In [218…

```
import cv2
import numpy as np
import os
import sys
from sklearn import tree
from sklearn import preprocessing
import graphviz
#I Have problems with packages try running without this command
os.environ['KMP_DUPLICATE_LIB_OK']='True'
import glob
import matplotlib.pyplot as plt
from skimage.feature import local_binary_pattern
import torch
import torchvision
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import pathlib
from torchvision.transforms import ToTensor
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn import semi_supervised
from sklearn.model_selection import GridSearchCV, train_test_split
```

In [219…

```
#checking for device
if torch.backends.mps.is_available():
    mps_device = torch.device("mps")
    x = torch.ones(1, device=mps_device)
    print (x)
else:
    print ("MPS device not found.")
```

```
tensor([1.], device='mps:0')
```

In [220…

```
def extract_color_features(image, target_size):
    # Convert the image to the HSV color space
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Define the number of bins for each channel in the histogram
    hue_bins = 8
    saturation_bins = 8
    value_bins = 8

    # Calculate the color histogram for each channel
    hue_hist = cv2.calcHist([hsv_image], [0], None, [hue_bins], [0, 180])
    saturation_hist = cv2.calcHist([hsv_image], [1], None, [saturation_bins]
    value_hist = cv2.calcHist([hsv_image], [2], None, [value_bins], [0, 256]

    # Normalize the histograms
    cv2.normalize(hue_hist, hue_hist, 0, 1, cv2.NORM_MINMAX)
    cv2.normalize(saturation_hist, saturation_hist, 0, 1, cv2.NORM_MINMAX)
    cv2.normalize(value_hist, value_hist, 0, 1, cv2.NORM_MINMAX)

    # Concatenate the histograms into a single feature vector
    color_features = np.concatenate((hue_hist.flatten(), saturation_hist.fla

    # Resize the color features to the target size
    if len(color_features) < target_size:
        color_features = np.pad(color_features, (0, target_size - len(color_
    elif len(color_features) > target_size:
        color_features = color_features[:target_size]

    #print("Color features:", color_features.shape)

    return color_features
```

In [221…

```
def extract_shape_features(image, target_size):
    # Convert the image to grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply binary thresholding to obtain a binary image
    _, binary_image = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY_IN

    # Find contours in the binary image
    contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL, cv2.CHAI

    # Initialize a list to store shape features
    shape_features = []

    # Iterate over the contours
    for contour in contours:
        # Calculate contour-based features
        area = cv2.contourArea(contour)
        perimeter = cv2.arcLength(contour, True)
        _, _, width, height = cv2.boundingRect(contour)
        aspect_ratio = width / float(height) if height != 0 else 0
        circularity = 4 * np.pi * area / (perimeter ** 2) if perimeter != 0

        # Append the shape features to the list
        shape_features.extend([area, perimeter, aspect_ratio, circularity])

    # Convert the shape features list to a numpy array
    shape_features = np.array(shape_features)

    # Resize the shape features to the target size
    if len(shape_features) < target_size:
        shape_features = np.pad(shape_features, (0, target_size - len(shape_
    elif len(shape_features) > target_size:
        shape_features = shape_features[:target_size]

    #print("Shape features:", shape_features.shape)
    return shape_features
```

In [222…
```
def extract_texture_features(image, target_size):
    # Convert the image to grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Calculate the Local Binary Pattern (LBP) for the grayscale image
    radius = 1
    n_points = 8 * radius
    lbp_image = local_binary_pattern(gray_image, n_points, radius, method='u

    # Calculate the histogram of the LBP image
    hist, _ = np.histogram(lbp_image.ravel(), bins=np.arange(0, n_points + 3

    # Normalize the histogram
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-7)

    # Flatten and return the histogram as the texture feature vector
    texture_features = hist.flatten()

    # Resize the texture features to the target size
    if len(texture_features) < target_size:
        texture_features = np.pad(texture_features, (0, target_size - len(te
    elif len(texture_features) > target_size:
        texture_features = texture_features[:target_size]

    #print("Texture features:", texture_features.shape)

    return texture_features
```

In [223…
```
def combine_features(image):
    # Load and preprocess the image
    # Assuming image is already loaded or you can use OpenCV to load it
    preprocessed_image = image
    #print(preprocessed_image)

    # Extract features using different methods
    color_features = extract_color_features(preprocessed_image,60)
    shape_features = extract_shape_features(preprocessed_image,50)
    texture_features = extract_texture_features(preprocessed_image,10)

    # Combine the features into a single vector
    #print(color_features.shape, shape_features.shape, texture_features.shap
    combined_features = np.concatenate((color_features, shape_features, text

    return combined_features
```

In [224…
```
#folder_path = "/Users/hadi/Desktop/Concordia/Comp 6721/AIproject/fruits/tra

def generate_features(image):
```

```
    #image = cv2.imread(image_path)
    #new_size = (32, 32)
    #image = cv2.resize(image, new_size)
    combined_features = combine_features(image)
    return combined_features

def check_label(element):
    if element == 'Banana_Training' or element == "Banana_Test" or element =
        return 1
    elif element == 'Kiwi_Training' or element == "Kiwi_Test" or element ==
        return 2
    elif element == 'Mango_Training' or element == "Mango_Test" or element =
        return 3
    elif element == 'Orange_Training' or element == "Orange_Test" or element
        return 4
    elif element == 'Plum_Training' or element == "Plum_Test" or element ==
        return 5
    elif element == 'Apple_Training' or element == "Apple_Test" or  element
        return 6
    else:
        return 0  # Return 0 if the element is not found in the list

def loadImages(folder_path,class_):
    #print(folder_path)
    folder_path = folder_path
    file_list = os.listdir(folder_path)
    class_features = np.empty((0,121))
    for file_name in file_list:
         if file_name.endswith(".jpg") or file_name.endswith(".png"):
             image_path = os.path.join(folder_path, file_name)
             # Perform your image processing tasks here
             image = cv2.imread(image_path)
             new_size = (32, 32)
             image = cv2.resize(image, new_size)
             combined_features = combine_features(image)
             #print(check_label(class_))
             combined_features = np.append(combined_features, check_label(cl
             #print(combined_features.shape)
             combined_features = np.expand_dims(combined_features, axis=0)
             class_features = np.append(class_features, combined_features,ax
    #print(class_features[0])
    return class_features




#Print the shape of the combined feature vector
#loadImages(folder_path,"Kiwi_Training")
```

In [225…

```
train_path = "/Users/hadi/Desktop/Concordia/Comp 6721/AIproject/fruits/trair
test_path = "/Users/hadi/Desktop/Concordia/Comp 6721/AIproject/fruits/testir
val_path = "/Users/hadi/Desktop/Concordia/Comp 6721/AIproject/fruits/validat

root_training=pathlib.Path(train_path)
root_testing=pathlib.Path(test_path)
root_val = pathlib.Path(val_path)


def generate_feature_vector(root,path):
    classes = []
    features = np.empty((0,121))
    labels = []
    for class_dir in root.iterdir():
        class_ = class_dir.name.split('/')[-1]
        print(class_)
        path_ = path +"/"
        if(class_!=".DS_Store"):
            print(class_)
            path_ = path+"/"+class_
            temp = loadImages(path_,class_)
            path_ = ""
            classes.append(class_)
            features = np.append(features, temp,axis=0)

    return features
#classes=sorted([j.name.split('/')[-1] for j in root.iterdir()])
#print(classes) #['Banana_Training', 'Kiwi_Training', 'Mango_Training', 'Ora
```

In [226…

```
#calculate size of training and testing images

f = generate_feature_vector(root_training,train_path) #total training featur
t = generate_feature_vector(root_testing,test_path) #total testing features
v = generate_feature_vector(root_val,val_path) #total val features

print(f.shape)
print(t.shape)
print(v.shape)
#train_count = len(glob.glob(train_path+"/**/*.png"))
#test_count = len(glob.glob(test_path+"/**/*.png"))

#print(train_count,test_count)
```

```
Banana_Training
Banana_Training
.DS_Store
Kiwi_Training
Kiwi_Training
Mango_Training
Mango_Training
Orange_Training
Orange_Training
Plum_Training
Plum_Training
Apple_Training
Apple_Training
.DS_Store
Kiwi_Test
Kiwi_Test
Plum_Test
Plum_Test
Orange_Test
Orange_Test
Apple_Test
Apple_Test
Mango_Test
Mango_Test
Banana_Test
Banana_Test
.DS_Store
Mango_Validation
Mango_Validation
Plum_Validation
Plum_Validation
Banana_Validation
Banana_Validation
Apple_Validation
Apple_Validation
Kiwi_Validation
Kiwi_Validation
Orange_Validation
Orange_Validation
(14676, 121)
(4583, 121)
(3677, 121)
```

In [255…

```
#Semi-supervised learning

xtrain = f[:,:-1] #14k, 54 feature vector 15x54
ytrain = f[:,-1] #14k , 1 label per image 6 classes

xval = v[:,:-1]
yval = v[:,-1]
```

```
xtest = t[:,:-1]
ytest = t[:,-1]

xtotal = np.vstack((xtrain, xval))
ytotal = np.concatenate((ytrain, yval))
ytotal_unlabled = ytotal.copy()

rng = np.random.RandomState(0)

# Creating unlabeled data by assigning "-1"
unl_pts = rng.rand(ytotal.shape[0]) > 0.1 #[0.012,,0.8...] size(ytotal) => [
ytotal_unlabled[unl_pts] = -1 #assign true values to -1
count = 0
for e in ytotal_unlabled:
    if(e==-1):
        count += 1
print("Inlabled points,", unl_pts)
print("Size of labels:" , ytotal_unlabled)
print("Count of unlabled data:" , count)
# Semi-Supervised learning
dtc = tree.DecisionTreeClassifier(criterion="entropy")

# Set the threshold for predicted probabilities
threshold = 0.99  # Example threshold value
max_iter = 1000  # Example maximum number of iterations

lbl = semi_supervised.SelfTrainingClassifier(dtc, threshold=threshold,max_it
lbl.fit(xtotal, ytotal_unlabled)

y_pred = lbl.predict(xtotal)
y_pred2 = lbl.predict(xtest)


# with np.printoptions(threshold=1000):
#     print("Labels for training", xtotal)
#     print("Actual labled data: ", ytotal[5000:6000])
#     print("Unlabled data: ", ytotal_unlabled[5000:6000])
#     print("Semi Supervised Labels: ", y_pred[5000:6000])

#Getting accuracy for original lables vs preicted labels

acu = classification_report(ytotal,y_pred) #Accuracy labled vs unlabled data
acu2 = classification_report(ytest,y_pred2) #Accuracy on testing data

print("Accuracy score:", acu)
print("Accuracy score:", acu2)
```

```
Inlabled points, [ True  True  True ...  True  True  True]
Size of labels: [-1. -1. -1. ... -1. -1. -1.]
```

```
Count of unlabled data: 16433
Actual labled data:  [3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3
. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.]
Unlabled data:  [-1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
-1. -1. -1.
 -1. -1. -1. -1.  3. -1. -1. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1.  3. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
```

```
 -1. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1.  3. -1. -1. -1. -1.  3.
 -1.  3. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1. -1. -1. -1. -1.  3.  3.
 -1. -1.  3. -1.  3.  3. -1. -1. -1. -1. -1. -1. -1.  3. -1. -1. -1. -1.
 -1.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1.  3. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1.
 -1.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3.
 -1. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1.  3. -1. -1.
 -1. -1. -1. -1. -1. -1.  3. -1. -1. -1.  3. -1. -1.  3. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1.  3. -1.  3. -1. -1. -1. -1. -1. -1. -1.  3. -1. -1.
 -1. -1.  3. -1. -1. -1. -1. -1. -1.  3. -1. -1. -1.  3. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1.  3. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1.
 -1. -1. -1. -1.  3. -1. -1.  3. -1. -1. -1.  3. -1. -1. -1. -1. -1. -1.
  3. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1.  3. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1.
  3. -1. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1.
 -1. -1. -1. -1.  3. -1. -1. -1. -1.  3.  3. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1.  3.  3. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1.  3. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1.
 -1. -1.  3.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1.
 -1. -1. -1. -1. -1. -1. -1.  3. -1.  3. -1. -1. -1. -1. -1.  3. -1. -1.
 -1. -1.  3. -1. -1.  3. -1. -1. -1. -1. -1. -1.  3. -1.  3. -1. -1. -1.
 -1. -1. -1. -1. -1.  3. -1.  3.  3. -1. -1. -1. -1. -1. -1.  3. -1. -1.
 -1. -1. -1. -1. -1.  3.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3.
 -1.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1.  3. -1. -1. -1. -1. -1. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3.  3. -1. -1. -1. -1. -1. -1.
 -1. -1.  3. -1. -1. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
  3. -1. -1. -1. -1. -1. -1. -1. -1.  3.  3. -1. -1. -1. -1. -1. -1. -1.
  3. -1.  3. -1. -1. -1. -1.  3. -1. -1. -1.  3. -1.  3. -1. -1. -1.  3.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  3. -1.
 -1. -1. -1. -1. -1. -1.  3. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1.  3. -1. -1. -1. -1. -1. -1. -1. -1.  3.  3. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
```

```
  3. -1. -1. -1. -1. -1. -1. -1. -1. -1.]
Semi Supervised Labels:  [3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 6. 1. 3. 3. 5.
3. 3. 2. 3. 3. 3. 3.
 3. 1. 6. 3. 3. 3. 3. 1. 1. 3. 1. 3. 3. 3. 3. 3. 3. 3. 1. 6. 3. 3. 3. 3.
 3. 1. 3. 3. 1. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 6. 3. 3. 3. 3. 1. 3. 3. 6.
 3. 4. 3. 3. 3. 1. 3. 3. 3. 3. 3. 2. 3. 3. 3. 3. 1. 6. 1. 3. 3. 3. 3. 1.
 1. 3. 3. 3. 3. 3. 1. 3. 3. 3. 6. 6. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 6.
 3. 3. 2. 1. 3. 3. 3. 3. 3. 3. 3. 6. 6. 1. 3. 1. 3. 2. 3. 1. 3. 6. 3. 3.
 2. 2. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 2. 3. 3. 3. 6. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 1. 6. 3. 1. 3. 3. 3. 3. 6. 3. 3. 6. 6. 3. 2. 6. 1. 3. 3. 3. 3.
 3. 1. 2. 3. 3. 3. 3. 3. 3. 3. 3. 3. 1. 1. 1. 1. 3. 3. 3. 6. 3. 3. 3. 3.
 6. 3. 1. 6. 1. 6. 1. 3. 2. 3. 3. 1. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 4. 3.
 3. 3. 4. 3. 1. 3. 3. 3. 3. 3. 3. 6. 1. 3. 3. 1. 3. 6. 1. 3. 3. 3. 3. 1.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 6. 3. 3. 3. 6. 1. 3. 1. 3. 3. 3.
 3. 6. 3. 3. 3. 3. 3. 3. 3. 1. 3. 3. 2. 3. 3. 3. 3. 3. 3. 3. 3. 6. 1. 3.
 3. 1. 3. 3. 3. 3. 3. 2. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 6. 3. 3. 2.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 2. 3. 3. 3. 1. 3. 3. 3. 3. 3. 6. 3. 3.
 3. 3. 3. 1. 3. 3. 3. 3. 3. 3. 3. 3. 3. 4. 1. 6. 3. 3. 1. 3. 3. 3. 3. 3.
 3. 3. 6. 1. 1. 6. 3. 3. 3. 3. 3. 3. 6. 3. 3. 3. 3. 1. 3. 2. 3. 3. 2. 6.
 3. 3. 3. 3. 3. 1. 1. 3. 2. 3. 3. 3. 3. 3. 1. 3. 3. 3. 6. 3. 4. 3. 6. 3.
 6. 3. 6. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 1. 4. 3. 2. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 1. 3.
 3. 1. 3. 3. 3. 3. 3. 3. 1. 3. 3. 3. 2. 2. 6. 3. 3. 3. 3. 1. 3. 3. 2. 4.
 3. 3. 3. 1. 1. 3. 3. 2. 1. 3. 6. 3. 6. 3. 1. 3. 2. 6. 3. 3. 3. 1. 3. 3.
 3. 1. 1. 3. 2. 3. 1. 6. 1. 1. 3. 3. 3. 4. 2. 3. 6. 1. 1. 3. 2. 1. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 1. 3. 6. 3. 2. 3. 3. 2. 3. 3. 3. 3.
 6. 3. 3. 1. 3. 3. 3. 3. 3. 3. 3. 2. 3. 3. 2. 3. 3. 1. 3. 1. 3. 1. 6. 3.
 6. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 1. 3. 3. 3. 3. 3. 3. 6.
 1. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 1.
 1. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 6. 1. 2. 3. 3. 3. 3. 1. 3. 1. 6. 1. 3.
 1. 3. 3. 3. 6. 1. 6. 3. 3. 3. 3. 3. 6. 1. 3. 2. 3. 6. 3. 3. 3. 3. 3. 5.
 3. 1. 3. 3. 3. 3. 3. 3. 3. 6. 3. 3. 3. 1. 3. 6. 4. 1. 3. 3. 3. 3. 3. 3.
 6. 2. 1. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 6. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 2. 3. 3. 2. 3. 3. 3. 3. 1. 3. 3. 3. 1. 3. 3. 3. 3. 1. 3. 3. 4. 3.
 3. 3. 2. 1. 1. 3. 3. 3. 3. 1. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 1. 1.
 3. 3. 6. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 6. 2. 3. 2. 1. 3.
 3. 3. 2. 3. 3. 6. 4. 3. 1. 3. 3. 3. 3. 3. 3. 3. 1. 6. 1. 1. 3. 1. 3. 3.
 1. 6. 1. 3. 3. 3. 3. 6. 3. 3. 3. 3. 3. 3. 3. 3. 3. 2. 3. 3. 3. 6. 1. 5.
 3. 3. 1. 5. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 5. 3. 6. 3. 3. 3. 3. 3. 1.
 3. 3. 2. 3. 3. 3. 3. 3. 6. 1. 3. 3. 1. 1. 5. 6. 2. 3. 1. 3. 3. 3. 3. 3.
 3. 3. 2. 3. 3. 3. 2. 3. 1. 6. 2. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 6.
 3. 3. 3. 3. 6. 3. 3. 3. 3. 1. 3. 3. 3. 3. 1. 2. 3. 3. 3. 3. 3. 3. 1. 1.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 2. 3. 3. 3. 3. 1. 3. 3. 3. 3. 3. 4. 3. 3. 1.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 1. 3. 3. 1. 1. 3. 3.]
Accuracy score:               precision    recall  f1-score   support

         1.0       0.73      0.69      0.71      2422
         2.0       0.85      0.83      0.84      3434
         3.0       0.72      0.76      0.74      3323
         4.0       0.92      0.91      0.92      2409
```

```
         5.0       0.90      0.89      0.90      1838
         6.0       0.84      0.84      0.84      4927

    accuracy                           0.82     18353
   macro avg       0.83      0.82      0.82     18353
weighted avg       0.82      0.82      0.82     18353

Accuracy score:               precision    recall  f1-score   support

         1.0       0.71      0.60      0.65       605
         2.0       0.75      0.75      0.75       858
         3.0       0.63      0.82      0.72       831
         4.0       0.92      0.92      0.92       603
         5.0       0.89      0.83      0.86       460
         6.0       0.81      0.73      0.77      1226

    accuracy                           0.77      4583
   macro avg       0.79      0.78      0.78      4583
weighted avg       0.78      0.77      0.77      4583
```

In [197…

```python
#training model

xtrain = f[:,:-1] #14k, 54 feature vector 15x54
ytrain = f[:,-1] #14k , 1 label per image 6 classes

xval = v[:,:-1]
yval = v[:,-1]

dtc = tree.DecisionTreeClassifier(criterion="entropy")
dtc.fit(xtrain, ytrain)

y_pred = dtc.predict(xval)

print(classification_report(yval, y_pred)) #metrics values
print("Confusion Matrix:\n", confusion_matrix(yval, y_pred)) #confusion matr
```

```
              precision    recall  f1-score   support

         1.0       0.68      0.69      0.69       485
         2.0       0.88      0.91      0.89       686
         3.0       0.75      0.77      0.76       665
         4.0       0.91      0.93      0.92       482
         5.0       0.96      0.96      0.96       368
         6.0       0.91      0.85      0.88       991

    accuracy                           0.85      3677
   macro avg       0.85      0.85      0.85      3677
weighted avg       0.85      0.85      0.85      3677

Confusion Matrix:
 [[336  10 121  12   2   4]
 [ 10 623  25   1   7  20]
 [ 76  25 515  10   1  38]
 [  9   2   7 448   0  16]
 [  0  11   0   0 354   3]
 [ 60  41  23  20   5 842]]
```

In [201…

```
# #Model Training and saving best model

xtest = t[:,:-1]
ytest = t[:,-1]

y_pred = dtc.predict(xtest)

print(classification_report(ytest, y_pred)) #metrics values
print("Confusion Matrix:\n", confusion_matrix(ytest, y_pred)) #confusion mat

num_leaves = dtc.tree_.n_leaves
num_nodes = dtc.tree_.node_count
maxd = dtc.tree_.max_depth

print("Number of leaves:", num_leaves)
print("Number of nodes:", num_nodes)
print("Max of depth:", maxd)

#plotting tree
# tree.plot_tree(dtc)
# elements = ["color"] * 24 + ["shape"] * 20 + ["texture"] * 10
# dot_data = tree.export_graphviz(dtc, out_file=None, feature_names=elements
# graph = graphviz.Source(dot_data)
# graph.render("mytree")
```

```
              precision    recall  f1-score   support

         1.0       0.74      0.66      0.70       605
         2.0       0.80      0.89      0.84       858
         3.0       0.73      0.83      0.78       831
         4.0       0.94      0.95      0.95       603
         5.0       0.93      0.97      0.95       460
         6.0       0.89      0.77      0.83      1226

    accuracy                           0.83      4583
   macro avg       0.84      0.84      0.84      4583
weighted avg       0.84      0.83      0.83      4583

Confusion Matrix:
 [[402  12 171   6   3  11]
 [ 20 763  15   1   5  54]
 [ 72  24 687  15   2  31]
 [  4   1  12 573   1  12]
 [  0  11   0   0 446   3]
 [ 44 148  53  14  25 942]]
Number of leaves: 729
Number of nodes: 1457
Max of number of nodes: 18
```

In [ ]:

In [ ]: