嵌入式前端人脸识别 SDK

使用手册

2018-11-15

OPEN AI LAB

变更记录

(Reversion Record)

日期	版本	说明	作者
2018-08-03	V1.0.0	第一个版本	Joey
2018-08-03	V1.1.0	加入cortex A7数据	张慧芳
2018-08-27	V1.2.0	加入linux接口描述	Joey
2018-09-07	V1.3.0	更新Android部分接口描述	吴猛
2018-09-14	V1.4.0	增加活体检测和人脸质量获取接口描述	Joey
2018-09-17	V1.5.0	增加活体检测和人脸质量Android接口描述	吴猛
2018-09-26	V1.6.0	文档新结构	吴猛
2018-09-27	V1.6.1	增加linux部分	Joey
2018-10-10	V1.7.0	增加SDK人脸数据库	Joey
2018-10-16	V1.7.1	增加获取多人脸API	Joey
2018-10-19	V1.7.2	Android新接口	吴猛
2018-10-22	V1.7.3	增加单识别人脸API	吴猛

2018-10-26	V1.7.4	增加android鉴权API	郑礼广
2018-10-31	V1.7.5	增加API新接口描述	吴猛
2018-11-02	V1.7.6	SDK版本获取接口描述	吴猛
2018-11-02	V1.7.7	增加获取android鉴权激活状态值	郑礼广
2018-11-15	V1.7.8	增加获取人脸属性	Joey
2018-11-19	V1.7.9	Android人脸属性接口和内部数据库操作接口	吴猛
2018-11-20	V1.7.10	Android增加faceinfo，替代原来用的byte[]posStr	崔鹏
2019-3-14	V1.7.11	增加Android的DelDB和DelAllDB接口，增加Linux的DelAllDB接口	崔鹏

目录

1 SDK 简介

本 SDK 开发指南指导您如何安装和配置开发环境，如何通过调用 SDK 提供的接口函 数(API)进行二次开发与系统集成。用户按照要求调用 SDK 提供的 API 即可实现使用 人脸检测/跟踪、活体识别、人脸识别等服务的目的。

文档适用对象：

- 软件工程师
- 硬件工程师
- 技术支持工程师

Android SDK 目录结构如下：

```
|   3399-sdkdemo.v0.2.0.apk            //可执行文件
|   FaceAPP.zip                        //demo 工程文件
|   mlkit.facerec.64bit.v0.2.0.aar     //SDK 库文件
```

FaceAPP 文件列表

```
└─FaceAPP
    |   .gitignore
    |   build.gradle
    |   gradle.properties
    |   gradlew
    |   gradlew.bat
    |   settings.gradle
    |
    ├─.idea
    |   |   compiler.xml
    |   |   gradle.xml
    |   |   misc.xml
    |   |   modules.xml
    |   |   runConfigurations.xml
    |   |
    |   └─copyright
    |           profiles_settings.xml
    |
    ├─app
    |   |   .gitignore
    |   |   build.gradle
    |   |   proguard-rules.pro
    |   |
    |   ├─libs
    |   |       facelibrary-release.aar   //算法 SDK 包
    |   |       openCVLibrary331-release.aar //OpenCV Android 下显示控件
```

```
    │   │
    │   └─src
    │       ├─androidTest
    │       │   └─java
    │       │       └─com
    │       │           └─openailab
    │       │               └─sdkdemo
    │       │                       ExampleInstrumentedTest.java
    │       │
    │       ├─main
    │       │   │  AndroidManifest.xml
    │       │   │
    │       │   ├─java
    │       │   │   └─com
    │       │   │       └─openailab
    │       │   │           └─sdkdemo
    │       │   │                   MainActivity.java  //主要显示界面
    │       │   │                   mixController.java //控制状态变化
    │       │   │                   myDrawRectView.java
    │       │   │                   PermissionsDelegate.java //申请权限
    │       │   │                   FileOperator.java //批量注册时对存取数据文件操作
    │       │   │                   GetSDTreeActivity.java//批量注册时浏览选择目录
    │       │   └─res
    │       │       ├─layout
    │       │       │       activity_main.xml
    │       │       │
    │       │       ├─mipmap-hdpi
    │       │       │       ic_launcher.png
    │       │       │
    │       │       ├─mipmap-mdpi
    │       │       │       ic_launcher.png
    │       │       │
    │       │       ├─mipmap-xhdpi
    │       │       │       ic_launcher.png
    │       │       │
    │       │       ├─mipmap-xxhdpi
    │       │       │       ic_launcher.png
    │       │       │
    │       │       ├─mipmap-xxxhdpi
    │       │       │       ic_launcher.png
    │       │       │
    │       │       ├─values
    │       │       │       colors.xml
    │       │       │       dimens.xml
    │       │       │       strings.xml
    │       │       │       styles.xml
    │       │       │
    │       │       └─values-w820dp
    │       │               dimens.xml
    │       │
    │       └─test
```

```
    │           └─java
    │               └─com
    │                   └─openailab
    │                       └─sdkdemo
    │                               ExampleUnitTest.java
    │
    └─gradle
        └─wrapper
                gradle-wrapper.jar
                gradle-wrapper.properties
```

Linux SDK 目录结构如下：

```
sample/
├── Makefile
├── README
├── demo.conf
├── include
│   └── faceapp.hpp
├── libs
│   ├── 32bit
│   │   ├── libface.so
│   │   └── libtengine.so
│   └── 64bit
│       ├── libface.so
│       └── libtengine.so
├── models
│   ├── det1.tmfile
│   ├── det2.tmfile
│   ├── det3.tmfile
│   └── mfn.tmfile
├── run.sh
└── src
    ├── AlgThread.cpp
    ├── AlgThread.h
    ├── cameraThread.h
    ├── config.cpp
    ├── config.h
    ├── lock.cpp
    ├── lock.h
    ├── main.cpp
    ├── thread.cpp
    ├── thread.h
    ├── util.cpp
    └── util.h
```

2 开发前准备

A. 获取 SDK

- 请联系 OPEN AI LAB 商务处获取，邮箱：business@openailab.com
- 正式版 SDK，申请前都需提供应用的包名

B. 激活 SDK

- 若您获得是测试版 SDK，无需激活。

C. 运行环境

- 推荐内存：1G 以上
- 推荐 CPU：1.2GHz 以上

Arm CPU架构类型	CPU 类型	操作系统
Arm v7	A7	Android5.1或以上 Linux, gcc 4.9或以上
	A17	Android5.1或以上 Linux, gcc 4.9或以上
Arm v8	A72	Android5.1或以上 Linux, gcc 4.9或以上
	A53	Android5.1或以上 Linux, gcc 4.9或以上

D. 调用关系

用户的应用程序调用 OPEN AI LAB SDK 提供的 API，就可以直接使用人脸检测、识别 等功能。

3 Android 快速入门

A. 添加依赖库

复制 facelibrary-release.aar openCVLibrary331-release.aar 到工程下 app/libs 目录。

B 添加编译 aar 信息

```
compile(name: 'facelibrary-release', ext: 'aar')
compile(name: 'openCVLibrary331-release', ext: 'aar')
```

到 build.gradle 的 dependencies

C 在程序中定义 FaceAPP 变量

```
private  FaceAPP face= FaceAPP.GetInstance(); //face 作为成员变量
```

D 通过 face 调用 Android API 接口

4 Linux 快速入门

A. 编译例子代码

```
cd sample
make
```

B. 运行例子代码

```
./run.sh
```

5 Android API 描述

```
public final static int SUCCESS=0;  //执行接口返回成功
public final static int  ERROR_INVALID_PARAM=-1; //非法参数
public final static int ERROR_TOO_MANY_REQUESTS=-2;//太多请求
public final static int ERROR_NOT_EXIST=-3;//不存在
public final static int ERROR_FAILURE=-4; // 执行接口返回失败
```

5.1 构造函数

static FaceAPP GetInstance()
功能 获取单例的对象，人脸识别类采用单例模式，一个类Class只有一个实例存在

参数 无

返回值 FaceAPP类型的对象

```
private  FaceAPP face= FaceAPP.GetInstance();
```

5.2 识别人脸特征

int Recognize(Image image, float featureArray[][128], int size, List<FaceInfo> faceinfos, int[] res)

功能 识别提交的Image中的人脸特征，然后和featureArray里这些特征数组进行比较，找出其中相似度最高的返回特征数组的二维数组的索引值。

参数 image： 人脸图片

featureArray：特征数组的二维数组，特征值数组是存储人脸特征信息的数组，由128个float组成。

size： 特征数组的二维数组大小

faceinfos： FaceInfo清单。<用于保存返回人脸在图片中的坐标信息>

res: 执行结果，

未发现人脸返回*ERROR_INVALID_PARAM*,

image中人脸不在feature数组中返回*ERROR_NOT_EXIST*， >=0特征数组的索引号

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
float[][] featurelist=new float[][]; //存储特征值的数组
int size= featurelist.lenth;
```

```
int[] ret=new int[1];
byte[] tmpPos = new byte[1024];

FaceAPP.Image image= FaceAPP.GetInstance().new Image();
image.matAddrframe=mRgbaFrame.getNativeObjAddr();
face.Recognize(image,featurelist,size, tmpPos,res);
```

5.3 识别人脸特征（根据特征值）

int Recognize(float[] feature, float featureArray[][128], int size, float[] high, int[] res)	
功能	根据已经存在的人脸特征，然后和featureArray里这些特征数组进行比较，找出其中相似度最高的返回特征数组的二维数组的索引值,返回相似度得分值。
参数	feature： 特征数组 featureArray：特征数组的二维数组，特征值数组是存储人脸特征信息的数组，由128个float组成。 size： 特征数组的二维数组大小 high： float[]型，返回最大得分值 res: 执行结果， 未发现人脸返回*ERROR_INVALID_PARAM*， image中人脸不在feature数组中返回*ERROR_NOT_EXIST*， >=0特征数组的索引号
返回值	执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
float[][] featurelist=new float[][]; //存储特征值的数组
int size= featurelist.lenth;
int[] ret=new int[1];
byte[] tmpPos = new byte[1024];
float[] feature;
float[] high=float[1];
FaceAPP.Image image= FaceAPP.GetInstance().new Image();
image.matAddrframe=mRgbaFrame.getNativeObjAddr();
face.Recognize(feature,featurelist , size, high, tmpPos , res);
```

5.4 检测人脸

int Detect(Image image，List<FaceInfo> faceinfos，int[] res)

功能 检测提交的图片中的是否有人脸

参数 image： 人脸图片，用于检测的图片

faceinfos： FaceInfo清单。<用于保存返回人脸在图片中的坐标信息>

res: 执行结果，未发现人脸返回*ERROR_INVALID_PARAM*

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

用法

```
int[] ret=new int[1];
byte[] tmpPos = new byte[1024];//byte数组用于存位置信息
FaceAPP.Image image= FaceAPP.GetInstance().new Image(); //初始化
```

```
image.matAddrframe=mRgbaFrame.getNativeObjAddr();//image赋值
if(success=face.Detect(image,tmpPos,res)){
//to do
};
```

5.5 比较特征数据

int Compare(float[] origin, float[] chose, float score)

功能 用于比较两个feature值相似度

参数 origin：待比较feature数组

chose：用于比较的feature数组

score ： origin和chose比较的相似度

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
Float score
float[] origin=new float[128];
Float[] chose=new float[128];
face.Compare(origin,chose,score);
```

5.6 双目活体

int GetFeature(Image image,Image grayImage,float[] feature，List<FaceInfo>

faceinfos , int[] res)

功能 获取image中的人脸特征值数组，特征值数组是存储人脸特征信息的数组，由128个float型数字组成，只获取图片中一个人的特征，多于一人会返回错误信息。

参数 image： 人脸图片，用于检测的图片

grayImage：红外摄像头获取的图片

feature：存储image中检测到的人脸特征信息，无人脸返回空数组

faceinfos： FaceInfo清单。<用于保存返回人脸在图片中的坐标信息>

res: 执行结果，未发现人脸返回*ERROR_INVALID_PARAM*

返回值 *SUCCESS*：正确获取人脸信息

ERROR_FAILURE：未获取到人脸信息

```
float[] feature=new float[128];
int[] ret=new int[1];
byte[] tmpPos = new byte[1024];//byte数组用于存位置信息
ret= face.GetFeature(image,grayImage,feature,tmpPos,res);
if(ret== SUCCESS){
//to do 成功获取到活体人脸特征值
}
```

5.7 双目校准

Int Calibration(Image image,Image grayImage,float[] scale,int[] Rect,int[] res);

功能 对红外和普通光组成的双摄像头识别进行校准，要求1个人在最佳位置（0.8-1米）

	站定，大约需要校验20次，得到人脸框修正参数用于人脸画框，返回红外摄像头相对普通光的显示区域坐标，该区域是有效识别和活体检测区域
参数	image：输入红外图像 grayImage：输入彩色图像 scale：输出 人脸框修正参数 rect：输出红外图像与彩色图像重叠区域（红外图像在彩色图像的对应区域/推荐的检测区域） res: 执行结果，未发现人脸返回*ERROR_INVALID_PARAM*
返回值	*SUCCESS*：正确校准 *ERROR_FAILURE*：校准失败

```
int[] ret=new int[1];
float[] scale=new float[1];
int[] rect=new int[4];
ret= face.Calibration(image,grayimage,scale,rect,res);
if(ret== SUCCESS){
//to do 校准成功
}
```

5.8 设备鉴权

int AuthorizedDevice(String uidStr，String password，Context activity)	
功能	设备鉴权

参数 uidStr : OEMID号+合同号

password: 用户密码

返回值 0：鉴权成功

```
String oem_id ="1000000000000001";//OEMID
String contract_id ="0001";//合同号
String password = "0123456789abcdef0123456789abcdef";//初始授权密码
String uidStr = oem_id+contract_id;
int res =face.AuthorizedDevice(uidStr,password, LoginActivity.this);
```

5.9 获取鉴权激活状态值

int getAuthStatus()

功能 获取鉴权激活状态值

参数

返回值 0：鉴权成功

```
//获取鉴权激活状态值，0为鉴权成功
public int getAuthStatus(){
    return authStatus;
}
```

5.10 提取人脸特征

int GetFeature(Image image, float[] feature， List<FaceInfo> faceinfos , int[] res)

功能	获取image中的人脸特征值数组，特征值数组是存储人脸特征信息的数组，由128个float型数字组成，只获取图片中一个人的特征。
参数	image： 人脸图片，用于检测的图片 feature：存储image中检测到的人脸特征信息，无人脸返回空数组 faceinfos： FaceInfo清单。<用于保存返回人脸在图片中的坐标信息> res: 执行结果，未发现人脸返回*ERROR_INVALID_PARAM*
返回值	*SUCCESS*：正确获取人脸信息 *ERROR_FAILURE*：未获取到人脸信息

```
float[] feature=new float[128];
int[] ret=new int[1];
byte[] tmpPos = new byte[1024];//byte数组用于存位置信息
ret= face.GetFeature(image,feature,tmpPos,res);
if(ret== SUCCESS){
//to do 成功获取到人脸特征值
}
```

5.11 提取人脸特征（根据人脸坐标信息）

int GetFeature(Image image, FaceInfo detectInfo,float[] feature, int[] res)	
功能	根据传入的人脸坐标和关键点坐标信息，获取image中的人脸特征值数组，特征值数组是存储人脸特征信息的数组，由128个float型数字组成，只获取图片中一个人的特征。

参数	image： 人脸图片，用于检测的图片 detectInfo：检测到的人脸信息，用于人脸特征提取 feature：存储image中检测到的人脸特征信息，无人脸返回空数组 res: 执行结果，未发现人脸返回*ERROR_INVALID_PARAM*
返回值	*SUCCESS*：正确获取人脸信息 *ERROR_FAILURE*：未获取到人脸信息

```
float[] feature=new float[128];
int[] ret=new int[1];
float[] detectinfo=new
float[]{x0,y0,x1,y1,landmarkx0,landmarky0,landmarkx1,landmarky1,
landmarkx2,landmarky2,landmarkx3,landmarky3, landmarkx4,landmarky4}
byte[] tmpPos = new byte[1024];//byte数组用于存位置信息
ret= face.GetFeature(image,detectinfo,feature,res);
if(ret== SUCCESS){
//to do 成功获取到人脸特征值
}
```

5.12 提取人脸关键点

int GetLandmark(Image image， float[] landmark，int[] res)	
功能	获取人脸关键点坐标信息
参数	Image：人脸图片，用于提取人脸关键点信息的图片，只提取一个人的关键点信息 Landmark：用于存储5个关键点坐标值，依次是左眼，右眼，鼻子，左侧嘴唇，右

	侧嘴唇 res: 执行结果，未发现人脸返回*ERROR_INVALID_PARAM*
返回值	*SUCCESS*：正确获取人脸关键点坐标信息 *ERROR_FAILURE*：未获取到人脸关键点信息

```
float[] landmark=new float[10];
int[] ret=new int[1];
ret=face.GetLandmark(image,landmark,res);
if(ret== SUCCESS){
    }
```

5.13 参数设置

bool SetParameter(const String[] name, float value[])

功能	设置输入的参数名和对应数值
参数	char[] name： 参数的名字 a： a参数值 内部参数，按示例设置，请不要随意修改 b： b参数值 内部参数，按示例设置，请不要随意修改 c： c参数值 内部参数，按示例设置，请不要随意修改 d： d参数值 内部参数，按示例设置，请不要随意修改 factor： 检测人脸放大比例 内部参数，按示例设置，请不要随意修改 min_size： 最小人脸框大小 范围 32-80

faceclarity： 照片清晰度阈值 范围 建议200-400

perfoptimize: 是否优化效果 范围 0或1

livenessdetect:是否活体检测 范围0-1

gray2colorscale:双目活体检测比值 范围 0.1-0.5

frame_num:优化的帧数，范围20-40

quality_thresh:图片质量阀值，建议范围0.7-0.8

mode:工作模式 0 闸机 1 门禁

facenum:检测最大人脸数，最多支持检测3张人脸识别1张脸，范围1-3

value[]： 参数的数值（可能多个）

返回值 参数设置是否成功

```
String[]
name={"a","b","c","d","factor","min_size","clarity","perfoptimize","livenessdetect","gray2colorscale","frame_num","quality_thresh","mode","facenum"};
double[] value={0.9,0.9,0.9,0.715,0.6,64,400,1,0,0.5, 20, 0.8, 1,1};
face.SetParameter(name,value);
```

5.14 参数设置（参数可变长度）

bool SetParameters(const String[] name, float value[])

功能 可变长度的设置输入的参数名和对应数值

参数 char[] name： 参数的名字（>=1）,详情见5.11,

value[]： 参数的数值（>=1个）

返回值 参数设置是否成功

```
String[]
name={"perfoptimize","livenessdetect","frame_num","quality_thresh","mode",
"facenum"};
double[] value={1,0,20, 0.8, 1,1};
face.SetParameter(name,value);
```

5.15 获取版本信息

public String GetVersion()

功能 获取当前SDK版本的信息

参数 无

返回值 返回当前版本信息的字符串

```
Face.GetVersion();
```

5.16 获取底层库信息

public String GetFacelibVersion()

功能 获取当前SDK底层库的信息

参数 无

返回值 返回当前版本底层库的字符串

```
Face.GetFacelibVersion();
```

5.17 打开人脸数据库

int OpenDB()

功能 使用人脸数据库,内部人脸数据库可实现1：N高效快速查询

参数 无

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
if(face.OpenDB () == SUCCESS){
// TODO
}
```

5.18 注册人脸

int AddDB(float[] feature, string name)

功能 注册人脸

参数 feature：人脸特征

name：登记名字

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
String name= "test";
int[]  res=new int[];
if(Face.GetFeature(image,feature,tmpPos,res);
==SUCCESS){
   Face.AddDB (feature,name);
}
```

5.19 删除人脸

int DelDB(string name)

功能 删除人脸

参数 name: 要删除的名字

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
String name= "test";
Face.DelDB (name);
```

5.20 删除所有已注册人脸

int DelAllDB()

功能 删除所有已注册人脸

参数 无

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
Face.DelAllDB ();
```

5.21 查询人脸特征对应名字

String QueryDB(float[] feature,float [] score)

功能 给定人脸特征最接近的数据库所登记的人脸，并给出相似度

参数 feature: 人脸特征

score: 相似度分值

返回值 执行成功 返回登记名字

执行失败 返回为unknown

```
float[] score=new float[1];
String name=Face.QueryDB(feature,score);
if(score>thresh_hold){


}
// TODO
}
```

5.22 关闭人脸数据库

int CloseDB()

功能 关闭人脸数据库

参数

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
if(Face.CloseDB () == SUCCESS){
// TODO
}
```

5.23 获取人脸属性

int GetFaceAttr(Image image,FaceInfo data,FaceAttribute face_attr,int *res)

功能 Detect后执行，通过检测获取的人脸位置和关键点信息，获取人脸属性包括年龄、性别和表情。

参数 image： 人脸图片，用于检测的图片

data：检测得到的人脸位置和人脸关键点信息

face_attr : 存储计算得到的人脸属性

res: 执行结果，未发现人脸返回*ERROR_INVALID_PARAM*

返回值 *SUCCESS*：正确获取人脸属性

ERROR_FAILURE：未获取到人脸属性

```
if(faceapp::FetchFaceAttr(Image,data,attr,&res) == SUCCESS){


// TODO
}
```

5.24 释放人脸识别资源

public void Destroy()

功能 释放初始化和设置参数时分配的资源

参数 无

返回值 无

```
Face.Destroy();
```

5.25 示例代码

```
 private  FaceAPP face= FaceAPP.GetInstance(); //face 作为成员变量
 public class MainActivity extends Activity implements CvCameraViewListener2 {
 ........
 @Override
protected void onCreate(Bundle savedInstanceState) {
........
String[] name={"a","b","c","d","factor","min_size","clarity","perf-
optimize","liveness-detect","gray2color-scale"};
double[] value={0.9,0.9,0.9,0.715,0.6,64,400,1,0,0.5};
face.SetParameter(name,value);
mainLoop = new Thread() { //人脸检测不要放在 Android 主线程
    public void run() {
........
float[] feature=new float[128];
byte[] tmpPos = new byte[1024];// byte 数组用于存位置信息
```

```
switch (mixController.curState){
......
case mixController.STATE_IDLE:

    FaceAPP.Image image= FaceAPP.GetInstance().new Image();
    image.matAddrframe=mRgbaFrame.getNativeObjAddr();
    int[] res=new int[1];
int ret;
ret= face.GetFeature(image,feature,tmpPos,res);
if(ret== SUCCESS){
//to do 成功获取到人脸特征值
}
}
```

6 Linux/Windows API 描述

```
#define SUCCESS 0  //执行接口返回成功
#define ERROR_INVALID_PARAM -1 //非法参数
#define ERROR_TOO_MANY_REQUESTS -2 //太多请求
#define ERROR_NOT_EXIST -3 //不存在
#define ERROR_FAILURE -4 // 执行接口返回失败
```

Linux/Windows SDK API 的命名空间(namespace)为 faceapp。

6.1 实例创建函数

FaceAPP faceapp::CreateFaceAPPInstance()	
功能	创建人脸检测和识别实例
参数	无
返回值	FaceAPP类型的对象

```
FaceAPP face= faceapp::CreateFaceAPPInstance();
```

6.2 检测识别人脸特征

int DetectRecognize(FaceAPP face,Image image, int policy, int *res)

功能 识别提交的Image中的人脸特征，作人脸检测和识别。

参数 face : FaceAPP实例

image： 人脸图片

policy： 0-实时检测 1-性能优先

res: 执行结果，

执行失败返回失败原因

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
if(faceapp::Recognize(face,Image(&frame),&res)==SUCCESS){
// TODO
}
```

6.3 检测识别人脸特征（双目活体识别）

int DetectRecognize(FaceAPP face,Image image,Image grayimage, int policy, int *res)

功能 识别提交的Image中的人脸特征，作人脸检测和识别。

参数 face : FaceAPP实例

image： 人脸图片

grayimage: 红外人脸图片（双目活体识别）

policy： 0-实时检测 1-性能优先

	res: 执行结果， 执行失败返回失败原因
返回值	执行成功 *SUCCESS* 执行失败 *ERROR_FAILURE*

```
if(faceapp::Recognize(face,Image(&frame),Image(&gray),&res)==
SUCCESS){
// TODO
}
```

6.4 检测人脸

int Detect(FaceAPP face,Image image，int* res)	
功能	检测提交的图片中的是否有人脸。只检测一个人的情况，多于一人会返回错误信息。只作检测不作识别。
参数	face : FaceAPP实例 image： 人脸图片，用于检测的图片 res: 执行结果， 未发现人脸返回*ERROR_INVALID_PARAM*， image中图片数多于1人返回*ERROR_TOO_MANY_REQUESTS*
返回值	执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

用法

```
if(faceapp::Detect(face,Image(&frame),&res)==SUCCESS){
// TODO
};
```

6.5 识别人脸

int Recognize(FaceAPP face,Image image,FaceDetctData* data,int *res)

功能 识别提交的图片中的人脸。

参数 face : FaceAPP实例

image： 人脸图片，用于检测的图片

data： 检测得到的人脸位置和人脸关键点信息

res: 执行结果，

执行失败返回失败状态，

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

用法

```
if(faceapp::Recognize(mFace,faceapp::Image(&mat),&data,&res)
==SUCCESS){
// TODO
};
```

6.6 比较特征数据

int Compare(FaceAPP face,float* origin, float* chose, float* score)

功能 用于比较两个feature值相似度

参数 face : FaceAPP实例

origin：待比较feature数组

chose：用于比较的feature数组

score ： origin和chose比较的相似度

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
float score;
if(faceapp::Compare(face,origin,chose,&score)== SUCCESS){
// TODO
}
```

6.7 打开人脸数据库

int FaceOpenDB(FaceAPP face)

功能 使用人脸数据库

参数 face : FaceAPP实例

返回值 执行成功 *注册人脸的总数*

执行失败 *ERROR_FAILURE*

```
if(faceapp:: FaceOpenDB (face) == SUCCESS){
// TODO
}
```

6.8 注册人脸

int FaceAddDB(FaceAPP face, float* feature, const char *name)

功能 注册人脸

参数 face : FaceAPP实例

feature：人脸特征

name: 登记名字

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
char *name= "test";
int res;
if(faceapp::GetFeature(mFace,feature,&res)==SUCCESS){
   faceapp::FaceAddDB (mFace,feature,name);
}
```

6.9 删除注册人脸

int FaceDelDB(FaceAPP face, const char *name)

功能 删除注册人脸

参数 face : FaceAPP实例

name: 登记名字

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
char *name= "test";
if(faceapp::FaceDelDB (mFace,name) == SUCCESS){
// TODO
}
```

6.10 删除所有已注册人脸

int FaceDelAllDB(FaceAPP face)

删除注册人脸

face : FaceAPP实例

执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
faceapp::FaceDelAllDB (mFace);
```

6.11 查询人脸特征对应名字

int FaceQueryDB(FaceAPP face,float* feature,float *score,const char **name)

功能 给定人脸特征最接近的数据库所登记的人脸，并给出相似度

参数 face : FaceAPP实例

feature: 人脸特征

score: 相似度分值

name: 登记名字

其中score和name为输出值

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
if(faceapp::FaceQueryDB(mFace,feature,&score,&name) == SUCCESS){
// TODO
}
```

6.12 查询最近登记的人脸名字

int FaceQueryRecentDB(FaceAPP face,int n,char *name)

功能 给出最近第n次登记的人脸名字

参数 face : FaceAPP实例

n: 第n次，n从0开始

name: 登记名字

其中name为输出值

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
if(faceapp::FaceQueryRecent(mFace,m,string_m) == SUCCESS){
// TODO
}
```

6.13 关闭人脸数据库

int FaceCloseDB(FaceAPP face)

功能 关闭人脸数据库

参数 face : FaceAPP实例

返回值 执行成功 *SUCCESS*

执行失败 *ERROR_FAILURE*

```
if(faceapp:: FaceCloseDB (face) == SUCCESS){
// TODO
}
```

6.14 提取人脸特征

int GetFeature(FaceAPP face,float* feature，int* res)

功能 获取人脸特征值数组，特征值数组是存储人脸特征信息的数组，由128个float型数字组成，只获取图片中一个人的特征，多于一人会返回错误信息。

参数 face : FaceAPP实例

feature：存储人脸检测中得到的人脸特征信息，无人脸返回空数组

res: 执行结果，

未发现人脸返回*ERROR_INVALID_PARAM*,

图片数多于1人返回*ERROR_TOO_MANY_REQUESTS*

返回值 *SUCCESS*：正确获取人脸信息

ERROR_FAILURE：未获取到人脸信息

图像质量比较差会导致获取人脸特征失败

使用活体识别功能，没有通过活体识别会导致获取人脸特征失败

```
int ret;
int res;
ret= faceapp::GetFeature(face,feature,&res);
if(ret== SUCCESS){
// TODO 成功获取到人脸特征值
}
```

6.15 获取检测状态数据

int GetDetectData(FaceAPP face,FaceDetctData* data,int *res)

功能 获取人脸关位置信息

参数 face : FaceAPP实例

data：用于存储位置坐标值和关键点数据。

res: 执行结果，

执行失败返回失败原因

返回值 *SUCCESS*：正确获取人脸位置坐标信息

ERROR_FAILURE：未获取到人脸位置坐标信息

```
float pos[4];
int res;
if(faceapp::GetDetectData(mFace,&data,&res) == SUCCESS){
// TODO
}
```

6.16 获取人脸属性

int FetchFaceAttr(FaceAPP face,Image image,FaceDetctData* data,FaceAttributeData *face_attr,int *res)

功能 计算人脸属性包括年龄、性别和表情。

参数 face : FaceAPP实例

image： 人脸图片，用于检测的图片

data：检测得到的人脸位置和人脸关键点信息

face_attr : 存储计算得到的人脸属性

res: 执行结果，

未发现人脸返回*ERROR_INVALID_PARAM*,

返回值 *SUCCESS*：正确获取人脸属性

ERROR_FAILURE：未获取到人脸属性

```
if(faceapp::FetchFaceAttr(mFace,faceapp::Image(&mat),&data,m_face.attr,&res) == SUCCESS){
// TODO
}
```

6.17 获取人脸位置信息

int GetFacePosition(FaceAPP face ,int *num,float *pos,int *res)

功能 获取多人脸关位置信息，支持最大人脸数为3。

参数 face : FaceAPP实例

num : 获取人脸的数量

它的数值会调整为实际人脸数量

pos：用于存储2个位置坐标值，依次是左上角和右下角的坐标。

res: 执行结果，

未发现人脸返回*ERROR_INVALID_PARAM*,

返回值 *SUCCESS*：正确获取人脸位置坐标信息

ERROR_FAILURE：未获取到人脸位置坐标信息

```
float pos[12];
int res;
int num=3;
if(faceapp::GetFacePosition(face,&num,pos,&res)== SUCCESS){
// TODO
}
```

6.18 提取人脸关键点

int GetLandmark(FaceAPP face, int *num, float* landmark，int* res)

功能 获取人脸关键点坐标信息

参数 face : FaceAPP实例

num : 获取人脸的数量

它的数值会调整为实际人脸数量

Landmark：用于存储5个关键点坐标值，依次是左眼，右眼，鼻子，左侧嘴唇，右侧嘴唇。

res: 执行结果，

未发现人脸返回*ERROR_INVALID_PARAM*,

返回值 *SUCCESS*：正确获取人脸关键点坐标信息

ERROR_FAILURE：未获取到人脸关键点信息

```
struct {
    float x[5];
    float y[5];
```

```
}landmark[3];
int res;
int num=3;
if( faceapp::GetLandmark(face,&num,(float*)landmark,&res)==SUCCESS){
// TODO
}
```

6.19 获取人脸质量评分

int GetFaceQuality(FaceAPP face,int *num,FaceQuality *quality,int *res)

功能 对于检测的图片，通过对人脸角度，图片清晰度的给以评分。对于评分小于0.85的图片，无法获得人脸特征数组。

参数 face : FaceAPP实例

num : 获取人脸的数量

它的数值会调整为实际人脸数量

quality：人脸质量

score 人脸质量的置信度

leftRight 左右角度

upDown 上下角度

horizontal 水平角度

clarity 图片清晰度(>250 表示图片清晰)

res: 执行结果，

未发现人脸返回*ERROR_INVALID_PARAM*

返回值 *SUCCESS*：正确获取人脸质量评分

ERROR_FAILURE：未获取到人脸质量评分

```
faceapp::FaceQuality quality;
int res;
int num=3;
if(faceapp::GetFaceQuality(mFace,&num,&quality,&res) == SUCCESS){
// TODO
}
```

6.20 获取活体检测状态

int GetLivenessStatus(FaceAPP face,int *liveness,int *res)

功能 检测是不是真人做人脸识别

参数 face : FaceAPP实例

liveness：活体状态

1 是真人做人脸识别

0 不是是真人做人脸识别

res: 执行结果，

未发现人脸返回*ERROR_INVALID_PARAM*

返回值 *SUCCESS*：正确获取活体状态

ERROR_FAILURE：未获获取活体状态

```
int liveness;
int ret;
int res;
ret= faceapp:: GetLivenessStatus(mFace,&liveness,&res);
if(ret== SUCCESS){
// TODO
}
```

6.21 参数设置

bool SetParameter(FaceAPP face, const char* name, float value[])

功能 设置输入的参数名和对应数值

参数 face : FaceAPP实例

name： 参数的名字

a： a参数值 内部参数，按示例设置，请不要随意修改

b： b参数值 内部参数，按示例设置，请不要随意修改

c： c参数值 内部参数，按示例设置，请不要随意修改

d： d参数值 内部参数，按示例设置，请不要随意修改

factor： 检测人脸放大比例 内部参数，按示例设置，请不要随意修改

min_size： 最小人脸框大小 范围 32-80

faceclarity： 照片清晰度阈值 范围 建议200-400

liveness： 是否做活体检测 范围 0-1

dbsize： 数据库最大支持人脸数， 默认为10000

dbname: 数据库文件名字，默认为facesdb.dat

mode：0-闸机 1-门禁

none：以none字符串表示要这次更改的参数结束

value[]： 参数的数值（可能多个）

返回值 参数设置是否成功

必须在参数名字最后加上none表示此次参数更新列表结束

必须在调用函数Recognize()或Detect()之前更改参数，之后调用就不会起作用，可以只更改部分参数。

```
const char* name[]={"a","b","c","d","factor","min_size","clarity", "none"};
float value[]={0.9,0.9,0.9,0.715,0.6,64,400};
faceapp::SetParameter(face,name,value);

更改部分参数

const char* name[]={"factor","min_size","clarity", "none"};
float value[]={0.6,64,400};
faceapp::SetParameter(face,name,value);
```

6.22 设备鉴权

int AuthorizedDevice(unsigned char *contract_id,unsigned char* user_password)

功能 设备鉴权

参数 contract_id : 客户合同号

user_password: 用户密码

返回值 0：鉴权成功

```
faceapp::AuthorizedDevice(contract_id,user_password);
```

6.23 释放人脸识别资源

void Destroy(FaceAPP face)

功能 释放初始化和设置参数时分配的资源

参数 face : FaceAPP实例

返回值 无

```
faceapp::Destroy(face);
```

6.24 示例代码

```
main()
{
    FaceAPP mFace= faceapp::CreateFaceAPPInstance ();
    const char* name[]={"a","b","c","d","factor","min_size","clarity","none"};
    float value[]={0.9,0.9,0.9,0.715,0.6, 64,400};
    faceapp::SetParameter(mFace,name,value);
```

```
    while(1){
        capture.read(frame);
        cv::resize(frame, mat, ResImgSiz, CV_INTER_LINEAR);
        ret= faceapp::Recognize (face,Image(&mat), &res);
        if(ret==SUCCESS){
           ret= faceapp::GetFeature(face,feature,&res);
           if(ret==SUCCESS){
             ret= faceapp::Compare(face,feature_ref,feature,&score);
            if(ret==SUCCESS){
              if(score>value){
                // TODO
               }
          }
         }
      }
   }
}
```

7 获取 Android 权限

如果您在自己的工程中集成 SDK，请确保已经在工程 AndroidManifest.xml 文件中添 加如下权限：

```
//开启摄像头
<uses-permission android:name="android.permission.CAMERA" />
//如果是 license 激活版本，需要网路
<uses-permission android:name="android.permission.INTERNET" />
//写入文件权限
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

8 注意事项

请严格按照本文中的 API 调用次序，方可实现功能，减少错误。

Android SDK

A. /sdcard/openailab 目录不可以删除
B. 对于 openCVLibrary331 这个 aar 包中默认读取的图片后的格式是 RGBA，本 SDK 支持的输入

格式就是 RGBA。

以下是代码示例

```
Mat mat1 = Imgcodecs.imread(SystemSupport.getInnerSDCardPath()+"/1.jpg");
float[] feature1=new float[128];
int[] ret=new int[1];
byte[] pos=new byte[1024];
FaceAPP.Image image1= face.new Image();
image1.matAddrframe = mat1.getNativeObjAddr();
 int res= face.GetFeature(image1,feature1,pos,ret);
```

Linux SDK

A. 使用 OpenCV3.3.0， 安装位置为/usr/local/AID/opencv3.3.0。 如果安装位置不同，需要把 Makefile opencv 的目录手工改成正确位置。
B. 编译成功后， 所有的运行环境安装在 bin 目录下。需要更改配置， 编辑 bin/demo.conf