FaceMatch SDK Documentation

Introduction

FaceMatch is a cross-platform software library for detecting and comparing faces of people in video and images, as well as storing them in a database for later (possibly offline) processing. The FaceMatch SDK can also be used to perform automatic face extraction (from e.g. a webcam input stream) in real time, and communicate the resulting bounding-box information to the third party application.

Dependencies

Windows

- Visual Studio C++11 compiler
- OpenCL library (optional, experimental). See **OpenCL acceleration** OpenCL section below

Linux

- Ubuntu 18.04 and up
- C++11 compiler and build tools: `sudo apt install build-essential`
- Lib Poco 1.8.0 and up : `sudo apt install libpoco-dev`
- FFmpeg library with de/encoders: `sudo apt install libavcodec-dev`
- FFmpeg library with (de)muxers: `sudo apt install libavformat-dev`
- OpenCL library (optional, experimental). See **OpenCL acceleration** OpenCL section below

OpenCL acceleration

The FaceMatch SDK comes with OpenCL support which can significantly increase its runtime performance (depending on available hardware) while reducing CPU utilization.

This was evaluated on Intel Iris+ integrated graphics hardware, for which client drivers can be installed by following the instructions for your specific operating system below. Once present on your system, the SDK will automatically detect these and make use of OpenCL acceleration when performing network inference. OpenCL drivers for other platforms (e.g. AMD) may also be compatible, but have not yet been tested. Note also that utilizing the GPU does not necessarily result in a speedup and may cause program instability depending on the specific hardware and driver combination.

- **Linux:** Follow the installation instructions at https://github.com/intel/compute-runtime/releases/latest under the "Installation procedure" section.
- **Windows:** Generally, Windows has already drivers that enable OpenCL support for the Intel integrated graphics hardware, but more performance can be obtained if the specific Intel drivers are installed.
  We have verified OpenCL functionality on NUC7i*BN and NUC8i7BE systems, for which suitable drivers can be downloaded here: https://downloadcenter.intel.com/download/28974/
  For the majority of other Intel processors, the Windows DCH drivers can be downloaded here: https://downloadcenter.intel.com/download/29058/Intel-Graphics-Windows-10-DCH-Drivers?product=80939

OpenCL acceleration can be enabled/disabled dynamically at runtime.

Minimum requirements

- **Windows 10** or **Ubuntu 18.04**
- Intel i5 or i7 4th generation
- 4 GB of RAM
- 500 MB of free disk space
- Input frames with a resolution of 640x480

Getting Started

After the dependencies are met, navigate to the samples directory and compile the example projects:

- **Linux** : cd samples; make

- **Windows** : Open the samples.sln sample project in the samples directory with Visual Studio and compile the examples

The example applications are a good starting point to understand the capabilities of the SDK, its usage and how to start a project with face analysis features.

FaceMatch SDK runtime workflow

A typical runtime workflow when using the FaceMatch SDK is:

- Create a fm::Settings instance with proper license key and neural networks path.
- Create a fm::FaceMatch instance with the fm::Settings instance.
- Authenticate the SDK via fm::FaceMatch::authenticate().
- Detect faces in an image with fm::FaceMatch::detectFaces().
- Compare detected faces with fm::FaceMatch::compareFaces().
- Process the fm::Face instances detected in each image, which will usually include computing a "faceprint" (ie facial datapoints) with fm::FaceMatch::computeFacePrint() and storing the face data with fm::FaceMatch::dbSaveFace() for later search / comparison operations.

Comparison Details

The fm::FaceMatch::compareFaces and fm::FaceMatch::dbCompareFaces functions return a similarity score in the range [0, 1], where 1 means that the two faces are identical and 0 means that the two faces are maximally different. Sightcorp provides two default thresholds to interpret the similarity score:

- **0.55**
- **0.85**

Similarity values less than **0.55** indicate "no match" between two given faces, while similarity values greater than **0.85** indicate a "good match". Similarity values between **0.55** and **0.85** can be considered a "fair match", but in these cases the comparison confidence is lower and human opinion remains advised.

Database

Detected faces can be stored in a database which is serializable to disk at any time. This allows persisting face data across FaceMatch runs and building up a library against which to compare new faces for similarity scoring applications. The database is indexed by faceprints in all functions prefixed by "db" and may be reloaded by subsequent instances of the SDK.

Prior to being commited to the database, faceprints computed by fm::FaceMatch::computeFacePrint are only stored in temporary memory and will not persist across SDK reloads. Temporary faces are not comparable by fm::FaceMatch::dbCompareFaces until fm::FaceMatch::dbSaveFace is called, but can be removed by calling fm::FaceMatch::releaseFacePrint.

A search initiated by fm::FaceMatch::dbSearchFaces() will scan through the entire database using a number of threads determined by fm::Settings::dbNumThreads, which defaults to the system's native CPU core-count. Searching with more threads is generally recommended for large databases.

NOTE: Although the FaceMatch SDK has been designed with thread-safety in mind, it offers no protection against multiple processes concurrently modifying the same database file. Such preventive measures should be taken separately when integrating the SDK.

License

The FaceMatch SDK needs to be initialized with a license key in order to work. If you don't have one yet, contact sales@sightcorp.com. To allow the SDK to compare faces (eg. fm::FaceMatch::compareFaces) a FaceMatch instance needs to be authenticated first through the fm::FaceMatch::authenticate() call.

Sample usage

```
#include <iostream>
#include <fm_facematch.h>
int main()
{
```

```
f           m::FaceMatch fm;
            fm::Settings s;
            s.neuralNetworksPath = "/path/to/neuralnetworks";
            s.licenseKey = "01234567890abcdef01234567890abcdef";
            fm.setSettings( s );
            fm.authenticate();
            for( const fm::Face & face : fm.detectFaces( cv::imread( "/path/to/an/image" ) ) )
            std::cout << face.boundingBox << std::endl;
            return 0;
}
```

Quality of estimates

The SDK has been thoroughly tested against several datasets to maintain overall accuracy. However, in real world scenarios a lot of different factors can influence results, such as backlight, shadows or poorly positioned cameras which can lower the quality of the estimates. Please make sure to take the following into account when configuring your scenario:

Camera Settings

Depending on your camera manufacturer, it should be possible to configure camera parameters for the best results. The most important part of the configuration is to enable anti-flicker for your region (50 or 60 Hz) and to disable automatic gain, white balance and focus control. This is a fundamental step, as the software will recognize any changes in the images as a change in the face, significantly affecting the classification and tracking results.

Positioning

An ideal positioning of the camera would be aimed at a narrow space where people pass one by one, such as a doorway. You can zoom in on the upper part of the door and capture each person's face one by one while they are entering or exiting. This allows for a high face image resolution as opposed to a scenario where people can pass by alongside each other and thus a lesser level of zoom must be applied to process multiple people at once.

Angle

The best age and gender results are obtained when people are captured frontally. The bigger the angle of the camera with respect to your ROI is, the bigger the error margins are on the age and gender estimates. For example, when using a ceiling mounted camera looking at a door, make sure that you try to minimize the angle of the camera looking down at the door by moving it backwards and apply zoom on your ROI.

Distance

The further the person is from the camera, the smaller the resolution of the face in the image will be and the harder it will be to detect the person. To detect people from a distance, two things are required. Firstly, the capture resolution of the camera must be sufficiently large, 720p or 1080p is advised. Secondly, optical or digital zoom can be used in order to detect faces with sufficient resolution.

Lighting

The faces must be sufficiently and frontally lit. In case of nonlinear light patterns on the face or strong sideways illumination, the SDK will have more difficulty to correctly process the features of a person. Back illumination, such as a window behind a person, darkens the face feature and is thus less desirable. This can be counteracted to some extent by disabling automatic camera adjustment such as white-balance, focus and exposure controls.