

```

from keras.models import load_model
import numpy as np
import cv2
import os

# Load pre-trained FaceNet model
model = load_model('facenet_keras.h5')

# Function to compute embeddings
def get_embedding(model, face_pixels):
    # Scale pixel values
    face_pixels = face_pixels.astype('float32')
    mean, std = face_pixels.mean(), face_pixels.std()
    face_pixels = (face_pixels - mean) / std
    # Transform face into one sample
    sample = np.expand_dims(face_pixels, axis=0)
    # Get embedding
    embedding = model.predict(sample)
    return embedding[0]

def detect_face(image):
    # Load Haar Cascade for face detection
    face_cascade = cv2.CascadeClassifier(cv2.data.harcascades + 'haarcascade_frontalface_default.xml')
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
    face_images = []
    for (x, y, w, h) in faces:
        face = image[y:y+h, x:x+w]
        face = cv2.resize(face, (160, 160)) # FaceNet requires 160x160 inputs
        face_images.append(face)
    return faces, face_images

#Build a Database of Known Faces
# Example face database
face_database = {
    "person1": get_embedding(model, cv2.imread("/content/FACE_IMGAE.jpg")),
    "person2": get_embedding(model, cv2.imread("/content/face 2.jpg"))
}

# Compare embeddings using cosine similarity
from numpy.linalg import norm

def compare_faces(known_embedding, candidate_embedding, threshold=0.5):
    similarity = np.dot(known_embedding, candidate_embedding) / (norm(known_embedding) * norm(candidate_embedding))
    return similarity > threshold

#Blurriness Detection
import matplotlib.pyplot as plt
def detect_blurriness(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    laplacian_var = cv2.Laplacian(gray, cv2.CV_64F).var()
    return laplacian_var

# Test blurriness
image = cv2.imread("/content/FACE_IMGAE.jpg")
blurriness = detect_blurriness(image)
threshold = 100.0 # Adjust threshold based on requirements
plt.imshow(image)

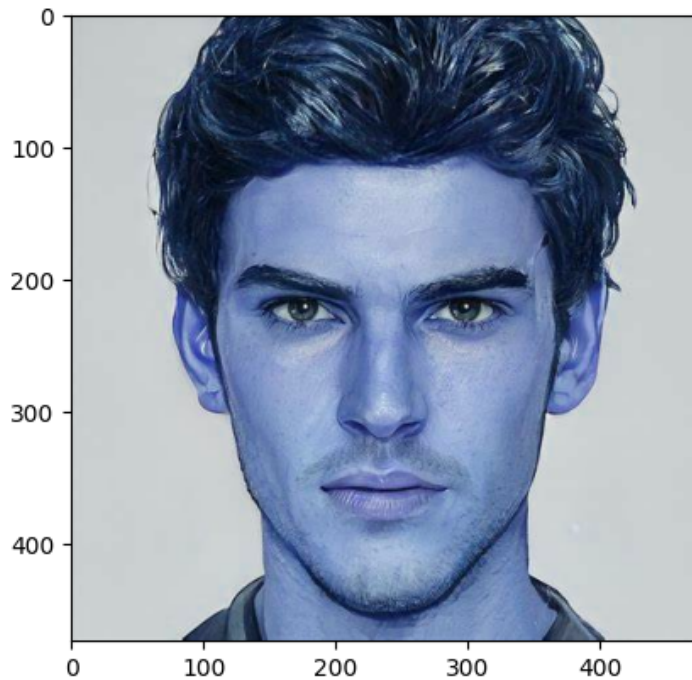
```

```

if blurriness < threshold:
    print("Image is blurry.")
else:
    print("Image is clear.")

```

⇒ Image is clear.



```

#Real-Time Recognition with Webcam
def recognize_face_from_webcam():
    cap = cv2.VideoCapture(0)

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        faces, face_images = detect_face(frame)

        for i, face_img in enumerate(face_images):
            embedding = get_embedding(model, face_img)
            recognized = False
            for name, db_embedding in face_database.items():
                if compare_faces(db_embedding, embedding):
                    recognized = True
                    # Draw a box and label around the face
                    x, y, w, h = faces[i]
                    cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
                    cv2.putText(frame, name, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
                    break

            if not recognized:
                x, y, w, h = faces[i]
                cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2)
                cv2.putText(frame, "Unknown", (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)

        cv2.imshow("Face Recognition", frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()

```

```

cv2.destroyAllWindows()

# Run the recognition
recognize_face_from_webcam()

def detect_faces(image):
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    return len(faces)

cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    if not ret:
        break
    face_count = detect_faces(frame)
    if face_count == 0:
        print("No face detected.")
    elif face_count > 1:
        print("Multiple faces detected.")
    else:
        print("Single face detected.")
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()

```

```

from cryptography.fernet import Fernet

```

```

# Generate and save a key
key = Fernet.generate_key()
cipher = Fernet(key)

```

```

def encrypt_data(data):
    encrypted = cipher.encrypt(data.encode())
    return encrypted

```

```


def decrypt_data(encrypted_data):
    decrypted = cipher.decrypt(encrypted_data).decode()
    return decrypted

```

```

# Example
response = "This is a test answer."
encrypted_response = encrypt_data(response)
print("Encrypted:", encrypted_response)
print("Decrypted:", decrypt_data(encrypted_response))

```

 Encrypted: b'gAAAAABnUnoev2n4ZT8cp_DiIWiv5khbT8IwrB6pGwJrmZsjSwdhrigiDwfT74SWyaMVkHBYXtL2JJ0eYeny8hTeb1
 Decrypted: This is a test answer.



Start coding or [generate](#) with AI.

