COMSATS University Islamabad
Wah Campus

Program/Class: BS(CS)
Date: 21-11-2024
Subject: Pattern Recognition
Instructor: Samia Riaz
Student Name: HUSSAIN ALI
Reg #:FA21-BCS-066

Q.NO.1.

Iris detection paper implementation

Code:

```
!pip install opencv-python numpy scipy
PyMuPDF


import cv2

import numpy as np

import fitz
```

```
from google.colab import files
from scipy.spatial.distance import hamming

uploaded = files.upload()

def extract_image_from_pdf(pdf_path):
    pdf_document = fitz.open(pdf_path)
    output_image_path = None
    for page_num in range(len(pdf_document)):
        page = pdf_document[page_num]
        images = page.get_images(full=True)
        for img_index, img in enumerate(images):
            xref = img[0]
            base_image =
pdf_document.extract_image(xref)
            image_bytes = base_image["image"]
            image_extension = base_image["ext"]
```

```
            image_filename =
f"page{page_num+1}_img{img_index+1}.{image
_extension}"
            with open(image_filename, "wb") as
image_file:
                image_file.write(image_bytes)
            if img_index == 0:
                output_image_path = image_filename
                break
        if output_image_path:
            break
    return output_image_path

pdf_path = "/content/iris.pdf"
iris1_path = extract_image_from_pdf(pdf_path)

if iris1_path is None:
    print("No image extracted from the PDF.")
```

```
else:
    print(f"Image saved to: {iris1_path}")


def preprocess_image(image_path):
    image = cv2.imread(image_path,
cv2.IMREAD_GRAYSCALE)
    resized_image = cv2.resize(image, (240, 240))
    normalized_image =
cv2.equalizeHist(resized_image)
    return normalized_image


def localize_iris(image):
    edges = cv2.Canny(image, 50, 150)
    circles = cv2.HoughCircles(edges,
cv2.HOUGH_GRADIENT, dp=1, minDist=20,
                           param1=50, param2=30,
minRadius=20, maxRadius=80)
    if circles is not None:
```

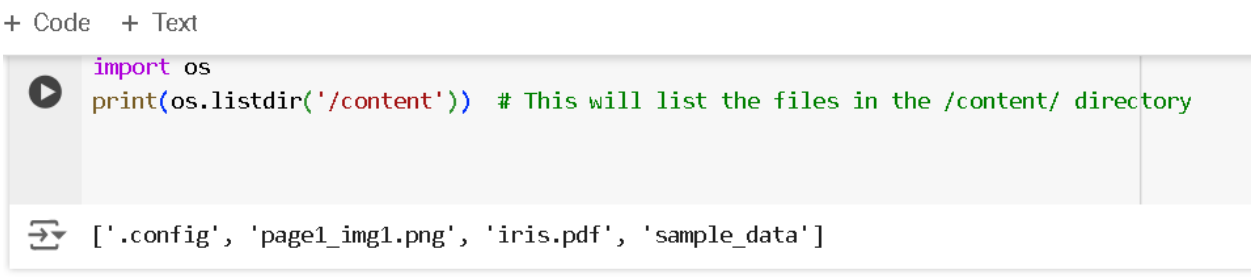
```
        circles = np.uint16(np.around(circles))
        iris_circle = circles[0][0]
        return iris_circle
    return None

def normalize_iris(image, circle):
    center_x, center_y, radius = circle
    polar_iris = cv2.linearPolar(image, (center_x, center_y), radius, cv2.WARP_FILL_OUTLIERS)
    return polar_iris

def gabor_filter(image, kernel_size=31, sigma=4.0, theta=0.0, lambd=10.0, gamma=0.5):
    kernel = cv2.getGaborKernel((kernel_size, kernel_size), sigma, theta, lambd, gamma, 0, ktype=cv2.CV_64F)
    filtered_image = cv2.filter2D(image, cv2.CV_8UC3, kernel)
```

```
    return filtered_image

def encode_iris(image):
    rows, cols = image.shape
    encoded = np.zeros((rows, cols),
dtype=np.uint8)
    gabor_kernels = [gabor_filter(image,
theta=np.pi * t / 4) for t in range(4)]
    for idx, kernel_image in
enumerate(gabor_kernels):
        encoded += ((kernel_image > 127) << idx)
    return encoded.flatten()

def match_iris(code1, code2):
    return hamming(code1, code2)

if __name__ == "__main__":
    iris1 = preprocess_image(iris1_path)
```

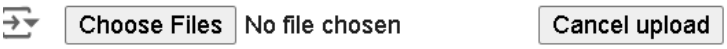
```
    circle1 = localize_iris(iris1)

    if circle1 is not None:
        normalized_iris1 = normalize_iris(iris1,
circle1)
        code1 = encode_iris(normalized_iris1)

        iris2_path = "/content/iris2.jpg"
        iris2 = preprocess_image(iris2_path)
        circle2 = localize_iris(iris2)

        if circle2 is not None:
            normalized_iris2 = normalize_iris(iris2,
circle2)
            code2 = encode_iris(normalized_iris2)

            distance = match_iris(code1, code2)
            print(f"Hamming Distance: {distance}")
```

```
            if distance < 0.3:
                print("Match Found!")
            else:
                print("No Match Found.")
        else:
            print("Iris localization failed for the second image.")
    else:
        print("Iris localization failed for the first image.")
```

Output:



```
        print("Iris localization failed for the first image.")
```



Description:

1. Code Breakdown:

1. **Library Installation**:
   The script installs the required Python libraries (`opencv-python`, `numpy`, `scipy`, `PyMuPDF`) that are necessary for image processing, numerical computations, and extracting images from PDF files.
2. **Extracting Image from PDF**:
   The function `extract_image_from_pdf()` opens a given PDF and extracts the first image from it. The image is saved locally in the `/content/` directory of Google Colab for further processing.
3. **Preprocessing the Image**:
   The `preprocess_image()` function reads the image, converts it to grayscale, resizes it to a standard size of 240x240 pixels, and improves

the contrast through histogram equalization to make features more prominent.

4. **Iris Localization**:
   The `localize_iris()` function identifies the iris and pupil boundaries using edge detection (via the Canny edge detector) and Hough Circle Transform to detect circular shapes within the image.
5. **Normalization of Iris**:
   The `normalize_iris()` function transforms the iris region from a Cartesian coordinate system into polar coordinates. This ensures that the iris image becomes invariant to variations in size and orientation, making it more consistent for further processing.
6. **Gabor Filter-Based Feature Encoding**:
   The `gabor_filter()` function applies Gabor filters to capture frequency-based features from the iris. The `encode_iris()` function uses multiple Gabor filters at various orientations to generate a robust and distinctive representation of the iris pattern.
7. **Iris Pattern Matching**:
   The `match_iris()` function compares two encoded iris patterns using **Hamming Distance**,

which quantifies how similar the two feature vectors are. A smaller Hamming distance indicates a higher degree of similarity between the irises.