版本日志

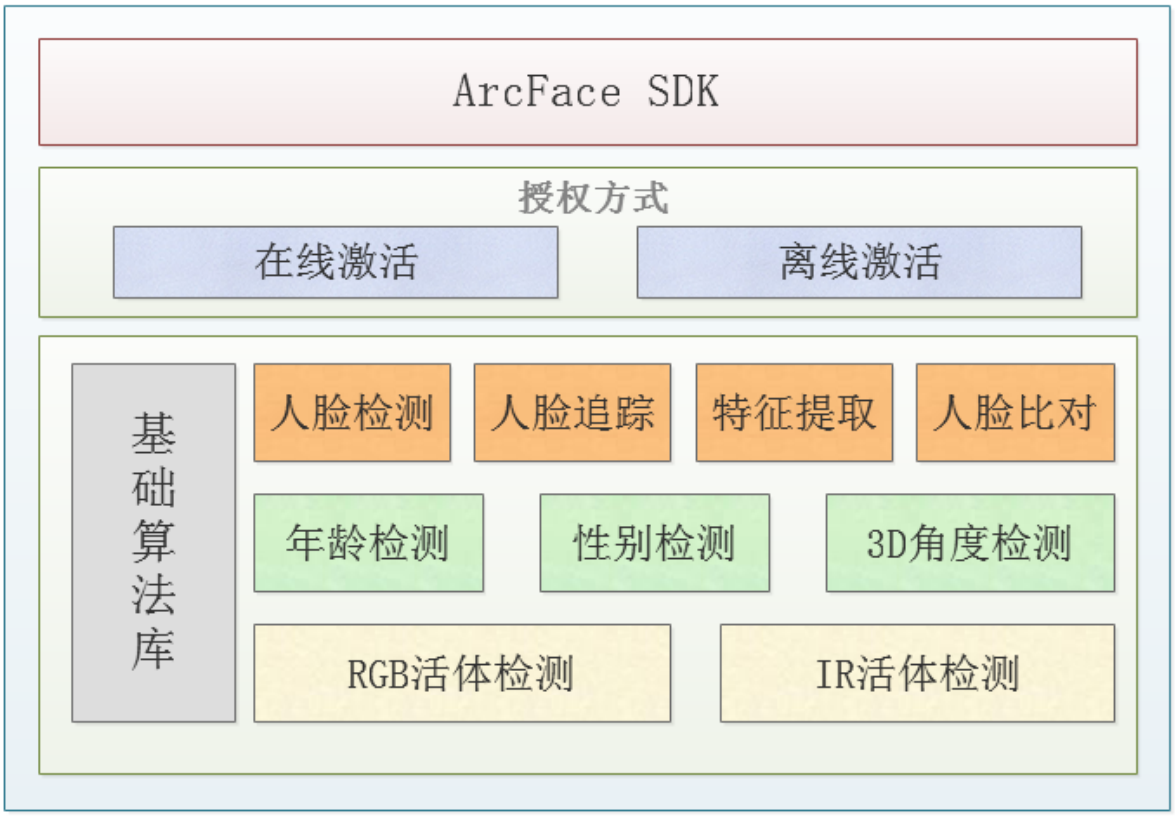
版本	日期	更新说明
3.0.12021020201.1	11/08/2019	1.激活和初始化接口加锁保护； 2.激活和初始化接口加锁限制不支持多线程调用； 3.更新活体、年龄、性别、人脸检测、人脸比对算法模型； 4.IMAGE模式支持全角度以及单一角度； 5.新增人证模式的人脸特征比对模型； 6.新增一组接口，用于兼容更高精度的图像； 7.初始化接口中检测角度参数变更为枚举类型； 8.初始化接口中检测模式修改为枚举类型，全角度检测名称由ASF_OP_0_HIGHER_EXT变更为ASF_OP_ALL_OUT； 9.此版本对人脸特征模型进行了升级，导致与2.X版本的模型不匹配，需要对2.X版本的人脸库进行重新注册; 10.活体检测推荐阈值变更，RGB：0.5 IR：0.7; 11.支持arm64-v8a架构;
3.0.12021020201.2	11/29/2019	1.修复图像处理库中BGR24格式图像数据转换为RGB565格式Bitmap时颜色不对的问题；
3.0.12021020201.3	12/25/2019	1.开放视频模式下活体阈值设置； 2.修复阈值设置的bug； 3.优化加密模块；
3.1.12021020201.1	04/30/2020	1.支持Android 10设备激活； 2.新增MAC地址网卡优先级设置接口； 3.新增获取设备信息接口； 4.新增图像质量检测； 5.支持多线程特征提取； 6.移除旧版active接口；
4.0.12021020201.3	12/11/2020	1. 更新算法库，优化性能和效果； 2. 优化图像质量接口，并支持口罩和非口罩的场景，基于场景精细化设置阈值； 3. 初始化接口删除scale参数，内部Image模式使用27的模型，Video模式使用16的模型； 4. 人脸比对兼容口罩场景，根据接口，结合应用场景进行使用； 5. 人脸检测接口输出睁闭眼、是否带眼镜； 6. 支持口罩、遮挡、额头区域检测； 7. 新增IMEI优先级设置； 8. 修改特征提取接口，兼容口罩模型； 9. 最大检测人脸数最多支持10张； 10. 优化接口结构，修改FaceInfo类； 11. 新增更新人脸信息接口，用于双目对齐等策略；
4.0.12021020201.6	02/22/2021	1. 支持android 4.4(API 19)； 2. 更新updateFaceData接口内存拷贝方式； 3. 修复异常文件导致离线授权接口carsh问题； 4. setDeviceIdPriority接口新增可不使用DeviceId作为设备指纹组成要素； 5. 修复特殊设备上内存异常的问题；

1. 简介

1.1 产品概述

ArcFace 离线SDK，包含人脸检测、性别检测、年龄检测、人脸识别、RGB活体检测、IR活体、图像质量、口罩检测等能力，初次使用时需联网激活，激活后即可在本地无网络环境下工作，可根据具体的业务需求结合人脸识别SDK灵活地进行应用层开发。

> 基础版本暂不支持离线激活，增值版本支持离线激活；



1.2 产品功能简介

1.2.1 人脸检测

对传入的图像数据进行人脸检测，返回人脸的边框以及朝向信息，可用于后续的人脸识别、特征提取、活体检测等操作;

- 支持IMAGE模式和VIDEO模式人脸检测。
- 支持单人脸、多人脸检测，最多支持检测人脸数为10。

1.2.2 人脸追踪

对来自于视频流中的图像数据，进行人脸检测，并对检测到的人脸进行持续跟踪。

1.2.3 图像质量检测

对图像数据中指定的人脸进行图像质量检测。

1.2.4 人脸特征提取

提取人脸特征信息，用于人脸的特征比对。

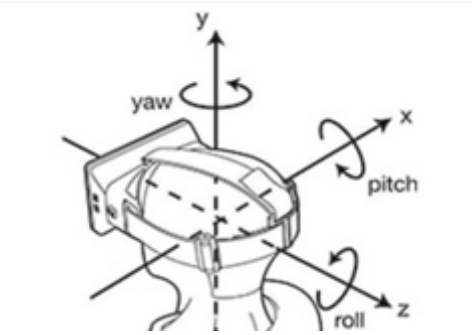
1.2.5 人脸特征比对

对两个人脸特征数据进行比对，返回比对相似度值。

1.2.6 人脸属性检测

人脸属性，支持检测年龄、性别、3D角度以及口罩、遮挡。

人脸3D角度：俯仰角（pitch），横滚角（roll），偏航角（yaw）。



1.2.7 活体检测

离线活体检测，静默式识别，在人脸识别过程中判断操作用户是否为真人，有效防御照片、视频、纸张等不同类型的作弊攻击，提高业务安全性，让人脸识别更安全、更快捷，体验更佳。支持单目RGB活体检测、双目（IR/RGB）活体检测，可满足各类人脸识别终端产品活体检测应用。

1.3 授权方式

SDK授权按设备进行授权，每台硬件设备需要一个独立的授权，此授权的校验是基于设备的唯一标识，被授权的设备在初次授权时需在线进行授权，授权成功后可以离线运行SDK。

1.3.1 在线授权

1. 确保可以正常访问公网；
2. 调用在线激活接口激活SDK;

注意事项：

1. 设备授权后，若设备授权信息被删除（重装系统/应用被卸载等），需联网重新激活；
2. 硬件信息发生变更，需要重新激活；

1.3.2 离线授权

1. 点击【查看激活码】->【离线激活】，进入离线激活操作界面;
2. 生成设备信息文件，使用`getActiveDeviceInfo`接口生成设备信息，保存到文件中；
3. 将设备信息文件上传到开发者中心，生成并获取离线授权文件；
4. 将离线授权文件拷贝到待授权设备上，调用离线激活接口激活SDK；

注意事项：

1. 设备授权后，若设备授权信息被删除（重装系统/应用被卸载等），可用原有激活码进行重新激活；
2. 激活码失效或硬件信息发生变更，需要使用新的激活码重新激活；

1.4 环境要求

1.4.1 运行环境

- Android armeabi-v7a、arm64-v8a

1.4.2 系统要求

- 支持Android 4.4 (API Level 19) 至 Android 10（API Level 29）

1.4.3 权限申明

- 获取设备唯一标识，用于SDK激活授权

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

- 允许应用联网，用于SDK联网激活授权

```
<uses-permission android:name="android.permission.INTERNET" />
```

- 允许应用读取本地文件，用于SDK离线激活授权

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

2. 接入须知

2.1 SDK的获取

2.1.1 注册开发者账号

- 访问ArcSoft AI开放平台门户：https://ai.arcsoft.com.cn，注册开发者账号并登录。

2.1.2 SDK下载

- 创建对应的应用，并选择需要下载的SDK、对应平台以及版本，确认后即可下载SDK和查看激活码。

- 点击【查看激活码】即可查看所需要APP_ID、SDK_KEY、ACTIVE_KEY，点击下载图标获取SDK开发包。

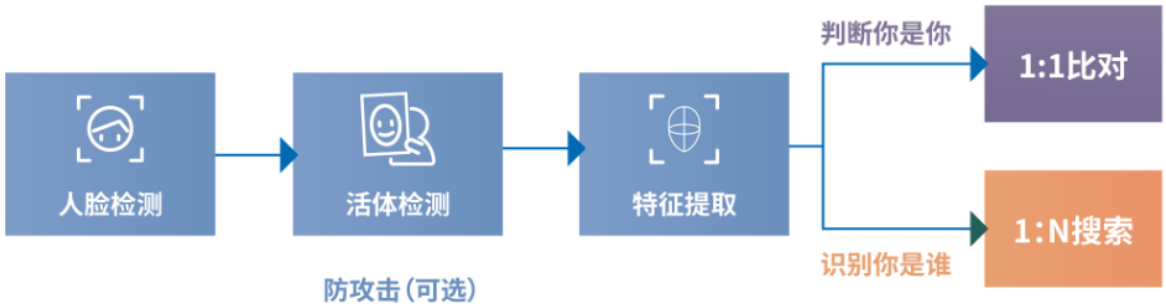
2.2 SDK包结构

```
|---doc
|   |---ARCSOFT_ARC_FACE_DEVELOPER'S_GUIDE.pdf          开发说明文档
|---libs
|   |---armeabi-v7a
|   |   |---libarcsoft_face.so                          算法库
|   |   |---libarcsoft_face_engine.so                   引擎库
|   |---arm64-v8a
|   |   |---libarcsoft_face.so                          算法库
|   |   |---libarcsoft_face_engine.so                   引擎库
|   |--arcsoft_face.jar                                 引擎接口
|---samplecode
|   |---ArcfaceDemo                                     示例工程
|---imageutil
|   |---ArcSoftImageUtil.zip                            图像处理SDK
|---releasenotes.txt                                    说明文件
```

2.3 业务流程

2.3.1 通用流程概述

> 根据实际应用场景以及硬件配置，活体检测和特征提取可选择是否采用并行的方式。



- 如上图所示，人脸识别的核心业务流程可以分为以下四个步骤：

1. 人脸检测：采集视频帧或静态图，传入算法进行检测，输出人脸数据，用于后续的检测。
2. 活体检测：在人脸识别过程中判断操作用户是否为真人，有效防御照片、视频、纸张等不同类型的作弊攻击，提高业务安全性，可根据应用场景选择是否接入；
3. 特征提取：对待比对的图像进行特征提取、人脸库的预提取，用于之后的比对；
4. 人脸识别：1：1比对主要是判断「你是你」，用于核实身份的真实性；1：N搜索主要识别「你是谁」，用于明确身份的具体所属。

2.3.2 活体检测

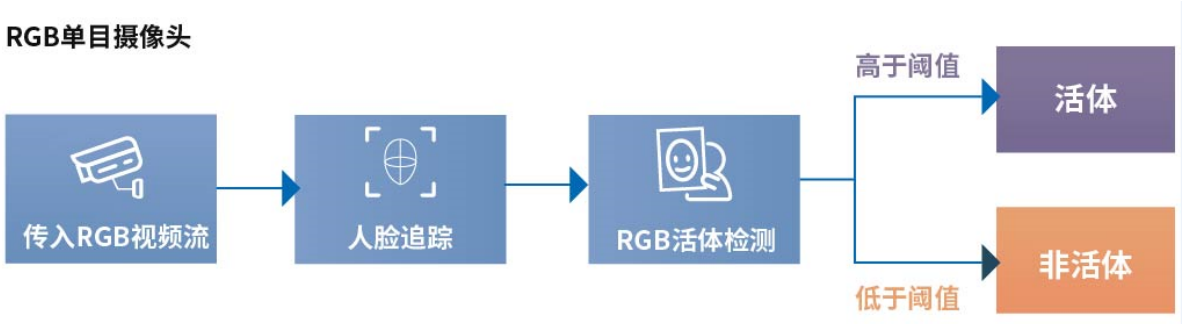
> 提示：此版本仅支持RGB、IR活体。

2.3.2.1 基础原理

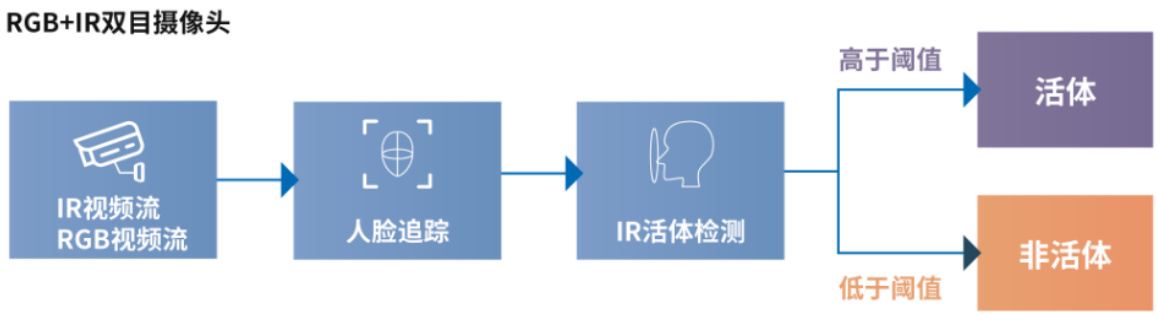
- RGB可见光活体：主要基于图片破绽，判断目标对象是否为活体。例如图像中的屏幕反光、成像畸形、二次翻拍边框背景等。RGB活体检测基于单目摄像头，采用静默式识别方式，用户体验较好。同时RGB活体受光线以及成像质量影响较大，在强光、暗光等场景，容易影响检测结果，可通过适当的补光、使用宽动态镜头抵消逆光等方式缓解。
- IR红外活体：主要基于红外光线反射成像原理，通过人脸成像甄别是否为活体。基于红外成像特点，对于屏幕类攻击，基本可以达到近100%的活体防御。IR采用静默式识别方式，体验较好，但需要增加红外摄像头成本，同时要结合场景考虑红外成像距离。

> 提示：在实际业务场景中，需要根据场景特点，灵活组合使用以上几种活体方案。

2.3.2.2 RGB 单目活体检测

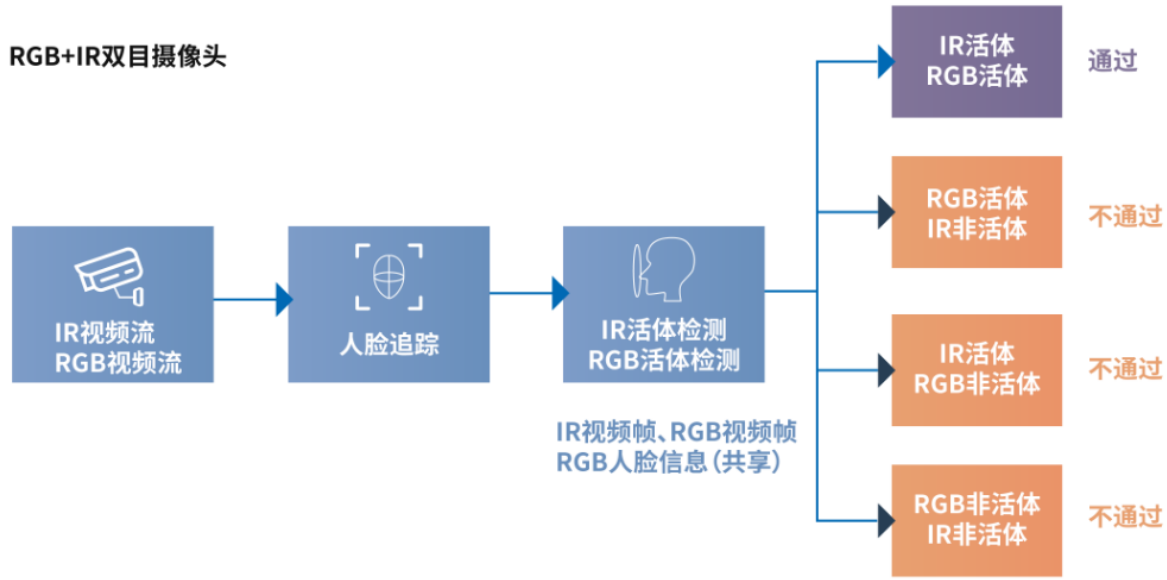


2.3.2.3 IR 双目活体检测



> 提示：为了保证取到有效帧，使用RGB视频流检测到人脸信息和IR图像，传入IR活体检测接口进行IR活体检测。

2.3.2.4 RGB+IR 双目活体检测



2.3.2.5 场景及应用方案

- 通行场景：此场景以考虑通行效率为主，并在此基础上增加活体检测。建议可采用RGB活体检测，体验较好，保障效率的同时仍可保证安全性。
- 身份核验场景：此场景优先确保业务安全性，需提升活体检测安全级别。可考虑采用 RGB+NIR双目活体检测，最大程度防御非法攻击。

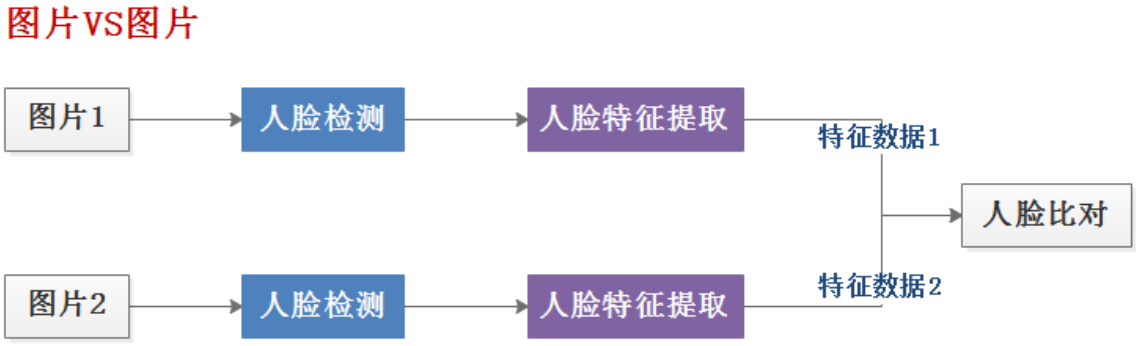
> 使用时尽量避免强光、暗光等场景，可通过补光灯、宽动态摄像头进行缓解。

2.3.3 人脸`1:1`比对

> `1:1`比对常见的应用场景可分为如下两种形式：

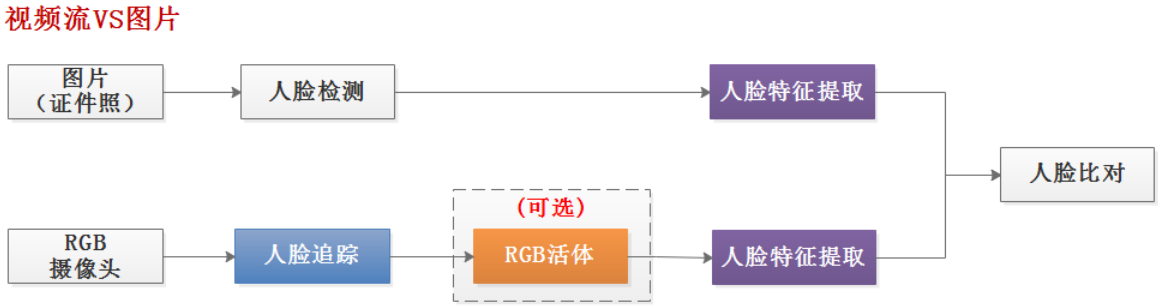
2.3.3.1 图片vs图片

> 两张静态图片分别提取特征，进行比对。



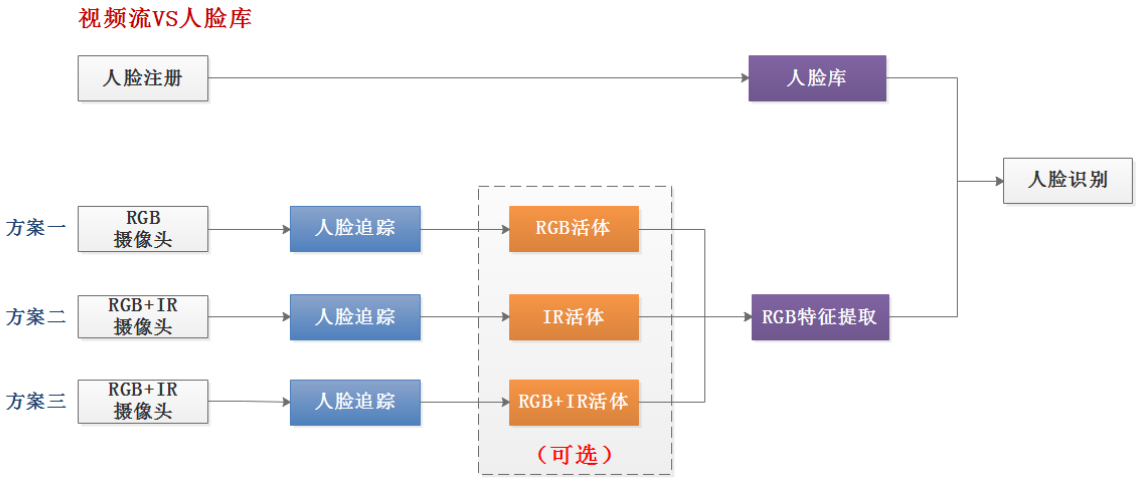
2.3.3.2 实时视频流vs图片

> 该场景比较常见，典型的应用场景为人证比对和人脸门禁，本SDK同时支持人证比对和生活照识别，在特征比对时选择对应的特征比对模型即可。



2.3.4 人脸1:N搜索

> 将需要识别的人脸图片集注册到本地人脸库中，当有用户需要识别身份时，从视频流中实时采集人脸图片，与人脸库中的人脸集合对比，得到搜索结果。



2.3.5 方案选型

> 可结合应用场景进行方案的选择

离线1：1比对

提供本地化的1：1人脸对比功能，证明你是你，可有效应用于车站、机场、大型活动、机关单位、银行、酒店、网吧等人员流动频繁场所或其它重点场景的人证核验，也可用于线上开户的人员身份验证等场景。

离线1：N检索

提供本地化的1：N人脸搜索功能，识别你是谁，可有效应用于社区、楼宇、工地、学校等较大规模的人脸考勤签到、人脸通行等应用场景。该版本人脸库在1万以内效果更佳。

2.4 指标

> 算法检测指标受硬件配置、图像数据质量、测试方法等因素影响，以下实验数据仅供参考，具体数据以实际应用场景测试结果为准。

2.4.1 算法性能

- 硬件信息：
  - 处理器：RK3288
  - 内存：4G
  - Android 版本：5.1
- 分辨率：1280 x 720

算法	性能(ms)
Detect	< 80
FeatureExtract	< 350
FeatureCompare	< 0.009
RGB Liveness	< 100
IR Liveness	< 30

2.4.2 阈值推荐

活体分值区间为[0~1]，推荐阈值如下，高于此阈值的即可判断为活体。

- RGB 活体：0.5
- IR 活体：0.7

图像质量分值区间为[0-1]，推荐阈值如下，高于此阈值图像质量即合格。

- 注册场景（要求不戴口罩）：0.63
- 识别场景、不戴口罩：0.49
- 识别场景、戴口罩：0.29

人脸遮挡分值区间为[0~1]，推荐阈值如下，高于此阈值的即可判断为人脸被遮挡。

- 人脸遮挡阈值：0.8

人脸比对分值区间为[0~1]，推荐阈值如下，高于此阈值的即可判断为同一人。

- 用于生活照之间的特征比对，推荐阈值0.8
- 用于证件照或生活照与证件照之间的特征比对，推荐阈值0.82

2.4.3 图像要求

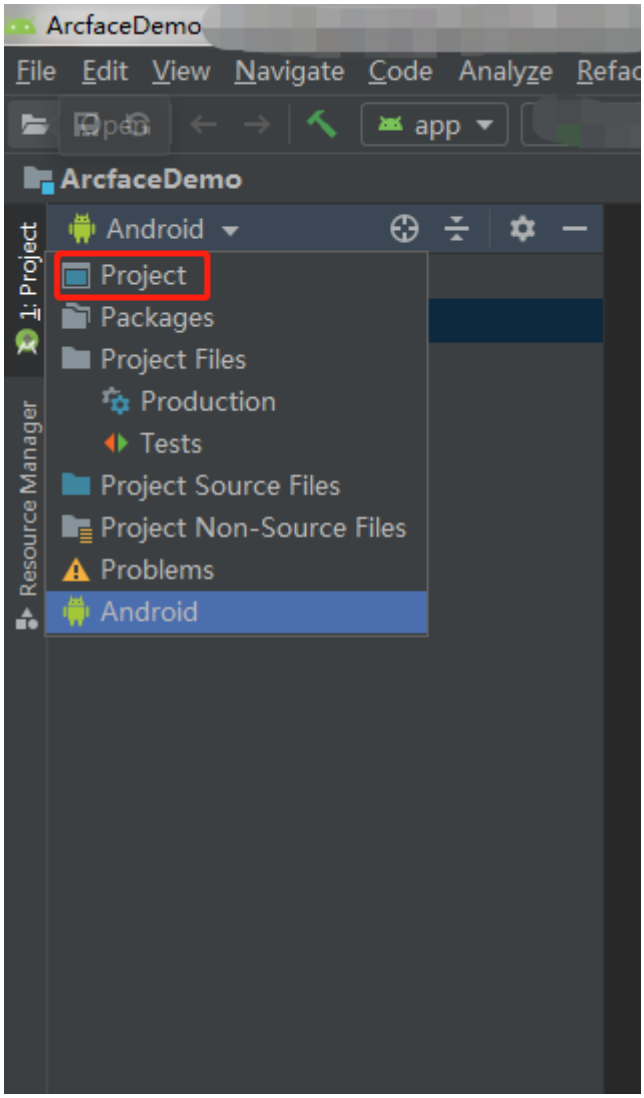
- 建议待检测的图像人脸角度上、下、左、右转向小于30度；
- 图像中人脸尺寸不小于80 x 80像素；
- 图像大小小于10MB；
- 图像清晰；

3. SDK的详细接入介绍

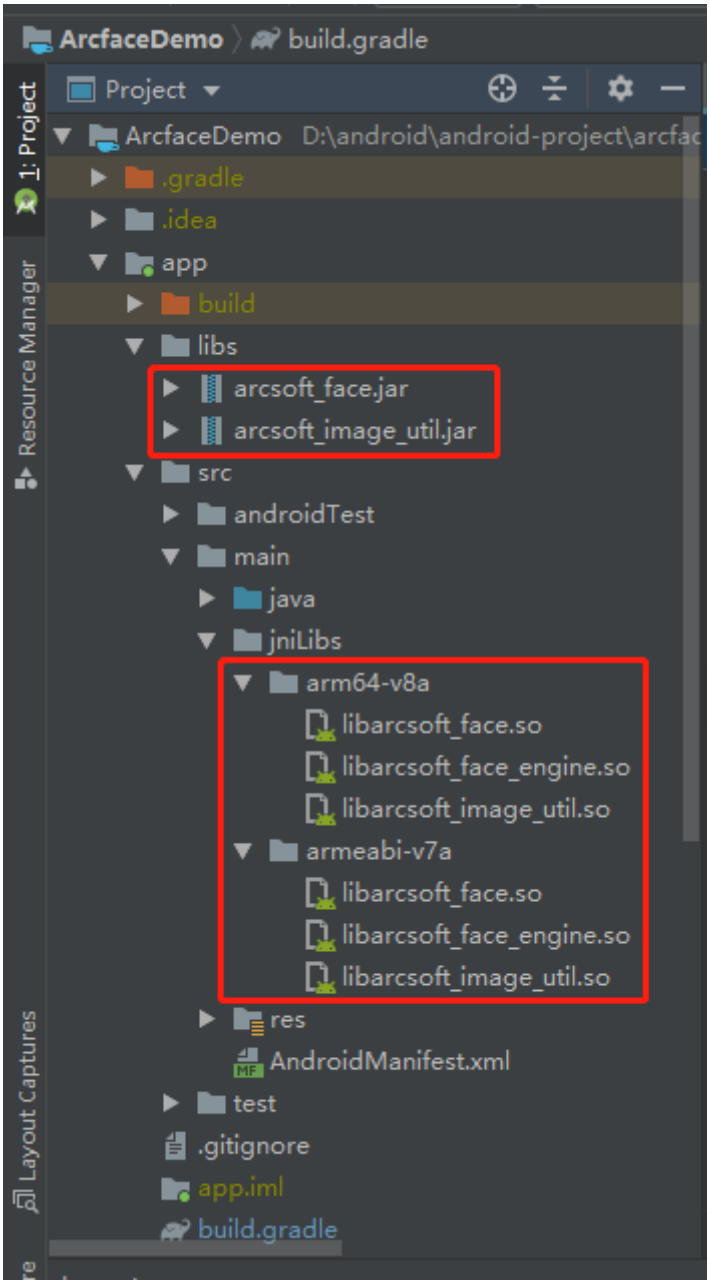
3.1 Demo工程配置

> 提示：以下部分流程在samplecode目录下的ArcfaceDemo工程中已处理。演示截图的Android Studio版本为3.5，如版本不一致可能界面显示会有不同，但操作步骤是一致的。

1. 打开SDK包中，`samplecode`目录下的`ArcfaceDemo`工程
2. 打开后切换到`Project`视图。



3. 在`app`目录下创建`libs`目录，添加`jar`文件，在`src/main`目录下新建`jniLibs`目录，添加`so`文件，如下图所示

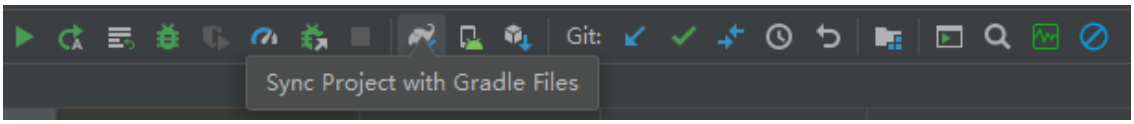


4. 确保 app 目录下的 build.gradle 中包含如下配置

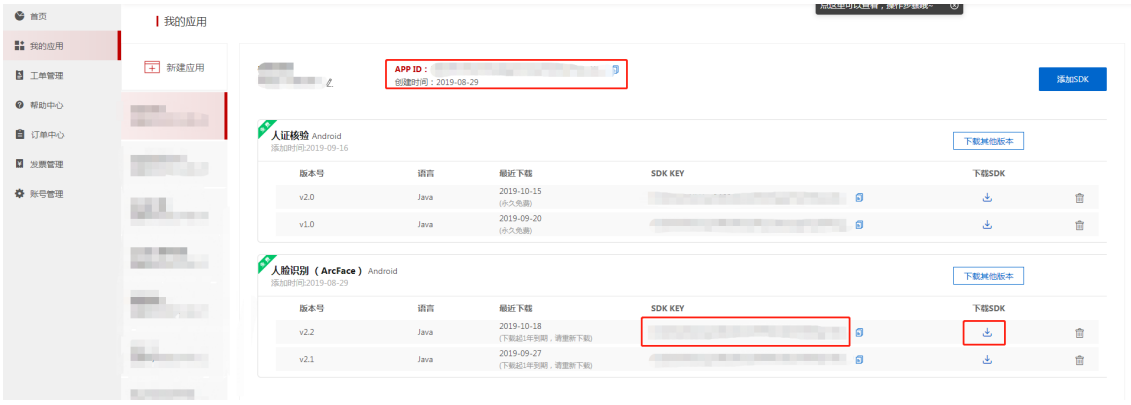
   在 dependencies 中：`implementation fileTree(dir: 'libs', include: ['*.jar'])`

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
    implementation 'com.android.support:recyclerview-v7:28.0.0'
    implementation 'com.github.bumptech.glide:glide:4.9.0'
    implementation 'io.reactivex.rxjava2:rxjava:2.2.6'
    implementation 'io.reactivex.rxjava2:rxandroid:2.1.0'
}
```

5. 点击 `Sync Project with Gradle Files` 进行同步



6. 前往虹软视觉开放平台获取 `APP_ID` 、 `SDK_KEY` 、 `ACTIVE_KEY`



7. 将获取到的 `APP_ID` 、 `SDK_KEY` 、 `ACTIVE_KEY` 填入工程的指定位置

将 `APP_ID` 与 `SDK_KEY` 填入 `com.arcsoft.arcfacedemo.common.Constants` 类中

Constants.java

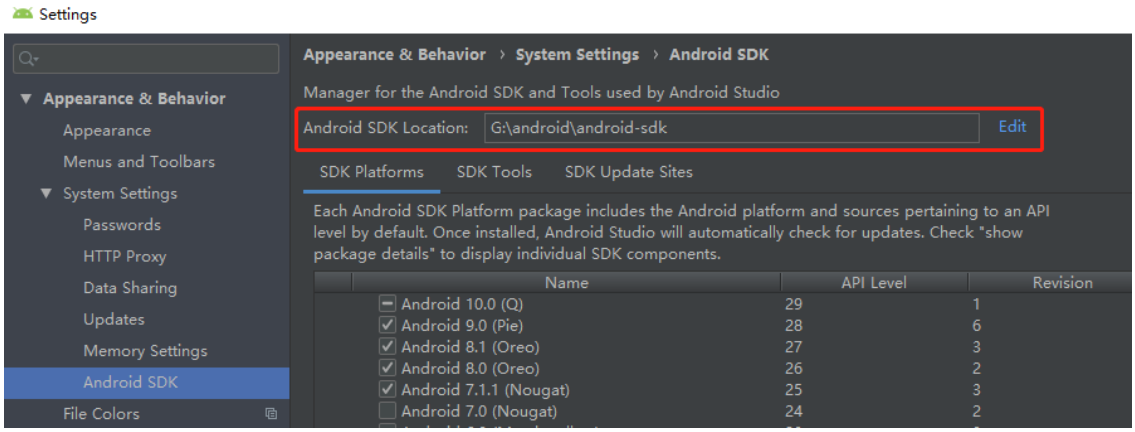
```
package com.arcsoft.arcfacedemo.common;

public class Constants {
    public static final String APP_ID = "官网申请的APP_ID";
    public static final String SDK_KEY = "官网申请的SDK_KEY";
    public static final String ACTIVE_KEY = "官网申请的ACTIVE_KEY";

    /**
     * IR预览数据相对于RGB预览数据的横向偏移量，注意：是预览数据，一般的摄像头的预览数据都是 width > height
     */
    public static final int HORIZONTAL_OFFSET = 0;
    /**
     * IR预览数据相对于RGB预览数据的纵向偏移量，注意：是预览数据，一般的摄像头的预览数据都是 width > height
     */
    public static final int VERTICAL_OFFSET = 0;
}

```

8. 一切准备就绪，点击运行即可

3.2 配置异常处理

- 若出现 `please select android sdk` 错误提示

点击 `Android Studio` 左上角的 `File` 选项，选择 `Settings -> Appearance & Behavior -> System Settings -> Android SDK` 页面，选择本地Android SDK即可



3.3 支持的颜色格式

> 提示：当前版本仅支持前3种颜色格式，`CP_PAF_DEPTH_U16`只是预留。

常量名	常量值	颜色格式说明
CP_PAF_NV21	2050	8-bit Y 通道，8-bit 2x2 采样 V 与 U 分量交织通道
CP_PAF_BGR24	513	RGB 分量交织，按 B, G, R, B 字节序排布
CP_PAF_GRAY	1793	8-bit IR图像
CP_PAF_DEPTH_U16	3074	16-bit IR图像（预留字段，暂不支持）

3.4 枚举类型

3.4.1 DetectMode

枚举描述

检测模式。

枚举值描述

枚举名	描述
ASF_DETECT_MODE_VIDEO	VIDEO检测模式，用于处理连续帧的图像数据
ASF_DETECT_MODE_IMAGE	IMAGE检测模式，用于处理单张的图像数据

3.4.2 DetectFaceOrientPriority

枚举描述

检测角度优先级。

枚举值描述

枚举名	描述
ASF_OP_0_ONLY	人脸检测角度，逆时针0度
ASF_OP_90_ONLY	人脸检测角度，逆时针90度
ASF_OP_180_ONLY	人脸检测角度，逆时针180度
ASF_OP_270_ONLY	人脸检测角度，逆时针270度
ASF_OP_ALL_OUT	人脸检测角度，全角度检测

3.4.3 CompareModel

枚举描述

比对模型。

枚举值描述

枚举名	描述
LIFE_PHOTO	用于生活照之间的特征比对
ID_CARD	用于证件照或生活照与证件照之间的特征比对

3.4.4 DetectModel

枚举描述

检测模型（预留扩展其他检测模型）

枚举值描述

枚举名	描述
RGB	RGB检测模型

3.4.5 RuntimeABI

枚举描述

运行时架构。

枚举值描述

枚举名	描述
ANDROID_ABI_ARM64	ARM64运行时环境
ANDROID_ABI_ARM32	ARM32运行时环境
ANDROID_ABI_UNSUPPORTED	不支持的运行时环境

3.4.6 MACPriority

枚举描述

获取mac地址的网卡优先级。

枚举值描述

枚举名	描述
ETH0_FIRST	有线网卡优先（SDK默认值）
WLAN0_FIRST	无线网卡优先

3.4.7 DeviceIdPriority

枚举描述

获取DeviceId的优先级。

枚举值描述

枚举名	描述
DEFAULT	系统默认优先级（SDK默认值）
IMEI_FIRST	IMEI优先
MEID_FIRST	MEID优先
NOT_USE_DEVICEID	不获取DeviceId

3.4.8 ExtractType

枚举描述

特征提取的类型。

枚举值描述

枚举名	描述
REGISTER	注册：在注册人脸时使用，耗时会比RECOGNIZE类型长
RECOGNIZE	识别：在识别人脸时使用

3.5 数据结构

3.5.1 VersionInfo

类描述

SDK版本信息。

成员描述

类型	变量名	描述
String	version	版本号信息
String	buildDate	构建日期
String	copyRight	版权信息

3.5.2 ActiveDeviceInfo

类描述

离线激活相关，设备信息。

成员描述

类型	变量名	描述
String	deviceInfo	设备信息

3.5.3 ActiveFileInfo

类描述

激活文件信息。

成员描述

类型	变量名	描述
String	appId	APP ID
String	sdkKey	SDK KEY
String	activeKey	激活码
String	platform	平台版本
String	sdkType	SDK类型
String	sdkVersion	SDK版本号
String	fileVersion	激活文件版本号
String	startTime	SDK开始时间
String	endTime	SDK截止时间

3.5.4 ArcSoftImageInfo

类描述

图像信息。其中图像宽度为4的倍数，图像高度在NV21格式下要求为2的倍数，BGR24\GRAY\DEPTH_U16格式无限制；

成员描述

类型	变量名	描述
int	width	图像宽度
int	height	图像高度
int	imageFormat	图像格式
byte[][]	planes	图像通道
int[]	strides	每个图像通道的步长

组成方式介绍

```
// arcSoftImageInfo组成方式举例：

// NV21格式数据，有两个通道，
// Y通道步长一般为图像宽度，若图像经过8字节对齐、16字节对齐等操作，需填入对齐后的图像步长
// VU通道步长一般为图像宽度，若图像经过8字节对齐、16字节对齐等操作，需填入对齐后的图像步长
ArcSoftImageInfo arcSoftImageInfo = new ArcSoftImageInfo(width, height,
FaceEngine.CP_PAF_NV21, new byte[][]{planeY, planeVU}, new int[]{yStride,
vuStride});
// GRAY，只有一个通道，
// 步长一般为图像宽度，若图像经过8字节对齐、16字节对齐等操作，需填入对齐后的图像步长
arcSoftImageInfo = new ArcSoftImageInfo(width, height, FaceEngine.CP_PAF_GRAY,
new byte[][]{gray}, new int[]{grayStride});
// BGR24，只有一个通道，
// 步长一般为图像宽度的三倍，若图像经过8字节对齐、16字节对齐等操作，需填入对齐后的图像步长
arcSoftImageInfo = new ArcSoftImageInfo(width, height, FaceEngine.CP_PAF_BGR24,
new byte[][]{bgr24}, new int[]{bgr24Stride});
// DEPTH_U16，只有一个通道，
// 步长一般为图像宽度的两倍，若图像经过8字节对齐、16字节对齐等操作，需填入对齐后的图像步长
arcSoftImageInfo = new ArcSoftImageInfo(width, height,
FaceEngine.CP_PAF_DEPTH_U16, new byte[][]{depthU16}, new int[]{depthU16Stride});
```

3.5.5 FaceInfo

类描述

人脸信息。

成员描述

类型	变量名	描述
Rect	rect	人脸框
int	orient	人脸角度
int	faceId	一张人脸从进入画面直到离开画面，faceId不变。 该参数在VIDEO模式下有效
float	wearGlasses	戴眼镜的置信度[0-1],推荐阈值：0.5
int	leftEyeClosed	左眼状态 -1:默认输出 0:未闭眼 1:闭眼
int	rightEyeClosed	右眼状态 -1:默认输出 0:未闭眼 1:闭眼
int	faceShelter	人脸是否被遮挡 "1" 遮挡, "0" 未遮挡, "-1" 不确定
byte[]	faceData	人脸数据

3.5.6 ImageQualitySimilar

类描述

图像质量检测信息。

成员描述

类型	变量名	描述
float	score	图像质量置信度

3.5.7 FaceFeature

类描述

人脸特征。

成员描述

类型	变量名	描述
byte[]	featureData	人脸特征数据

3.5.8 AgeInfo

类描述

年龄信息。

成员描述

类型	变量名	描述
int	age	AgeInfo.UNKNOWN_AGE： 未知 其他： 年龄值

3.5.9 GenderInfo

类描述

性别信息。

成员描述

类型	变量名	描述
int	gender	GenderInfo.MALE：男性 GenderInfo.FEMALE：女性 GenderInfo.UNKNOWN：未知

3.5.10 Face3DAngle

类描述

3D角度信息。

成员描述

类型	变量名	描述
float	yaw	偏航角
float	roll	横滚角
float	pitch	俯仰角
int	status	0: 正常 非0: 异常

3.5.11 LivenessParam

类描述

活体置信度。

成员描述

类型	变量名	描述
float	rgbThreshold	RGB活体置信度
float	irThreshold	IR活体置信度

3.5.12 LivenessInfo

类描述

活体信息。

成员描述

类型	变量名	描述
int	liveness	LivenessInfo.NOT_ALIVE：非真人 LivenessInfo.ALIVE：真人 LivenessInfo.UNKNOWN：不确定 LivenessInfo.FACE_NUM_MORE_THAN_ONE：传入人脸数 > 1 LivenessInfo.FACE_TOO_SMALL：人脸过小 LivenessInfo.FACE_ANGLE_TOO_LARGE：角度过大 LivenessInfo.FACE_BEYOND_BOUNDARY：人脸超出边界 LivenessInfo.ERROR_DEPTH：深度图错误 LivenessInfo.TOO_BRIGHT_IR_IMAGE：红外图像太亮了

3.5.13 MaskInfo

类描述

口罩佩戴信息。

成员描述

类型	变量名	描述
int	mask	MaskInfo.WORN：已佩戴 MaskInfo.NOT_WORN：未佩戴 MaskInfo.UNKNOWN：不确定

3.5.14 LandmarkInfo

类描述

人脸额头区域信息。

成员描述

类型	变量名	描述
PointF[]	landmarks	人脸额头关键点，数组长度为4，各个下标对应的关键点如下： 0：左上角 1：右上角 2：右下角 3：左下角

3.6 常量描述

3.6.1 检测到的人脸角度

> 注：按逆时针方向。

```
public static final int ASF_OC_0 = 0x1;         // 0度
public static final int ASF_OC_90 = 0x2;        // 90度
public static final int ASF_OC_270 = 0x3;       // 270度
public static final int ASF_OC_180 = 0x4;       // 180度
public static final int ASF_OC_30 = 0x5;        // 30度
public static final int ASF_OC_60 = 0x6;        // 60度
public static final int ASF_OC_120 = 0x7;       // 120度
public static final int ASF_OC_150 = 0x8;       // 150度
public static final int ASF_OC_210 = 0x9;       // 210度
public static final int ASF_OC_240 = 0xa;       // 240度
public static final int ASF_OC_300 = 0xb;       // 300度
public static final int ASF_OC_330 = 0xc;       // 330度
```

3.7 功能接口

当前版本使用同一个引擎句柄不支持多线程调用同一个算法接口，若需要对同一个接口进行多线程调用需要启动多个引擎。

例如：若需要做多人脸门禁系统，我们希望提升识别速度，一般会使用同一个引擎进行人脸检测，我们可以提前初始化多个引擎用于人脸特征提取，这些引擎只需要初始化`ASF_FACERECOGNITION`属性即可。同时要根据硬件硬件配置来控制最多启动的线程数。

3.7.1 activeOnline

功能描述

用于在线激活SDK。

方法

```
int activeOnline(Context context, String activeKey, String appId, String sdkKey)
```

> 初次使用SDK时需要对SDK先进行激活，激活后无需重复调用；
> 调用此接口时必须为联网状态，激活成功后即可离线使用；

参数说明

参数	类型	描述
context	in	上下文信息
activeKey	in	官网获取的ACTIVE_KEY
appId	in	官网获取的APP_ID
sdkKey	in	官网获取的SDK_KEY

返回值

成功返回`ErrorInfo.MOK`、`ErrorInfo.MERR_ASF_ALREADY_ACTIVATED`，失败详见`4.2`错误码列表。

示例代码

```
//该组数据无效，仅做示例参考
String APP_ID = "D617np8jyKt1jN9gMr7ENbT3gjFdaeDet3Pp8yfwHxnt";
String SDK_KEY = "8FdAtZSffuvS6kuzPP4PrxyxkQMGpSi4yE3Amo61sbF2";
String ACTIVE_KEY = "8511-F112-J5V2-TJ7F";
```

```
int code = FaceEngine.activeOnline(context, ACTIVE_KEY, APP_ID, SDK_KEY);
if(code == ErrorInfo.MOK){
    Log.i(TAG, "activeOnline success");
}else if(code == ErrorInfo.MERR_ASF_ALREADY_ACTIVATED){
    Log.i(TAG, "already activated");
}else{
    Log.i(TAG, "activeOnline failed, code is : " + code);
}
```

3.7.2 activeOffline

功能描述

用于离线激活SDK。

方法

```
int activeOffline(Context context, String filePath)
```

参数说明

参数	类型	描述
context	in	上下文信息
filePath	in	离线激活文件路径

返回值

成功返回`ErrorInfo.MOK`、`ErrorInfo.MERR_ASF_ALREADY_ACTIVATED`，失败详见 4.2 错误码列表。

示例代码

```
int code = FaceEngine.activeOffline(context, filePath);
if(code == ErrorInfo.MOK){
    Log.i(TAG, "activeOffline success");
}else if(code == ErrorInfo.MERR_ASF_ALREADY_ACTIVATED){
    Log.i(TAG, "already activated");
}else if(code == MERR_ASF_LOCAL_EXIST_USEFUL_ACTIVE_FILE){
    Log.i(TAG, "activeOffline failed, but a useful local active file exists");
}else{
    Log.i(TAG, "activeOffline failed, code is : " + code);
}
```

3.7.3 setMACPriority

功能描述

获取设备信息相关，设置获取mac地址的网卡优先级。

方法

```
int setMACPriority(MACPriority macPriority)
```

参数说明

参数	类型	描述
macPriority	in	获取mac地址的网卡优先级，可以是MACPriority.ETH0_FIRST或MACPriority.WLAN0_FIRST

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2`错误码列表。

示例代码

```
// 默认以有线网卡优先

// 设置为优先获取有线网卡MAC地址
FaceEngine.setMACPriority(MACPriority.ETH0_FIRST);
// 设置为优先获取无线网卡MAC地址
FaceEngine.setMACPriority(MACPriority.WLAN0_FIRST);
```

3.7.4 setDeviceIdPriority

功能描述

获取设备信息相关，设置获取DeviceId的优先级。

方法

```
int setDeviceIdPriority(DeviceIdPriority deviceIdPriority)
```

参数说明

参数	类型	描述
deviceIdPriority	in	获取DeviceId的优先级，可以是： DeviceIdPriority.DEFAULT DeviceIdPriority.IMEI_FIRST DeviceIdPriority.MEID_FIRST DeviceIdPriority.NOT_USE_DEVICEID

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2`错误码列表。

示例代码

```
// 默认以系统为准

// 设置为优先获取IMEI
FaceEngine.setDeviceIdPriority(DeviceIdPriority.IMEI_FIRST);
// 设置为优先获取MEID
FaceEngine.setDeviceIdPriority(DeviceIdPriority.MEID_FIRST);
// 设置为不获取DeviceId
FaceEngine.setDeviceIdPriority(DeviceIdPriority.NOT_USE_DEVICEID);
```

3.7.5 getActiveDeviceInfo

功能描述

离线激活相关，获取设备信息。

方法

```
int getActiveDeviceInfo(Context context, ActiveDeviceInfo activeDeviceInfo)
```

参数说明

参数	类型	描述
context	in	上下文对象
activeDeviceInfo	out	设备信息内容，若获取成功，则将设备信息保存在`activeDeviceInfo.deviceInfo`中

返回值

成功返回`ErrorInfo.MOK`，失败详见 `4.2` 错误码列表。

示例代码

```
ActiveDeviceInfo activeDeviceInfo = new ActiveDeviceInfo();
int code = FaceEngine.getActiveDeviceInfo(context, activeDeviceInfo);
....
if (code == ErrorInfo.MOK) {
   // 可复制信息保存到文件
   ...
} else {
   // 获取信息失败
   Log.i(TAG,"getActiveDeviceInfo failed, code is : " + code)
}
```

3.7.6 getActiveFileInfo

功能描述

获取激活文件信息。

方法

```
int getActiveFileInfo(Context context, ActiveFileInfo activeFileInfo)
```

参数说明

参数	类型	描述
context	in	上下文信息
activeFileInfo	out	激活文件信息

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2`错误码列表。

示例代码

```
ActiveFileInfo activeFileInfo = new ActiveFileInfo();
int code = FaceEngine.getActiveFileInfo(context, activeFileInfo);
if (code == ErrorInfo.MOK) {
    Log.i(TAG, activeFileInfo.toString());
}else{
    Log.i(TAG, "getActiveFileInfo failed, code is : " + code);
}
```

3.7.7 init

功能描述

初始化引擎。

> 该接口至关重要，清楚的了解该接口参数的意义，可以避免一些问题以及对项目的设计都有一定的帮助。

方法

```
int init(
    Context context,
    DetectMode detectMode,
    DetectFaceOrientPriority detectFaceOrientPriority,
    int detectFaceMaxNum,
    int combinedMask
)
```

参数说明

参数	类型	描述
context	in	上下文信息
detectMode	in	VIDEO模式：处理连续帧的图像数据 IMAGE模式：处理单张的图像数据
detectFaceOrientPriority	in	人脸检测角度，推荐单一角度检测；
detectFaceMaxNum	in	最大需要检测的人脸个数，取值范围[1,10]
combinedMask	in	需要启用的功能组合，可多选

重点参数详细说明

- **detectMode**

枚举名	说明
ASF_DETECT_MODE_VIDEO	VIDEO模式
ASF_DETECT_MODE_IMAGE	IMAGE模式

VIDEO 模式

1. 对视频流中的人脸进行追踪，人脸框平滑过渡，不会出现跳框的现象。
2. 用于预览帧数据的人脸追踪，使用该模式比使用IMAGE模式处理速度更快，可避免出现卡顿问题。
3. 在视频模式下人脸追踪会带有一个`faceId`值，该值用于标记一张人脸，当一个人脸从进入画面直到离开画面，`faceId`值不变，这可用于业务中优化程序性能。

IMAGE 模式

1. 针对单张图片进行人脸检测精度更高。
2. 在注册人脸库时，我们建议使用精度更高的IMAGE模式。

模式选择

1. 从摄像头中获取数据并需要预览显示，推荐选择VIDEO模式；
2. 处理静态图像数据，类似注册人脸库时，推荐使用IMAGE模式；
3. 若同时需要进行IMAGE模式人脸检测和VIDEO模式人脸检测，需要创建一个VIDEO模式的引擎和一个IMAGE模式的引擎；

- **detectFaceOrientPriority**

枚举名	说明
ASF_OP_0_ONLY	人脸检测角度，逆时针0度
ASF_OP_90_ONLY	人脸检测角度，逆时针90度
ASF_OP_180_ONLY	人脸检测角度，逆时针180度
ASF_OP_270_ONLY	人脸检测角度，逆时针270度
ASF_OP_ALL_OUT	人脸检测角度，全角度检测

> 注：设置0度方向并不是说只有0度角可以检测到人脸，而是大体在这个方向上的人脸均可以，我们也提供了全角度方向检测，但在应用场景确定的情况下我们还是推荐使用单一角度进行检测，因为单一角相对于全角度性能更好、精度更高。


0度角


90度角


180度角


270度角

- **combinedMask**

> 针对算法功能会有常量值与之一一对应，可根据业务需求进行自由选择，不需要的属性可以不用初始化，否则会占用多余内存。

```
public static final int ASF_FACE_DETECT = 0x00000001;          //人脸检测
public static final int ASF_FACE_RECOGNITION = 0x00000004;     //人脸特征
public static final int ASF_AGE = 0x00000008;                  //年龄
public static final int ASF_GENDER = 0x00000010;               //性别
public static final int ASF_FACE3DANGLE = 0x00000020;          //3D角度
public static final int ASF_FACELANDMARK = 0x00000040;         //人脸额头关键点信
息
public static final int ASF_LIVENESS = 0x00000080;             //RGB活体
public static final int ASF_IMAGEQUALITY = 0x00000200;         //图像质量检测
public static final int ASF_IR_LIVENESS = 0x00000400;          //IR活体
public static final int ASF_FACE_SHELTER = 0x00000800;         //人脸遮挡检测
public static final int ASF_MASK_DETECT = 0x00001000;          //口罩检测
public static final int ASF_UPDATE_FACEDATA = 0x00002000;      //人脸信息检测
```

说明：

1. 人脸识别时一般都需要`ASF_FACE_DETECT`和`ASF_FACERECOGNITION`这两个属性。
2. 需要防止纸张、屏幕等攻击可以传入`ASF_LIVENESS`和`ASF_IR_LIVENESS`，`RGB`和`IR`活体检测根据用户的摄像头类型及实际的业务需求来决定如何选择。
3. `ASF_AGE`/`ASF_GENDER`/`ASF_FACE3DANGLE`根据业务需求进行选择即可。

> 这些属性均是以常量值进行定义，可通过`|`位运算符进行组合使用。
>
> 例如：int combinedMask = FaceEngine.ASF_FACE_DETECT | FaceEngine.ASF_FACERECOGNITION | FaceEngine.ASF_LIVENESS;

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2`错误码列表。

示例代码

```
private int maxFaceNum = 5; //人脸数取值范围[1-10]
// 如下的组合，初始化的功能包含：人脸检测、人脸识别、RGB活体检测、年龄、性别、人脸3D角度
private int initMask = FaceEngine.ASF_FACE_DETECT |
FaceEngine.ASF_FACERECOGNITION | FaceEngine.ASF_LIVENESS | FaceEngine.ASF_AGE |
FaceEngine.ASF_FACE3DANGLE | FaceEngine.ASF_GENDER | FaceEngine.ASF_FACE3DANGLE;

faceEngine = new FaceEngine();
code = faceEngine.init(getApplicationContext(),
DetectMode.ASF_DETECT_MODE_VIDEO, DetectFaceOrientPriority.ASF_OP_0_ONLY,
maxFaceNum, initMask);
if (code != ErrorInfo.MOK) {
    Toast.makeText(this, "init failed, code is : " + code,
Toast.LENGTH_SHORT).show();
} else {
    Log.i(TAG, "init success");
}
```

3.7.8 人脸检测

功能描述

检测人脸信息。

> 该功能依赖初始化的模式选择，VIDEO模式下使用的是人脸追踪功能，IMAGE模式下使用的是人脸检测功能。初始化中`detectFaceOrientPriority`、`detectFaceMaxNum`参数的设置，对能否检测到人脸以及检测到几张人脸都有决定性的作用。

3.7.8.1 detectFaces（传入分离的图像信息数据）

方法

```
int detectFaces(
    byte[] data,
    int width,
    int height,
    int format,
    List<FaceInfo> faceInfoList
)
```

参数说明

参数	类型	描述
data	in	图像数据
width	in	图像宽度，为4的倍数
height	in	图像高度，在NV21格式下要求为2的倍数； BGR24/GRAY/DEPTH_U16格式无限制；
format	in	图像的颜色格式
faceInfoList	out	检测到的人脸信息（新增睁闭眼、是否戴眼镜）

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2`错误码列表。

> `detectFaceMaxNum` 参数的设置，对能否检测到人脸以及检测到几张人脸都有决定性的作用。

示例代码

> - 对Camera的数据进行检测

```
List<FaceInfo> faceInfoList = new ArrayList<>();
int code = faceEngine.detectFaces(nv21, previewSize.width, previewSize.height,
FaceEngine.CP_PAF_NV21, faceInfoList);
if (code == ErrorInfo.MOK && faceInfoList.size() > 0) {
    Log.i(TAG, "detectFaces, face num is : "+ faceInfoList.size());
} else {
    Log.i(TAG, "no face detected, code is : " + code);
}
```

> - 对静态图片进行检测。若图像数据宽高不符合要求，可使用ArcSoftImageUtil中的getAlignedBitmap函数获取对齐后的图像。

```
// 原始图像
Bitmap originalBitmap = BitmapFactory.decodeFile("图片地址");
// 获取宽高符合要求的图像
Bitmap bitmap = ArcSoftImageUtil.getAlignedBitmap(originalBitmap, true);
// 为图像数据分配内存
byte[] bgr24 = ArcSoftImageUtil.createImageData(bitmap.getWidth(),
bitmap.getHeight(), ArcSoftImageFormat.BGR24);
// 图像格式转换
int transformCode = ArcSoftImageUtil.bitmapToImageData(bitmap, bgr24,
ArcSoftImageFormat.BGR24);
if (transformCode != ArcSoftImageUtilError.CODE_SUCCESS) {
    Log.i(TAG, "transform failed, code is : " + transformCode);
    return;
}
List<FaceInfo> faceInfoList = new ArrayList<>();
int code = faceEngine.detectFaces(bgr24, width, height, FaceEngine.CP_PAF_BGR24,
faceInfoList);
if (code == ErrorInfo.MOK && faceInfoList.size() > 0) {
    Log.i(TAG, "detectFaces, face num is : "+ faceInfoList.size());
} else {
    Log.i(TAG, "no face detected, code is : " + code);
}
```

3.7.8.2 detectFaces（传入ArcSoftImageInfo图像信息数据）

方法

```
int detectFaces(
    ArcSoftImageInfo arcSoftImageInfo,
    List<FaceInfo> faceInfoList
)
```

参数说明

参数	类型	描述
arcSoftImageInfo	in	图像信息数据
faceInfoList	out	检测到的人脸信息

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2`错误码列表。

示例代码

> - 对分通道的数据进行检测

```
// 原始图像是NV21，并分为两个通道

// y通道
byte[] y = planeY;
// vu通道
byte[] vu = planeVU;
// y通道的步长
int yStride = strideOfPlaneY;
// vu通道的步长
int vuStride = strideOfPlaneVU;
// 组成ArcSoftImageInfo图像信息
ArcSoftImageInfo arcSoftImageInfo = new
ArcSoftImageInfo(width,height,FaceEngine.CP_PAF_NV21,new byte[][]{y,vu},new
int[]{yStride , vuStride});
List<FaceInfo> faceInfoList = new ArrayList<>();
// 传入ArcSoftImageInfo对象进行人脸检测
int code = faceEngine.detectFaces(arcSoftImageInfo, faceInfoList);
if (code == ErrorInfo.MOK && faceInfoList.size() > 0) {
    Log.i(TAG, "detectFaces, face num is : "+ faceInfoList.size());
} else {
    Log.i(TAG, "no face detected, code is : " + code);
}
```

3.7.9 更新人脸数据

功能描述

更新人脸数据。

> 该接口主要用于在需要修改人脸框时候更新人脸数据，用于之后的算法检测。一般常用与双目摄像头对齐，对齐之后的人脸框传入该接口更新人脸数据用于之后的红外活体检测。

3.7.9.1 updateFaceData（传入分离的图像信息数据）

方法

```
int updateFaceData(
    byte[] data,
    int width,
    int height,
    int format,
    List<FaceInfo> faceInfoList
)
```

参数说明

参数	类型	描述
data	in	图像数据
width	in	图像宽度，为4的倍数
height	in	图像高度，在NV21格式下要求为2的倍数； BGR24/GRAY/DEPTH_U16格式无限制；
format	in	图像的颜色格式；
faceInfoList	in/out	该参数即是入参也是出参，会更新人脸数据(faceData)；

返回值

成功返回`ErrorInfo.MOK`，失败详见 4.2 错误码列表。

示例代码

```
List<FaceInfo> faceInfoList = new ArrayList<>();
//经过detectFaces接口的处理输出 faceInfoList
int code = faceEngine.updateFaceData(nv21, previewSize.width,
previewSize.height, FaceEngine.CP_PAF_NV21, faceInfoList);
if (code == ErrorInfo.MOK && faceInfoList.size() > 0) {
    Log.i(TAG, "updateFaceData, face num is : "+ faceInfoList.size());
} else {
    Log.i(TAG, "code is : " + code);
}
```

3.7.9.2 updateFaceData（传入ArcSoftImageInfo图像信息数据）

方法

```
int updateFaceData(
    ArcSoftImageInfo arcSoftImageInfo,
    List<FaceInfo> faceInfoList
)
```

参数说明

参数	类型	描述
arcSoftImageInfo	in	图像信息数据
faceInfoList	out	检测到的人脸信息

返回值

成功返回`ErrorInfo.MOK`，失败详见 4.2 错误码列表。

示例代码

> - 对分通道的数据进行检测

```
// 原始图像是NV21，并分为两个通道
```

```
// y通道
byte[] y = planeY;
// vu通道
byte[] vu = planeVU;
// y通道的步长
int yStride = strideOfPlaneY;
// vu通道的步长
int vuStride = strideOfPlaneVU;
// 组成ArcSoftImageInfo图像信息
ArcSoftImageInfo arcSoftImageInfo = new
ArcSoftImageInfo(width,height,FaceEngine.CP_PAF_NV21,new byte[][]{y,vu},new
int[]{yStride , vuStride});
List<FaceInfo> faceInfoList = new ArrayList<>();
// 经过detectFaces接口的处理输出 faceInfoList
// 传入ArcSoftImageInfo对象进行人脸检测
int code = faceEngine.updateFaceData(arcSoftImageInfo, faceInfoList);
if (code == ErrorInfo.MOK && faceInfoList.size() > 0) {
    Log.i(TAG, "updateFaceData, face num is : "+ faceInfoList.size());
} else {
    Log.i(TAG, "code is : " + code);
}
```

3.7.10 图像质量检测

功能描述

根据人脸信息，对图像的指定区域进行图像质量检测。

3.7.10.1 imageQualityDetect（传入分离的图像信息数据）

方法

```
public int imageQualityDetect(
    byte[] data,
    int width,
    int height,
    int format,
    FaceInfo faceInfo,
    int isMask,
    ImageQualitySimilar imageQualitySimilar
)
```

参数说明

参数	类型	描述
data	in	图像数据
width	in	图片宽度，为4的倍数
height	in	图片高度，在NV21格式下要求为2的倍数； BGR24/GRAY/DEPTH_U16格式无限制；
format	in	图像的颜色格式
faceInfo	in	单人脸信息
isMask	in	是否戴口罩
imageQualitySimilar	out	图像质量置信度

返回值

成功返回`ErrorInfo.MOK`，失败详见 4.2 错误码列表。

示例代码

> 依赖`detectFaces`接口成功检测到人脸，将检测到的人脸信息和使用的图像信息传入到该接口进行图像质量检测。

```
// 首先进行人脸检测
List<FaceInfo> faceInfoList = new ArrayList<>();
int code = faceEngine.detectFaces(nv21, width, height, FaceEngine.CP_PAF_NV21,
faceInfoList);
......
int mask = 0; //这里应该是获取口罩检测的结果，仅在此定义为0
ImageQualitySimilar imageQualitySimilar = new ImageQualitySimilar();
// 对人脸信息进行图像质量检测
int imageQualityDetectCode = faceEngine.imageQualityDetect(nv21,
previewSize.width, previewSize.height, FaceEngine.CP_PAF_NV21,
faceInfoList.get(0),mask, imageQualitySimilar);
if (imageQualityDetectCode == ErrorInfo.MOK) {
    Log.i(TAG, "imageQualityDetect success: imageQuality Of face is : " +
imageQualitySimilar.getScore());
} else {
    Log.e(TAG, "imageQualityDetect code: " + imageQualityDetectCode);
}
```

3.7.10.2 imageQualityDetect（传入ArcSoftImageInfo图像信息数据）

方法

```
public int imageQualityDetect(
    ArcSoftImageInfo arcSoftImageInfo,
    FaceInfo faceInfo,
    int isMask,
    ImageQualitySimilar imageQualitySimilar
)
```

参数说明

参数	类型	描述
arcSoftImageInfo	in	图像数据
faceInfo	in	单人脸信息
isMask	in	是否戴口罩
imageQualitySimilar	out	图像质量置信度

返回值

成功返回`ErrorInfo.MOK`，失败详见 4.2 错误码列表。

示例代码

> 依赖`detectFaces`接口成功检测到人脸，将检测到的人脸信息和使用的图像信息传入到该接口进行图像质量检测。

```
// 首先进行人脸检测

// 原始图像是NV21，并分为两个通道

// y通道
byte[] y = planeY;
// vu通道
byte[] vu = planeVU;
// y通道的步长
int yStride = strideOfPlaneY;
// vu通道的步长
int vuStride = strideOfPlaneVU;
// 组成ArcSoftImageInfo图像信息
ArcSoftImageInfo arcSoftImageInfo = new
ArcSoftImageInfo(width,height,FaceEngine.CP_PAF_NV21,new byte[][]{y,vu},new
int[]{yStride , vuStride});
List<FaceInfo> faceInfoList = new ArrayList<>();
// 传入ArcSoftImageInfo对象进行人脸检测
int code = faceEngine.detectFaces(arcSoftImageInfo, faceInfoList);
......
int mask = 0; //这里应该是获取口罩检测的结果，仅在此定义为0
ImageQualitySimilar imageQualitySimilar = new ImageQualitySimilar();
// 对人脸信息进行图像质量检测
int imageQualityDetectCode = faceEngine.imageQualityDetect(arcSoftImageInfo,
faceInfoList.get(0),mask, imageQualitySimilar);
if (imageQualityDetectCode == ErrorInfo.MOK) {
    Log.i(TAG, "imageQualityDetect success: imageQuality Of face is : " +
imageQualitySimilar.getScore());
} else {
    Log.e(TAG, "imageQualityDetect code: " + imageQualityDetectCode);
}
```

3.7.11 人脸特征提取

功能描述

单人脸特征信息提取。

3.7.11.1 extractFaceFeature（传入分离的图像信息数据）

方法

```
int extractFaceFeature(
    byte[] data,
    int width,
    int height,
    int format,
    FaceInfo faceInfo,
    ExtractType extractType,
    int mask,
    FaceFeature feature
)
```

参数说明

参数	类型	描述
data	in	图像数据
width	in	图片宽度，为4的倍数
height	in	图片高度，在NV21格式下要求为2的倍数； BGR24/GRAY/DEPTH_U16格式无限制；
format	in	图像的颜色格式
faceInfo	in	人脸信息（人脸框、人脸角度）
extractType	in	特征提取类型， ExtractType.REGISTER：注册 ExtractType.RECOGNIZE：识别
mask	in	MaskInfo.WORN：已佩戴口罩 其他：未佩戴口罩
feature	out	提取到的人脸特征信息

返回值

成功返回`ErrorInfo.MOK`，失败详见 4.2 错误码列表。

示例代码

> 依赖`detectFaces`接口成功检测到人脸，将检测到的人脸信息取单张人脸信息和使用的图像信息传入到该接口进行特征提取。

> 注：若应用场景不需要支持戴口罩模式下的人脸比对以及对人脸入库性能有较高要求时，可在识别和注册时特征提取接口统一设置为`extractType=RECOGNIZE`与`mask=0`。后续如需兼容口罩场景，人脸注册需使用`extractType=REGISTER`与`mask=0`模式重新提取入库。

```
// 首先进行人脸检测
List<FaceInfo> faceInfoList = new ArrayList<>();
int code = faceEngine.detectFaces(nv21, width, height, FaceEngine.CP_PAF_NV21,
faceInfoList);
......
```

```
// 这里取检测到的第一张人脸进行特征提取，也可以对人脸大小进行排序，取最大人脸检测，可根据实际应用场景进行选择。如需提取所有人脸的人脸特征数据，循环处理即可
if (code == ErrorInfo.MOK && faceInfoList.size() > 0) {
    FaceFeature faceFeature = new FaceFeature();
    // 在FaceFeature的二进制数组中保存获取到的人脸特征数据
    // 注意：注册时建议使用未佩戴口罩的清晰人脸
    int extractCode = faceEngine.extractFaceFeature(nv21, width, height,FaceEngine.CP_PAF_NV21, faceInfoList.get(0), ExtractType.REGISTER, MaskInfo.NOT_WORN, faceFeature);
    if (extractCode == ErrorInfo.MOK){
        Log.i(TAG, "extract face feature success");
    }else{
        Log.i(TAG, "extract face feature failed, code is : " + extractCode);
    }
}else {
    Log.i(TAG, "no face detected, code is : " + code);
}
```

3.7.11.2 extractFaceFeature（传入ArcSoftImageInfo图像信息数据）

方法

```
int extractFaceFeature(
    ArcSoftImageInfo arcSoftImageInfo,
    FaceInfo faceInfo,
    ExtractType extractType,
    int mask,
    FaceFeature feature
)
```

参数说明

参数	类型	描述
arcSoftImageInfo	in	图像数据
faceInfo	in	人脸信息（人脸框、人脸角度）
extractType	in	特征提取类型， ExtractType.REGISTER：注册 ExtractType.RECOGNIZE：识别
mask	in	MaskInfo.WORN：已佩戴口罩 其他：未佩戴口罩
feature	out	提取到的人脸特征信息

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2`错误码列表。

示例代码

> 依赖`detectFaces`接口成功检测到人脸，将检测到的人脸信息取单张人脸信息和使用的图像信息传入到该接口进行特征提取。

> 注：若应用场景不需要支持戴口罩模式下的人脸比对以及对人脸入库性能有较高要求时，可在识别和注册时特征提取接口统一设置为`extractType=RECOGNIZE`与`mask=0`。后续如需兼容口罩场景，人脸注册需使用`extractType=REGISTER`与`mask=0`模式重新提取入库。

```
// 首先进行人脸检测

// 原始图像是NV21，并分为两个通道

// y通道
byte[] y = planeY;
// vu通道
byte[] vu = planeVU;
// y通道的步长
int yStride = strideOfPlaneY;
// vu通道的步长
int vuStride = strideOfPlaneVU;
// 组成ArcSoftImageInfo图像信息
ArcSoftImageInfo arcSoftImageInfo = new
ArcSoftImageInfo(width,height,FaceEngine.CP_PAF_NV21,new byte[][]{y,vu},new
int[]{yStride , vuStride});
List<FaceInfo> faceInfoList = new ArrayList<>();
// 传入ArcSoftImageInfo对象进行人脸检测
int code = faceEngine.detectFaces(arcSoftImageInfo, faceInfoList);
......

// 这里取检测到的第一张人脸进行特征提取，也可以对人脸大小进行排序，取最大人脸检测，可根据实际应用场景进行选择。如需提取所有人脸的人脸特征数据，循环处理即可
if (code == ErrorInfo.MOK && faceInfoList.size() > 0) {
    FaceFeature faceFeature = new FaceFeature();
    // 在FaceFeature的二进制数组中保存获取到的人脸特征数据
    // 注意：注册时建议使用未佩戴口罩的清晰人脸
    int extractCode = faceEngine.extractFaceFeature(arcSoftImageInfo,
faceInfoList.get(0), ExtractType.REGISTER, MaskInfo.NOT_WORN, faceFeature);
    if (extractCode == ErrorInfo.MOK){
        Log.i(TAG, "extract face feature success");
    }else{
        Log.i(TAG, "extract face feature failed, code is : " + extractCode);
    }
}else {
    Log.i(TAG, "no face detected, code is : " + code);
}
```

3.7.12 人脸特征比对

功能描述

人脸特征比对，输出相似度。

3.7.12.1 compareFaceFeature（可选择比对模型）

方法

```
int compareFaceFeature (
    FaceFeature feature1,
    FaceFeature feature2,
    CompareModel compareModel,
    FaceSimilar faceSimilar
)
```

参数说明

参数	类型	描述
feature1	in	人脸特征
feature2	in	人脸特征
compareModel	in	比对模型
faceSimilar	out	比对相似度

返回值

成功返回`ErrorInfo.MOK`，失败详见 4.2 错误码列表。

示例代码

> - 人证比对

```
FaceSimilar faceSimilar = new FaceSimilar();
int compareCode = faceEngine.compareFaceFeature(previewFaceFeature,
idCardFaceFeature, CompareModel.ID_CARD, faceSimilar);
if (compareCode == ErrorInfo.MOK){
    //两个人脸的相似度
    int score = faceSimilar.getScore();
}else{
    Log.i(TAG, "compare failed, code is : " + compareCode);
}
```

3.7.12.2 compareFaceFeature（默认LIFE_PHOTO比对模型）

方法

```
int compareFaceFeature (
    FaceFeature feature1,
    FaceFeature feature2,
    FaceSimilar faceSimilar
)
```

参数说明

参数	类型	描述
feature1	in	人脸特征
feature2	in	人脸特征
faceSimilar	out	比对相似度

返回值

成功返回`ErrorInfo.MOK`，失败详见 4.2 错误码列表。

示例代码

> - 多人脸特征比对

```
/**
  * 在人脸特征库中搜索，获取相似度最高者的下标
  *
  * @param faceFeature     人脸特征数据
  * @param faceFeatureList 人脸特征库
  * @return 相似度最高者的下标
  */
public int searchFace(FaceFeature faceFeature, List<FaceFeature>
faceFeatureList) {
    FaceSimilar faceSimilar = new FaceSimilar();
    float maxSimilar = 0;
    int maxSimilarIndex = -1;
    for (int i = 0; i < faceFeatureList.size(); i++) {
    // 同faceEngine.compareFaceFeature(faceFeature, faceFeatureList.get(i),
CompareModel.LIFE_PHOTO, faceSimilar);
        int compareCode = faceEngine.compareFaceFeature(faceFeature,
faceFeatureList.get(i), faceSimilar);
        if (compareCode == ErrorInfo.MOK) {
            if (faceSimilar.getScore() > maxSimilar) {
                maxSimilar = faceSimilar.getScore();
                maxSimilarIndex = i;
            }
        } else {
            Log.i(TAG, "compare failed, code is : " + compareCode);
        }
    }
    return maxSimilarIndex;
}
```

3.7.13 setLivenessParam

功能描述

设置RGB/IR活体阈值，若不设置内部默认RGB：0.5, IR：0.7。

方法

```
int setLivenessParam(LivenessParam livenessParam)
```

参数说明

参数	类型	描述
LivenessParam	in	活体阈值信息，推荐值： rgbThreshold：0.5 irThreshold：0.7

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2`错误码列表。

示例代码

```
float rgbThreshold = 0.5f;
float irThreshold =  0.7f;
LivenessParam livenessParam = new LivenessParam(rgbThreshold, irThreshold);
int code = faceEngine.setLivenessParam(livenessParam);
Log.i(TAG,"setLivenessParam code is : " + code);
```

3.7.14 setFaceShelterParam

功能描述

设置人脸遮挡阈值，若不设置，内部默认值为0.8。

方法

```
int setFaceShelterParam(float faceShelterThreshold)
```

参数说明

参数	类型	描述
faceShelterThreshold	in	人脸遮挡阈值，推荐值0.8，可根据实际需求调整

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2`错误码列表。

示例代码

```
int code = faceEngine.setFaceShelterParam(0.8f);
if(code == ErrorInfo.MOK){
    Log.i(TAG,"setFaceShelterParam success");
}else{
    Log.i(TAG,"setFaceShelterParam failed, code is : " + code);
}
```

3.7.15 人脸属性检测

> 该接口仅支持可见光图像检测。

功能描述

人脸属性检测（年龄/性别/人脸3D角度），最多支持10张人脸信息检测，超过部分返回未知（活体仅支持单张人脸检测，超出返回未知），该接口不支持IR图像检测，且额头关键点信息检测仅支持0度人脸。

3.7.15.1 process（传入分离的图像信息数据）

方法

```
int process(
    byte[] data,
    int width,
    int height,
    int format,
    List<FaceInfo> faceInfoList,
    int combinedMask
)
```

参数说明

参数	类型	描述
data	in	图像数据
width	in	图片宽度，为4的倍数
height	in	图片高度，在NV21格式下要求为2的倍数； BGR24格式无限制；
format	in	支持NV21/BGR24
faceInfoList	in	人脸信息列表
combinedMask	in	1.检测的属性（ASF_AGE、ASF_GENDER、 ASF_FACE3DANGLE、ASF_LIVENESS），支持多选 2.检测的属性须在引擎初始化接口的combinedMask参数中启用

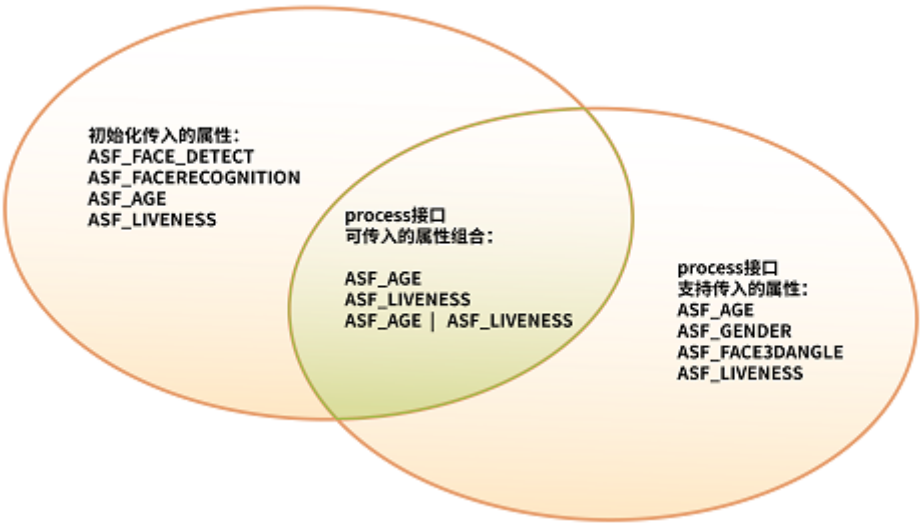
重要参数说明

- **combinedMask**

> process接口中支持检测`ASF_AGE`、`ASF_GENDER`、`ASF_FACE3DANGLE`、`ASF_LIVENESS`四种属性，但是想检测这些属性，必须在初始化引擎接口中对想要检测的属性进行初始化。

关于初始化接口中`combinedMask`和`process`接口中`combinedMask`参数之间的关系，举例进行详细说明，如下图所示：

1. `process`接口中`combinedMask`支持传入的属性有`ASF_AGE`、`ASF_GENDER`、`ASF_FACE3DANGLE`、`ASF_LIVENESS`。
2. 初始化中传入了`ASF_FACE_DETECT`、`ASF_FACERECOGNITION`、`ASF_AGE`、`ASF_LIVENESS`属性。
3. process可传入属性组合只有`ASF_AGE`、`ASF_LIVENESS`、`ASF_AGE | ASF_LIVENESS`。



返回值

成功返回`ErrorInfo.MOK`，失败详见 `4.2` 错误码列表。

示例代码

> 下面代码要求初始化引擎时包含了 ASF_AGE | ASF_GENDER | ASF_FACE3DANGLE | ASF_LIVENESS

```
int processMask = FaceEngine.ASF_AGE | FaceEngine.ASF_GENDER |
FaceEngine.ASF_FACE3DANGLE;
int faceProcessCode = faceEngine.process(nv21, width, height,
FaceEngine.CP_PAF_NV21, faceInfoList, processMask);
if (faceProcessCode == ErrorInfo.MOK) {
    Log.i(TAG, "process success");
}else{
    Log.i(TAG, "process failed, code is : " + faceProcessCode);
}
```

3.7.15.2 process（传入ArcSoftImageInfo图像信息数据）

方法

```
int process(
    ArcSoftImageInfo arcSoftImageInfo,
    List<FaceInfo> faceInfoList,
    int combinedMask
)
```

参数说明

参数	类型	描述
arcSoftImageInfo	in	图像信息数据
faceInfoList	in	人脸信息列表
combinedMask	in	1.检测的属性（ASF_AGE、ASF_GENDER、 ASF_FACE3DANGLE、ASF_LIVENESS），支持多选 2.检测的属性须在引擎初始化接口的combinedMask参数中启用

返回值

成功返回`ErrorInfo.MOK`，失败详见 4.2 错误码列表。

示例代码

> 下面代码要求初始化引擎时包含了 ASF_AGE | ASF_GENDER | ASF_FACE3DANGLE | ASF_LIVENESS

```
// 原始图像是NV21，并分为两个通道

// y通道
byte[] y = planeY;
// vu通道
byte[] vu = planeVU;
// y通道的步长
int yStride = strideOfPlaneY;
// vu通道的步长
int vuStride = strideOfPlaneVU;
// 组成ArcSoftImageInfo图像信息
ArcSoftImageInfo arcSoftImageInfo = new ArcSoftImageInfo(width, height,
FaceEngine.CP_PAF_NV21, new byte[][]{y, vu}, new int[]{yStride, vuStride});
List<FaceInfo> faceInfoList = new ArrayList<>();
// 传入ArcSoftImageInfo对象进行人脸检测
int code = faceEngine.detectFaces(arcSoftImageInfo, faceInfoList);
if (code == ErrorInfo.MOK && faceInfoList.size() > 0){
    int processMask = FaceEngine.ASF_AGE | FaceEngine.ASF_GENDER
|FaceEngine.ASF_FACE3DANGLE;
    int faceProcessCode = faceEngine.process(arcSoftImageInfo, faceInfoList,
processMask);
    if (faceProcessCode == ErrorInfo.MOK) {
        Log.i(TAG, "process success");
    }else{
        Log.i(TAG, "process failed, code is : " + faceProcessCode);
    }
}
```

3.7.16 getAge

功能描述

获取年龄信息。

方法

```
int getAge(List<AgeInfo> ageInfoList)
```

参数说明

参数	类型	描述
ageInfoList	out	检测到的年龄信息数组

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2` 错误码列表。

示例代码

> 前提：`process`接口调用成功，且combinedMask参数中包含`ASF_AGE`属性

```
List<AgeInfo> ageInfoList = new ArrayList<>();
int ageCode = faceEngine.getAge(ageInfoList);
// 获取第一个人脸的年龄信息，多人脸情况进行循环即可
if (ageCode == ErrorInfo.MOK && ageInfoList.size() > 0) {
    if (ageInfoList.size() > 0){
        Log.i(TAG, "age of the first face is : " + ageInfoList.get(0).getAge());
    }else{
        Log.i(TAG, "no face processed");
    }
}else {
    Log.i(TAG, "get age failed, code is : " + ageCode);
}
```

3.7.17 getGender

功能描述

获取性别信息。

方法

```
int getGender(List<GenderInfo> genderInfoList)
```

参数说明

参数	类型	描述
genderInfoList	out	检测到的性别信息数组

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2` 错误码列表。

示例代码

> 前提：`process`接口调用成功，且combinedMask参数中包含`ASF_GENDER`属性

```
List<GenderInfo> genderInfoList = new ArrayList<>();
int genderCode = faceEngine.getGender(genderInfoList);
// 获取第一个人脸的性别信息，多人脸情况进行循环即可
if (genderCode == ErrorInfo.MOK) {
    if (genderInfoList.size() > 0){
        Log.i(TAG, "gender of the first face is : " +
genderInfoList.get(0).getGender());
    }else{
        Log.i(TAG, "no face processed");
    }
}else {
    Log.i(TAG, "get gender failed, code is : " + genderCode);
}
```

3.7.18 getFace3DAngle

功能描述

获取3D角度信息。

方法

```
int getFace3DAngle(List<Face3DAngle> face3DAngleList)
```

参数说明

参数	类型	描述
face3DAngleList	out	检测到的3D角度信息数组

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2`错误码列表。

示例代码

> 前提：`process`接口调用成功，且combinedMask参数中包含`ASF_FACE3DANGLE`属性

```
List<Face3DAngle> face3DAngleList = new ArrayList<>();
int face3DAngleCode = faceEngine.getFace3DAngle(face3DAngleList);
// 获取第一个人脸的角度信息，多人脸情况进行循环即可
if (face3DAngleCode == ErrorInfo.MOK && face3DAngleList.size() > 0) {
    if (face3DAngleList.size() > 0){
        // 可使用getYaw、getRoll、getPitch等函数获取三维角度
        int yaw = face3DAngleList.get(0).getYaw();
        int roll = face3DAngleList.get(0).getRoll();
        int pitch = face3DAngleList.get(0).getPitch();

        // 查看status
        int status = face3DAngleList.get(0).getStatus();

        Log.i(TAG, "getFace3DAngle, status of the first face is : " + status);
        Log.i(TAG, "getFace3DAngle, yaw of the first face is : " + yaw);
        Log.i(TAG, "getFace3DAngle, roll of the first face is : " + roll);
        Log.i(TAG, "getFace3DAngle, pitch of the first face is : " + pitch);
    }else{
        Log.i(TAG, "no face processed");
```

```
    }
}else {
    Log.i(TAG, "get face3DAngle failed, code is : " + face3DAngleCode);
}
```

3.7.19 getLiveness

功能描述

获取RGB活体信息。

方法

```
int getLiveness(List<LivenessInfo> livenessInfoList)
```

参数说明

参数	类型	描述
livenessInfoList	out	检测到的活体信息

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2`错误码列表。

示例代码

> 前提：`process`接口调用成功，且combinedMask参数中包含`ASF_LIVENESS`属性

```
List<LivenessInfo> livenessInfoList = new ArrayList<>();
int livenessCode = faceEngine.getLiveness(livenessInfoList);
// RGB活体不支持多人脸，因此只能拿第1个活体信息
if (livenessCode == ErrorInfo.MOK) {
    if (livenessInfoList.size() > 0){
        Log.i(TAG, "liveness of the first face is : " +
livenessInfoList.get(0).getLiveness());
    }else{
        Log.i(TAG, "no face processed");
    }
}else {
    Log.i(TAG, "get liveness failed, code is : " + livenessCode);
}
```

3.7.20 getMaskInfo

功能描述

获取口罩佩戴信息。

方法

```
public int getMask(List<MaskInfo> maskInfoList)
```

参数说明

参数	类型	描述
maskInfoList	out	口罩佩戴信息列表，用于输出口罩佩戴信息结果数据

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2` 错误码列表。

示例代码

> 前提：`process`接口调用成功，且combinedMask参数中包含`ASF_MASK_DETECT`属性

```
List<MaskInfo> maskInfoList = new ArrayList<>();
int maskCode = faceEngine.getMask(maskInfoList);
if (maskCode == ErrorInfo.MOK) {
    if (maskInfoList.size() > 0){
        Log.i(TAG, "maskInfo of the first face is : " +
maskInfoList.get(0).getMask());
    }else{
        Log.i(TAG, "no face processed");
    }
}else {
    Log.i(TAG, "get mask failed, code is : " + maskCode);
}
```

3.7.21 getFaceLandmark

功能描述

获取额头关键点信息。

方法

```
public int getFaceLandmark(List<LandmarkInfo> faceLandmarkList)
```

参数说明

参数	类型	描述
faceLandmarkList	out	额头关键点信息列表，用于输出额头关键点信息结果数据

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2` 错误码列表。

示例代码

> 前提：`process`接口调用成功，且combinedMask参数中包含`ASF_FACELANDMARK`属性

```
List<LandmarkInfo> landmarkList = new ArrayList<>();
int landmarkCode = faceEngine.getFaceLandmark(landmarkList);
if (landmarkCode == ErrorInfo.MOK) {
    if (landmarkList.size() > 0){
        Log.i(TAG, "landmark of the first face is : " +
Arrays.asList(landmarkInfoList.get(0).getLandmarks()));
    }else{
        Log.i(TAG, "no face processed");
    }
}else {
    Log.i(TAG, "get face landmark failed, code is : " + landmarkCode);
}
```

3.7.22 IR活体检测

功能描述

该接口仅支持单人脸`IR`活体检测，超出返回未知。

3.7.22.1 processIr（传入分离的图像信息数据）

方法

```
int processIr(
    byte[] data,
    int width,
    int height,
    int format,
    List<FaceInfo> faceInfoList,
    int combinedMask
)
```

参数说明

参数	类型	描述
data	in	图像数据
width	in	图片宽度，为4的倍数
height	in	图片高度
format	in	图像颜色格式
faceInfoList	in	人脸信息列表
combinedMask	in	目前仅支持`ASF_IR_LIVENESS`

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2`错误码列表

示例代码

> 下面代码要求初始化引擎时包含了 ASF_IR_LIVENESS

```
int processIrCode = faceEngine.processIr(irData, width, height,
FaceEngine.CP_PAF_NV21, faceInfoList,FaceEngine.ASF_IR_LIVENESS);
if (processIrCode == ErrorInfo.MOK) {
    Log.i(TAG, "processIr success" );
}else{
    Log.i(TAG, "processIr failed, code is " + processIrCode);
}
```

3.7.22.2 processIr（传入ArcSoftImageInfo图像信息数据）

方法

```
int processIr(
    ArcSoftImageInfo arcSoftImageInfo
    List<FaceInfo> faceInfoList,
    int combinedMask
)
```

参数说明

参数	类型	描述
arcSoftImageInfo	in	图像信息数据
faceInfoList	in	人脸信息列表
combinedMask	in	目前仅支持`ASF_IR_LIVENESS`

返回值

成功返回`ErrorInfo.MOK`，失败详见`4.2`错误码列表。

示例代码

> 下面代码要求初始化引擎时包含了 ASF_IR_LIVENESS

```
int processIrCode = faceEngine.processIr(arcSoftImageInfo, faceInfoList,
FaceEngine.ASF_IR_LIVENESS);
if (processIrCode == ErrorInfo.MOK) {
    Log.i(TAG, "processIr success" );
}else{
    Log.i(TAG, "processIr failed, code is " + processIrCode);
}
```

3.7.23 getIrLiveness

功能描述

获取IR活体信息。

方法

```
int getIrLiveness(List<LivenessInfo> irLivenessInfoList)
```

参数说明

参数	类型	描述
irLivenessInfoList	out	检测到的IR活体信息

返回值

成功返回`ErrorInfo.MOK`，失败详见 `4.2` 错误码列表。

示例代码

> 前提：`processIr`接口调用成功，且combinedMask参数中包含`ASF_IR_LIVENESS`属性

```
List<LivenessInfo> irLivenessInfoList = new ArrayList<>();
int irLivenessCode = faceEngine.getIrLiveness(irLivenessInfoList);
// IR活体不支持多人脸，因此只能拿第1个活体信息
if (irLivenessCode == ErrorInfo.MOK) {
    if (irLivenessInfoList.size() > 0){
        Log.i(TAG, "irLiveness of the first face is : " +
irLivenessInfoList.get(0).getLiveness());
    }else{
        Log.i(TAG, "no face processed");
    }
}else {
    Log.i(TAG, "get irLiveness failed, code is : " + irLivenessCode);
}
```

3.7.24 getVersion

功能描述

获取SDK版本信息。

方法

```
int getVersion(VersionInfo versionInfo)
```

参数说明

参数	类型	描述
versionInfo	out	版本信息

返回值

成功返回`ErrorInfo.MOK`，失败详见 `4.2` 错误码列表。

示例代码

> 注意：无需激活或初始化即可调用。

```
VersionInfo versionInfo = new VersionInfo();
int code = FaceEngine.getVersion(versionInfo);
Log.i(TAG, "getVersion, code is : " + code + ", version is :" + versionInfo);
```

3.7.25 getRuntimeABI

功能描述

获取运行时架构。

方法

> 注意：无需激活或初始化即可调用。

```
RuntimeABI getRuntimeABI()
```

返回值

运行时架构。

3.7.26 unInit

功能描述

销毁SDK引擎。

方法

```
int unInit()
```

返回值

成功返回`ErrorInfo.MOK`，失败详见 `4.2` 错误码列表。

示例代码

```
int code = faceEngine.unInit();
Log.i(TAG, "unInit code is : " + code);
```

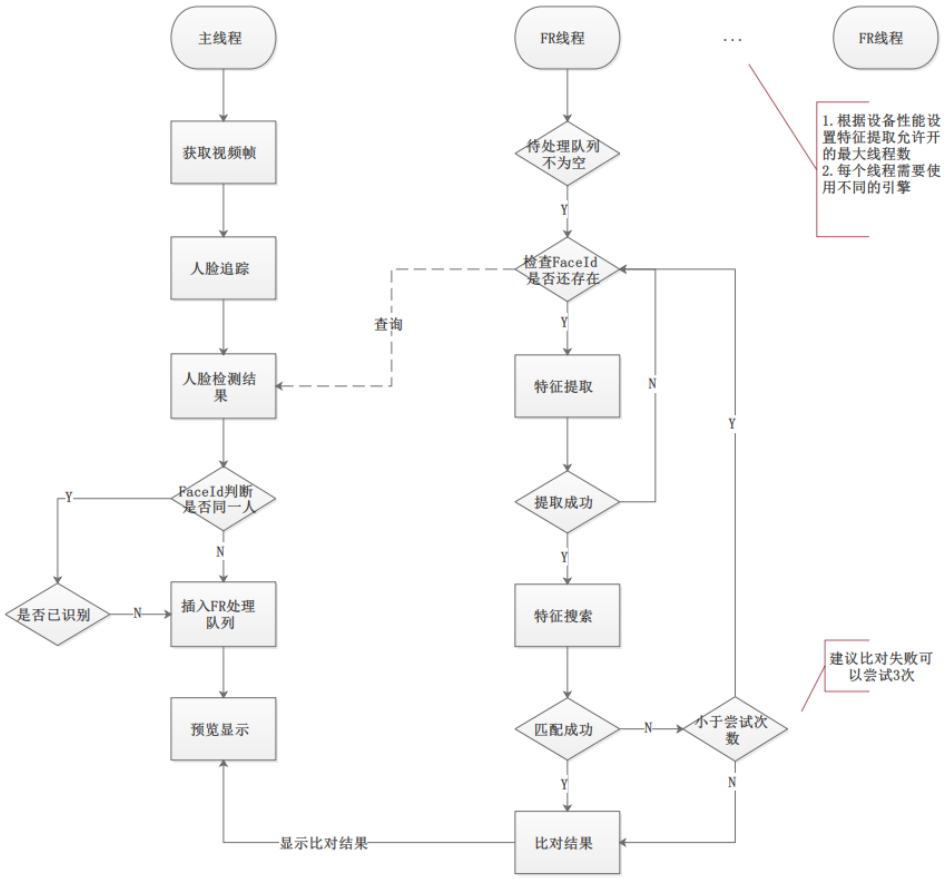
4. 常见问题

4.1 常用接入场景流程

一般情况下使用人脸识别SDK需对每一帧图像数据做人脸检测、特征提取等操作，但这样往往消耗系统资源，这里引入一些优化策略作为参考：

- `faceId`标记一张人脸从进入画面到离开画面这个值不变，可以使用`faceId`来判断，用户从进入画面到离开画面只需做一次人脸比对即可。
- 通过摄像头获取图像数据，人是动态移动的，图像质量不能保证，可以引入尝试策略，连续几次识别失败才认为比对失败，但不能无限制的尝试，在极限情况下可能会影响效果，推荐尝试3-5次即可。

4.1.1 常用比对流程



4.1.2 常用比对流程 + 活体


活体线程
...
活体线程
主线程
FR线程
...
FR线程
1. 根据设备性能设置活体检测允许开的最大线程数
2. 每个线程需要使用不同的引擎
活体检测使用IMAGE模式
获取视频帧
待处理队列不为空
人脸追踪
检查FaceId是否还存在
查询
人脸检测结果
查询
特征提取
提取成功
特征搜索
匹配成功
小于尝试次数
建议比对失败可以尝试3次
活体检测
FaceId判断是否同一人
检测成功
小于尝试次数
是否已识别
插入待处理队列
活体结果
输出结果
预览显示
输出结果
比对结果
Y
N

4.2 错误码列表

错误码名	十六进制	十进制	描述
MOK	0x0	0	成功
MERR_UNKNOWN	0x1	1	错误原因不明
MERR_INVALID_PARAM	0x2	2	无效的参数
MERR_UNSUPPORTED	0x3	3	引擎不支持
MERR_NO_MEMORY	0x4	4	内存不足
MERR_BAD_STATE	0x5	5	状态错误
MERR_USER_CANCEL	0x6	6	用户取消相关操作
MERR_EXPIRED	0x7	7	操作时间过期
MERR_USER_PAUSE	0x8	8	用户暂停操作
MERR_BUFFER_OVERFLOW	0x9	9	缓冲上溢
MERR_BUFFER_UNDERFLOW	0xA	10	缓冲下溢
MERR_NO_DISKSPACE	0xB	11	存贮空间不足
MERR_COMPONENT_NOT_EXIST	0xC	12	组件不存在
MERR_GLOBAL_DATA_NOT_EXIST	0xD	13	全局数据不存在
MERRP_IMGCODEC	0xE	14	暂未用到
MERR_FILE_GENERAL	0xF	15	暂未用到
MERR_FSDK_INVALID_APP_ID	0x7001	28673	无效的APP_ID
MERR_FSDK_INVALID_SDK_ID	0x7002	28674	无效的SDK_KEY
MERR_FSDK_INVALID_ID_PAIR	0x7003	28675	APP_ID和SDK_KEY不匹配
MERR_FSDK_MISMATCH_ID_AND_SDK	0x7004	28676	SDK_KEY和使用的SDK不匹配（注意：调用初始化引擎接口时，请确认激活接口传入的参数，并重新激活）
MERR_FSDK_SYSTEM_VERSION_UNSUPPORTED	0x7005	28677	系统版本不被当前SDK所支持
MERR_FSDK_FR_INVALID_MEMORY_INFO	0x12001	73729	无效的输入内存
MERR_FSDK_FR_INVALID_IMAGE_INFO	0x12002	73730	无效的输入图像参数
MERR_FSDK_FR_INVALID_FACE_INFO	0x12003	73731	无效的脸部信息
MERR_FSDK_FR_NO_GPU_AVAILABLE	0x12004	73732	当前设备无GPU可用
MERR_FSDK_FR_MISMATCHED_FEATURE_LEVEL	0x12005	73733	待比较的两个人脸特征的版本不一致
MERR_FSDK_FACEFEATURE_UNKNOWN	0x14001	81921	人脸特征检测错误未知
MERR_FSDK_FACEFEATURE_MEMORY	0x14002	81922	人脸特征检测内存错误
MERR_FSDK_FACEFEATURE_INVALID_FORMAT	0x14003	81923	人脸特征检测格式错误
MERR_FSDK_FACEFEATURE_INVALID_PARAM	0x14004	81924	人脸特征检测参数错误
MERR_FSDK_FACEFEATURE_LOW_CONFIDENCE_LEVEL	0x14005	81925	人脸特征检测结果置信度低
MERR_FSDK_FACEFEATURE_EXPIRED	0x14006	81926	人脸特征检测结果操作过期
MERR_FSDK_FACEFEATURE_MISSFACE	0x14007	81927	人脸特征检测人脸丢失人脸特征检测结果置信度低
MERR_FSDK_FACEFEATURE_NO_FACE	0x14008	81928	人脸特征检测没有人脸
MERR_FSDK_FACEFEATURE_FACEDATA	0x14009	81929	人脸特征检测人脸信息错误
MERR_ASF_EX_FEATURE_UNSUPPORTED_ON_INIT	0x15001	86017	Engine不支持的检测属性
MERR_ASF_EX_FEATURE_UNINITED	0x15002	86018	需要检测的属性未初始化
MERR_ASF_EX_FEATURE_UNPROCESSED	0x15003	86019	待获取的属性未在process中处理过
MERR_ASF_EX_FEATURE_UNSUPPORTED_ON_PROCESS	0x15004	86020	PROCESS不支持的检测属性，例如FR，有自己独立的处理函数
MERR_ASF_EX_INVALID_IMAGE_INFO	0x15005	86021	无效的输入图像
MERR_ASF_EX_INVALID_FACE_INFO	0x15006	86022	无效的脸部信息
MERR_ASF_ACTIVATION_FAIL	0x16001	90113	SDK激活失败，请打开读写权限
MERR_ASF_ALREADY_ACTIVATED	0x16002	90114	SDK已激活

错误码名	十六进制	十进制	描述
MERR_ASF_NOT_ACTIVATED	0x16003	90115	SDK未激活
MERR_ASF_SCALE_NOT_SUPPORT	0x16004	90116	detectFaceScaleVal不支持
MERR_ASF_ACTIVEFILE_SDKTYPE_MISMATCH	0x16005	90117	激活文件与SDK类型不匹配，请确认使用的sdk
MERR_ASF_DEVICE_MISMATCH	0x16006	90118	设备不匹配
MERR_ASF_UNIQUE_IDENTIFIER_ILLEGAL	0x16007	90119	唯一标识不合法
MERR_ASF_PARAM_NULL	0x16008	90120	参数为空
MERR_ASF_VERSION_NOT_SUPPORT	0x1600A	90122	版本不支持
MERR_ASF_SIGN_ERROR	0x1600B	90123	签名错误
MERR_ASF_DATABASE_ERROR	0x1600C	90124	激活信息保存异常
MERR_ASF_UNIQUE_CHECKOUT_FAIL	0x1600D	90125	唯一标识符校验失败
MERR_ASF_COLOR_SPACE_NOT_SUPPORT	0x1600E	90126	颜色空间不支持
MERR_ASF_IMAGE_WIDTH_HEIGHT_NOT_SUPPORT	0x1600F	90127	图片宽高不支持，宽度需四字节对齐
MERR_ASF_READ_PHONE_STATE_DENIED	0x16010	90128	android.permission.READ_PHONE_STATE权限被拒绝
MERR_ASF_ACTIVATION_DATA_DESTROYED	0x16011	90129	激活数据被破坏,请删除激活文件，重新进行激活
MERR_ASF_SERVER_UNKNOWN_ERROR	0x16012	90130	服务端未知错误
MERR_ASF_INTERNET_DENIED	0x16013	90131	android.permission.INTERNET权限被拒绝
MERR_ASF_ACTIVEFILE_SDK_MISMATCH	0x16014	90132	激活文件与SDK版本不匹配,请重新激活
MERR_ASF_DEVICEINFO_LESS	0x16015	90133	设备信息太少，不足以生成设备指纹
MERR_ASF_LOCAL_TIME_NOT_CALIBRATED	0x16016	90134	客户端时间与服务器时间（即北京时间）前后相差在30分钟以上
MERR_ASF_APPID_DATA_DECRYPT	0x16017	90135	数据校验异常
MERR_ASF_APPID_APPKEY_SDK_MISMATCH	0x16018	90136	传入的APP_ID和AppKey与使用的SDK版本不一致
MERR_ASF_NO_REQUEST	0x16019	90137	短时间大量请求会被禁止请求,30分钟之后解封
MERR_ASF_ACTIVE_FILE_NO_EXIST	0x1601A	90138	激活文件不存在
MERR_ASF_CURRENT_DEVICE_TIME_INCORRECT	0x1601B	90139	当前设备时间不正确，请调整设备时间
MERR_ASF_DETECT_MODEL_UNSUPPORTED	0x1601C	90140	检测模型不支持，请查看对应接口说明，使用当前支持的检测模型
MERR_ASF_NETWORK_COULDNT_RESOLVE_HOST	0x17001	94209	无法解析主机地址
MERR_ASF_NETWORK_COULDNT_CONNECT_SERVER	0x17002	94210	无法连接服务器
MERR_ASF_NETWORK_CONNECT_TIMEOUT	0x17003	94211	网络连接超时
MERR_ASF_NETWORK_UNKNOWN_ERROR	0x17004	94212	网络未知错误
MERR_ASF_ACTIVEKEY_COULDNT_CONNECT_SERVER	0x18001	98305	无法连接激活服务器
MERR_ASF_ACTIVEKEY_SERVER_SYSTEM_ERROR	0x18002	98306	服务器系统错误
MERR_ASF_ACTIVEKEY_POST_PARM_ERROR	0x18003	98307	请求参数错误
MERR_ASF_ACTIVEKEY_PARM_MISMATCH	0x18004	98308	ACTIVE_KEY与APP_ID、SDK_KEY不匹配
MERR_ASF_ACTIVEKEY_ACTIVEKEY_ACTIVATED	0x18005	98309	ACTIVE_KEY已经被使用
MERR_ASF_ACTIVEKEY_ACTIVEKEY_FORMAT_ERROR	0x18006	98310	ACTIVE_KEY信息异常
MERR_ASF_ACTIVEKEY_APPID_PARM_MISMATCH	0x18007	98311	ACTIVE_KEY与APP_ID不匹配
MERR_ASF_ACTIVEKEY_SDK_FILE_MISMATCH	0x18008	98312	SDK与激活文件版本不匹配
MERR_ASF_ACTIVEKEY_EXPIRED	0x18009	98313	ACTIVE_KEY已过期
MERR_ASF_LICENSE_FILE_NOT_EXIST	0x19001	102401	离线授权文件不存在或无读写权限
MERR_ASF_LICENSE_FILE_DATA_DESTROYED	0x19002	102402	离线授权文件已损坏
MERR_ASF_LICENSE_FILE_SDK_MISMATCH	0x19003	102403	离线授权文件与SDK版本不匹配
MERR_ASF_LICENSE_FILEINFO_SDKINFO_MISMATCH	0x19004	102404	离线授权文件与SDK信息不匹配
MERR_ASF_LICENSE_FILE_FINGERPRINT_MISMATCH	0x19005	102405	离线授权文件与设备指纹不匹配

错误码名	十六进制	十进制	描述
MERR_ASF_LICENSE_FILE_EXPIRED	0x19006	102406	离线授权文件已过期
MERR_ASF_LOCAL_EXIST_USEFUL_ACTIVE_FILE	0x19007	102407	离线授权文件不可用，本地原有激活文件可继续使用
MERR_ASF_LICENSE_FILE_VERSION_TOO_LOW	0x19008	102408	离线授权文件版本过低，请使用新版本激活助手重新进行离线激活

4.3 FAQ

Q：如何将人脸识别1:1进行开发改为1:n？

A：先将人脸特征数据用本地文件、数据库或者其他的方式存储下来，若检测出结果需要显示图像可以保存对应的图像。之后循环对特征值进行对比，相似度最高者若超过您设置的阈值则输出相关信息。

Q：Android人脸检测结果的人脸框绘制到View上为何位置不对？

A：人脸检测结果的人脸框位置是基于输入图像的，例如在竖屏模式下，假设View的宽高是1080x1920，相机是后置相机，为了适配画面，预览画面相对于相机的旋转角度为90度，并且预览数据宽高为1920x1080，有一个被检测到的人脸位置是（left,top,right,bottom），那么需要绘制到View上的Rect就是（bottom,left,1080-top,right），相当于顺时针旋转90度，其他角度可用类似的方法计算。另外，在一般情况下，安卓调用前置相机时在View上的显示的画面和实际预览数据成镜像关系。

Q：初始化引擎时检测方向应该怎么选择？

A：SDK初始化引擎中可选择仅对0度、90度、180度、270度单角度进行人脸检测，对于VIDEO模式也可选择全角度进行检测；根据应用场景，推荐使用单角度进行人脸检测，因为选择全角度的情况下，算法会对每个角度检测一遍，导致性能相对于单角度较慢。IMAGE模式下为了提高识别率不支持全角度检测。

Q：初始化引擎时（detectFaceScaleVal）参数多大比较合适？

A：用于数值化表示的最小人脸尺寸，该尺寸代表人脸尺寸相对于图片长边的占比。VIDEO 模式有效值范围[2,32]，推荐值为16 ；IMAGE 模式有效值范围[2,32]，推荐值为 30，特殊情况下可根据具体场景进行设置。

Q：初始化引擎之后调用其他接口返回错误码86018，该怎么解决？

A：86018即需要检测的属性未初始化，需要查看调用接口的属性值有没有在初始化引擎时在combinedMask参数中加入。

Q：调用detectFaces、extractFaceFeature和process接口返回90127错误码，该怎么解决？

A：ArcFace SDK对图像尺寸做了限制，宽高大于0，宽度为4的倍数，NV21格式的图片高度为2的倍数，BGR24/GRAY/DEPTH_U16格式的图片高度不限制；如果遇到90127请检查传入的图片尺寸是否符合要求，若不符合可对图片进行适当的裁剪。

Q：人脸检测结果的人脸框Rect为何有时会溢出传入图像的边界？

A：Rect溢出边界可能是人脸只有一部分在图像中，算法会对人脸的位置进行估计。

Q：SDK是否支持多线程调用？

A：SDK支持多线程调用，但是在不同线程使用同一算法功能时需要使用不同引擎对象，且调用过程中不能销毁。

Q：MERR_FSDK_FACEFEATURE_LOW_CONFIDENCE_LEVEL，人脸检测结果置信度低是什么情况导致的？

A：图片模糊或者传入的人脸框不正确。若是使用了RGB摄像头人脸检测 + IR摄像头活体检测的方案，则很有可能是两者成像差距很大或两者画面成镜像或旋转的关系。

Q：哪些因素会影响人脸检测、人脸跟踪、人脸特征提取等SDK调用所用时间？

A：硬件性能、图片质量等。

Q：如何进行IR活体检测？

A：推荐的方案是采用双目（RGB/ IR）摄像头，RGB摄像头数据用于人脸检测，将人脸检测的结果用于IR活体检测。需要注意的是，IR活体检测不支持BGR24颜色格式的图像数据。