FAF.FIA16.1 -- Artificial Intelligence Fundamentals

Lab 4: Processing Images with OpenCV \ **Performed by:** Furdui Alexandru, group FAF-192 \ **Verified by:** Mihail Gavrilita, asist. univ.

Imports and Utils

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import csv
```

Task 1 -- Write blurring function and sharpening function

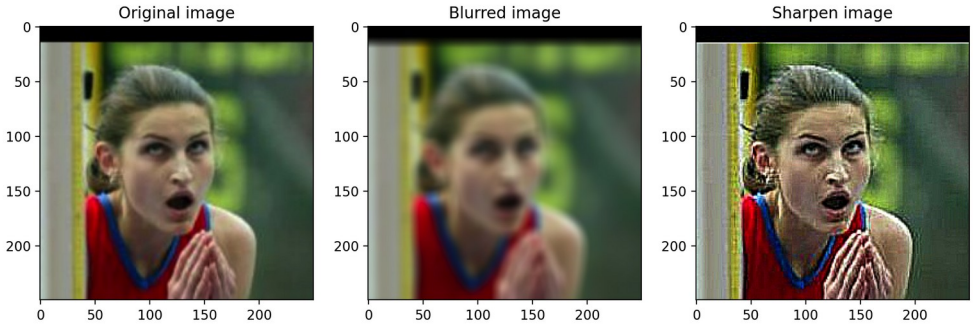
```
def blur_image(image_path, ksize=(10, 10)):
    image = cv2.imread(image_path)

    # Using cv2.blur() method
    blurred_image = cv2.blur(image, ksize)
    # Plot both images using subplots
    fig, axs = plt.subplots(1, 2, figsize=(10, 5))
    axs[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    axs[0].set_title('Original image')
    axs[1].imshow(cv2.cvtColor(blurred_image, cv2.COLOR_BGR2RGB))
    axs[1].set_title('Blurred image')
    plt.show()

def sharpen_image(image_path):
    image = cv2.imread(image_path)
    sharpen_filter = np.array([[-1,-1,-1],
                               [-1,9,-1],
                               [-1,-1,-1]])

    #apply the filter to the original photo
    sharp_image = cv2.filter2D(image, -1, sharpen_filter)
    fig, axs = plt.subplots(1, 2, figsize=(10, 5))
    axs[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    axs[0].set_title('Original image')
    axs[1].imshow(cv2.cvtColor(sharp_image, cv2.COLOR_BGR2RGB))
    axs[1].set_title('Blurred image')
    plt.show()
```

The goal of this task is to implement 2 functions: blurring and sharpening. \ Both were achieved by using OpenCV pre-defined blur and filter functions. It is possible to use just one function for both blurring and sharpening but you need to use 2 different matrices to be used in the convolution process. The most important thing is that the sum of all elements in the matrix should be 1, if it will not be satisfied, the image luminosity will be modified.



In the pictures above, the result of both blurring and sharpening function can be seen

Task 2 -- Implement a face detection system using OpenCV

1. The photo should be colored
2. The head of a person should represent 20% to 50% of the area of the photo
3. The eyes of a subject should be at the same level (with a max error of 5 pixels)
4. The photo should contain only one person
5. The photo should be in portrait orientation or square. Assume that the image given as input is not rotated;

```
def detect_face(image_path):
    cascade_path='haarcascade_frontalface_default.xml'
    face_cascade = cv2.CascadeClassifier(cascade_path)

    image = cv2.imread(image_path)

    #convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5, minSize=(30, 30))

    #if faces are detected, return the face coordinates
    if len(faces) > 0:
        return faces.tolist()
    else:
        return None
```

The detect_face function uses the cascade clasifier to apply it on th eprocessed photo, it hte function detects at least one face, it returns its coordinates, else None.

```
def is_gray(imgpath):
    img = cv2.imread(imgpath)
    if len(img.shape) < 3: return True
    if img.shape[2]  == 1: return True
    b,g,r = img[:,:,0], img[:,:,1], img[:,:,2]
    if (b==g).all() and (b==r).all(): return True
    return False
```

To satisfy the condition that the photo is colored, i implemented a function to detect if a photo is gray, pretty straight forward

```
def detect_eyes(imgpath):
    cascade_path='haarcascade_eye.xml'
    eyes_cascade = cv2.CascadeClassifier(cascade_path)
    image = cv2.imread(imgpath)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    eyes = eyes_cascade.detectMultiScale(gray, 1.1, 4)

    if len(eyes) >= 2:
        return abs(eyes[0][1] - eyes[1][1]) <= 5
    return False
```

Using the same method to detect faces, fucntion to detect eyes was implemented just using different cascade file, it also verifies if photo satisfies the condition that eyes should be at the same level with an error of max 5 px. Returns true or false.

```
def has_one_face(image_path):
    cascade_path='haarcascade_frontalface_default.xml'
    face_cascade = cv2.CascadeClassifier(cascade_path)

    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5, minSize=(30, 30))

    if len(faces) == 1:
        return True
    else:
        return False
```

The template from detect_face function was used to count the number of detected faces in a photo. Returns true or false

```
def is_portrait(image_path):
    height, width = image_path.shape[:2]

    if height == width:
        return True
    elif height > width:
        return True
    else:
        return False
```

THe funcion to check if the image is squared or is in portrait mode, i just compared image width and height and returning true or false, regarding the result

```
def calculate_face_area(image_path):
    cascade_path='haarcascade_frontalface_default.xml'
```

```
    face_cascade = cv2.CascadeClassifier(cascade_path)
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)

    total_image_area = image.shape[0] * image.shape[1]
    for (x, y, w, h) in faces:
        face_area = w * h
        if face_area >= 0.2 * total_image_area and face_area <= 0.5 *
total_image_area:
            return True
    return False
```

Lastly, the function to calculate the face area from a photo, if the face was detected, it just uses its coordinates to calculate the area ocupied. returns true or false, depending on if the face ocupies between 20 and 50% of total area.

Task 3 -- Test how good your system performs on a test dataset.

```
def calculate_result(img_path):
  if is_portrait(img_path) and calculate_face_area(img_path) and
has_one_face(img_path) and detect_eyes(img_path) and not
is_gray(img_path):
    return True
  else:
    return False

correct_detected = 0
total_images = 0

with open('test.csv', 'r') as file:
    reader = csv.reader(file)
    next(reader)  # skip the header

    for row in reader:
        result = calculate_result(row[0])
        expected_result = row[1]
        if str(result) == str(expected_result):
          correct_detected = correct_detected + 1
        total_images = total_images + 1

accuracy = correct_detected / total_images
print("{:.2f}%".format(accuracy * 100))
```

And putting all together, i have a loop that runs all the images through the system and check if they satisfy all conditions, if it does not, then the photo doesnt satisfy the requirements to be a passport photo. \ Calculating the accuracy, i just divided the number of correct detected by total number of photos.

Conclusions:

Concluding, after implementing this laboratory work, i achieved a system accuracy of 72.92%.

Bibliography:

https://www.geeksforgeeks.org/python-opencv-cv2-blur-method/ -- How to blur images in python using OpenCV

https://www.codespeedy.com/how-to-sharpen-an-image-in-python-using-opencv/ -- How to sharpen images in python using OpenCV

https://www.geeksforgeeks.org/face-detection-using-cascade-classifier-using-opencv-python/ -- How to use cascade clasifier to detect faces in a photo