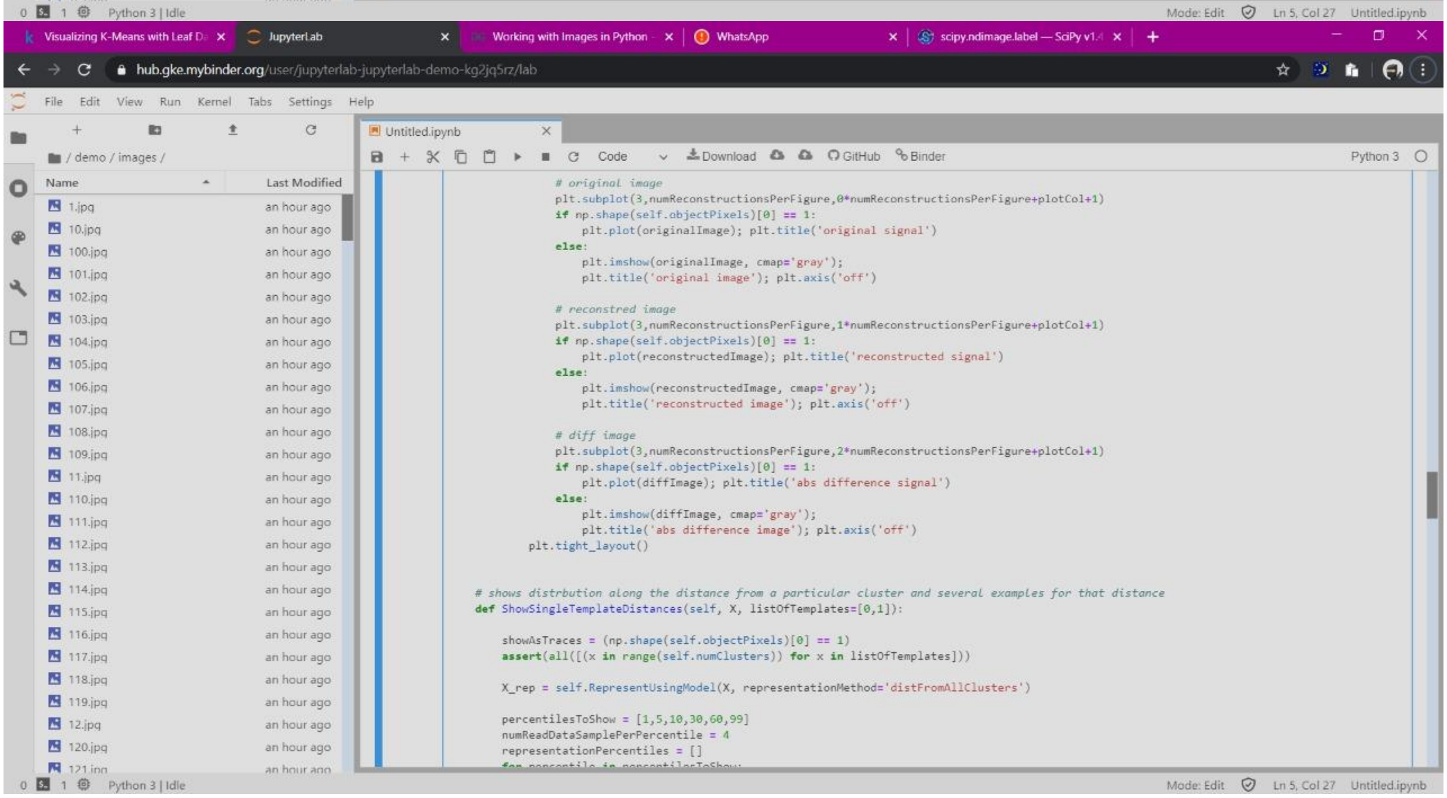
Leaf Detection



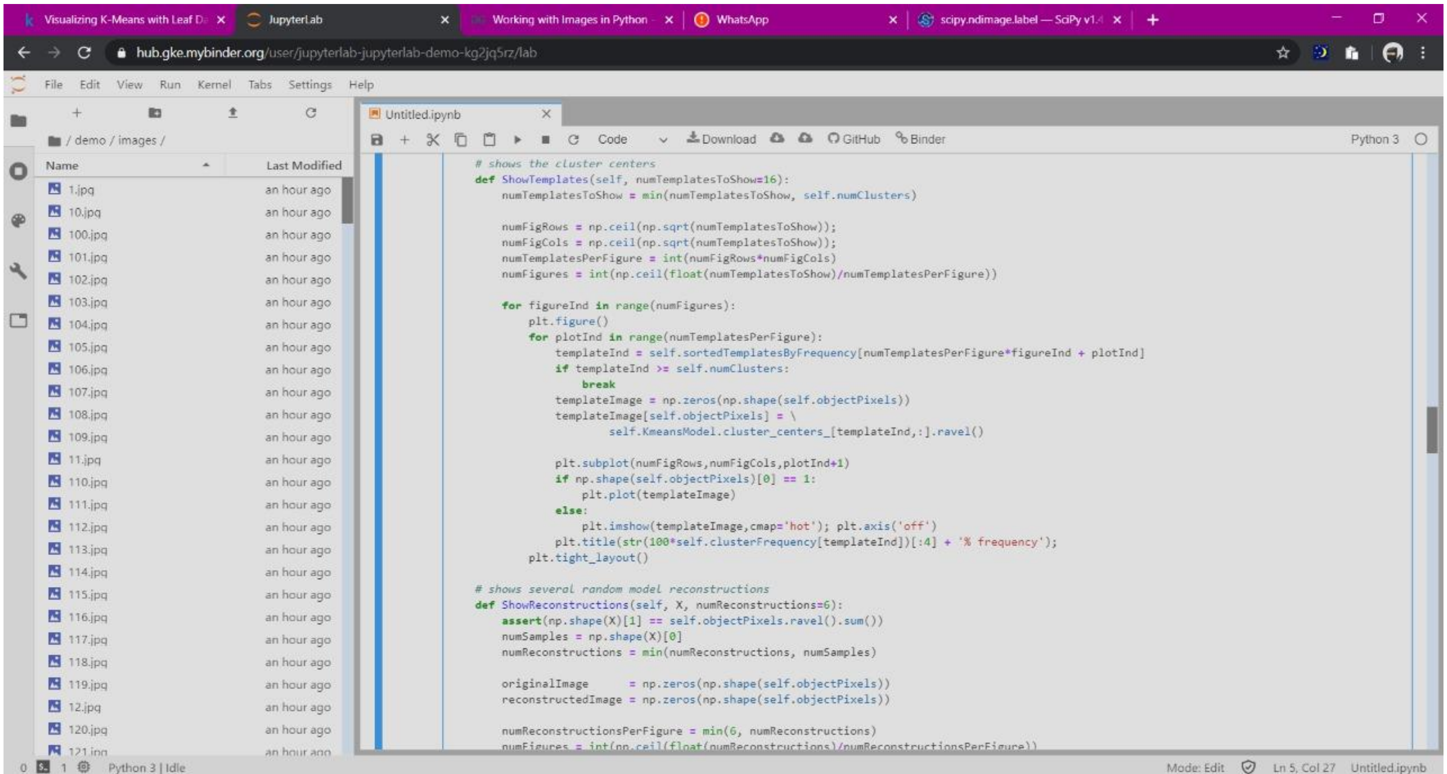


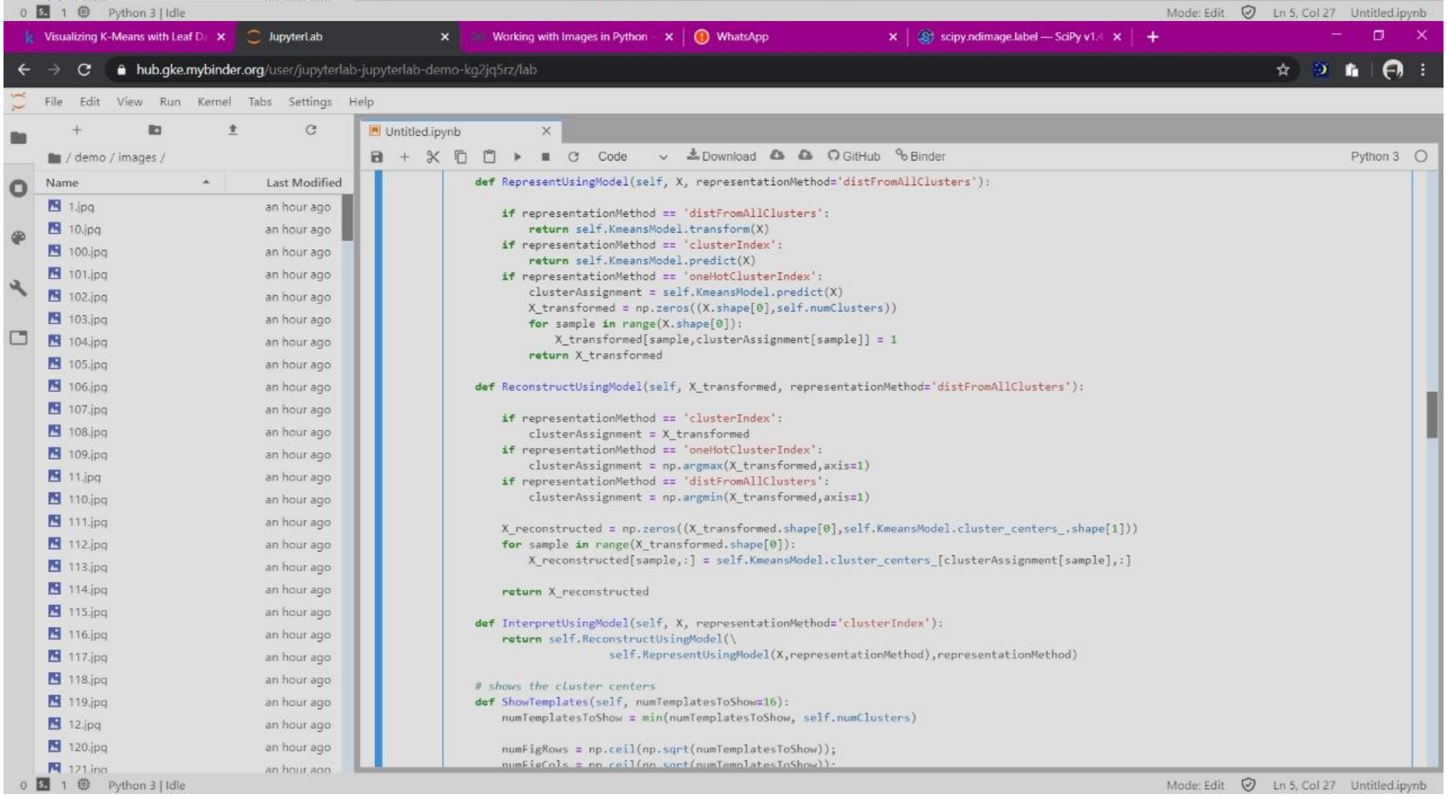
Leaf Detection



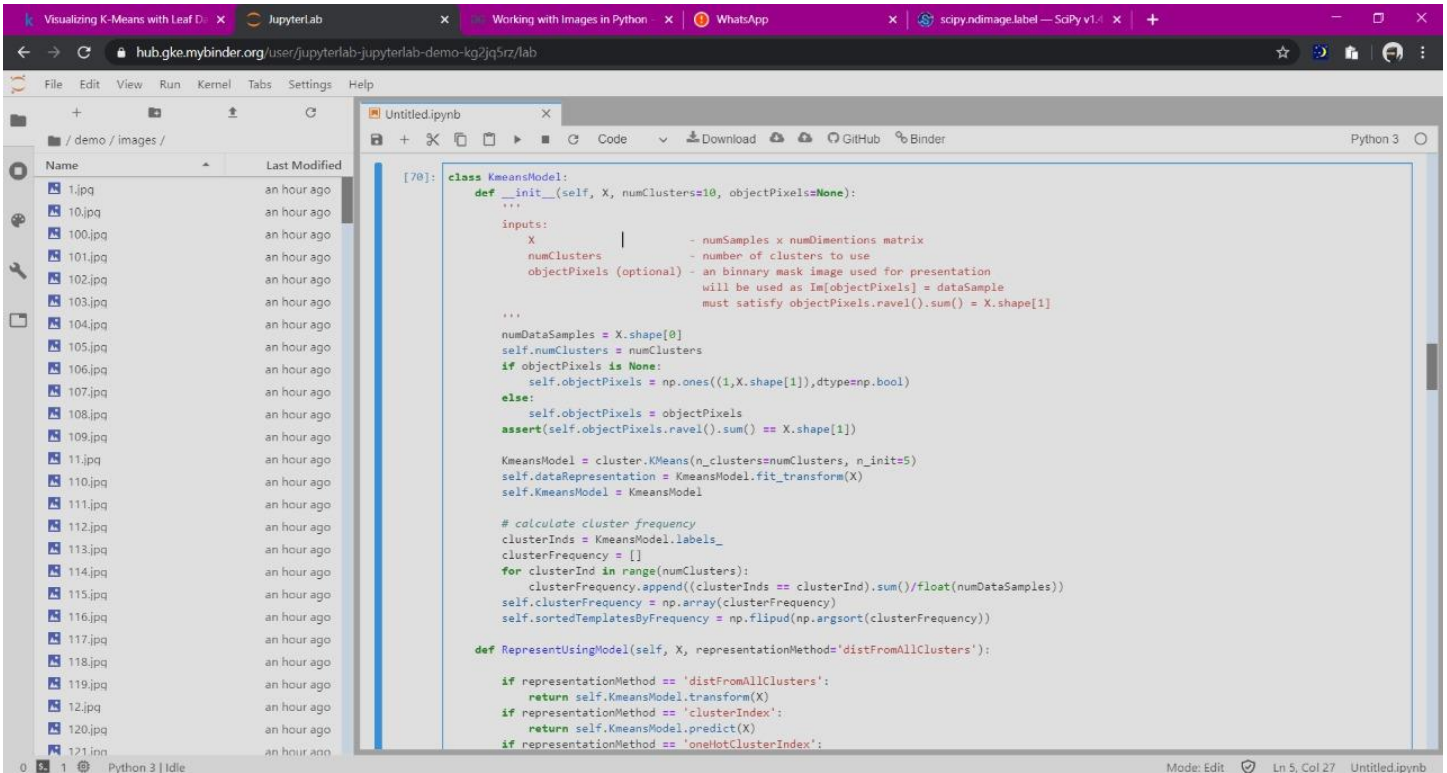


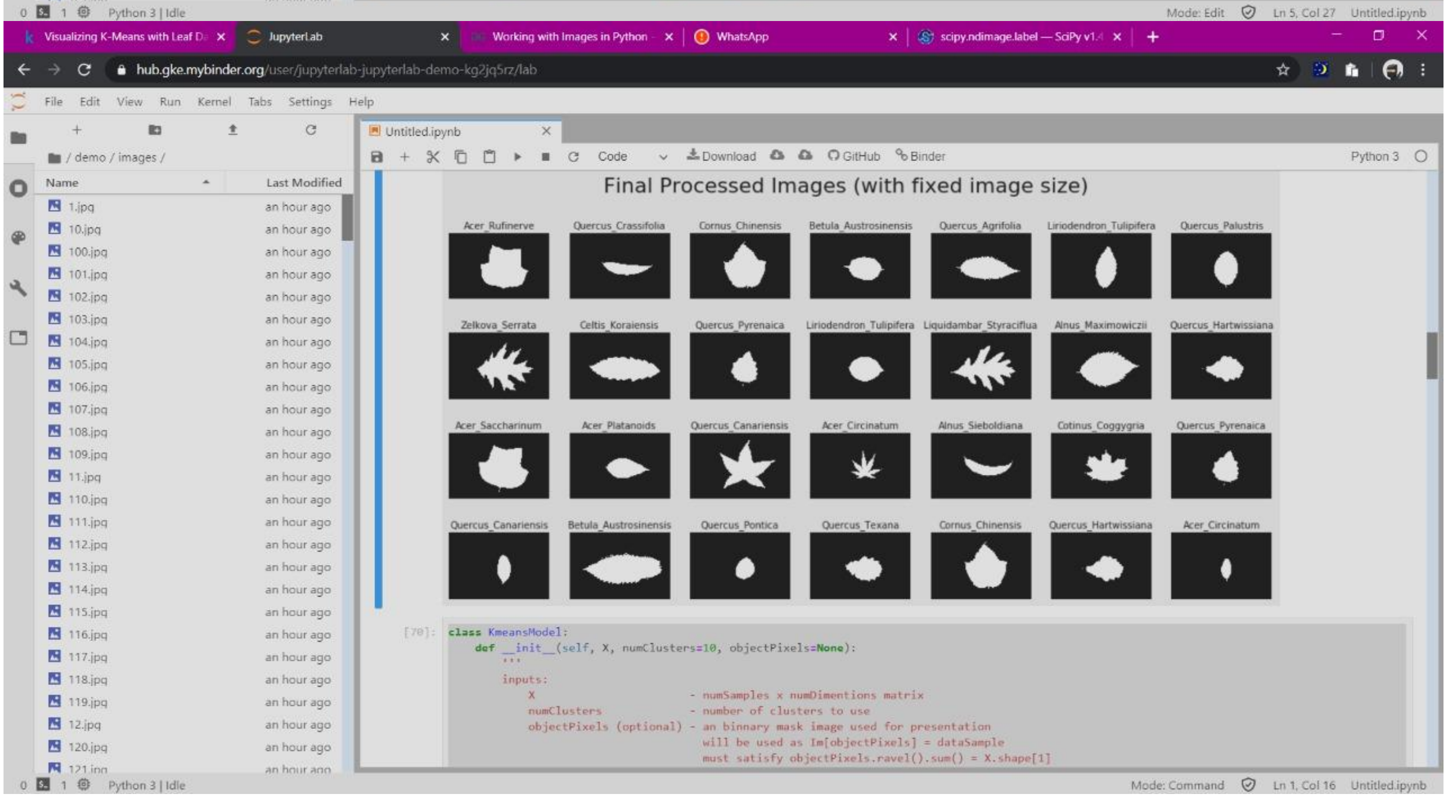
Leaf Detection



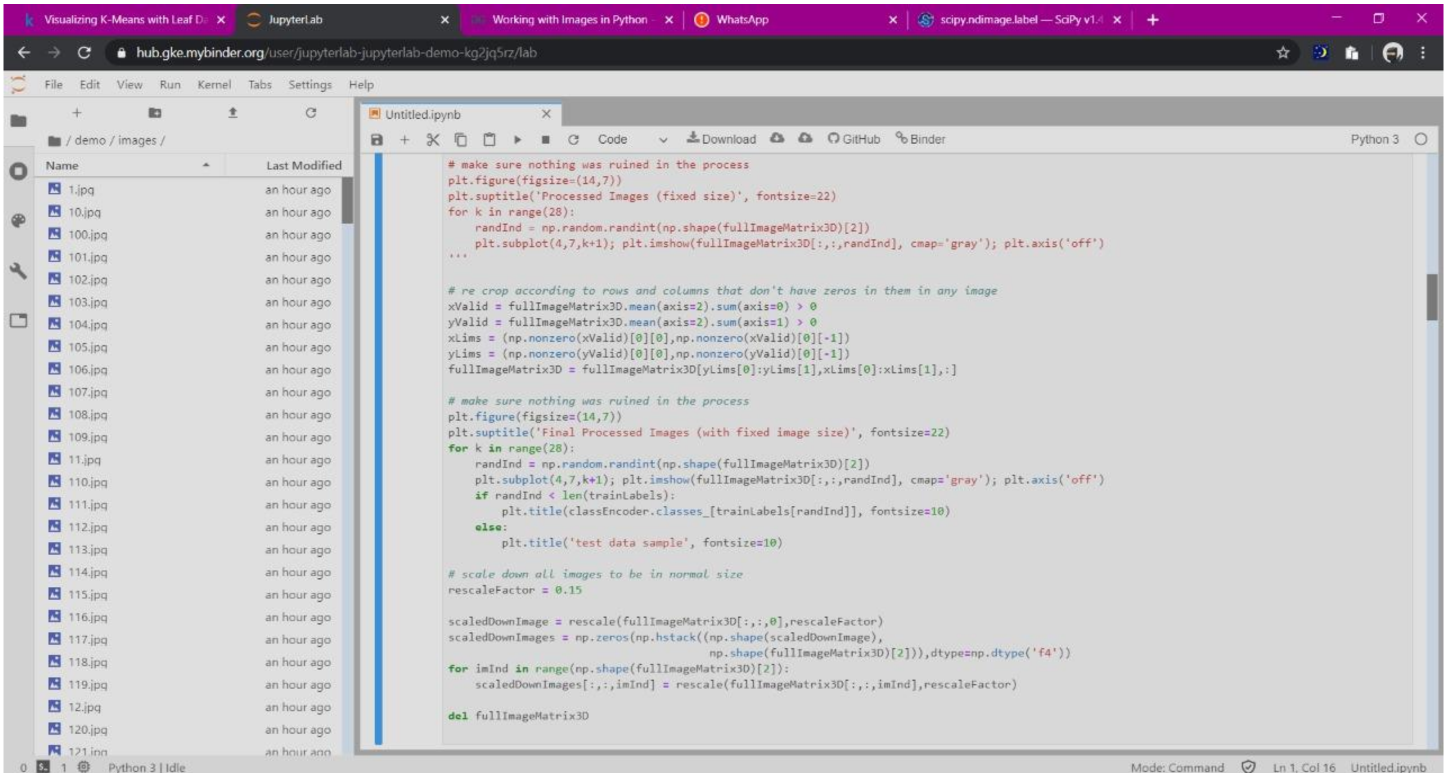


Leaf Detection

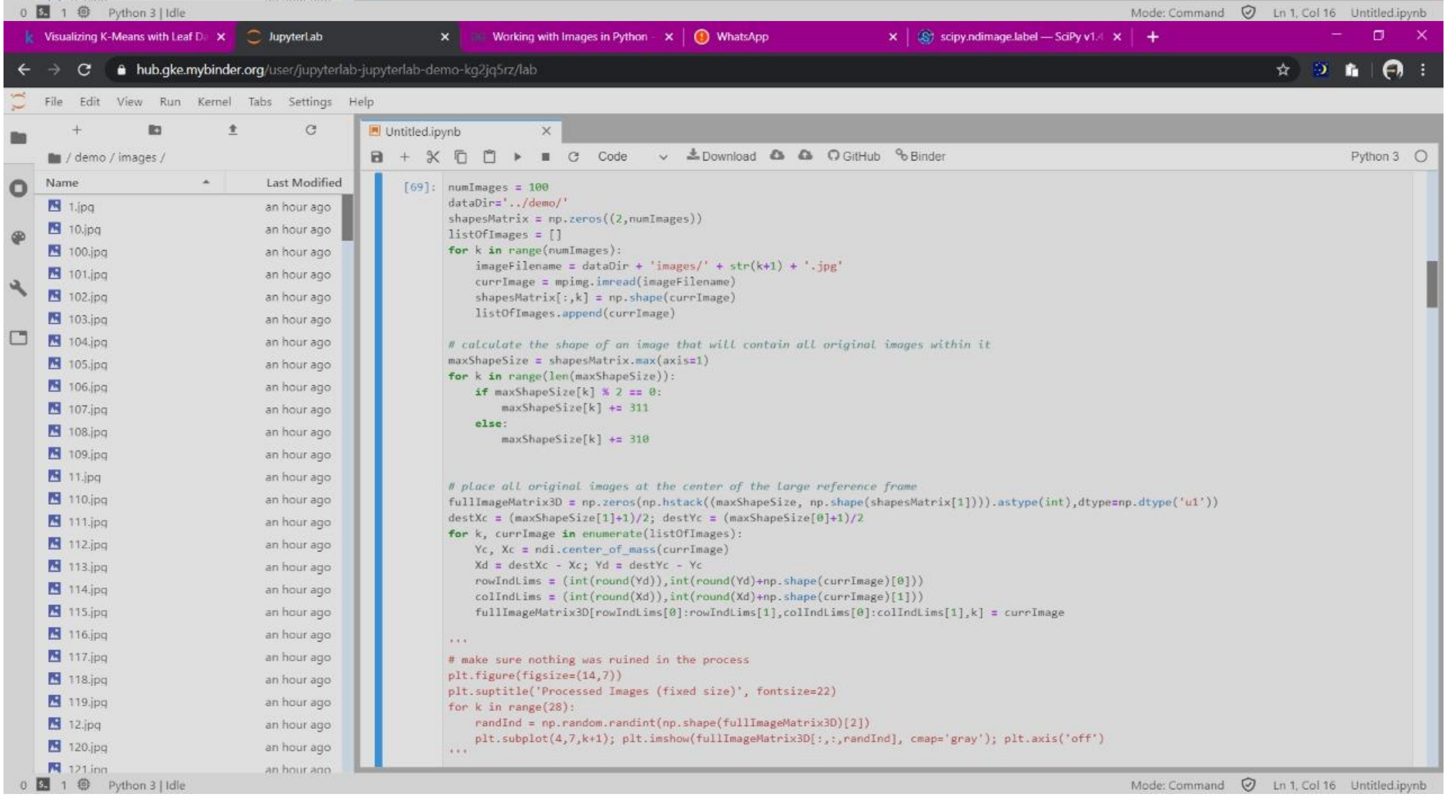




Leaf Detection

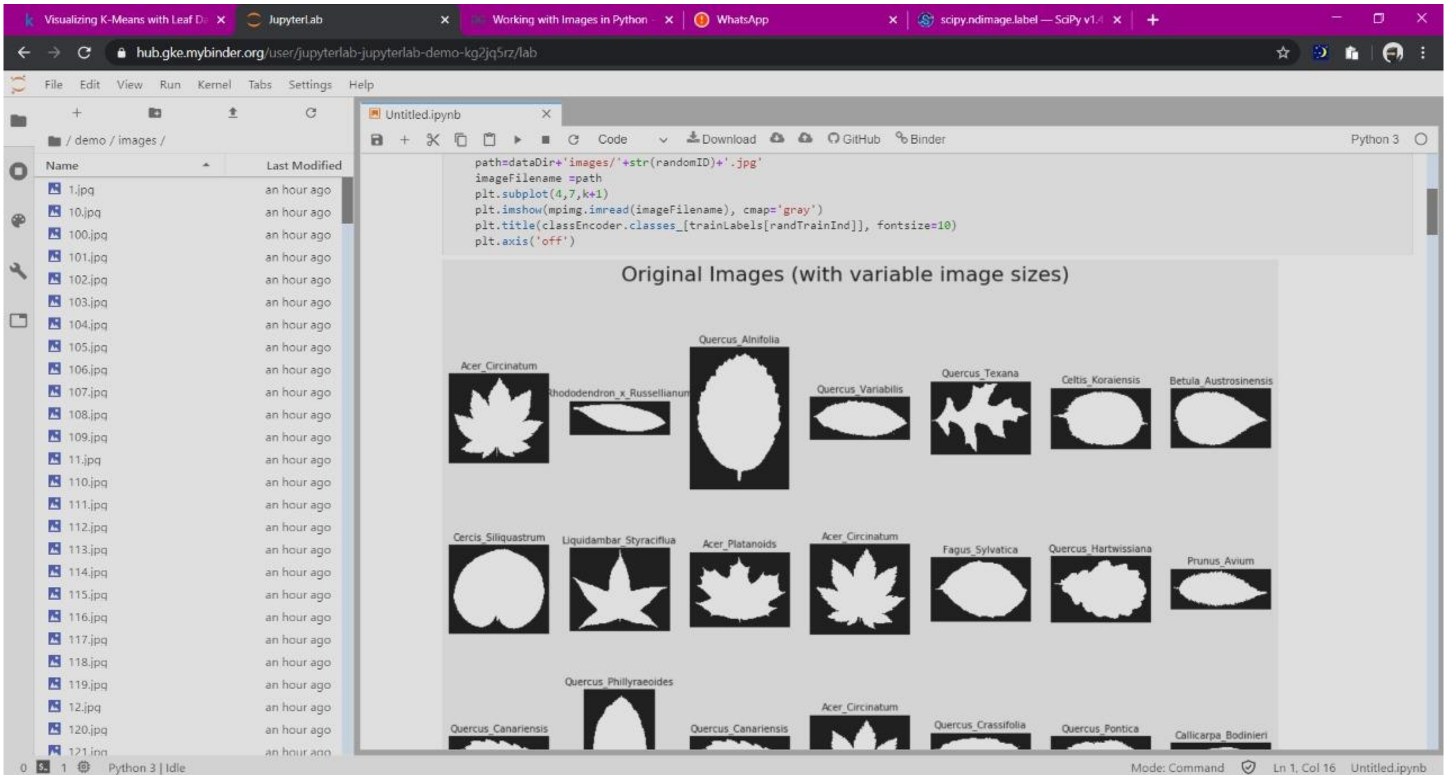


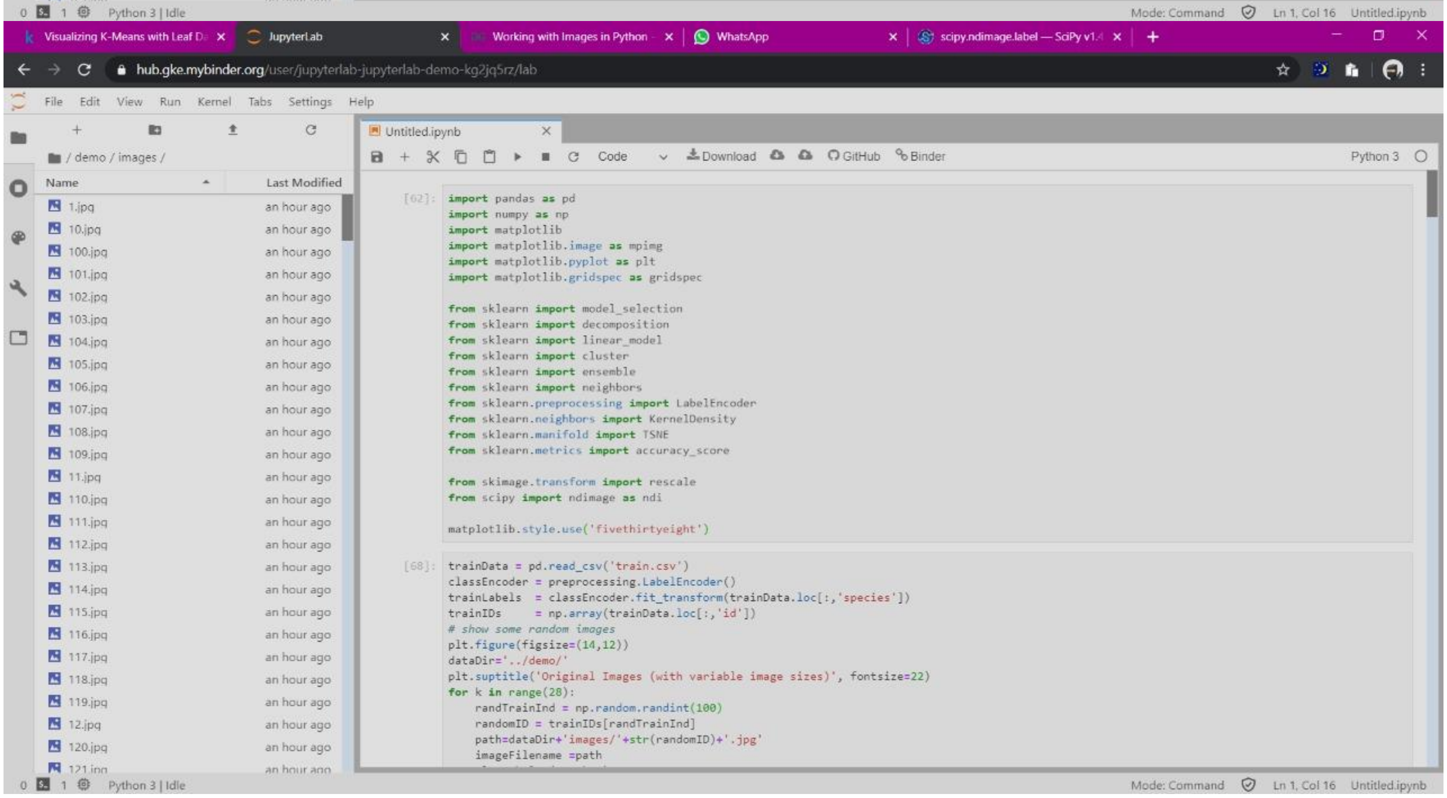
```
# make sure nothing was ruined in the process
plt.figure(figsize=(14,7))
plt.suptitle('Processed Images (fixed size)', fontsize=22)
for k in range(28):
    randInd = np.random.randint(np.shape(fullImageMatrix3D)[2])
    plt.subplot(4,7,k+1); plt.imshow(fullImageMatrix3D[:,:,randInd], cmap='gray'); plt.axis('off')
'''

# re crop according to rows and columns that don't have zeros in them in any image
xValid = fullImageMatrix3D.mean(axis=2).sum(axis=0) > 0
yValid = fullImageMatrix3D.mean(axis=2).sum(axis=1) > 0
xLims = (np.nonzero(xValid)[0][0],np.nonzero(xValid)[0][-1])
yLims = (np.nonzero(yValid)[0][0],np.nonzero(yValid)[0][-1])
fullImageMatrix3D = fullImageMatrix3D[yLims[0]:yLims[1],xLims[0]:xLims[1],:]

# make sure nothing was ruined in the process
plt.figure(figsize=(14,7))
plt.suptitle('Final Processed Images (with fixed image size)', fontsize=22)
for k in range(28):
    randInd = np.random.randint(np.shape(fullImageMatrix3D)[2])
    plt.subplot(4,7,k+1); plt.imshow(fullImageMatrix3D[:,:,randInd], cmap='gray'); plt.axis('off')
    if randInd < len(trainLabels):
        plt.title(classEncoder.classes_[trainLabels[randInd]], fontsize=10)
    else:
        plt.title('test data sample', fontsize=10)

# scale down all images to be in normal size
rescaleFactor = 0.15

scaledDownImage = rescale(fullImageMatrix3D[:,:,0],rescaleFactor)
scaledDownImages = np.zeros(np.hstack((np.shape(scaledDownImage),
                                       np.shape(fullImageMatrix3D)[2])),dtype=np.dtype('f4'))
for imInd in range(np.shape(fullImageMatrix3D)[2]):
    scaledDownImages[:,:,imInd] = rescale(fullImageMatrix3D[:,:,imInd],rescaleFactor)

del fullImageMatrix3D
```



```
[69]: numImages = 100
dataDir='../demo/'
shapesMatrix = np.zeros((2,numImages))
listOfImages = []
for k in range(numImages):
    imageFilename = dataDir + 'images/' + str(k+1) + '.jpg'
    currImage = mpimg.imread(imageFilename)
    shapesMatrix[:,k] = np.shape(currImage)
    listOfImages.append(currImage)

# calculate the shape of an image that will contain all original images within it
maxShapeSize = shapesMatrix.max(axis=1)
for k in range(len(maxShapeSize)):
    if maxShapeSize[k] % 2 == 0:
        maxShapeSize[k] += 311
    else:
        maxShapeSize[k] += 310


# place all original images at the center of the large reference frame
fullImageMatrix3D = np.zeros(np.hstack((maxShapeSize, np.shape(shapesMatrix[1]))).astype(int),dtype=np.dtype('u1'))
destXc = (maxShapeSize[1]+1)/2; destYc = (maxShapeSize[0]+1)/2
for k, currImage in enumerate(listOfImages):
    Yc, Xc = ndi.center_of_mass(currImage)
    Xd = destXc - Xc; Yd = destYc - Yc
    rowIndLims = (int(round(Yd)),int(round(Yd)+np.shape(currImage)[0]))
    colIndLims = (int(round(Xd)),int(round(Xd)+np.shape(currImage)[1]))
    fullImageMatrix3D[rowIndLims[0]:rowIndLims[1],colIndLims[0]:colIndLims[1],k] = currImage

'''
# make sure nothing was ruined in the process
plt.figure(figsize=(14,7))
plt.suptitle('Processed Images (fixed size)', fontsize=22)
for k in range(28):
    randInd = np.random.randint(np.shape(fullImageMatrix3D)[2])
    plt.subplot(4,7,k+1); plt.imshow(fullImageMatrix3D[:,:,randInd], cmap='gray'); plt.axis('off')
'''
```

Leaf Detection





CHAPTER 5

REFERENCES

- https://www.google.com/
- https://www.javatpoint.com/
- https://stackoverflow.com/
- https://www.kaggle.com/