



SAFR[®] Android Embedded SDK Documentation

Documentation Version = 3.048

Publish Date = August 19, 2022

Copyright © 2022 RealNetworks, Inc. All rights reserved.

SAFR® is a trademark of RealNetworks, Inc. Patents pending.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

Note: Documentation pertaining to the macOS platform is no longer being actively maintained.

Contents

1	SAFR Embedded SDK Overview	3
2	Android eSDK Overview	4
3	Use the Android eSDK	6
4	Use Android eSDK Detection and Recognition	8
5	Understand Android eSDK Object Detection and Recognition	9
6	Understand Android eSDK Object Face Estimation and Signature Compositing	11
7	Mask Detection and RGB Liveness Detection	13
8	FaceFind Sample App	16
9	FaceCompare Android Java Sample App	19
10	Panoptes Command Line Tool	34
11	Panoptes Actions	36
12	Panoptes Benchmark Output	43

1 SAFR Embedded SDK Overview

1.1 Purpose

The Embedded SDK is for use when the facial recognition software needs to run on a device with limited resources — RAM, CPU, and memory — without any internet or cloud connectivity. However, the device must still perform, for example, detection and recognition of images with varying conditions with a minimum response time.

Using this SDK, developers can create applications that can:

- Provide information on the total number of known and unknown faces, the relative coordinates of each person detected, the size in pixels for each face, and more.
- Learn new faces through training (labeling).

The SAFR Embedded SDK ships as a `tar.gz` file that contains the necessary libraries, recognizer, and detector model files, and a sample command line tool called Panoptes.

Note: For SAFR Embedded SDK system requirements, go to SAFR.com.

2 Android eSDK Overview

Download the SAFR Embedded SDK package for Android from the Developers tab of the SAFR Download Portal. The eSDK is distributed as a set of (native code) shared objects (`.so`) and resource files. Make sure that you have the Android development tools and the Android NDK version 20b or newer installed on the computer on which you intend to build your application.

Copy the library and model files from the SAFR eSDK into the directory containing your final application. Place the shared object files inside a directory named `lib` and the resource files inside a directory named `model`. Assuming that your application would be called `Unique` and that you ship it inside a directory called `UniqueApp` then the structure of your final application directory would look like this:

Copy the library and model files from the SAFR eSDK into the directory containing your final application. Place the shared object files inside a directory named `lib` and the resource files inside a directory named `model`. Assuming that your application would be called `Example` and that you ship it inside a directory called `ExampleApp` then the structure of your final application directory would look like this:

```
ExampleApp/  
  Example  
  lib/  
  ...  
  model/  
  ...
```

Note: The application has to be built as a 64-bit application. 32-bit applications are not supported.

2.1 Sample Applications

SDK comes with 2 sample applications: `panoptes` and `facefind`. These are console applications demonstrating how to use the eSDK's C API. To build them, you need to do the following:

1. The `ANDROID_NDK` environmental variable needs to be set to the location of the Android NDK. (i.e. `ANDROID_NDK=/path/android-ndk-r20b/`).
2. The `build.sh` script in the `panoptes` or `facefind` folder needs to be run.

The sample apps can only be built on Linux and macOS machines. The `panoptes/bin` folder needs to be copied to the target platform and then the `panoptes` executable run from the device shell. The folder needs to have executable permission.

2.2 Using the eSDK from Java Code

You can develop Android mobile apps with Android Studio and use the eSDK from Java (or Kotlin) code by using Java Native Interface (JNI). JNI is basically a bridge between a byte code that runs in the Java Virtual Machine and the native code.

To start, we've prepared a sample native Java/C++ Android project called `FaceCompare`. The project consists of:

- Native code which directly invokes the eSDK API. (The eSDK is packed as native shared libraries, `.so` files.) In the project, this is implemented in `app/src/main/cpp/native-lib.cpp`.
- JNI method declarations. These are method signatures which are invoked from Java virtual machine (JVM) layer but are implemented in the native layer. There are counterpart methods with same signatures in the native code. In the project, this is implemented in `app/src/main/java/com/real/face-compare/ESDKWrapper.java`.
- Makefile which includes rules to build the wrapper library. Basically, it's an `.so` file which is deployed as part of the apk and which includes definitions for the JNI methods. In the project, this is in `app/src/main/cpp/CMakeLists.txt`.
- Other Java code and classes that include business logic for the sample app.

Note: When building a release version of the application, any new JNI method declarations or classes used in those methods should be added to Proguard rules. These are accessed via reflection from the native code so their signatures must remain unchanged, rather than being obfuscated.

3 Use the Android eSDK

Use the C API provided by the SDK by including the `eArgusKit.h` header file into your C files. The following code example shows how to do this and includes basic steps for setting up and using a face detector and face recognizer:

```
#include "eArgusKit.h"

// Detects as many faces as possible in the image 'pBitmap'.
// \param pLicense the SDK license
// \param pBitmap the image description
void detect_faces(const EARLicense* pLicense, const EARBitmap* pBitmap)
{
    EARFaceDetectionInitializationConfiguration initConfig;

    EARFaceDetectorRef pDetector = EARFaceDetectorCreate(pLicense,
        kEARFaceDetectionType_Normal, &initConfig, 0);
    EARFaceDetectionConfiguration config;
    EARDetectedFaceArrayRef pFaces;

    EARFaceDetectionConfigurationInitWithPreset(&config,
        kEARFaceDetectionConfigurationPreset_Generic);

    pFaces = EARFaceDetectorDetect(pDetector, pBitmap, &config);
    for (size_t i = 0; i < EARDetectedFaceArrayGetCount(pFaces); i++) {
        const EARDetectedFace* pFace =
            EARDetectedFaceArrayGetAtIndex(pFaces, i);

        // Process the face data
        ...
    }

    EARDetectedFaceArrayDestroy(pFaces);
    EARFaceDetectorDestroy(pDetector);
}

// Recognizes as many faces as possible out of the set of detected faces
// 'pDetectedFaces'. Expects that the the faces were previously detected
// in the
// image 'pBitmap'.
// \param pLicense the SDK license
// \param pBitmap the image description
// \param pDetectedFaces the previously detected faces
void recognize_faces(const EARLicense* pLicense, const EARBitmap*
    pBitmap, EARDetectedFaceArrayRef pDetectedFaces)
{
    EARPersonStoreRef pStore = EARPersonStoreCreate(<path to the store
        file>, <store file password>);
    EARFaceRecognizerRef pRecognizer = EARFaceRecognizerCreate(pLicense,
        pStore, kEARFaceRecognitionModelType_Regular,
        kEARFaceRecognitionModelPerformance_SpeedOptimized, 0);
    EARFaceRecognitionConfiguration config;
    EARRecognizedFaceArrayRef pFaces;
```

```

    EARFaceRecognitionConfigurationInitWithPreset(&config,
        kEARFaceRecognitionConfigurationPreset_Recognize);

    pFaces = EARFaceRecognizerRecognizeDetectedFaces(pRecognizer,
        pDetectedFaces, pBitmap, &config, kEARAllowRecognition);
    for (size_t i = 0; i < EARRecognizedFaceArrayGetCount(pFaces); i++) {
        const EARRecognizedFace* pFace =
            EARRecognizedFaceArrayGetAtIndex(pFaces, i);

        // Process the face data
        ...
    }

    EARRecognizedFaceArrayDestroy(pFaces);
    EARFaceRecognizerDestroy(pRecognizer);
    EARPersonStoreDestroy(pStore);
}

```

Note: This sample code does not check for errors. As you should always check for errors in your code, use the `EARGetLastError()` function to get the most recent error.

Your code should create the face detector, face recognizer, and person store objects once and reuse them. Do not recreate these objects for every single detection and recognition operation.

4 Use Android eSDK Detection and Recognition

Execute the following steps to detect and recognize faces:

1. Create a person store object. You specify a file system location and file name. A new and empty person store is created if none already exists.
2. Create a face detector object.
3. Create a face recognizer object and pass it a reference to the person store.
4. Use the face detector to find faces in an image. This returns an array of detected faces. Each face has a bounding box and information about the quality of the detected face.
5. Use the face recognizer on the faces returned by the face detector. You can attempt to recognize all faces but more typically you should first filter the array of detected faces down to the set that you want to attempt recognition on given your use case constraints.
6. Save the updated person store to disk.

Execute Steps 1 to 3 once in a typical application. The person store, face detector, and recognizer objects should be reused whenever possible.

Note: The face detector and recognizer objects are not thread safe. Make sure only one face detection/recognition operation is executed on a single detector/recognizer object at any given time. However, you may create multiple separate detector/recognizer objects and use them in parallel from different threads.

When processing a video file, execute Step 4 on every frame or every other frame of the video file. Inspect the detected faces and decide which ones are of sufficient quality to attempt recognition on. Face recognition should be rate limited to approximately one (1) recognition per face since it is significantly more costly than face detection.

Note: Changes to a person store are not automatically saved. The changes are applied in memory and you must explicitly save the person store when it makes sense for your application.

5 Understand Android eSDK Object Detection and Recognition

This page describes the SAFR Embedded SDK objects you use to detect and recognize objects.

5.1 EARFaceDetector

A face detector instance allows you to detect faces in an image. All face detector APIs are synchronous. Parallelism/asynchronous behavior can be achieved by the caller by creating a thread and assigning the detector instance to that thread. This thread should then listen to detection requests and process them with the help of the face detector object.

5.1.1 Threading

A face detector instance is owned by a single thread. Only this thread may use the face detection services.

5.1.2 EARFaceDetection Types

Two face detector types are supported.

- **kEARFaceDetectionType_Normal**: Normal is the default face detector. It's faster, but less accurate than HighSensitivity face detector.
- **kEARFaceDetectionType_HighSensitivity**: HighSensitivity face detector has better accuracy compared to Normal, especially when detecting faces wearing masks. Because the input resolution is fixed, it's faster than Normal on higher resolution images.

Similar face attribute values are expected for both face detectors.

5.1.3 kEARFaceDetectionHighSensitivityResolutions Property

You can configure the resolution of the HighSensitivity face detector by setting the **kEARFaceDetectionHighSensitivityResolutions** property. This property is similar to the **minFaceSearchSize** configuration parameter of the Normal face detector in that it depends on expected face resolutions. Unlike **minFaceSearchSize**, however, it's set when the face detector is initialized and can't be changed later on.

The default value for the HighSensitivity face detector resolution is 640x480: **kEARFaceDetectionHighSensitivityResolution**

5.1.4 enableLiveness Property

Enables liveness detection. If enabled, the face detector will produce two additional output values: *liveness* (i.e. liveness score) and *livenessStatus*. For liveness detection to output a score for a detected face, certain liveness thresholds must be satisfied. See the list of liveness thresholds and respective default values in **EARFaceDetectionConfiguration**.

Liveness detection is supported only on high sensitivity face detector, **kEARFaceDetectionType_HighSensitivity**.

5.2 EARFaceRecognizer

A face recognizer instance allows you to recognize faces/persons in an image. All face recognizer APIs are synchronous. Parallelism/asynchronous behavior can be achieved by the caller by creating a thread and assigning the recognizer instance to that thread. This thread should then listen to recognition requests and process them with the help of the face recognizer object.

5.2.1 Threading

A face recognizer instance is owned by a single thread. Only this thread may use the face recognizer services.

5.2.2 EARFaceRecognition Model Type Options

The face recognizer supports two recognition models,

- **kEARFaceRecognitionModelType_Regular**: Regular model is the default, performs well on variety of use cases.
- **kEARFaceRecognitionModelType_MaskOptimized**: MaskOptimized recognition model has similar performance to Regular model, but performs better on faces wearing masks.

5.2.3 EARRecognition Model Performance Options

The face recognizer model has 2 options for optimizing performance.

- **kEARFaceRecognitionModelPerformance_SpeedOptimized**: Speed optimized model has slightly lower accuracy due to loss of precision from 32-bit to 8-bit. It is the default option and recommended to use since the loss of accuracy is in most cases not as important compared to gain in speed.
- **kEARFaceRecognitionModelPerformance_AccuracyOptimized**: Accuracy optimized model is recommended to use when enough computational resources are available and when the database contains significant number of persons and the extra bit of accuracy can help.

5.3 EARPersonStore

A person store persistently stores information about a person. Every person is identified by a globally unique ID (UUID). Each person also has a signature the face recognizer generates. This signature property is an internal property that is not exposed to the user.

The user of the API may add arbitrary key-value pairs to a person. A key is always represented by a string, while a value is a sequence of arbitrary bytes. The person store does not interpret these bytes.

5.3.1 Threading

A single person store instance may be accessed from multiple threads at the same time.

6 Understand Android eSDK Object Face Estimation and Signature Compositing

This page describes the SAFR eSDK objects you use to estimate certain characteristics of detected faces as well as the signature compositor object which can be used to create a composite signature from a set of source signatures.

6.1 EARAgeEstimator

An age estimator instance allows you to estimate the age of detect faces in an image. All face estimator APIs are synchronous. Parallism/asynchronous behavior can be achieved by the caller by creating a thread and assigning the estimator instance to that thread. This thread should then listen to estimation requests and process them with the help of the face estimator object.

6.1.1 EARAgeEstimator Threading

An estimator instance is owned by a single thread. Only this thread may use the estimator services.

6.2 EAREmotionEstimator

An emotion estimator instance allows you to estimate the expressed emotion of detect faces in an image. All face estimator APIs are synchronous. Parallism/asynchronous behavior can be achieved by the caller by creating a thread and assigning the estimator instance to that thread. This thread should then listen to estimation requests and process them with the help of the face estimator object.

6.2.1 EAREmotionEstimator Threading

An estimator instance is owned by a single thread. Only this thread may use the estimator services.

6.3 EAROcclusionEstimator

An occlusion estimator instance allows you to estimate the amount of occlusion of detect faces in an image. All face estimator APIs are synchronous. Parallism/asynchronous behavior can be achieved by the caller by creating a thread and assigning the estimator instance to that thread. This thread should then listen to estimation requests and process them with the help of the face estimator object.

6.3.1 EAROcclusionEstimator Threading

An estimator instance is owned by a single thread. Only this thread may use the estimator services.

6.4 EARGenderDetector

A gender detector instance allows you to detect the gender of detect faces in an image. All face detector APIs are synchronous. Parallism/asynchronous behavior can be achieved by the caller by creating a thread and assigning the detector instance to that thread. This thread should then listen to detection requests and process them with the help of the face detector object.

6.4.1 EARGenderDetector Threading

A detector instance is owned by a single thread. Only this thread may use the detector services.

6.5 EARMaskEstimator

A mask estimator instance allows you to estimate the probability of a face being covered by a mask. All face detector APIs are synchronous. Parallism/asynchronous behavior can be achieved by creating a thread and

assigning the estimator instance to that thread. This thread should then listen to estimation requests and process them with the help of the face estimator object.

6.5.1 EARMaskEstimator Threading

An estimator instance is owned by a single thread. Only this thread may use the estimator services.

6.6 EARSignatureCompositor

A signature compositor is used to create a composite signature from a set of source signatures. The source signatures should all represent the same physical person. For example you may have a set of person objects in the person store which all represent the same physical person. In this case you could create a singular merged person object in the person store for all the existing person objects by doing the following:

1. Create a signature compositor.
2. Get the signatures of all the person objects which represent the same physical person. (These are the input person objects.)
3. Use the signature compositor to create the composite signature.
4. Apply whatever algorithms are necessary to generate a correct merged representation of the metadata key-value pairs of the input persons.
5. Create a new person in the person store from the composite signature.
6. Add the merged metadata key-value pairs to the newly create composite person.
7. Remove the input person objects from the person store.

7 Mask Detection and RGB Liveness Detection

7.1 Recognition of Faces with Masks

The same `EARFaceRecognitionModelType` used for face registration should be used for face recognition.

When registering a face without a mask that is to be matched with faces with and without masks, register the face with `kEARFaceRecognitionModelType_Regular` and `kEARFaceRecognitionModelType_MaskOptimized` into two separate person stores.

When recognizing a face, first determine if the face is wearing a mask by using mask detection. If the face is masked, recognize it with `kEARFaceRecognitionModelType_MaskOptimized` and the corresponding person store. If a face does not have a mask, recognize it with `kEARFaceRecognitionModelType_Regular` and the corresponding person store.

If also registering faces with a mask, register the faces with masks with `kEARFaceRecognitionModelType_MaskOptimized` into a third, separate person store. When recognizing a face, first determine if the face has a mask using mask detection. If the face does have a mask, recognize it with `kEARFaceRecognitionModelType_MaskOptimized` and the corresponding person store containing masked faces. If this fails, attempt to recognize it with `kEARFaceRecognitionModelType_MaskOptimized` and the person store containing unmasked faces.

If a face does not have a mask, recognize it with `kEARFaceRecognitionModelType_Regular` and the corresponding person store. If this fails, attempt to recognize it with `kEARFaceRecognitionModelType_MaskOptimized` and the person store containing masked faces.

7.2 RGB Liveness Detection

RGB liveness detection is calculated internally within the face detector (`EARFaceDetector`) as part of the face detection operation. It's only supported on high sensitivity face detectors (`kEARFaceDetectionType_HighSensitivity`).

RGB liveness detection is disabled by default. To enable it, set the `enableLiveness` parameter in the face detection configuration object (`EARFaceDetectionConfiguration`) to `true`.

Using the `EARFaceDetectorDetect` API will result in computing both face detection and RGB liveness detection on a single low resolution image. For best results, the `EARFaceDetectorDetectHighRes` API should be used if both low resolution and high resolution images can be obtained. Low resolution images (e.g. 480p) should be used for face detection while high resolution images (e.g. 1080p) should be used for RGB liveness detection, which improves the RGB liveness detection accuracy.

Detected face info (`EARDetectedFace`) returned by the face detection operation contains four liveness-related parameters:

- **liveness:** The RGB liveness score of the detected face. This value ranges from 0.0 (i.e. definitely fake) to 1.0. (i.e. there's a real person in front of the camera). We recommend that you use a liveness threshold of 0.6.
- **livenessStatus:** Status of the RGB liveness detection.
 - *kEARLivenessOK*: RGB liveness has been successfully calculated.
 - *kEARLivenessLowQuality*: The image quality of the detected face is below the required quality level.
 - *kEARLivenessModelNotLoaded*: RGB liveness could not be evaluated. This is either because the `EARFaceDetector` instance wasn't initialized with RGB liveness support (i.e. the platform doesn't support RGB liveness detection) or because the `enableLiveness` parameter was set to `false`.
 - *kEARLivenessError*: An unspecified error occurred.
- **livenessStage1:** The RGB liveness score of a detected face on a texture model. This value ranges from 0.0 (i.e. definitely fake) to 1.0. (i.e. there's a real person in front of the camera).
- **livenessStage2:** The RGB liveness score of a detected face on a context model. This value ranges from 0.0 (i.e. definitely fake) to 1.0. (i.e. there's a real person in front of the camera).

7.2.1 Parameters

The face detection configuration object (`EARFaceDetectionConfiguration`) contains multiple liveness-related parameters. Some parameters are used to define the minimum required image quality of the detected face image for RGB liveness to be evaluated. When the image quality is too low, the RGB liveness value is set to 0.0 and `livenessStatus` is set to `kEARLivenessLowQuality`. The following parameters are used as thresholds for low image quality.

- **liveness_minFaceSize**: Minimum face size, in pixels, required for the RGB liveness model to be evaluated. The default value for this parameter is 150.
- **liveness_minCPQ**: Minimum center pose quality for the RGB liveness model to be evaluated. This parameter ranges from 0.0 (worst) to 1.0 (best). Its default value is 0.2.
- **liveness_minSharpness**: Minimum sharpness required for the RGB liveness model to be evaluated. This parameter ranges from 0.0 (worst) to 1.0 (best). Its default value is 0.45.
- **liveness_minContrast**: Minimum contrast required for the RGB liveness model to be evaluated. This parameter ranges from 0.0 (worst) to 1.0 (best). Its default value is 0.45.
- **liveness_minFaceContext**: Minimum context required around the detected face for the RGB liveness model to be evaluated. This parameter ranges from 0 to 1000. Its default value is 1.0.

Two additional parameters can be configured: `liveness_mode` and `liveness_initial_threshold`.

The `liveness_mode` parameter is used to select the mode used to calculate the liveness response in multi-model setup. The following modes are allowed:

- `kEARLivenessModeNone`: Only the texture-based model value is returned.
- `kEARLivenessModeLast`: Only the context-based model value is returned.
- `kEARLivenessModeAverage`: The average liveness value of both models is returned.
- `kEARLivenessModeMin`: Minimum liveness value from both models is returned. This is the default mode.
- `kEARLivenessModeMax`: Maximum liveness value from both models is returned.

The `liveness_initial_threshold` parameter is used to short-circuit stage 2 liveness. It ranges from 0.0 to 1.0. Its default value is 0.19.

7.2.2 Additional Info

7.2.2.1 Improve Accuracy We recommended you calculate liveness score on multiple consecutive frames (e.g. 10 frames) to confirm that the face is live or fake.

7.2.2.2 Recommended Camera Settings To ensure reliable RGB liveness detection, the following camera settings are recommended.

- High color saturation. (i.e. vibrant colors)
- Low contrast. (i.e. no grain)
- Focus into the distance with a reasonable focus in target range.
- Exposure setting should yield properly exposed faces.

7.3 Sample Code

```
#include "eArgusKit.h"

// Detects as many faces as possible in the image 'pBitmap' and calculate
// liveness score for each detected face.
// \param pLicense the SDK license
// \param pBitmap the image description
void detect_faces(const EARLicense* pLicense, const EARBitmap* pBitmap,
                 const EARBitmap* pBitmapHighRes)
{
```

```

EARFaceDetectionInitializationConfiguration initConfig;
//Enable liveness detection as it is disabled by default
initConfig.enableLiveness = true;

EARFaceDetectionConfiguration config;
EARFaceDetectionConfigurationInitWithPreset(&config,
      kEARFaceDetectionConfigurationPreset_Generic);

//Adjust some default liveness parameters
config.liveness_initial_threshold = 0.25;
config.liveness_mode = kEARLivenessModeAverage;

//Only HighSensitivity face detector type supports liveness detection
EARFaceDetectionType FDType = kEARFaceDetectionType_HighSensitivity;

EARFaceDetectorRef pDetector = EARFaceDetectorCreate(pLicense, FDType,
      &initConfig, 0);

EARDetectedFaceArrayRef pFaces;

pFaces = EARFaceDetectorDetectHighRes(pDetector, pBitmap,
      pBitmapHighRes, &config);
for (size_t i = 0; i < EARDetectedFaceArrayGetCount(pFaces); i++) {
    const EARDetectedFace* pFace =
        EARDetectedFaceArrayGetAtIndex(pFaces, i);

    // Process the face data
    ...
}

EARDetectedFaceArrayDestroy(pFaces);
EARFaceDetectorDestroy(pDetector);
}

```

Additional liveness instructions can be found in Panoptes sample documentation, with liveness sample images in panoptes/sample_images directory.

8 FaceFind Sample App

The facefind sample app allows you to learn persons from a provided image and recognize learned persons in images contained in a provided directory.

The syntax of facefind is described below.

```
facefind usage:
<path to image with face which should be learned for recognition:
  "face.jpg"> <path to image(s) on which face should be recognized:
  "dirWithImages"> [-r] [-sort] [-min-size=value]
  [-min-search-size=value] [-min-confidence=value] [-fd-type=value]
  [fd-retina=resolution=value] [fr-model=value] [-fr-optimization=value]
```

Where:

```
[-r]                Recursively search all subdirectories in
  given path
[-sort]            Output all matches sorted by confidence
  score
[-min-confidence=value] Required minimum confidence[0 - 1], default
  = 1
[-min-search-size=value] Minimum face width/height in pixels used
  for facesearch during detection[15-720], default = 40
[-min-size=value]    Minimum face width/height in pixels for
  recognition acceptance[1 - 720], default = 146
[-fr-model=value]    Face recognition model (regular, masked).
  regular - default
[-fr-optimization=value] Face recognition variant optimized for
  speed vs accuracy (speed, accuracy). speed - default
[-fd-type=value]     Face detector type (normal,
  high_sensitivity). normal - default
[-fd-high-sensitivity-resolution=value] High sensitivity face detector
  model input resolution. 640x480 - default
  Available resolutions: 1280x720, 720x1280, 640x480, 640x640,
  480x640, 480x480, 480x320, 320x480, 320x240, 320x320, 240x320,
  240x240, 128x128
```

Environment variables:

```
PANOPTES_LICENSE    If set, specifies the panoptes license
```

- **-r** will recursively search through all sub-directories of the specified directory.
- **-sort** will sort all face matches found in images in the specified directory in descending order.
- The learning/recognition configuration options are listed below. (Defaults can be overridden using command-line arguments.)

Setting	Default	Range	Description
-min-confidence	1	0-1.0	Required minimum match confidence.
-min-search-size	40	15 - 720	Minimum face width/height in pixels used for face search in mtcnn face detector.

Setting	Default	Range	Description
-min-size	146	1 - 720	Minimum required face width/height in pixels used for face recognition.
-fd-type	normal	normal, high_sensitivity	Face detector to use.
-fd-high-sensitivity-resolution	640x480	1280x720, 720x1280, 640x480, 640x640, 480x640, 480x480, 480x320, 320x480, 320x240, 320x320, 240x320, 240x240, 128x128	Resolution used for face search with high_sensitivity face detector, similar to min-search-size.
-fr-model	regular	regular, masked	Face recognition model to use.
-fr-optimization	speed	speed, accuracy	Face recognition model optimized for speed or accuracy.

Note: You have to set the PANOPTES_LICENSE environment variable to the license key that you've received from RealNetworks before you invoke Panoptes. On a POSIX comparability system you can do this by executing the following line in a shell window:

```
export PANOPTES_LICENSE=<license key>
```

This line should be added to your shell startup script to ensure that it will be executed for every new shell window you open.

8.1 Installation

Decompress the tar.gz file that you have received. The result will be a directory with the name 'facefind' in your current directory. Push this directory and all its subdirectories with 'adb push' to a suitable home folder on your device. The facefind app can be found inside the 'Facefind' folder.

8.2 Usage

Once you are positioned in the 'Facefind' folder, you can invoke facefind without recursive directory search using the following command:

```
./Facefind /home/images/personA/personA1.jpg /home/images/personA
```

If recursive directory search is desired, use the "-r" option:

```
./Facefind /home/images/personA/personA1.jpg /home/images/personA -r
```

If you'd like to override the defaults for face learning and recognition, run the following command:

```
./Facefind /home/images/personA/personA1.jpg /home/images/personA -r
-min-search-size=30 -min-size=30 -min-confidence=0.5
```

If you'd like to set the face detector type to "high_sensitivity" with a resolution of 640x640, and set the face recognition model to "masked" and the face recognition optimization to "accuracy", run the following command:

```
./Facefind /home/images/personA/personA1.jpg /home/images/personA -r
-fd-type=high_sensitivity -fd-high-sensitivity-resolution=640x640
-fr-model=masked -fr-optimization=accuracy
```

Note: When using the High Sensitivity face detector, this message can occur:

```
ERROR: OpenCL library not loaded - dlopen failed: library
"libOpenCL-pixel.so" not found
ERROR: Falling back to OpenGL
```

This error is just a warning that fallback to OpenGL is taking place and can be ignored.

Output face matches sorted by confidence scores for all detected faces (set -min-confidence to 0):

```
./Facefind /home/images/personA/personA1.jpg /home/images/personA -r
-min-confidence=0 -sort
```

8.3 Output

When the app is processing images, it logs the progress as:

```
Analyzing image 2 of 23 image(s)...
```

If a face is learned from a given image and recognized in an image from a provided folder, the output contains the confidence, location, and size of the recognized face:

```
C:\images\personA\personA2.jpg - 98% match @ x=321, y=211, width=89,
height=102
```

Otherwise, the app will output the following message:

```
No faces recognized in image.
```

If -sort is used, all face matches will be printed out, sorted by their confidence scores in descending order.

```
Total matches: 5
1. /home/images/personA/compare_5.jpg,      100.00% match,      (x,
   y, width, height): (356, 67, 304, 389)
2. /home/images/personA/compare_4.jpg,      89.30% match,      (x,
   y, width, height): (728, 547, 987, 1334)
3. /home/images/personA/compare_2.jpg,      41.32% match,      (x,
   y, width, height): (1122, 195, 391, 494)
4. /home/images/personA/compare_1.jpg,      32.25% match,      (x,
   y, width, height): (1106, 55, 564, 575)
5. /home/images/personA/compare_3.jpg,      26.15% match,      (x,
   y, width, height): (1136, 159, 281, 376)
```

8.4 Stopping

If you want to stop the app during execution, press Ctrl-Z.

9 FaceCompare Android Java Sample App

The sample app demonstrates how to use the Android eSDK to create a Java Android app.

The app is organised into:

1. A Java layer which includes the UI & presentation logic.
2. The native layer which directly invokes the embedded SDK API and implements some of the app's business logic.

The layers communicate via the Java Native Interface (JNI). All JNI calls are done synchronously.

9.1 App's Features

- Face detection
- Face recognition
- Face learning
- Face compare (in terms of the confidence attribute)
- Face recogniser configuration

9.2 The Use Case

1. User picks an image which is used as a source.
2. The source image is processed to detect and learn the largest face.
3. User picks another image which is used as a target.
4. The target image is processed to detect all faces which are compared against the face from the first image.

The results, in terms of match percentage, are displayed for every detected face.

9.2.1 IDE System Requirements

- Android Studio 4.1
- With CMake and NDK

For Windows:

- Microsoft Windows 7/8/10 (32- or 64-bit)
- 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
- Minimum 2 GB of available disk space (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
 - 4 GB is recommended
- 1280 x 800 minimum screen resolution

For macOS:

- Mac OS X 10.13 High Sierra or later
- 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
- Minimum 2 GB of available disk space minimum
 - 4 GB is recommended
- 1280 x 800 minimum screen resolution

For Linux:

- GNOME or KDE desktop
- Tested on Ubuntu 14.04 LTS, Trusty Tahr (64-bit distribution capable of running 32-bit applications)
- 64-bit distribution capable of running 32-bit applications
- GNU C Library (glibc) 2.19 or later
- 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
- Minimum 2 GB of available disk space

- 4 GB is recommended
- 1280 x 800 minimum screen resolution

Supported Device Architecture:

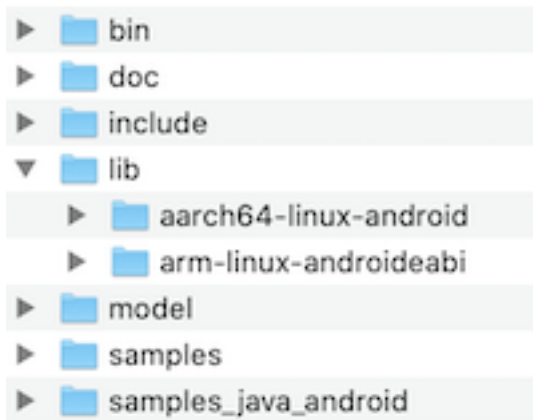
- ARM (armeabi-v7a, arm64-v8a)

9.3 Prerequisites

Download the SAFR eSDK package for Android from the SAFR Download Portal. We recommended you download both armeabi-v7a and arm64-v8a packages because the sample project is set up to build both architectures and the PlayStore actually expects both of them.

9.3.1 Set Up

1. Download and unpack *SAFREMBSDK_android-lite_xxx.tgz* and *SAFREMBSDK_android64-lite_xxx.tgz* from the SAFR Download Portal.
2. Copy *SAFREMBSDK_android-lite_xxx/lib/arm-linux-androideabi* into *SAFREMBSDK_android64-lite_xxx/lib/arm-linux-androideabi*.
3. *SAFREMBSDK_android64-lite_xxx/* should look like this:



The project's location is *SAFREMBSDK_android64-lite_xxx/samples_java_android/facecompare/*

Before importing into Android Studio, the project's dependency files must be set. This can be done automatically by executing the *samples_java_android/facecompare/source/setup_dependency_tree.sh* script from within its directory, or by manually copying the files.

Once completed, the final directory structure should look as follows:

```
eArgusSDK\samples_java_android\facecompare\
|--- doc - docs directory
|--- source - app directory
|--- esdk_distribution
|   |--- include - header files
|   |--- model - model files
|   |--- lib- libraries - directory names must match to Android ABIs
|       |--- arm64-v8a - content from aarch64-linux-android package
|           if available
|       |--- armeabi-v7a - content from arm-linux-androideabi package
|           if available
```

9.4 Android Studio Setup

To import the project into Android Studio, go to **File -> New -> Import Project...** Navigate to "package/samples_java_android/facecompare/source" - import.

Once the indexing is done, open *MainActivity.java* and replace **OBTAIN_FROM_RN** with the license key you've obtained from RealNetworks.

```
private static final String LICENSE = "OBTAIN_FROM_RN";
```

If needed, update LIB_VERSION inside the *app/src/main/cpp/CMakeLists.txt* file:

```
set(LIB_VERSION 3) # It must match to libArgusKit's version number i.e.
                   libArgusKit-3.so version is 3
```

The app is now ready to build and run.

9.4.1 Source Files

app/src/main/cpp/native-lib.cpp

Includes native code which does eSDK invocation.

Detector, Recogniser, and Store initialization:

```
const char *_modelsPath = env->GetStringUTFChars(modelsPath,
    JNI_FALSE);
const char *_storePath = env->GetStringUTFChars(storePath, JNI_FALSE);
const char *_licence = env->GetStringUTFChars(licence, JNI_FALSE);

EARFaceRecognitionModelType faceRecognitionModelType =
    EARFaceRecognitionModelType::kEARFaceRecognitionModelType_MaskOptimized;
EARRecognitionModelPerformance recognitionModelPerformance =
    EARRecognitionModelPerformance::kEARFaceRecognitionModelPerformance_AccuracyOpt

if (configOptions != nullptr) {

    jclass optionsClass =
        env->FindClass("com/real/facecompare/ConfigOptions");

    faceRecognitionModelType =
        getFaceRecognitionModelType(env->GetIntField(configOptions,
            env->GetFieldID(optionsClass, "faceRecognitionModelType",
                "I")));
    recognitionModelPerformance =
        getRecognitionModelPerformance(env->GetIntField(configOptions,
            env->GetFieldID(optionsClass, "recognitionModelPerformance",
                "I")));
}

EARSetModelFilesSearchPath(_modelsPath);

const size_t _length = env->GetStringUTFLength(licence);
char *_licenceCopy = (char *) malloc(_length);

strcpy(_licenceCopy, _licence);

EARLicense pLicense = EARLicense();
```

```

pLicense.key = _licenceCopy;
pLicense.length = _length;

// TODO handle errors
g_pStore = EARPersonStoreCreate(_storePath, nullptr); // Don't use
               password for encryption
g_lastError = EARGetLastError();

EARFaceDetectionInitializationConfiguration
    detectionInitializationConfiguration;

g_pDetector = EARFaceDetectorCreate(&pLicense,
    EARFaceDetectionType::kEARFaceDetectionType_Retina,
    &detectionInitializationConfiguration, 0);

EARFaceDetectionConfigurationInitWithPreset(&g_faceDetectionConfiguration,
    kEARFaceDetectionConfigurationPreset_Generic);

g_lastError = EARGetLastError();

g_pRecognizer = EARFaceRecognizerCreate(&pLicense, g_pStore,
    faceRecognitionModelType, recognitionModelPerformance, 0);

g_lastError = EARGetLastError();

free(_licenceCopy);

env->ReleaseStringUTFChars(licence, _licence);
env->ReleaseStringUTFChars(modelsPath, _modelsPath);
env->ReleaseStringUTFChars(storePath, _storePath);

```

Calculate match using confidence attribute and return results to the Java layer:

```

jobject confidenceScore(JNIEnv *env, jclass type, jobject
sourceByteBuffer, jint sourceWidth, jint sourceHeight, jint
sourceStride, jobject imageBuffer, jint imageWidth, jint imageHeight,
jint imageStride) {

    jobject result;
    jobjectArray facesResult = nullptr;

    internalClearPersonStore();

    auto *sourceBytes = static_cast<jbyte>
        *(>(env->GetDirectBufferAddress(sourceByteBuffer)));
    jbyte *_sourceBytes = argbToRgba(sourceBytes, sourceWidth,
        sourceHeight, sourceStride);

    auto *const pImageToLearn = new EARBitmap();
    pImageToLearn->pixels = _sourceBytes;
    pImageToLearn->bytesPerRow = sourceStride;
    pImageToLearn->width = sourceWidth;
    pImageToLearn->height = sourceHeight;
    pImageToLearn->pixelFormat = EARPixelFormat::kEAR_RGBA32;

```

```

auto *imageBytes = static_cast<jbyte
    *>(env->GetDirectBufferAddress(imageBuffer));
jbyte *_imageBytes = argbToRgba(imageBytes, imageWidth, imageHeight,
    imageStride);

auto *const pImageToCompareAgainst = new EARBitmap();
pImageToCompareAgainst->pixels = _imageBytes;
pImageToCompareAgainst->bytesPerRow = imageStride;
pImageToCompareAgainst->width = imageWidth;
pImageToCompareAgainst->height = imageHeight;
pImageToCompareAgainst->pixelFormat = EARPixelFormat::kEAR_RGBA32;

EARFaceRecognitionConfiguration learnConfig;
EARFaceRecognitionConfigurationInitWithPreset(&learnConfig,
    kEARFaceRecognitionConfigurationPreset_Learn);
learnConfig.minConfidence = 1;
learnConfig.minCenterPoseQuality = 0.0;
learnConfig.minContrastQuality = 0.0;
learnConfig.minSharpnessQuality = 0.0;
learnConfig.minFaceSearchSize = 40;
learnConfig.minFaceSize = 40;

// First need to add faces from the target image to make sure we get
// unique pids
EARDetectedFaceArrayRef pDetectedFacesToLearnLargest =
    EARFaceDetectorDetect(g_pDetector, pImageToLearn,
        &g_faceDetectionConfiguration);

EARRecognizedFaceArrayRef pRecognizedFacesToLearn =
    EARFaceRecognizerRecognizeDetectedFaces(g_pRecognizer,
        pDetectedFacesToLearnLargest, pImageToLearn, &learnConfig,
        kEARAllowLearning);
// Get a list of stored unique faces
size_t learntTargetFacesCount =
    EARRecognizedFaceArrayGetCount(pRecognizedFacesToLearn);

if (learntTargetFacesCount != 0) {

    EARFaceRecognitionConfiguration recogniseConfig;
    EARFaceRecognitionConfigurationInitWithPreset(&recogniseConfig,
        kEARFaceRecognitionConfigurationPreset_Recognize);
    recogniseConfig.minConfidence = 0;
    recogniseConfig.minCenterPoseQuality = 0.0;
    recogniseConfig.minContrastQuality = 0.0;
    recogniseConfig.minSharpnessQuality = 0.0;
    recogniseConfig.minFaceSearchSize = 40;
    recogniseConfig.minFaceSize = 40;

    // Store the source face
    EARDetectedFaceArrayRef pDetectedFacesToCompare =

```



```

    EARFaceDetectorDetect(g_pDetector, pImageToCompareAgainst,
        &g_faceDetectionConfiguration);

    const size_t found =
        EARDetectedFaceArrayGetCount(pDetectedFacesToCompare);

    EARRecognizedFaceArrayRef pRecognisedFacesToCompare =
        EARFaceRecognizerRecognizedDetectedFaces(g_pRecognizer,
            pDetectedFacesToCompare, pImageToCompareAgainst,
            &recogniseConfig, kEARAllowRecognition);

    size_t facesToCompareCount =
        EARRecognizedFaceArrayGetCount(pRecognisedFacesToCompare);

    if (facesToCompareCount != 0 || found != 0)
        facesResult = env->NewObjectArray(facesToCompareCount,
            g_pJNIClassInfo->detectedFaceClass, nullptr);

    for (size_t j = 0; j < facesToCompareCount; j++) {
        const EARRecognizedFace *face =
            EARRecognizedFaceArrayGetAtIndex(pRecognisedFacesToCompare,
                j);

        // Using confidence here as a measure of match
        const double match = std::min(1.0, face->confidence) * 100;

        jobject leftEyeCenter =
            env->NewObject(g_pJNIClassInfo->pointfClass,
                g_pJNIClassInfo->pointCtr, face->leftEyeCenter.x,
                face->leftEyeCenter.y);
        jobject rightEyeCenter =
            env->NewObject(g_pJNIClassInfo->pointfClass,
                g_pJNIClassInfo->pointCtr, face->rightEyeCenter.x,
                face->rightEyeCenter.y);
        jobject noseTip =
            env->NewObject(g_pJNIClassInfo->pointfClass,
                g_pJNIClassInfo->pointCtr, face->noseTip.x,
                face->noseTip.y);
        jobject leftMouthCorner =
            env->NewObject(g_pJNIClassInfo->pointfClass,
                g_pJNIClassInfo->pointCtr, face->leftMouthCorner.x,
                face->leftMouthCorner.y);
        jobject rightMouthCorner =
            env->NewObject(g_pJNIClassInfo->pointfClass,
                g_pJNIClassInfo->pointCtr, face->rightMouthCorner.x,
                face->rightMouthCorner.y);

        jobject faceObject =
            env->NewObject(g_pJNIClassInfo->detectedFaceClass,
                g_pJNIClassInfo->detectedFaceMatchCtr, face->localId,
                face->confidence, match, face->centerPoseQuality,
                face->contrastQuality, face->sharpnessQuality,

```

```

        face->boundsX, face->boundsY, face->boundsWidth,
        face->boundsHeight, leftEyeCenter, rightEyeCenter,
        noseTip, leftMouthCorner, rightMouthCorner);

    env->SetObjectArrayElement(facesResult, j, faceObject);

    env->DeleteLocalRef(leftEyeCenter);
    env->DeleteLocalRef(rightEyeCenter);
    env->DeleteLocalRef(noseTip);
    env->DeleteLocalRef(leftMouthCorner);
    env->DeleteLocalRef(rightMouthCorner);
    env->DeleteLocalRef(faceObject);
}

    EARRecognizedFaceArrayDestroy(pRecognisedFacesToCompare);
    EARDetectedFaceArrayDestroy(pDetectedFacesToCompare);
}

    EARRecognizedFaceArrayDestroy(pRecognizedFacesToLearn);
    EARDetectedFaceArrayDestroy(pDetectedFacesToLearnLargest);

    if (!facesResult) {
        facesResult = env->NewObjectArray(0,
            g_pJNIClassInfo->detectedFaceClass, nullptr);
    }

    result = env->NewObject(g_pJNIClassInfo->detectorResultClass,
        g_pJNIClassInfo->detectorResultCtr, facesResult, g_lastError);

    delete _sourceBytes;
    delete pImageToLearn;
    delete _imageBytes;
    delete pImageToCompareAgainst;

    return result;
}

```

app/src/main/java/com/real/facecompare/ESDKWrapper.java

Includes JNI API definitions and does JNI invocations:

```

public ArrayList<FaceInfo> confidenceScore(Bitmap imageToLearn, Bitmap
    imageToCompare) {

    final int imageToLearnWidth = imageToLearn.getWidth();
    final int imageToLearnHeight = imageToLearn.getHeight();
    final int imageToLearnByteSize =
        imageToLearn.getAllocationByteCount();
    final int imageToLearnBytesPerPixes = (int) (1f *
        imageToLearnByteSize / (imageToLearnWidth *
        imageToLearnHeight));
    final int imageToLearnStride = imageToLearnBytesPerPixes *

```

```

        imageToLearnWidth;

final ByteBuffer imageToLearnBuffer =
    ByteBuffer.allocateDirect(imageToLearnByteSize);
imageToLearn.copyPixelsToBuffer(imageToLearnBuffer);

final int imageToCompareWidth = imageToCompare.getWidth();
final int imageToCompareHeight = imageToCompare.getHeight();
final int imageToCompareByteSize =
    imageToCompare.getAllocationByteCount();
final int imageToCompareBytesPerPixes = (int) (1f *
    imageToCompareByteSize / (imageToCompareWidth *
    imageToCompareHeight));
final int imageToCompareStride = imageToCompareBytesPerPixes *
    imageToCompareWidth;

final ByteBuffer imageBuffer =
    ByteBuffer.allocateDirect(imageToCompareByteSize);
imageToCompare.copyPixelsToBuffer(imageBuffer);

final EARResult result = confidenceScore(imageToLearnBuffer,
    imageToLearnWidth, imageToLearnHeight, imageToLearnStride,
    imageBuffer, imageToCompareWidth, imageToCompareHeight,
    imageToCompareStride);

final ArrayList<FaceInfo> faces = new ArrayList<>();

if (result.result == 0) {
    Arrays.sort(result.faces, matchComparator);

    for (EARDetectedFace detectedFace : result.faces) {
        faces.add(new FaceInfo(detectedFace,
            ImageUtils.extractFaceBitmap(detectedFace,
            imageToCompare, true)));
    }
}

return faces;
}

```

app/src/main/cpp/CMakeLists.txt

Native build configuration:

```

# For more information about using CMake with Android Studio, read the
# documentation:
#   https://d.android.com/studio/projects/add-native-code.html

# Sets the minimum version of CMake required to build the native library.

# ---|
#   |--- doc - docs directory
#   |--- source - app directory
#   |--- esdk_distribution - binaries
#   |--- include - header files

```

```

#           |--- model - model files
#           |--- lib- libraries - directory names must match to Android
#           ABIs
#           |--- arm64-v8a - content from aarch64-linux-android package
#           |--- armeabi-v7a - content from arm-linux-androideabi
#           package
#
cmake_minimum_required(VERSION 3.10.2)

# Declares and names the project.

project("facecompare")

# esdk directories
set(ESDK_ROOT_DIR
    ${CMAKE_CURRENT_SOURCE_DIR}/../../../../../../esdk_distribution)
set(LIBS_DIR ${ESDK_ROOT_DIR}/lib/)

# JNI wrapper
set(SOURCE ${CMAKE_CURRENT_SOURCE_DIR}/native-lib.cpp)

# headers location
set(HEADERS ${ESDK_ROOT_DIR}/include)

# dependency libs

add_library(ArgusKit-2 SHARED IMPORTED)
set_target_properties(ArgusKit-2
    PROPERTIES IMPORTED_LOCATION
        ${LIBS_DIR}/${ANDROID_ABI}/libArgusKit-2.so)

add_library(ArgusKit SHARED IMPORTED)
set_target_properties(ArgusKit
    PROPERTIES IMPORTED_LOCATION
        ${LIBS_DIR}/${ANDROID_ABI}/libArgusKit.so)

add_library(facedetector SHARED IMPORTED)
set_target_properties(facedetector
    PROPERTIES IMPORTED_LOCATION
        ${LIBS_DIR}/${ANDROID_ABI}/libfacedetector.so)

add_library(facerecognizer SHARED IMPORTED)
set_target_properties(facerecognizer
    PROPERTIES IMPORTED_LOCATION
        ${LIBS_DIR}/${ANDROID_ABI}/libfacerecognizer.so)

add_library(jpeg SHARED IMPORTED)
set_target_properties(jpeg
    PROPERTIES IMPORTED_LOCATION
        ${LIBS_DIR}/${ANDROID_ABI}/libjpeg.so)

add_library(c++_shared SHARED IMPORTED)
set_target_properties(c++_shared

```

```

        PROPERTIES IMPORTED_LOCATION
        ${LIBS_DIR}/${ANDROID_ABI}/libc++_shared.so)

add_library(libtensorflowlite_gpu_delegate SHARED IMPORTED)
set_target_properties(libtensorflowlite_gpu_delegate
    PROPERTIES IMPORTED_LOCATION
        ${LIBS_DIR}/${ANDROID_ABI}/libtensorflowlite_gpu_delegate.so)

add_library(libtensorflowlite SHARED IMPORTED)
set_target_properties(libtensorflowlite
    PROPERTIES IMPORTED_LOCATION
        ${LIBS_DIR}/${ANDROID_ABI}/libtensorflowlite.so)

# The android stuff
find_library(
    log-lib
    log)

# include headers
include_directories(${HEADERS})

# wrapper
add_library( # Sets the name of the library.
    native-lib
    SHARED
    ${SOURCE})

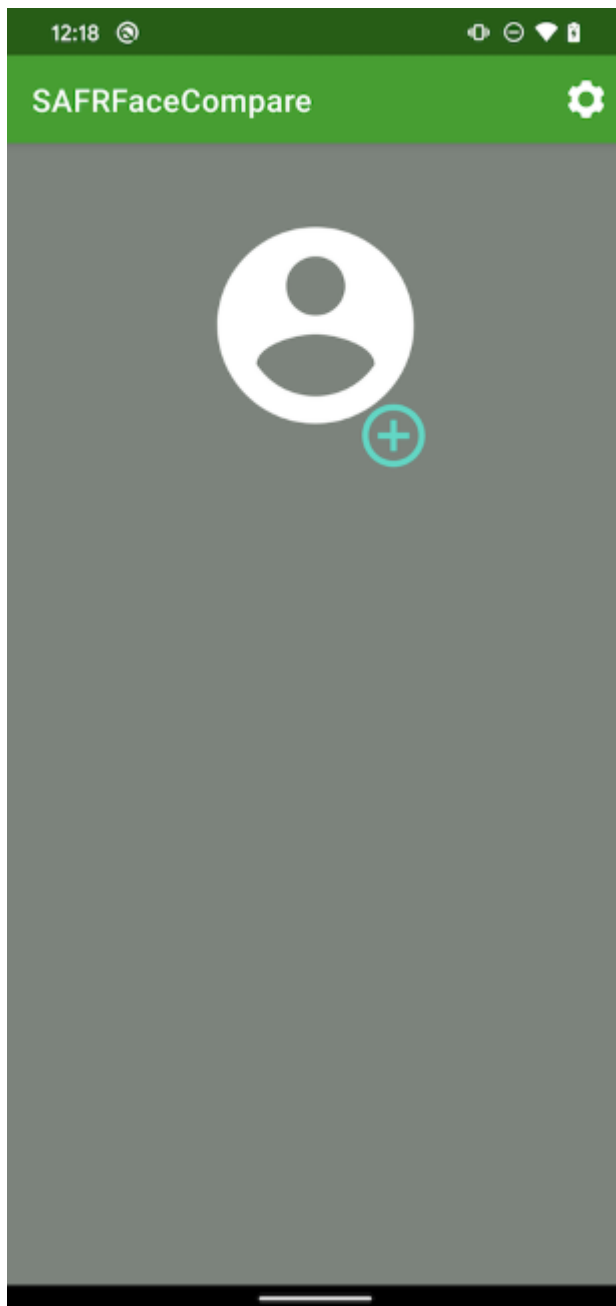
# Link everything.
target_link_libraries(
    native-lib
    ArgusKit-2
    ArgusKit
    facedetector
    facerecognizer
    jpeg
    c++_shared
    libtensorflowlite_gpu_delegate
    libtensorflowlite
    ${log-lib})

```

The rest of the code does Android-specific such as handling image picker, RecyclerView, navigation stuff, etc.

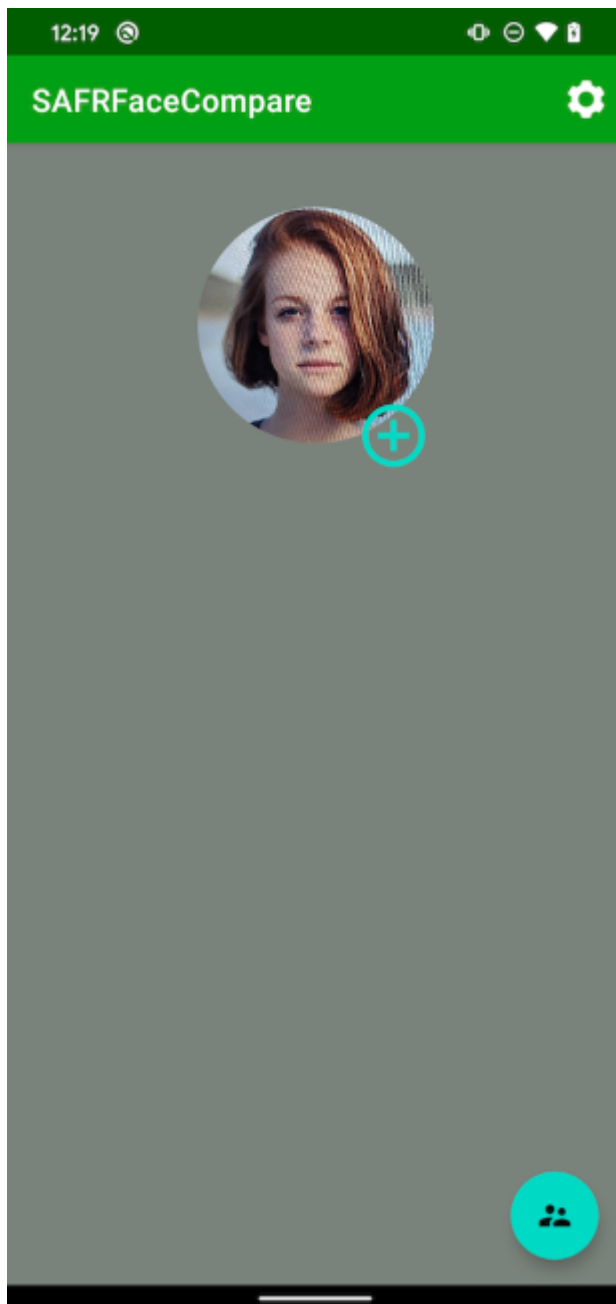
9.5 Application Screens

1. Start



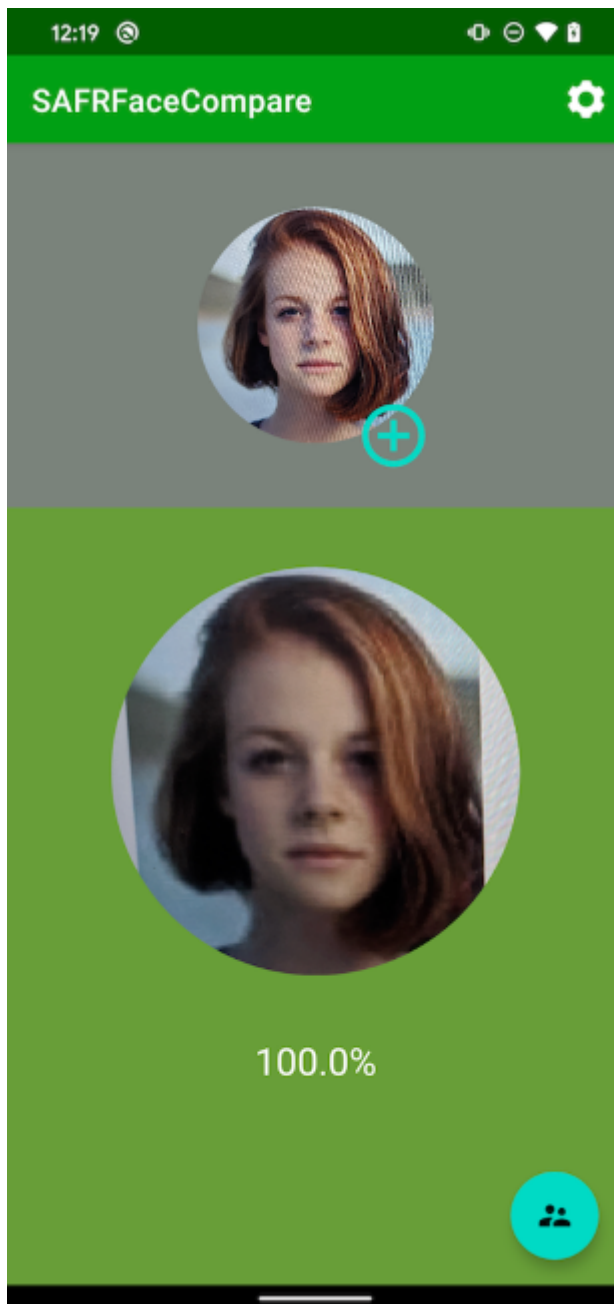
Shows when the app starts. Clicking the plus button will open an image picker to select a source image.

2. **The largest found face is displayed in the image view.**

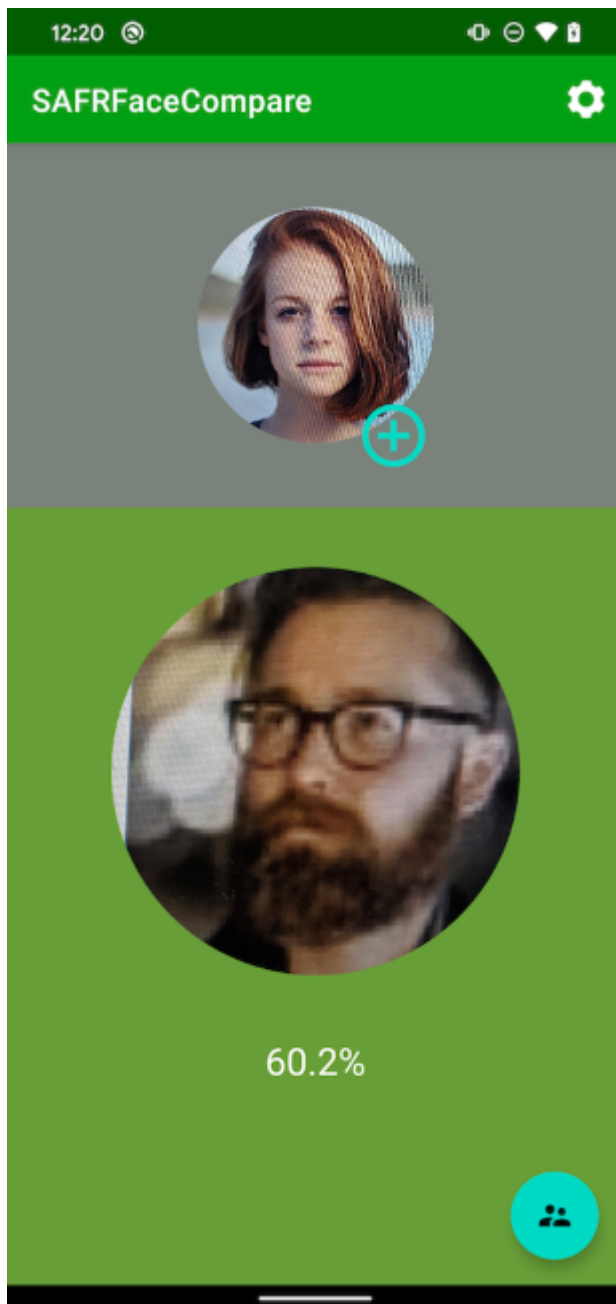


The Floating Action Button (FAB) is displayed when the source face is ready. Clicking the FAB will show an image picker to select a target image. After the target image is selected, the face from the source image is learned and compared with faces being found on the target image. The resultant faces with their matching scores are displayed in the horizontal list.

3. Display results



A 100% match.



Another face.

4. Configure settings



There is a settings page to configure the face recogniser model's type and performance. By default it's set to MaskOptimised/Speed settings. When returned to the main screen, it takes some time for the eSDK to re-initialise, so please wait some time before processing new images.

9.6 Notes

- The app doesn't include error handling.
- The same JNI content will work for Android applications written in the Kotlin language.
- The app includes the code to demonstrate usage of `EARPersonStoreGetMatchingConfidenceForPersonAPI` (see `app/src/main/cpp/native-lib.cpp`, `matchScore` function - line 127-307).

10 Panoptes Command Line Tool

The panoptes command line tool allows you to learn and recognize persons in images. It also provides you with tools to manage persons in a person directory.

Panoptes stores person-related information in a database file named `safr_person.db` that is by default stored in your home directory.

To install panoptes:

1. Decompress the `tar.gz` file you have received. The result is a directory with the name `panoptes` in your current directory.
2. Push this directory and all its subdirectories with `adb push` to a suitable home folder on your device. The Panoptes tool can be found inside the `panoptes` folder.

Note: Many embedded Linux versions set home directory to `/`.

Panoptes automatically creates the database file if it doesn't already exist, and it updates the database file if it does already exist. You can point panoptes to a database file stored in a different location by setting the environment variable `PANOPTES_DB_PATH` before you invoke panoptes. The value of this variable should be an absolute path to the desired database location.

You invoke panoptes with an *action* and *action specific parameters*. You can see what commands panoptes supports by running `panoptes --help`, as shown below.

```
> panoptes --help
panoptes <action>
```

Where <action> is one of:

```
analyze    <path to image(s)> [-Ckey=value ...]
analyze-highres <path to low-res image> <path to high-res image>
            [-Ckey=value ...]
learn      <path to image(s)> [-Dkey=value ...] [-Ckey=value ...]
recognize  <path to image(s)> [-Ckey=value ...]
benchmark <path to image(s)> [-Dkey=value ...] [-Ckey=value ...]\n
remove     <person id>
list       [person id]
edit       <person id> [-Dkey=value ...] [-Rkey ...]
encrypt    <new password>
```

```
-Dkey=value      define person metadata key-value pair
-Rkey            remove person metadata key-value pair
```

```
-Cmin-cpq=value    the required minimum center pose quality [0 - 1]
-Cmin-cq=value     the required minimum contrast quality [0 - 1]
-Cmin-sq=value     the required minimum sharpness quality [0 - 1]
-Cmin-confidence=value the required minimum confidence [0 - 1]
-Cmin-search-size=value the minimum face width/height in pixels for
                    detection
-Cmin-size=value   the minimum face width/height in pixels for
                    acceptance
-Cselect=value     the face selection policy: 'any', 'largest' or
                    'at-location'
-Cloc-x=value      the X coordinate of the location if selection
                    policy is at-location
-Cloc-y=value      the Y coordinate of the location if selection
                    policy is at-location
```

```

-Cmax-threads=value      the maximum number of threads to process
                           detection and recognition (0 - auto detect, 1-100 actual number of
                           threads)
-Cfr-model=value         face recognition model (regular, masked).
                           regular - default
-Cfr-optimization=value  face recognition variant optimized for speed vs
                           accuracy (speed, accuracy). speed - default
-Cfd-type=value          face detector type (normal, high_sensitivity).
                           normal - default
-Cfd-enable-liveness=value use liveness detection (true / false),
                           default - false
-Cfd-liveness-mode=value liveness detection mode (none, last, avg, min,
                           max), default - min
-Cfd-high-sensitivity-resolution=value high sensitivity face detector
                           model input resolution. 640x480 - default
                           Available resolutions: 1280x720, 720x1280, 640x480, 640x640,
                           480x640, 480x480, 480x320, 320x480, 320x240, 320x320, 240x320,
                           240x240, 128x128

-b                        generate benchmark output (used with learn or
                           recognize actions)
-do                        Run detection only (used only with benchmark
                           action)
-wo                        Write output of all models to stdout (used only
                           with benchmark action)
-Brd=N                    Repeat detection (and recognition if applicable)
                           N times per image.
-Omodel-size=(n|t)        Model size for object/person detector (benchmark
                           action only) (n - normal, t - tiny)
-Oinput-size=(s|n|l)      Input size for object/person detector (benchmark
                           action only) (s - small, n - normal, l - large)
-Omodel-precision=(fp32|fp16|int8) Model precision for
                           object/person detector (benchmark action only)

Environment variables:
PANOPTES_DB_PATH          if set, specifies the path and file name for the
                           person store file
PANOPTES_DB_PWD           if set, specifies the password for the person
                           store file
PANOPTES_LICENSE          if set, specifies the panoptes license

```

Important: You have to set the PANOPTES_LICENSE environment variable to the license key you have received from RealNetworks before you invoke panoptes. On a POSIX comparability system, you can do this by executing the following command in a shell window:

```
export PANOPTES_LICENSE=<license key>
```

This command should be added to your shell startup script to ensure it is executed for every new shell window you open.

11 Panoptes Actions

11.1 Analyze Action

The *Analyze* action instructs panoptes to run a face recognition action on an image or a set of images for the purpose of finding out what kind of center pose quality, contrast, and sharpness values can be expected for faces. Notethat panoptes does not learn those faces. Neither does it attempt to match those faces against face information stored in an Identity Database. It tries to find as many faces as it can in the images, and it prints information about the found faces to the console.

Use this action on a set of images you have never processed before to get an idea about what center pose quality (cpq), contrast (cq), sharpness (sq) and face size values you can expect. You can then use those values as a basis for choosing the `min-cpq`, `min-cq`, `min-sq`, and `min-size` values for the *Learn* and *Recognize* actions.

If the `fd-enable-liveness` parameter is set to `true`, RGB liveness detection is enabled and the face detector will produce two additional values for the detected face: a liveness score and a liveness status. RGB liveness detection can be used only with high sensitivity face detector. (i.e. with `-Cfd-type=high_sensitivity`)

Panoptes uses the settings below in analyze mode which can be overridden via the `-C` command line options. (e.g. `-Cmin-search-size`, `-Cfd-enable-liveness`, etc.)

Setting	Default	Range	Description
fd-type	normal	normal, high_sensitivity	Face detector to use. <i>high_sensitivity</i> is slower than <i>normal</i> , but has better accuracy, especially on masked faces.
fd-enable-liveness	false	true, false	When enabled, RGB liveness detection will be used. (Available only with <i>high_sensitivity</i> face detector.)
fd-liveness-mode	min	none, last, avg, min, max	Liveness detection mode used to calculate liveness response in multi-model setup.
min-search-size	40	15 - 720	Minimum face width/height, in pixels, used for face search with the normal face detector.
fd-high-sensitivity-resolution	640x480	1280x720, 720x1280, 640x480, 640x640, 480x640, 480x480, 480x320, 320x480, 320x240, 320x320, 240x320, 240x240, 128x128	Resolution used for face search with the <i>high_sensitivity</i> face detector.

Below is an invocation of panoptes that instructs it to analyze an image and print recognition information about every person found in the image:

```
> panoptes analyze image_with_person.jpeg
```

Note: The path to the image file/image directory must immediately follow the action word.

Invocation of panoptes that will analyze an image and additionally calculate liveness score for every person found in the image. Liveness mode is changed to the average of the *Texture* and *Context* models:

```
> panoptes analyze image_with_person.jpeg -Cfd-type=high_sensitivity
-Cfd-enable-liveness=true -Cfd-liveness-mode=avg
```

11.1.1 Examples

The directory *sample_images* contains sample images that can be used to test RGB liveness detection.

RGB liveness detection achieves best results if a high resolution image is provided for liveness detection. Invocation of panoptes that will provide low resolution image for face detection and high resolution image for RGB liveness detection:

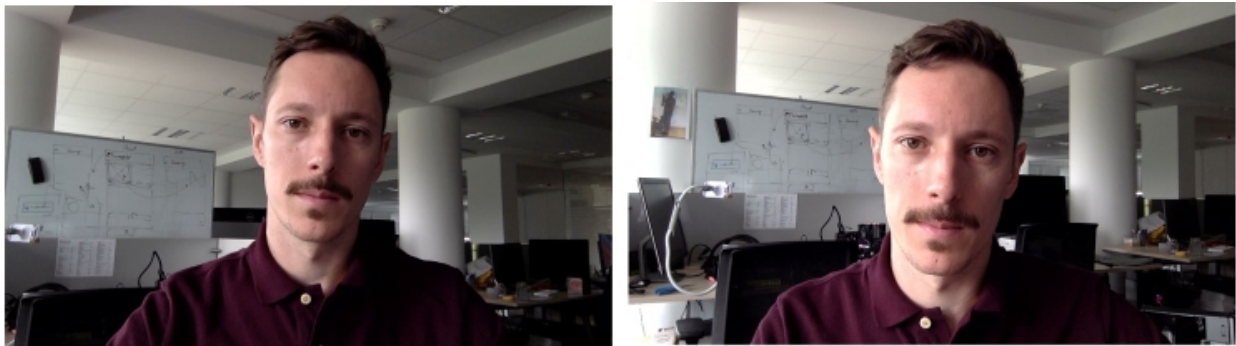
```
> panoptes analyze-highres ../sample_images/Live_LowRes_240x320.jpg
../sample_images/Live_HighRes_1920x2560.jpg
-Cfd-type=high_sensitivity -Cfd-enable-liveness=true
```

If only *Live_LowRes_240x320.jpg* was provided to the *Analyze* action, RGB liveness would not be calculated because the face size wouldn't satisfy the default requirements. For all of the following examples, RGB liveness detection and face detection are done on a single image, but if possible RGB liveness detection should be provided with a higher resolution image.

Live person images are expected to return:

- liveness: 1.00 (or close to 1)
- livenessStatus: 0 (liveness detection has completed successfully)

```
> panoptes analyze ../sample_images/Live_1.jpg -Cfd-type=high_sensitivity
-Cfd-enable-liveness=true
> panoptes analyze ../sample_images/Live_2.jpg -Cfd-type=high_sensitivity
-Cfd-enable-liveness=true
```

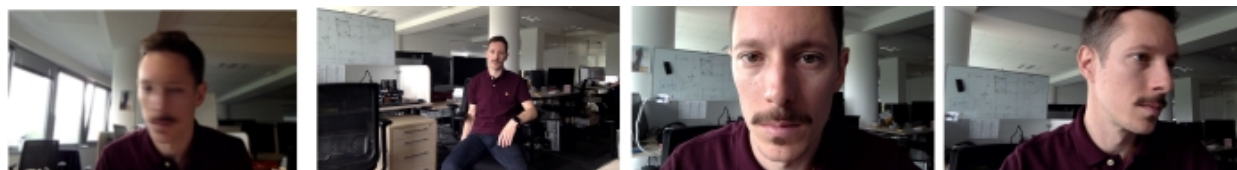


Live person images that don't satisfy face image quality requirements:

- liveness: 0.00
- livenessStatus: 1 (image does not satisfy quality requirements - low quality)
- sq/cpq/cq/face context - below given threshold

```
> panoptes analyze ../sample_images/Live_Blur.jpg
-Cfd-type=high_sensitivity -Cfd-enable-liveness=true
> panoptes analyze ../sample_images/Live_MinFaceSize.jpg
-Cfd-type=high_sensitivity -Cfd-enable-liveness=true
> panoptes analyze ../sample_images/Live_FaceContext.jpg
-Cfd-type=high_sensitivity -Cfd-enable-liveness=true
```

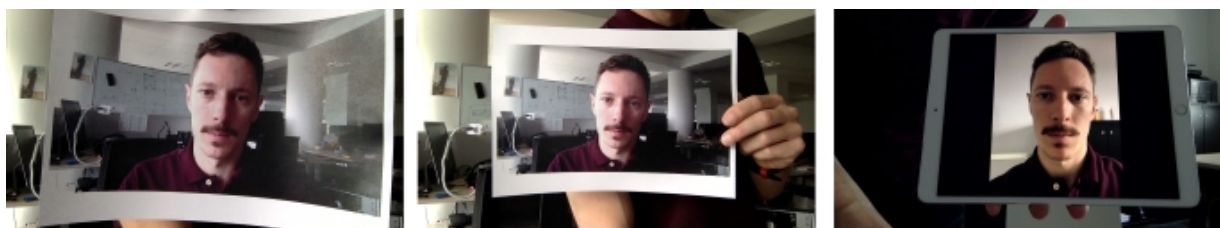
```
> panoptes analyze ../sample_images/Live_CPQ.jpg
-Cfd-type=high_sensitivity -Cfd-enable-liveness=true
```



Fake person images:

- liveness: 0.00 (or close to 0)
- livenessStatus: 0

```
> panoptes analyze ../sample_images/Fake_Paper_1.jpg
-Cfd-type=high_sensitivity -Cfd-enable-liveness=true
> panoptes analyze ../sample_images/Fake_Paper_2.jpg
-Cfd-type=high_sensitivity -Cfd-enable-liveness=true
> panoptes analyze ../sample_images/Fake_Device_1.jpg
-Cfd-type=high_sensitivity -Cfd-enable-liveness=true
```



11.2 Learn Action

This action instructs panoptes to learn a face/person/identity from a JPEG image or a set of persons from a directory with JPEG images. Panoptes focuses on the person with the largest projected face in the image and learns only this person. You may specify metadata for the person. This metadata is stored together with the person's face signature in the person database. Panoptes assigns a unique identifier (GUID) to the person. This GUID can be used with other actions to refer to this person.

Panoptes uses a default set of face recognizer settings, but you can override the settings via `-C` command line argument described previously. The following table lists the supported settings and their default values:

Setting	Default	Range	Description
min-cpq	0.59	0 - 1	Required minimum center pose quality.
min-cq	0.45	0 - 1	Required minimum contrast quality.
min-sq	0.45	0 - 1	Required minimum sharpness quality.
min-confidence	1	0 - 1	Required minimum confidence.
min-search-size	40	15 - 720	Minimum face width/height in pixels used for face search during detection.

Setting	Default	Range	Description
min-size	146	1 - 720	Minimum face width/height in pixels for recognition acceptance.
select	largest	any, largest, at-location	Face selection policy.
loc-x	0.5	0 - 1	X-coordinate of the location if selection policy is at-location.
loc-y	0.5	0 - 1	Y-coordinate of the location if selection policy is at-location.
fd-type	normal	normal, high_sensitivity	Face detector to use. <i>high_sensitivity</i> is slower than <i>normal</i> , but has better accuracy, especially on masked faces.
fd-enable-liveness	false	true, false	When enabled, RGB liveness detection is used. (Only available with <i>high_sensitivity</i> face detector.)
fd-high-sensitivity-resolution	640x480	1280x720, 720x1280, 640x480, 640x640, 480x640, 480x480, 480x320, 320x480, 320x240, 320x320, 240x320, 240x240, 128x128	Resolution used for face search with the <i>high_sensitivity</i> face detector.
fr-model	regular	regular, masked	Face recognition model to use. The <i>masked</i> model is optimized for recognition on masked faces.
fr-optimization	speed	speed, accuracy	Face recognition model optimized for either speed or accuracy.

Below is an invocation of panoptes, instructing it to learn the person visible in the image and to add the metadata `name` with the value `Foo` to the person.

```
> panoptes learn image_with_person.jpeg -Dname=Foo
```

Note: As mentioned previously, the path to the image file/image directory must immediately follow the action word.

11.3 Recognize Action

This action instructs panoptes to recognize a person or a set of persons. You may pass a path to a single JPEG image or a path to a directory containing one or more JPEG images. Panoptes runs a face recognition operation on every image to extract person information, and matches up this person information with the person information stored in the person database. It then lists the unique person ID (GUID) and metadata for every recognized person.

Panoptes uses a default set of face recognizer settings, but you can override the settings. The following table lists the supported settings and their default values:

Setting	Default	Range	Description
min-cpq	0.0	0 - 1	Required minimum center pose quality.
min-cq	0.0	0 - 1	Required minimum contrast quality.
min-sq	0.0	0 - 1	Required minimum sharpness quality.
min-confidence	0.93	0 - 1	Required minimum confidence.
min-search-size	40	15 - 720	Minimum face width/height in pixels used for face search during detection.
min-size	0	1 - 720	Minimum face width/height in pixels for recognition acceptance.
select	largest	any, largest, at-location	Face selection policy.
loc-x	0.5	0.5	X-coordinate of the location if selection policy is at-location.
loc-y	0.5	0.5	Y-coordinate of the location if selection policy is at-location.
fd-type	normal	normal, <i>high_sensitivity</i>	Face detector to use. <i>high_sensitivity</i> is slower than <i>normal</i> , but has better accuracy, especially on masked faces.
fd-enable-liveness	false	true, false	When enabled, RGB liveness detection is used. (Only available with <i>high_sensitivity</i> face detector.)
fd-high-sensitivity-resolution	640x480	1280x720, 720x1280, 640x480, 640x640, 480x640, 480x480, 480x320, 320x480, 320x240, 320x320, 240x320, 240x240, 128x128	Resolution used for face search with the <i>high_sensitivity</i> face detector.
fr-optimization	speed	speed, accuracy	Face recognition model optimized for either speed or accuracy.
fr-model	regular	regular, masked	Face recognition model to use. The <i>masked</i> model is optimized for recognition on masked faces.

Below is an invocation of panoptes, instructing it to print out all recognized persons visible in the given JPEG image. This invocation also shows how to override one of the recognizer settings with a custom value:

```
> panoptes recognize image_with_person.jpeg -Cmin-confidence=0.86
```

Below is an invocation of panoptes which instructs it to print out all recognized persons visible in the given JPEG image. This invocation also shows how to override face detector and face recognition default settings:

```
> panoptes recognize image_with_person.jpeg -Cfd-type=high_sensitivity
-Cfd-high-sensitivity-resolution=128x128 -Cfr-model=masked
-Cfr-optimization=accuracy -Cfd-enable-liveness=true
```

If a person was learned using the regular/masked face recognition model, the same model should be used when attempting to recognize the face. Different face recognition models produce different facial signatures for the same face and should not be compared. If a different model is used when attempting to recognize a face, no error will be returned but the person is unlikely to be recognized.

When using the *high_sensitivity* face detector, this message can occur:

```
ERROR: OpenCL library not loaded - dlopen failed: library
"libOpenCL-pixel.so" not found
ERROR: Falling back to OpenGL
```

This error is actually only a warning and can be ignored.

11.4 Benchmark Action

This action instructs panoptes to instantiate all eSDK capabilities (i.e. face detection, face recognition, emotion estimation, age estimation, gender detection, and object/person detection), measure the duration of each process, and generate a benchmark output before ending execution. You may pass a path to a single JPEG image or a path to a directory which contains one or more JPEG images.

Face recognition is configurable in the same way as it is in the *Recognize* action, described above. In addition, the object/person detector can be configured by specifying the model type (object detector or person detector), model size (normal or tiny), input size (normal, large, or small) and precision.

When running panoptes benchmark for the first time, it may take a while (depending on the device, up to 10 minutes) for benchmark to start as all the models have to be optimized before the first run.

Setting	Default	Options	Description
model-size	Normal	Normal, Tiny	Model size. The tiny model is smaller and faster but has lower accuracy.
input-size	Normal	Small, Normal, Large	Size to which the input image is resized to. Normal = 416px Small = 320px Large = 608px
model-precision	Half	Float, Half, Int8	Precision of the model. The "Float" value represents "float32", while the "Half" value represents "float16".

Below is an invocation of panoptes which will benchmark the eSDK on a single image with a person detector

using fp16 precision:

```
> panoptes benchmark image_with_person.jpeg -Omodel-type=p  
-Omodel-precision=fp16
```

11.4.1 Interpreting Confidence Values

Confidence Value	Interpretation
≥ 1.0	Certain match. (Values > 1.00 are possible indicating even greater certainty.)
[0.93 1.0>	Close match.
[0.86 0.93>	Possible match with low confidence.
[0.82 0.86>	Similar face with no confidence of match.
[0.00 0.82>	Different face.

Note: As mentioned previously, the path to the image file/image directory must immediately follow the action word.

11.5 List Action

The *List* action instructs panoptes to display the person ID and metadata of all persons in the person database. You can optionally specify a known person ID to limit the display to just this person.

11.6 Remove Action

The *Remove* action takes a person ID and instructs panoptes to delete all information about this person from the person database.

11.7 Edit Action

The *Edit* action instructs panoptes to edit the metadata of a person, specified by their person ID. Metadata key-value pairs can be added or updated with the *-D* switch. A metadata key-value pair may be deleted with the *-R* switch.

11.8 Encrypt Action

The *Encrypt* action instructs panoptes to encrypt the person store database file with the provided password. This action also allows you to change the password of an existing database file. Note that the `PANOPTES_DB_PWD` environment variable must be set to the old password so that panoptes is able to decrypt the existing database file before encrypting it with the new password.

Important: Don't forget to update the `PANOPTES_DB_PWD` environment variable with the new password after the *Encrypt* action has completed and before you try to execute a *Learn* or *Recognition* action.

12 Panoptes Benchmark Output

The **Learn** and **Recognize** actions support the generation of benchmark output. You enable this mode by passing the **-b** switch to panoptes.

Below is an invocation of panoptes instructing it to print out benchmark data into a tab-delimited file for all recognized persons visible in all JPEG images in a specified directory:

```
> panoptes recognize <path to image(s)> -b [-Ckey=value ...]
```

The format of the benchmark output is:

```
<image file name> <person ID> <confidence percentage> <recognition time>
```

where:

- Fields are tab delimited.
- **image file name**: The file name of the JPEG image.
- **person ID**: The person ID.
- **confidence percentage**: The recognition confidence expressed as a percentage.
- **recognition time**: The time in milliseconds it took to run recognition on the image expressed.

Note: Confidence percentage can range from 0 to approximately 180. A value of 100 indicates near certainty of a match.

You can redirect the benchmark output to a file as shown below to open it in a spreadsheet application. The data is stored as a tab-separated list.

```
> panoptes recognize directory_with_jpegs -b > my_data.tsv
```