Dokumentation Gesichtserkennung

1. QR-Code-Erkennung

Die QR-Code-Erkennung findet in der KI-Engine (app.py) der Flask-Applikation statt. Es wurde die Python-Bibliothek „OpenCV“ verwendet. Mithilfe der Python-Bibliothek „OpenCV“ und ihrer Funktion *cv2.VideoCapture* wird eine Videoerfassung aus der Kamera gestartet. Dann wird mit ihrer Funktion *cv2.QRCodeDetector()* der QR-Code erkannt und decodiert. Anschließend wird das Ergebnis auf der URL der Web-Applikation wiedergegeben. Dafür wird die Funktion *parse2()* verwendet (siehe Listing 1).

```
  @app.route('/Qrcode')
def parse2():
    cap = cv2.VideoCapture(0)
    # initialize the cv2 QRCode detector

    detector = cv2.QRCodeDetector()

    while True:
        success, img = cap.read()
        # detect and decode
        data, bbox, _ = detector.detectAndDecode(img)
        # check if there is a QRCode in the image
        if data:
            verify = data
            print(verify)
            break
        if success:
            print('sucesss!!!')
            cv2.imshow("QRCODEscanner", img)

        if cv2.waitKey(1) == ord("q"):

            break

    cap.release()
    cv2.destroyAllWindows()

    return render_template('index.html', verify=verify)
```

Listing 1.: Funktion *parse2()*

2. Liveness-Detection und Gesichtserkennung

Die Liveness-Detection (erkennt, ob eine Person real ist) wurde anhand von vortrainierten Modellen, die erkennen können, ob es ein Video (basierend auf Lichtverhältnissen) oder ein Foto (basierend auf Augenbewegungen) ist, realisiert.

Die Gesichtserkennung wurde anhand von vortrainierten Modellen der Python-Bibliothek „OpenCV“ realisiert. Es handelt sich um einen auf Machine Learning basierenden Ansatz, bei dem eine Kaskadenfunktion aus einer Vielzahl positiver und negativer Bilder trainiert wird. Diese wird dann verwendet, um Gesichter in anderen Bildern zu erkennen.

Mithilfe der Python-Bibliothek „OpenCV“ und ihrer Funktion *cv2.VideoCapture* wird eine Videoerfassung aus der Kamera gestartet. Dann wird mit den vortrainierten Modellen der Liveness-Detection eine Wahrscheinlichkeit berechnet, welche angibt, ob die Person echt ist oder nicht. Ist diese Wahrscheinlichkeit kleiner als der Schwellenwert (*thresh = 0.92*) wird es als *„fake“* klassifiziert und in die Videodarstellung live-notiert. Ist diese Wahrscheinlichkeit größer als der Schwellenwert (*thresh = 0.92*) wird es als *„true“* klassifiziert und in die Videodarstellung live-notiert. Anschließend muss der Benutzer die Taste „q“ betätigen, damit dann die Summe der „fake“ und „true“ berechnet wird. Ist die Summe von „true“ kleiner als 10, wird die Berechnung abgebrochen und das Ergebnis wird als „nicht echte Person“ auf der URL der Web-Applikation wiedergegeben.
Ist die Summe von „true“ großer als 10, wird die Gesichtserkennung gestartet und die erkannten Gesichter im blauen Rahmen werden in die Videodarstellung live-notiert. Anschließend muss der Benutzer die Leertaste betätigen, um die Gesichter, die erkannt wurden, temporell lokal zu speichern. Abschließend wird das Ergebnis auf der URL der Web-Applikation wiedergegeben. Dafür wird die Funktion *parse1()* verwendet (siehe Listing 2).

```
  @app.route('/Gesichtserkennung')
def parse1():
    print('[DEBUG] call cv2.VideoCapture(0) from PID', os.getpid())
    print("[INFO] starting video stream...")
    vs = VideoStream(src=0).start()



    print('vs', vs)
    # vs = cv2.VideoCapture('1.mp4')
    time.sleep(1.0)

    while True:
        print('hier')

        frame = vs.read()


        #print('frame', frame)
        # frame = vs.read()[1]
        frame = imutils.resize(frame, width=600)
        #print('frame...', frame)
        # frame = imutils.rotate(frame, -90)
        (h, w) = frame.shape[:2]
        blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0,
```

```
(300, 300), (104.0, 177.0, 123.0))
        net.setInput(blob)

        detections = net.forward()
        print('detections', detections)

        for i in range(0, detections.shape[2]):

            confidence = detections[0, 0, i, 2]
            print('confidence', confidence)
            if confidence > 0.5:
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")
                sx = startX
                sy = startY
                ex = endX
                ey = endY

                ww = (endX - startX) // 10
                hh = (endY - startY) // 5

                startX = startX - ww
                startY = startY + hh
                endX = endX + ww
                endY = endY + hh

                startX = max(0, startX)
                startY = max(0, startY)
                endX = min(w, endX)
                endY = min(h, endY)

                x1 = int(startX)
                y1 = int(startY)
                x2 = int(endX)
                y2 = int(endY)

                roi = frame[y1:y2, x1:x2]
                gary_frame = cv2.cvtColor(roi, cv2.COLOR_RGB2GRAY)
                resize_mat = np.float32(gary_frame)
                m = np.zeros((40, 40))
                sd = np.zeros((40, 40))
                mean, std_dev = cv2.meanStdDev(resize_mat, m, sd)
                new_m = mean[0][0]
                new_sd = std_dev[0][0]
                new_frame = (resize_mat - new_m) / (0.000001 + new_sd)
                blob2 = cv2.dnn.blobFromImage(cv2.resize(new_frame, (40,
40)), 1.0, (40, 40), (0, 0, 0))
                net2.setInput(blob2)
                align = net2.forward()

                aligns = []
                alignss = []
                for i in range(0, 68):
                    align1 = []
                    x = align[0][2 * i] * (x2 - x1) + x1
                    y = align[0][2 * i + 1] * (y2 - y1) + y1
                    cv2.circle(frame, (int(x), int(y)), 1, (0, 0, 255), 2)
                    align1.append(int(x))
                    align1.append(int(y))
                    aligns.append(align1)
                cv2.rectangle(frame, (sx, sy), (ex, ey), (0, 0, 255), 2)
```

```
                aligns = aligns.append(aligns)

                ldmk = np.asarray(alignss, dtype=np.float32)
                ldmk = ldmk[np.argsort(np.std(ldmk[:, :, 1], axis=1))[-1]]
                img = crop_with_ldmk(frame, ldmk)

                attack_prob = demo(img)

                true_prob = 1 - attack_prob
                if attack_prob > thresh:
                    label = 'fake'
                    testfake.append(label)

                    cv2.putText(frame, label + ' :' + str(attack_prob), (sx,
sy - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                                (0, 0, 255), 2)
                else:
                    label = 'true'
                    testfake.append(label)

                    cv2.putText(frame, label + ' :' + str(true_prob), (sx,
sy - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                                (0, 0, 255), 2)


        #print('frame3', frame)
        print('cv2.imshow("Frame", frame)!!!!!!!!!!!!', cv2.imshow("Frame",
frame))

        cv2.imshow("Frame", frame)

        key = cv2.waitKey(20) & 0xFF
        if key == ord("q"):
            sfake = 'fake'
            strue = 'true'
            countfake = testfake.count(sfake)
            counttrue = testfake.count(strue)
            print('countfake', countfake)
            print('counttrue', counttrue)
            if (len(testfake) - countfake) < 10:
                verify = "Nicht echte Person"

                vs.stop()
                vs.stream.release()
                cv2.destroyAllWindows()

                break
            else:

                print('ich bin da')
                vs.stop()
                vs.stream.release()
                cv2.destroyAllWindows()

                verify = "Echte Person"
                print('richtige erkennung')
                ########
                ctr = 0
                # import face detection cascade
                face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

```
                # create capture object
                cap = cv2.VideoCapture(0)

                while True:
                    # capture frame-by-frame
                    ret, img1 = cap.read()
                    # convert image to grayscale
                    gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
                    faces = face_cascade.detectMultiScale(img1, 1.3, 5)
                    # for each face draw a rectangle around and copy the
face

                    for (x, y, w, h) in faces:
                        cv2.rectangle(img1, (x, y), (x + w, y + h), (255, 0,
0), 2)
                        roi_color = img1[y:y + h, x:x + w]
                        # print("[INFO] Object found. Saving locally.")
                        # cv2.imwrite(str(w) + str(h) + '_faces.jpg',
roi_color)
                    # display the resulting frame
                    cv2.imshow('frame', img1)

                    # key = cv2.waitKey(1)
                    key = cv2.waitKey(10) & 0xFF
                    if key == 32:  # if space key is pressed

                        status = cv2.imwrite(
                            'faces_detected.jpg',
                            img1)
                        # print("[INFO] Image faces_detected.jpg written to
filesystem: ", status)

                        image = cv2.imread(
                            'faces_detected.jpg')
                        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

                        faces = face_cascade.detectMultiScale(
                            gray,
                            scaleFactor=1.3,
                            minNeighbors=3,
                            minSize=(30, 30)
                        )

                        print("[INFO] Sind {0} Faces
gefunden.".format(len(faces)))

                        for (x, y, w, h) in faces:
                            cv2.rectangle(image, (x, y), (x + w, y + h), (0,
255, 0), 2)
                            roi_color = image[y:y + h, x:x + w]
                            print("[INFO] Object gefunden. Lokal
gespeichert.")
                            cv2.imwrite(
                                "cropped_face/face_" + str(
                                    w) + str(h) + ".jpg", roi_color)


                        cap.release()
                        cv2.destroyAllWindows()

                        break
```

```
                vs.stop()
                vs.stream.release()
                cap.release()
                cv2.destroyAllWindows()
                break

            #cap.release()
            #cv2.destroyAllWindows()


    # when everything done, release the capture
    cap.release()
    cv2.destroyAllWindows()


    #vs.stop()
    #vs.stream.release()

    return render_template('index.html', verify=verify)
```

Listing 2.: Funktion parse1()

3. Gesichtsvergleich

Der Gesichtsvergleich (Vergleich des Gesichtes mit dem biometrischen Bild auf dem Personalausweis) wurde mithilfe der Python-Bibliothek „Deepface“ realisiert.
Mit der Funktion *parse()* werden die Bilder (Gesicht und biometrisches Bild auf dem Personalausweis), die von der Funktion *parse1()* gespeichert wurden, miteinander auf Ähnlichkeit verglichen. Abschließend werden alle gespeicherten Bilder gelöscht und das Ergebnis wird auf der URL der Web-Applikation wiedergegeben. Dafür wird die Funktion *parse()* verwendet (siehe Listing 3).

```
@app.route('/Verifizierung')
def parse():
    image1 = 'cropped_face/' + os.listdir('cropped_face/')[0]
    print('image1', image1)
    verify1 = ''
    verify2 = ''
    verify3 = ''
    verify4 = ''
    try:
        image2 = 'cropped_face/' + os.listdir('cropped_face/')[1]
        resp = DeepFace.verify(image1, image2, enforce_detection=False)
        print(resp["verified"])
        if resp["verified"] == True:
            verify = 'Gleiche Person'
            print('Gleiche Person')
        else:
            verify = 'Nicht gleiche Person'
            print('Nicht gleiche Person')
        for f in os.listdir('cropped_face/'):
            os.remove(
                os.path.join('cropped_face/', f))
```

```
    except IndexError:
        verify = 'Bitte den Vorgang wiederholen! '
        verify1 = 'Achten Sie dabei auf folgende Punkte:'
        verify2 = '1. Positionieren Sie sich nicht zu weit von der Kamera
entfernt. '
        verify3 = '2. Es darf nur eine Person im Kamerabild sichtbar sein.'
        verify4 = '3. Halten Sie Ihren Personalausweis neben Ihr Gesicht.
Der Personalausweis darf das Gesicht nicht verdecken.'

        for f in os.listdir('cropped_face/'):
            os.remove(
                os.path.join('cropped_face/', f))

    os.remove(os.path.join('faces_detected.jpg'))

    #os._exit(0)


    return render_template('index.html', verify=verify, verify1=verify1,
verify2=verify2, verify3=verify3,
                           verify4=verify4)
```

Listing 3.: Funktion parse()