Paquete FaceRecognition

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

Clase para aplicar el reconocimiento facial en imagen.

Método: **clase** identify_face_image.

Método para aplicar reconocimiento facial a la imagen.

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

Valores entrada: self, ruta

- **valor self:** None.
- **valor ruta:** Ruta del archivo de imagen.

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.imread:** Carga la imagen 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 result_name: Devuelve el nombre de la persona reconocida.

```
class identify_face_image:
    def resultado(self,ruta):
        result_names="No se detecto ninguna cara"
        img_path = ruta
        modeldir = os.path.realpath("./FaceRecognition/model/20170511-185253.pb")
        classifier_filename = os.path.realpath("./FaceRecognition/class/classifier.pkl")
        npy = os.path.realpath("./FaceRecognition/npy")
        train_img = os.path.realpath("./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)
                image = cv2.imread(img_path)
                umbral = 0.35
                h,w,_=image.shape
                minsize = 8  # minimum size of face
                threshold = [0.7, 0.7, 0.7]  # three steps's threshold
```

```
                factor = 0.709  # scale factor
                margin = 20
                frame_interval = 3
                batch_size = 1000
                image_size = 182
                input_image_size = 160

                HumanNames = os.listdir(train_img)
                HumanNames.sort()

                print('Loading feature extraction model')
                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)

                c = 0

                print('Start Recognition!')
                prevTime = 0
                frame = cv2.imread(img_path, 0)


                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('Face Detected: %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 too 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)
                            # print(best_class_indices)
                            best_class_probabilities = predictions[np.arange(len(best_class_indices)), best_class_indices]
                            print(best_class_probabilities)
                            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
                            print('Result Indices: ', best_class_indices[0])
                            print(HumanNames)
                            print("Mejor Indicie",best_class_indices[0])
                            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:
                                for H_i in HumanNames:

                                    if HumanNames[best_class_indices[0]] == H_i:
                                        result_names = HumanNames[best_class_indices[0]]# + str(i)
                                        cv2.putText(frame, result_names, (text_x, text_y), cv2.FONT_HERSHEY_COMPLEX_SMALL,
                                        1, (0, 0, 255), thickness=1, lineType=2)

                            print("La cara",i,"pertene a",result_names,"con un indice de probabilidad de",best_class_probabilities)
                    else:
                        print('Unable to align')
                #dim=int(frame.shape[1]-200),int(frame.shape[0]-200)#height,width
                #frame = cv2.resize(frame, dim, fx=0.5, fy=0.5)  #el resize es opcional, pero si se hace mejor despues de la deteccion ya que si no perdemos información
                #cv2.imshow('Image', frame)
                cv2.imwrite(ruta, frame)
                return result_names

                if cv2.waitKey(1000000) & 0xFF == ord('q'):
                    cv2.destroyAllWindows()
                    #sys.exit("Thanks")
                cv2.destroyAllWindows()
```