```python
#importing the necesary libraries
import cv2
from PIL import Image #used for PPM format images
import os #used to iterate over files and open them
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt #visualzie the grayscale + binary images
import warnings#command to supress warnings
warnings.filterwarnings('ignore', category=FutureWarning)
from collections import defaultdict
import math
import tempfile #used for uploading the files
import objc
import av
import tkinter as tk #used to create GUI for the users
from tkinter import filedialog
import random #used to isolate random data

#Initializing global variables in order to save the user-selected points & the filepath when
uploaded
x_global = []
y_global = []
window = None
user_input = ""
window_result = None
click_count = 0

#Reads the image in the train directory and returns an array of the stored images
def read_image(folder,l):
    print("soze")
    print(len(l))
    #L represents the unique indicies that will extract randomly selected images
    if l is not None:
        #Initialize an empty array to store the images
        img_total = []
        #The folder path is the images in the train folder.
        #The folder parameter is either benign or malignant
        folder_path = "train/" + folder + "/"
        file_names = os.listdir(folder_path)
        for i in range(0,600):
            #Iterate through all the files in the train directory
            if i in l:
                #Only isolate the filename if the L is equal to the iteration
                filename = file_names[i]
                path = folder_path + filename
                #Read the image with the specified path
                img = cv2.imread(path)
                #Append to array
                img_total.append(img)
        #Return the final array
        return img_total
    #If L is not none, and all train images should be returned then use this method.
    else:
        img_total = []
        folder_path = "train/" + folder + "/"
        file_names = os.listdir(folder_path)
        for i in range(0,600):
            filename = file_names[i]
            path = folder_path + filename
            img = cv2.imread(path)
            img_total.append(img)

        return img_total
```

```python
#This function is responsible for reading the test images
def read_image_test(folder,l):
    #L is randomly selected indicies for the test images to be extracted
    if l is not None:
        #Initalize an empty array
        img_total = []
        #the folder parameter is either malignant or benign
        folder_path = "test/" + folder + "/"
        file_names = os.listdir(folder_path)
        #Iterate through all the files in the folder
        for i in range(0,100):
            #If that index is equal to the iteration...
            if i in l:
                filename = file_names[i]
                path = folder_path + filename
                #Read the selected image based on the specified path
                img = cv2.imread(path)
                #Append to the array
                img_total.append(img)
        #Return the final array of stored images
        return img_total
    #If l is not specified, then read all the test images
    else:
        img_total = []
        folder_path = "test/" + folder + "/"
        file_names = os.listdir(folder_path)
        for i in range(0,100):
            filename = file_names[i]
            path = folder_path + filename
            img = cv2.imread(path)
            img_total.append(img)

        return img_total

#This method applies GaussianBlur to the image
def gaussianFilter(img):
    #Applying the blur and returning the new image
    img_filtered = cv2.GaussianBlur(img, (5,5), 0)
    return img_filtered

#This method is responsbile for attaining the coordinates that a user clicked on
def get_click_coordinates(event, x, y, flag, params):
    #Intialize global variable of count
    global click_count
    if event == cv2.EVENT_LBUTTONDOWN:
        #If they pressed less than four times, then obtain the coordinates of the
        #clicked points and the threshold value
        if click_count < 4:
            global y_global
            global x_global
            # get the pixel value at the clicked point
            # use the pixel value as the threshold value
            y_global.append(y)
            x_global.append(x)
            #Draw a dot on the image and show the updated image to the user
            cv2.circle(params, (x, y), 5, (0, 0, 255), -1)
            cv2.imshow('image', params)
        #Update the count
        click_count +=1


#This method calculates the binary image using Otsu threshold for the training image set
```

```
def otsuThreshold(img):
    #Convert the image into a gray scale
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #Calculate the threshold using Otsu's method
    _, img_thresh = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    #Reverse the white and black pixels
    img_thresh = cv2.bitwise_not(img_thresh)
    #return the new image
    return img_thresh

#This method calculates the otsu threshold for the user-inputted image
def otsuThreshold1(img):
    #Initalize the globsl variables that contain the pixel coordinates
    global y_global
    global x_global
    global click_count
    click_count= 0
    y_global = []
    x_global = []
    #Create a copy of the image
    new_img = img.copy()
    #Prompt users to click on the border points of the mole
    cv2.namedWindow("image")
    cv2.putText(new_img, 'Click on the 4 points on the border of the mole', (50, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2, cv2.LINE_AA)
    cv2.setMouseCallback('image', get_click_coordinates, new_img)
    #End the session after the user has selected on four points
    while click_count < 4:
        cv2.imshow('image', new_img)
        cv2.waitKey(1)
    cv2.destroyAllWindows()

    #Append on the pixel values to array uisng the coordinates
    pixel_value = []
    for i,j in zip(y_global, x_global):
        pixel_value.append(img[i,j][0])
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    #Apply a threshold using the mean of the pixel values
    _, img_thresh = cv2.threshold(img, np.mean(pixel_value), 255, cv2.THRESH_BINARY)
    #Reverse the white and black pixels
    img_thresh = cv2.bitwise_not(img_thresh)
    y_min = max(0, min(y_global)-300)
    x_min = max(0, min(x_global)-300)
    width = max(y_global) - y_min + 300
    height = max(x_global) - x_min + 300
    #Cropping the bounding rectangle of the image
    img_crop_thresh = img_thresh[y_min:y_min+width, x_min:x_min+height]

    #return the new image
    return img_crop_thresh, [y_min, x_min, width, height]

#Define a contour around the mole
def contour(img):
    #Find all the possible contours using the binary image
    contours, hierarchy = cv2.findContours(img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    img_contours = np.zeros(img.shape)
    if len(contours) == 0:
        return None, None
    #Choose the largest contour out of the possible contour
    c = max(contours, key = cv2.contourArea)

    cv2.drawContours(img_contours, c, -1, (255,0, 0), 3)
```

```python
    #Finding the bounding Rectangle of the contour
    x, y, w, h = cv2.boundingRect(c)
    #Return the contour and the bounding rectangle
    return c, (x,y,w,h)

#This method calculates the assymetry score of the image using the contour
def assym_score(contour):
    # Calculate the center of mass of the object
    if contour is not None:
        M = cv2.moments(contour)
        cx = int(M['m10'] / M['m00'])
        cy = int(M['m01'] / M['m00'])

        # Calculate the distance between the center of mass and each point on the contour
        distances = []
        #Complete this by iterating through a loop
        for point in contour:
            #Use pointPolygon test to iterate through the contour and find the distances
            distance = cv2.pointPolygonTest(point, (cx, cy), True)
            distances.append(distance)
        #Return the standard deviation of the points which wil be used as the asym_score.
        return np.std(distances)
    return "none"

#This function compares the contour with a perfect circle contour
def border(img,c,x,y,w,h):
    #Define the center of the circle based on the bounding rectangle x, y quadrant_coordinates
    #and the provided height, width
    center = (x+w/2, y+h/2)
    s = min(w,h)
    #The radius of the circle is the min of the width & height of the rectangle by 2
    radius = s/2
    #Define the circle contour
    circle_contour = np.array([[center[0] + radius*np.cos(theta), center[1] +
radius*np.sin(theta)] for theta in np.linspace(0, 2*np.pi, 100)]).astype(np.int32)
    #Find the match score between the circle contour and the mole contour
    match_score = cv2.matchShapes(circle_contour, c, cv2.CONTOURS_MATCH_I2, 0.0)
    return match_score

#Returning the diameter of the mole
def diameter_check(w,h):
    #Either choosing the min width or height of the bounding rectangle as the diameter of the mole
    s = min(w,h)
    #Finding the length by converting pixels to inches to mm
    millimeters = (s / 550) * 25.4
    return millimeters

#This method finds the color variation of the image
def colorfulness(img):
    # split the image into its respective RGB components
    (B, G, R) = cv2.split(img.astype("float"))
    # compute rg = R - G
    rg = np.absolute(R - G)
    # compute yb = 0.5 * (R + G) - B
    yb = np.absolute(0.5 * (R + G) - B)
    # compute the mean and standard deviation of both `rg` and `yb`
    (rb_mean, rb_std) = (np.mean(rg), np.std(rg))
    (yb_mean, yb_std) = (np.mean(yb), np.std(yb))
    # combine the mean an d standard deviations
    std_root = np.sqrt((rb_std ** 2) + (yb_std ** 2))
    mean_root = np.sqrt((rb_mean ** 2) + (yb_mean ** 2))
    # Return the color metric (CITE HERE)
    return (std_root + (0.3 * mean_root))/(img.shape[0]*img.shape[1])
```

```python
#This method extracts all the ABCD scores and checks for a diagnosis
def extract(images, mal):
    #Initialize an empty array that holds all the diagnsosi in the array
    values = []
    #Intialize array that contains all the scores of the array so that the
    #training percentile scores can be extracted
    ss = []
    dd = []
    cc = []
    bb = []
    #iterate through all the stored images
    for i,img in enumerate(images):
        #Save the orginal image to use later
        img_old = img
        #Apply gaussian filter for smoothing
        img = gaussianFilter(img)
        #find the binary image
        img= otsuThreshold(img)
        max_contour, rectangle = contour(img)
        #Crop the image based on the bounding rect of the mole
        x,y,w,h = rectangle
        img_crop = img[y:y+h, x:x+w]
        #Find the assymmetry score
        asym = assym_score(max_contour)
        #Find the border score
        b = border(img_crop, max_contour, x,y,w,h)
        #Find the color variation
        color_threshold = colorfulness(img_old[y:y+h, x:x+w])
        #Return the diameter
        d = diameter_check(w,h)
        #Based on the abcd values, check the diagnosis
        __, value = check_the_diagnosis(asym, b, color_threshold, d)
        #Append the diagnosis
        values.append(value)
        #Append the scores to the arrays
        ss.append(asym)
        dd.append(d)
        cc.append(color_threshold)
        bb.append(b)

    return values

#Checks to see what ABCD features are flags for skin cancer
def check_the_diagnosis(a, border, color_threshold, d):
    #Find four boolean values for each of the abcd features
    bool1 = asym(a)
    bool2 = border_check(border)
    bool3 = color_check(color_threshold)
    bool4 = d_check(d)
    #Arrange in an array
    arr_of_bools = [bool1, bool2, bool3, bool4]

    #Return the percentage based on the amount of boolean variables that returned true
    if arr_of_bools.count(True) == 4:
        value =  "100"
    elif arr_of_bools.count(True) == 3:
        value = "75"
    elif arr_of_bools.count(True) == 2:
        value =  "50"
    elif arr_of_bools.count(True) == 1:
        value = "25"
    elif arr_of_bools.count(True) == 0:
```

```python
        value = "0"

    #Also return the array that contains which features out of abcd were flagged
    if bool1 == True:
        arr_of_bools[0] = "assymetry"
    if bool2 == True:
        arr_of_bools[1] = "border"
    if bool3 == True:
        arr_of_bools[2] = "color"
    if bool4 == True:
        arr_of_bools[3] = "diameter"
    return arr_of_bools, value

#This method calculates the performance of the train & test based on the labels provided
def performance(arr):
    #Benign is counted as if 0 or 1 feature were flagged
    len_of_ben = arr.count("0") + arr.count("25")
    len_of_mal = len(arr) - len_of_ben
    return len_of_ben/len(arr), len_of_mal/len(arr)


#The function returns true if the assymm score is higher than the threshold
def asym(num):
    if num > 10.5:
        return True

#The function returns true if the border score is higher than the threshold
def border_check(num):
    if num > 0.08:
        return True

#The function returns true if the color variation socre is higher than the threshold
def color_check(num):
    if num > 0.00608:
        return True
#The function returns true if the diameter is higher than the threshold of 6.4
def d_check(num):
    if num > 6:
        return True

#This method is responsbile for using the uploaded image by the user
#and providing a result through feature extraction
def new_image_from_user(path):

    #Read the image
    img_read = cv2.imread(path)
    #Apply Gaussian blue_convert
    img_read = gaussianFilter(img_read)
    #Apply Otsu's threshold through the user selection process
    img_read1, c= otsuThreshold1(img_read)
    #Find the contour of the image and the bounding rectangle
    max_contour, rectangle = contour(img_read1)
    font = cv2.FONT_HERSHEY_COMPLEX_SMALL
    #Error: if no mole is detected then ask the user to try again
    if max_contour is None:
        text = "No mole detected. Please try again"
        cv2.putText(img_read,text,(50,250), font, 1,(255,0,0),2,cv2.LINE_AA)
        cv2.imshow("Text",img_read)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    #Otherwise, then perform feature extraction
    else:
        x,y,w,h = rectangle
```

```
        #Crop the image into its bounding rectangle
        img_read2 = img_read1[y:y+h, x:x+w]
        #Extract the assymetry score
        asymm = assym_score(max_contour)
        #Extract the color score
        color_threshold = colorfulness(img_read[y:y+h, x:x+w])
        #Extract the border value
        b = border(img_read2, max_contour, x,y,w,h)
        #Extract the length of the diameter
        d = diameter_check(w,h)
        #Check the diagnosis using the extracted features
        arr, value = check_the_diagnosis(asymm, b, color_threshold, d)
        #Only return features that are not none
        arr = [x for x in arr if x is not None]
        #If the length of the array is 0, then no features were flagged
        text1 = "Out of the ABCD, the computer flagged: "
        if len(arr) == 0:
            text2 = 'nothing'
        #Otherwise, convert the features to a string
        else:
            text2 = str(arr)

        #Identify the risk level using the length of the array or how many features are present
        if len(arr) == 0:
            var = "no"
        if len(arr) == 1:
            var = "a low"
        if len(arr) == 2:
            var = "a medium"
        if len(arr) == 3 or len(arr) == 4:
            var = "a high"
        text3 = "there is " + var + " risk of skin cancer."
        #Display the image with its contouring for the user to evaluate
        cv2.drawContours(img_read[c[0]:c[0]+c[2], c[1]:c[1]+c[3]], max_contour, -1, (255,0, 0),
3)

        cv2.imshow("image",img_read[c[0]:c[0]+c[2], c[1]:c[1]+c[3]])
        #Present the resulting GUI screen using the risk level and features
        result_GUI(text1, text2, text3)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

#Destorying the window if button clicked
def on_button_click_destory():
    global window_result
    window_result.destroy()

#This method builds the resulting GUI that contains the risk level and features
def result_GUI(text1, text2, text3):
    global window_result
    # Create the GUI window
    window_result = tk.Tk()
    window_result.title("My GUI")
    window_result.geometry("400x200")  # Set the size of the window

    # Create a label to display the different texts
    text_label = tk.Label(window_result, text="Diagnosis", font=("Helvetica", 14, "bold italic"))
    text_label.pack()
    #Displaying the intro feature sentence
    text_label = tk.Label(window_result, text=text1)
    text_label.pack()
    #Displaying the flagged features
    text_label = tk.Label(window_result, text=text2)
```

```
    text_label.pack()
    #Displaying risk level
    text_label = tk.Label(window_result, text=text3)
    text_label.pack()

    # Create a button to quit.
    button = tk.Button(window_result, text="Done", command=on_button_click_destory)
    button.pack()

    # Run the GUI loop
    window_result.mainloop()

def open_image():
    # Open a file dialog box to select an image file
    file_path = filedialog.askopenfilename()
    #Initializing global variables of user_input and window to be able capture the file name from
the window
    global user_input
    global window

    user_input = file_path
    #Destorying the window once it is captured
    window.destroy()

#Main method
def main():
    #Loads the randomly selected indicies for the training and testing sets
    x0 = np.load('x0.npy')
    x1 = np.load('x1.npy')
    x2 = np.load('x2.npy')
    x3 = np.load('x3.npy')
    print(len(x0) + len(x1))
    print(len(x2) + len(x3))
    #Reads the benign images from the training set
    read_train_images_benign = read_image("benign", x0)

    #Reads the malignant images from the training set
    read_train_images_malignant = read_image("malignant", x1)
    #Reads the benign images from the test set
    read_test_images_benign = read_image_test("benign", x2)
    #Reads the malignant images from the test set
    read_test_images_malignant = read_image_test("malignant", x3)

    # Extract the diagnsosi of each of the image sets and return the accuracy
    arr= extract(read_train_images_benign, False)
    print(performance(arr))

    arr = extract(read_train_images_malignant, True)
    print(performance(arr))

    arr = extract(read_test_images_benign, False)
    print(performance(arr))

    arr= extract(read_test_images_malignant, True)
    print(performance(arr))

    #Running for the user inputed images
    global window
    window = tk.Tk()
    window.title("My GUI")
    window.geometry("400x200")

    # Create an input box for the user to type in
```

```
    global user_input

    # Create a button for the user to click
    button = tk.Button(window, text ='Open Image', command = open_image)
    button.pack()

    # Run the GUI loop
    window.mainloop()
    #Extracting the features from the user.
    new_image_from_user(user_input)

#Running the main function
main()
```