Baidu人脸识别的API

void initLicenseOnLine(final Context context, final String licenseKey, final AuthCallback callback)

在线激活（上下文，licenseKey，激活码{0成功，1失败}）

void initLicenseOffLine(final Context context, final Callback callback)

离线激活（上下文，激活码{0成功，1失败}）

void setActiveLog(BDFaceLogInfo logInfo)

底层打印Log

setAnakinConfigure(BDFaceSDKCommon.BDFaceCoreRunMode runMode, int coreNum)

code == 0 成功； code == 1 context 为null； code == -1 非法的参数；

code == -2 内存分配失败； code == -3 实例对象为空；

code == -4 模型内容为空；code == -5 不支持的能力类型；

code == -6 不支持预测类型；code == -7 预测库对象创建失败；

code == -8 预测库初始化失败；code == -9 图像数据为空；

code == -10 人脸能力初始化失败；code == -11 能力未加载 ；

code == -12 人脸能力已加载；code == -13 未授权；

code == -14 人脸能力运行异常 ；code == -15 不支持的图像类型；

code == -16 图像转换失败；

// 无参构造调用（执行默认创建的instance ）

public FaceDetect()

// 有参构造调用（执行自己创建的instance ）

public FaceDetect(BDFaceInstance thisBdFaceInstance)

根据创建的实例进行面目检测

void initModel(final Context context, final String visModel,final String nirModel,final String alignModel, final Callback callback)

。。。只看懂了初始化模型以及一个调用类

void initQuality(final Context context, final String blurModel, final String occlurModel, final Callback callback)

这个是对图像质量进行检测的（参数遮挡模型的理解为固定训练好的参数）

void initAttrEmo(final Context context,final String atttibuteModel,final String emotionModel,final Callback callback)

加载情绪人脸基础的属性方法

void initFaceClose(final Context context,

final String eyecloseModel,final String mouthcloseModel,final Callback callback)

人脸眼睛闭合，嘴巴模型初始化

void loadConfig(BDFaceSDKConfig config)

加载配置信息，config是配置参数实体

FaceInfo[] detect(BDFaceSDKCommon.DetectType type,BDFaceImageInstance imageInstance)

根据图片进行检测（检测类型，图片数据信息，返回参数）

FaceInfo[] detect(BDFaceSDKCommon.DetectType type,BDFaceSDKCommon.AlignType alignType, BDFaceImageInstance imageInstance,FaceInfo[] faceInfos,BDFaceDetectListConf bdFaceDetectListConfig)

同名构造器（多了对齐类型，图片数据信息，人脸框数据，功能开关）

FaceInfo[] track(DetectType detectType, BDFaceImageInstance imageInstance)

人脸追踪（检测类型，图片数据信息，返回信息）

FaceInfo[] track(BDFaceSDKCommon.DetectType detectType, BDFaceSDKCommon.AlignType alignType,BDFaceImageInstance imageInstance)

多人脸的追踪（检测类型，对齐信息，图片数据信息，返回参数）

public BDFaceImageInstance cropFace(BDFaceImageInstance imageInstance, float[] landmark)

人脸抠图（图片数据信息，检测后的产生数据）

void uninitModel()

检测模型卸载

void initModel(final Context context,final String visModel,final String nirModel, final String depthModel,final Callback callback)

活体模型加载（上下文，可见光模型，深度图片活体模型，红外图片活体模型，调用返回值）

public float silentLive(LiveType type, BDFaceImageInstance bdFaceImageInstance, float[] landmarks)

人脸静默活体检测（type,图像对象，检查后的landmark）

返回值float0-1 0.9是活体

void uninitModel()

静默活体卸载

// 无参构造调用（执行默认创建的instance ）

public FaceFeature()

// 有参构造调用（执行自己创建的instance ）

public FaceFeature(BDFaceInstance thisBdFaceInstance)

initModel(final Context context,final String idPhotoModel,final String visModel,final String nirModel,final String rgbdModel, final Callback callback)

初始化模型（上下文，证件照图片模型，可见光模型，红外图片模型，红外图片模型，RGBD图片模型，调用）

float feature(FeatureType featureType, BDFaceImageInstance imageInstance,float[] landmarks, byte[] feature)

提取图片特征值（类型，图像信息，检测后的数据，特征数组，返回特征数据）

float featureRGBD(FeatureType featureType, BDFaceImageInstance imageInstance,BDFaceImageInstance imageInstance\_depth, float[] landmarks, byte[] feature)

深度图片特征提取（RGBD）

float featureCompare(FeatureType featureType, byte[] feature1, byte[] feature2, boolean isPercent)

离线特征值对比（分值0-100）比较出的结果

public int featurePush(List<? extends Feature> features)

不懂

// 无参构造调用（执行默认创建的instance ）

public FaceGaze()

// 有参构造调用（执行自己创建的instance ）

public FaceGaze(BDFaceInstance thisBdFaceInstance)

注意力模型主要是分析眼睛是否集中，眼球状态以及是否闭合

// 无参构造调用（执行默认创建的instance ）

public FaceMouthMask()

// 有参构造调用（执行自己创建的instance ）

public FaceMouthMask(BDFaceInstance thisBdFaceInstance)

以一定置信度去判断对象是否戴口罩（？？）

FaceInfo

人脸的的信息中心店坐标，人脸的宽高，头角度，是否模糊，光照，遮挡，年龄，戴眼镜状态，性别，情绪，嘴巴眼睛闭合程度

 Feature 实体类

特征值的族信息，更新时间相似得分...判断的结果

// 设置相机的ID

CameraPreviewManager.getInstance().setCameraFacing(CameraPreviewManager.CAMERA\_USB);

// 开启预览

CameraPreviewManager.getInstance().startPreview(this, mAutoCameraPreviewView,

PREFER\_WIDTH, PERFER\_HEIGH, new CameraDataCallback())

public void onDetectCheck(final byte[] rgbData,

final byte[] nirData,

final byte[] depthData,

final int srcHeight,

final int srcWidth,

final int liveCheckMode,

final FaceDetectCallBack faceDetectCallBack)

人脸搜索

人脸检测

private void faceDetect(byte[] data, final int width, final int height) {

// 摄像头预览数据进行人脸检测

int liveType = SingleBaseConfig.getBaseConfig().getType();

if (liveType == 1) { // 无活体检测

FaceTrackManager.getInstance().setAliving(false);

} else if (Integer.valueOf(liveType) == 2) { // 活体检测

FaceTrackManager.getInstance().setAliving(true);

}

FaceTrackManager.getInstance().faceTrack(data, width, height, new FaceDetectCallBack() {

@Override

public void onFaceDetectCallback(LivenessModel livenessModel) {

checkResult(livenessModel);

}

});

}

活体判断

int liveType = SingleBaseConfig.getBaseConfig().getType();

// 无活体

if (Integer.valueOf(liveType) == 1) {

displayResult(model, null);

// 注册

register(model);

} else if (Integer.valueOf(liveType) == 2) { // RGB活体检测

displayResult(model, "livess");

boolean livenessSuccess = false;

float rgbLiveThreshold = SingleBaseConfig.getBaseConfig().getRgbLiveScore();

livenessSuccess = (model.getRgbLivenessScore() > rgbLiveThreshold) ? true : false;

if (livenessSuccess) {

// 注册

register(model);

}

}

人脸特征值提取

// 获取选择的特征抽取模型

int modelType = SingleBaseConfig.getBaseConfig().getActiveModel();

if (modelType == 1) {

// 生活照

FaceSDKManager.getInstance().onFeatureCheck(model.getBdFaceImageInstance(), model.getLandmarks(),

BDFaceSDKCommon.FeatureType.BDFACE\_FEATURE\_TYPE\_LIVE\_PHOTO, new FaceFeatureCallBack() {

@Override

public void onFaceFeatureCallBack(float featureSize, byte[] feature) {

displayCompareResult(featureSize, feature);

Log.e("qing", String.valueOf(feature.length));

}

});

} else if (Integer.valueOf(modelType) == 2) {

// 证件照

FaceSDKManager.getInstance().onFeatureCheck(model.getBdFaceImageInstance(), model.getLandmarks(),

BDFaceSDKCommon.FeatureType.BDFACE\_FEATURE\_TYPE\_ID\_PHOTO, new FaceFeatureCallBack() {

@Override

public void onFaceFeatureCallBack(float featureSize, byte[] feature) {

displayCompareResult(featureSize, feature);

}

});

}

注册到数据库

// 根据特征抽取的结果 注册人脸

private void displayCompareResult(float ret, byte[] faceFeature) {

// 特征提取成功

if (ret == 128) {

String imageName = groupId + "-" + username + ".jpg";

// 注册到人脸库

boolean isSuccess = FaceApi.getInstance().registerUserIntoDBmanager(groupId, username, imageName,

userInfo, faceFeature);

if (isSuccess) {

// 关闭摄像头

CameraPreviewManager.getInstance().stopPreview();

Log.e("qing", "注册成功");

// 数据变化，更新内存

FaceApi.getInstance().initDatabases(true);

// 防止重复注册

username = null;

groupId = null;

}else {

// 注册失败

}

} else if (ret == -1) {

displayTip("特征提取失败");

} else {

// 特征提取失败

}

}

打开预览

public void startPreview(Context context, AutoTexturePreviewView textureView, int width,int height, CameraDataCallback cameraDataCallback)

人脸搜索

public void onDetectCheck(final byte[] rgbData,

final byte[] nirData,

final byte[] depthData,

final int srcHeight,

final int srcWidth,

final int liveCheckMode,

final FaceDetectCallBack faceDetectCallBack)

接口信息

FaceSDKManager.getInstance().onDetectCheck(data, null, null,height, width, mLiveTyenew, FaceDetectCallBack()

private void syncFeature(final Bitmap bitmap, final byte[] feature, final int index, boolean isFromPhotoLibrary) {

float ret = -1;

BDFaceImageInstance rgbInstance = new BDFaceImageInstance(bitmap);

FaceInfo[] faceInfos = FaceSDKManager.getInstance().getFaceDetect()

.detect(BDFaceSDKCommon.DetectType.DETECT\_VIS, rgbInstance);

// 检测结果判断

if (faceInfos != null && faceInfos.length > 0) {

// 判断质量检测，针对模糊度、遮挡、角度

if (qualityCheck(faceInfos[0], isFromPhotoLibrary)) {

ret = FaceSDKManager.getInstance().getFaceFeature().feature(BDFaceSDKCommon.FeatureType.

BDFACE\_FEATURE\_TYPE\_ID\_PHOTO, rgbInstance, faceInfos[0].landmarks, feature);

Log.i("qing", "ret:" + ret);

if (ret == 128 && index == 1) {

firstFeatureFinished = true;

} else if (ret == 128 && index == 2) {

secondFeatureFinished = true;

}

if (ret == 128) {

toast("图片" + index + "特征抽取成功");

} else if (ret == -100) {

toast("未完成人脸比对，可能原因，图片1为空");

} else if (ret == -101) {

toast("未完成人脸比对，可能原因，图片2为空");

} else if (ret == -102) {

toast("未完成人脸比对，可能原因，图片1未检测到人脸");

} else if (ret == -103) {

toast("未完成人脸比对，可能原因，图片2未检测到人脸");

} else {

toast("未完成人脸比对，可能原因，"

+ "人脸太小（小于sdk初始化设置的最小检测人脸）"

+ "人脸不是朝上，sdk不能检测出人脸");

}

}

} else {

toast("未检测到人脸,可能原因人脸太小");

}

}

**2**）比对两张人脸图片

private void match() {

if (!firstFeatureFinished) {

toast("图片一特征抽取失败");

return;

}

if (!secondFeatureFinished) {

toast("图片二特征抽取失败");

return;

}

int idFeatureValue = SingleBaseConfig.getBaseConfig().getThreshold();

float score = 0;

// 比较两个人脸

score = FaceSDKManager.getInstance().getFaceFeature().featureCompare(

BDFaceSDKCommon.FeatureType.BDFACE\_FEATURE\_TYPE\_ID\_PHOTO,

firstFeature, secondFeature, true);

if (score > idFeatureValue) {

tvState.setTextColor(getResources().getColor(R.color.buttonBg));

tvState.setText("核验通过");

} else {

tvState.setTextColor(getResources().getColor(R.color.red));

tvState.setText("核验不通过");

}

}

**FaceSDKManager**

功能：负责初始检测类FaceAuth、FaceDetector、FaceFeature、FaceLiveness com.baidu.idl.face.main.manager

**FaceDetector**

功能：人脸检测封装类，包含人脸检测功能初始化、人脸检测接口调用 com.baidu.idl.main.facesdk

**FaceFeatures**

功能：人脸特征抽取封装类，包含人脸特征抽取功能初始化、人脸特征抽取接口调用 com.baidu.idl.main.facesdk

**FaceLiveness**

功能：人脸活体相关操作封装类，包含人脸rgb、ir、depth活体检测 com.baidu.idl.main.facesdk

**FaceTrackManager**

功能：主要是对人脸检测、质量检测的封装 com.baidu.idl.face.main.manager

**ImportFileManager**

功能：批量导入的相关操作 com.baidu.idl.face.main.manager

**UserInfoManager**

功能：人脸库管理相关操作 com.baidu.idl.face.main.manager

**DBManager**

功能：数据库相关操作 com.baidu.idl.face.main.db

人脸的质量检测非常重要，活体检测和环境光确定你是否是真实的你，这样避免安全信息被盗，毕竟现在没有办法用瞳孔检测，所以人脸识别中是否是活体就非常重要了

人脸角度，面部光线，人脸模糊度，图像分变率，人脸的完整度，人脸表清，人脸大小，图像几何失真

拒绝率，up 通过率 down 误判率up 通过率 up 阈值门限