Paquete FaceRecognition

En este módulo se encuentran todos los métodos necesarios para aplicar el reconocimiento facial en video.

En esta clase se encuentran todos los métodos utilizados para el funcionamiento del analisis del video.

Método: **clase** identify_face_video.

parámetro global: **frame** contiene el frame de video manejado por la clase.

Métodos necesarios para el analisis del vídeo.

Método: **función** resultado(self,input_video).

Valores entrada: self, ruta

- **valor self:** None.
- **valor input_video:** Ruta o Ip de vídeo.

Lista de parámetros: parámetros variables.

- **threshold:** Umbral para que cada una de las redes [Pnet, Rnet, Onet] decida que la ventana facial es valida.
- **factor:** Valor con la que la imagen se va a reducir en cada interacción.
- **minsize:** Tamaño mínimo para detectar el rostro.
- **umbral:** Valor mínimo para marcar a la persona como desconocida.

Lista de métodos: métodos para el desarrollo del reconocimiento facial.

- **método cv2.VideoCapture:** Accede el vídeo a analizar.
- **método video_capture.read:** Carga la vídeo en el sistema.
- **método pickle.load:** Carga el modelo para realizar las prediciones SVC
- **método detect_face.detect_face:** Obtiene las bounding_boxes y los puntos de los rostros.
- **método model.predict_proba:** Obtiene las prediciones del modelo.

return frame: frame analizado procesado y listo para el envio a través de HTTP como tipo jpeg.

```
global frame
class identify_face_video:
    def resultado(self,input_video):
            #input_video="http://admin:admin@192.168.1.33:8081"#Ejemplo de Dir Ip
            modeldir = os.path.realpath(
                "./FaceRecognition/model/20170511-185253.pb")  # 'C:/Users/Corgito/Desktop/untitled/FaceRecognition/model/20170511-185253.pb'
            classifier_filename = os.path.realpath(
```

```
                "./FaceRecognition/class/classifier.pkl")  # 'C:/Users/Corgito/Desktop/untitled/FaceRecognition/class/classifier.pkl'
            npy = os.path.realpath(
```

```
        "./FaceRecognition/npy")  # 'C:/Users/Corgito/Desktop/untitled/FaceRecognition/npy'
    train_img = os.path.realpath(
        "./FaceRecognition/train_img")  # "C:/Users/Corgito/Desktop/untitled/FaceRecognition/train_img"
    with tf.Graph().as_default():
        gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.6)
        sess = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options, log_device_placement=False))
        with sess.as_default():
            pnet, rnet, onet = detect_face.create_mtcnn(sess, npy)

            minsize = 35  # minimum size of face
            threshold = [0.8, 0.8, 0.8]  # three steps's threshold
            factor = 0.709  # scale factor
            margin = 44
            frame_interval = 3
            batch_size = 1000
            image_size = 182
            input_image_size = 160
            umbral = 0.3
            HumanNames = os.listdir(train_img)
            HumanNames.sort()

            print('Loading Modal')
            facenet.load_model(modeldir)
            images_placeholder = tf.get_default_graph().get_tensor_by_name("input:0")
            embeddings = tf.get_default_graph().get_tensor_by_name("embeddings:0")
            phase_train_placeholder = tf.get_default_graph().get_tensor_by_name("phase_train:0")
            embedding_size = embeddings.get_shape()[1]


            classifier_filename_exp = os.path.expanduser(classifier_filename)
            with open(classifier_filename_exp, 'rb') as infile:
                (model, class_names) = pickle.load(infile)

            video_capture = cv2.VideoCapture(input_video)
            c = 0

            print('Start Recognition')
            prevTime = 0
            while True:
                ret, frame = video_capture.read()
                print(frame.shape[1])
                if frame.shape[1]>1000:
                    frame = cv2.resize(frame, (0, 0), fx=0.5, fy=0.5)  # resize frame (optional)

                curTime = time.time() + 1  # calc fps
                timeF = frame_interval

                if (c % timeF == 0):
                    find_results = []

                    if frame.ndim == 2:
                        frame = facenet.to_rgb(frame)
                    frame = frame[:, :, 0:3]
                    bounding_boxes, _ = detect_face.detect_face(frame, minsize, pnet, rnet, onet, threshold, factor)
                    nrof_faces = bounding_boxes.shape[0]
                    print('Detected_FaceNum: %d' % nrof_faces)
```

```
                    if nrof_faces > 0:
                        det = bounding_boxes[:, 0:4]
                        img_size = np.asarray(frame.shape)[0:2]

                        cropped = []
                        scaled = []
                        scaled_reshape = []
                        bb = np.zeros((nrof_faces, 4), dtype=np.int32)

                        for i in range(nrof_faces):
                            emb_array = np.zeros((1, embedding_size))

                            bb[i][0] = det[i][0]
                            bb[i][1] = det[i][1]
                            bb[i][2] = det[i][2]
                            bb[i][3] = det[i][3]

                        # inner exception
                            if bb[i][0] <= 0 or bb[i][1] <= 0 or bb[i][2] >= len(frame[0]) or bb[i][3] >= len(frame):
                                print('Face is very close!')
                                continue

                            cropped.append(frame[bb[i][1]:bb[i][3], bb[i][0]:bb[i][2], :])
```

```
                            cropped[i] = facenet.flip(cropped[i], False)
                            scaled.append(misc.imresize(cropped[i], (image_size, image_size), interp='bilinear'))
                            scaled[i] = cv2.resize(scaled[i], (input_image_size, input_image_size),
                                                   interpolation=cv2.INTER_CUBIC)
                            scaled[i] = facenet.prewhiten(scaled[i])
                            scaled_reshape.append(scaled[i].reshape(-1, input_image_size, input_image_size, 3))
                            feed_dict = {images_placeholder: scaled_reshape[i], phase_train_placeholder: False}
                            emb_array[0, :] = sess.run(embeddings, feed_dict=feed_dict)
                            predictions = model.predict_proba(emb_array)
```

```
                            print(predictions)
                            best_class_indices = np.argmax(predictions, axis=1)
                            best_class_probabilities = predictions[np.arange(len(best_class_indices)), best_class_indices]
```

```
                    # print("predictions")
                            print(best_class_indices, ' with accuracy ', best_class_probabilities)

                    # print(best_class_probabilities)
                            if best_class_probabilities > 0.1:
                                cv2.rectangle(frame, (bb[i][0], bb[i][1]), (bb[i][2], bb[i][3]), (0, 255, 0),
                                    2)  # boxing face

                                # plot result idx under box
                                    text_x = bb[i][0]
                                    text_y = bb[i][3] + 20
                                    for H_i in HumanNames:
                                        if best_class_probabilities < umbral:
                                            result_names = "Desconocido"
                                            cv2.putText(frame, result_names+str(i), (text_x, text_y), cv2.FONT_HERSHEY_COMPLEX_SMALL,
                                                1, (0, 0, 255), thickness=1, lineType=2)
                                        else: # HumanNames[best_class_indices[0]] == H_i:
                                                result_names = HumanNames[best_class_indices[0]]
                                                cv2.putText(frame, result_names, (text_x, text_y), cv2.FONT_HERSHEY_COMPLEX_SMALL,
                                                    1, (0, 0, 255), thickness=1, lineType=2)


                    else:
                        print('Alignment Failure')
            # c+=1
                    #cv2.imshow('Video', frame)


                    evento, cuadro = cv2.imencode('.jpg', frame)
                    ventana=cuadro.tobytes()

                    yield (b'--frame\r\n'
                        b'Content-Type: image/jpeg\r\n\r\n' + ventana + b'\r\n\r\n')


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

                video_capture.release()
                cv2.destroyAllWindows()
```