FaceMatch SDK Documentation

始めに

FaceMatch は、ビデオとイメージ内の顔を検出・比較すると同時に、後の処理（おそらくオフラインで）のために結果を保存する、クロスプラットフォームのソフトウェアライブラリィです。FaceMatch SDK はリアルタイムでウェブキャムカメラや入力ストリームなどから自動的に顔を抽出して、結果として得られる情報をサードパーティのアプリケーションとコミュニケートするように使えます。

クロスプラットフォーム：　Windows・Linux・Macintosh・iOS・Android の複数のオペレーティングシステム(OS)に対応しています。

制約条件

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 の加速

The FaceMatch SDK は、ランタイム性能を著しく増す（利用するハードウェアに依存しますが）ことが出来る OpenCL のサポートの下に提供され、CPU の使用を減少させます。Sightcorp 社は、下記のオペレーティングシステム毎のインストラクションに基づいて、クライアントドライバがインストールされている Intel Iris+統合グラフィックスハードウェアの上で評価しました。貴社のシステムにそれらが存在すれば、SDK は自動的にこれらを検出し、ネットワークの推測を実行する時に OpenCL 加速を利用します。例えば AMD のような他のプラットフォームの場合、OpenCL ドライバと整合性があるかもしれませんが、テストしたことはありませんのでご注意ください。GPU を利用してもスピードアップしなかったりするかもしれませんが、特定のハードウェアとドライバの組み合わせにより不安定を引き起こしているかもしれません。ご注意ください。

Linux: "Installation procedure"の文節に記述されている https://github.com/intel/compute-runtime/releases/latest のインスタレーションインストラクションに従ってください。

Windows: 一般に Intel の統合型グラフィックスハードウェアは OpenCL をサポートするドライバを既に備えていますが、もし特殊な Intel ドライバがインストールされれば一層性能が増すでしょう。Sightcorp では以下の URL から適切なドライバをダウンロードして、NUC7i*BN と NUC8iBF システム上で OpenCL を評価しました： https://downloadcenter.intel.com/download/28974/
他の大部分の Intel プロセッサでは、Windows DCH ドライバは、次の URL からダウンロードできます： https://downloadcenter.intel.com/download/29058/Intel-Graphics-Windows-10-DCH-Drivers?product=80939

OpenCL 加速機能はランタイムでダイナミックに動作させたり、動作不可にすることが出来ます。

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

システムが制約条件と一致したら、サンプルディレクトリィを参照し example projects をコンパイルしてください:

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

例題のアプリケーションから始めることにより、SDK の能力・使い方を最適化できますのでプロジェクトを理解できるでしょう。

FaceMatch SDK ランタイムワークフロー

FaceMatch SDK を使ったときの典型的なランタイムワークフローは以下のようになります：

＊ 適切なライセンスキーとニューラルネトワークパスを使って、fm::Settings instance を作成してください。

*fm： FaceMatch instance.を使って fm::FaceMatch instance を作成してください。

*fm::FaceMatch::authenticate()を介して、SDK を認証してください。

*fm::FaceMatch::detectFaces()によってイメージ内の顔を検出してください。

＊ fm::FaceMatch::compareFaces()によって顔面を比較してください。

*fm::FaceMatch::computeFacePrint()によりイメージ内で検出された fm::Face instances を処理してください。このイメージには通常”faceprint“（所謂顔のデータポイント）の計算を含んでおり、fm::FaceMatch::dbSaveFace()を使って後の検索/比較処理のための顔データを保存します。

顔面比較結果

fm::FaceMatch::compareFaces と fm::FaceMatch::dbCompareFaces 機能は、[0, 1]の範囲の similarity スコアで表示されます。

“1”： 二つの顔が完全に一致していること

“0”： 2 つの顔が最大限異なっている

ことを表しています。Sightcorp はデフォルトとして次の値を提供していますが、similarity スコアはユーザにより自由に変更可能です。

- 0.55
- 0.85

0.55 以下の similarity の値は"no match"（不一致）を示します。

0.85 以上の similarity スコアは"good match"を示します。

0.55 と 0.85 の間の similarity 値は"fair match"を示していますが、これらのケースの場合、比較 confidence（信頼度）は低く人間の参考意見を聞く必要があるでしょう。

データベース

検出された顔はデータベースに保存可能ですが、常にディスクに対して連続して保存します。こうすることで FaceMatch の動作中に顔面データを存続させたり、アプリケーションに従って新しい顔との similarity の比較するライブラリィを構築することが可能です。データベースは"db"でプリフィックスした全てのファンクション内で faceprint によりインデックスを付けられ、SDK の連続したインスタンスにより再ロードされるかもしれません

データベースに引き渡す前に、fm::FaceMatch::computeFacePrint により計算された faceprints は一時的なメモリに格納され、SDK の再ロードに亘って持続する訳ではありません。一時的な顔は fm::FaceMatch::dbSaveFace がコールされるまで、fm::FaceMatch::dbCompareFaces により比較されることはありませんが、fm::FaceMatch::releaseFacePrint をコールすることで削除可能です。

fm::FaceMatch::dbSearchFaces()により起動される検索は、fm::Settings::dbNumThreads により決められるスレッド数分のデータベース全体をスキャンします。これはシステムのネーティブ CPU コアカウントにデフォールトします。それ以上の検索は一般に大規模なデータベース向けです。

注： FaceMatch SDK はスレッドの安全を考慮して設計されましたが、同じデータベースファイルを同時に修復するのに対して保護していません。そのような予防的調整は、SDK を統合する時に別途考慮しておくべきでしょう。

ライセンス

FaceMatch SDK を動かすにはライセンスキーで起動しなければなりません。もしそれを持っていない場合には、販売代理店に連絡してください。SDK を利用して顔面の比較

（eg. fm::FaceMatch::compareFaces）を許可するには、FaceMatch インスタンスは fm::FaceMatch::authenticate() コールで一度認証する必要があります。

サンプルプログラム

```
#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;
}
```

推測の品質

SDK は全体の精度を維持するために数種のデータセットを利用して徹底的にテストを行いました。しかしながら、実際の世界のシナリオでは多くの異なるファクタが結果に影響を与えます。例えば、バックライト・影・まずい位置に設置されたカメラなどは推測の品質を低下させます。実用化の際には以下の示すような事柄に配慮してください：

カメラの設定

カメラメーカによっては、最高の結果を提供するカメラパラメータの構築が可能となるかもしれません。構築の際の最も重要な部分は自分の地域（50 または 60 ヘルツ）のフリッカーレスに設定し、自動ゲイン・ホワイトバランス・フォーカスコントロールを不可とすることです。ソフトウェアはクラシフケーションと追跡結果に極めて強く影響を与える顔面内の変化をイメージの変化として認識しますので、これは基本的なステップです。

配置

カメラの最適な配置とは、玄関口のように人が一人ずつ通過するように狭いスペースを対象にして下さい。人の顔を一人一人取得したら退出するように、ドアの上部にズームしてください。このような設定をするとイメージ解像度を高めることが可能です。一方人が並んで通過するような形で配置すると、ズームが効かず複数の人間を認識しなければならなくなります。

角度

人を正面で取得すると年齢も性別も正しく認識することが出来ます。ROI(Region of Interest=対象領域)に関してカメラの角度が広角だと、年齢と性別の推定誤差が大きくなります。例えば、ドアを見ている天井設置カメラの場合には、カメラを後ろに動かし ROI にズームしてドアを見下ろすようにカメラ角度を最小化してください。

距離

カメラからの距離が遠いと、イメージとして顔の解像度は小さくなり、人を検出するのが厳しくなります。遠方から人を検出するには、2 つの点に注意してください:

1） カメラで取得する解像度は出来るだけ大きく、720pixel とか 1,080pixel が望ましい。

2） 十分な解像度で顔を検出するためにオプティカルまたはディジタルズームを使うことを推奨します。

照明

顔面は出来るだけ正面から照明が当たるようにして下さい。顔面に対して非線形照明パターンあるいは強力な間接照明の場合には、SDK は人の特徴を正確に処理できません。人の背後に窓があるようなバックイルミネーションは顔の特徴をくらくして望ましくありません。ホワイトバランス・焦点や露出のようなカメラの自動調節を不可にして、調整してください。

FaceMatch

Detailed Description

FaceMatch SDK engine class.

fm::FaceMatch クラスは SDK のエンジンです。fm::FaceMatch オブジェクトを例示することにより、SDK エンジンは設定されイメージの処理準備 OK となります。fm::FaceMatch クラスは、外部構築パラメータを持つ fm::Settings により制御されます。fm::FaceMatch クラスの主たる目的はイメージである顔を検出し、顔のユニークな文様を生成する音です。顔の文様は "faceprint"と呼ばれます。"Faceprint"は内部に保存され、外部でユニークな ID（UUID）で参照され fm::FaceMatch::computeFacePrint ファンクションにより戻されます。compareFaces ファンクション(fm::FaceMatch::dbCompareFaces and **fm::FaceMatch::compareFaces)**によって異なる faceprints の similarity スコアを計算することが可能です。

Constructor & Destructor Documentation

◆ FaceMatch() [1/2]

fm::FaceMatch::FaceMatch (const fm::Settings & settings)

fm::FaceMatch constructor.

Parameters

settings A fm::Settings instance containing the configuration to be used by the FaceMatch SDK engine.

Main fm::FaceMatch constructor. May throw `std::runtime_error` exceptions if the fm::Settings::neuralNetworksPath specified in the settings does not contain the required networks or the fm::Settings::faceDatabasePath is not a valid database file.

Example usage:

```
fm::Settings s;
s.neuralNetworksPath = "/path/to/neuralnetworks";
```

```
try
{
fm::FaceMatch fm{ s };
}
catch( const std::runtime_error & e )
{
std::cerr << "Could not construct FaceMatch object : " << e.what() << std::endl;
}
```

Example usage:

```
fm::FaceMatch fm;
fm::Settings s;
s.neuralNetworksPath = "/path/to/neuralnetworks";
try
{
fm.setSettings( s );
}
catch( const std::runtime_error & e )
{
std::cerr << "Invalid settings : " << e.what() << std::endl;
}
```

◆ FaceMatch() [2/2]

fm::FaceMatch::FaceMatch (**fm::FaceMatch** && fm) noexcept

fm::FaceMatch move constructor.

A **fm::FaceMatch** object cannot be copied, but it can be moved. The move constructor allows for this.

Example usage:

```
fm::FaceMatch fm;
fm::FaceMatch fm2 = std::move( fm );
```

Member Function Documentation

◆ authenticate()

void fm::FaceMatch::authenticate ( )

Authenticates the SDK.

Authenticate the **FaceMatch** SDK against Sightcorp's licensing servers to allow for its usage. Throws an `std::runtime_error` if the authentication fails.

Example usage:

```
fm::Settings s;
s.neuralNetworksPath = "/path/to/neuralnetworks";
s.licenseKey = "license key";
fm::FaceMatch fm{ s };
try
{
fm.authenticate();
}
catch( const std::runtime_error & e )
{
std::cerr << "Could not authenticate the SDK : " << e.what() << std::endl;
}
```

◆ compareFaces() [1/2]

float
fm::FaceMatch::compareFaces (const cv::Mat & image1,
const cv::Rect & faceLocation1,
const std::string & uuid2
)

Compares a face contained in the input image against a faceprint previously computed.

Parameters

image1	A 3-dimensional cv::Mat array representing a BGR image. Must be non empty.
faceLocation1	The face location (bounding box) calculated by detectFace for image1.
uuid2	Unique identifier of the second face to be compared.

Returns

A similarity-score between 0 and 1 computed over the face detected in image1 at faceLocation1 and the face identified by uuid2.

Refer to **Comparison Details** section for an interpretation of the similarity score.

◆ compareFaces() [2/2]

```
float fm::FaceMatch::compareFaces ( const cv::Mat &  image1,
                                    const cv::Rect &  faceLocation1,
                                    const cv::Mat &  image2,
                                    const cv::Rect &  faceLocation2
                                  )
```

Compares two faces contained in the two input images.

Parameters

image1	A 3-dimensional cv::Mat array representing a BGR image. Must be non empty.
faceLocation1	The face location (bounding box) calculated by detectFace for image1.
image2	A 3-dimensional cv::Mat array representing a BGR image. Must be non empty.
faceLocation2	The face location (bounding box) calculated by detectFace for image2.

Returns

A similarity-score between 0 and 1 computed over the faces detected in image1 at faceLocation1 and in image2 at faceLocation2.

Refer to **Comparison Details** section for an interpretation of the similarity score.

◆ computeFacePrint()

```
std::string fm::FaceMatch::computeFacePrint ( const cv::Mat &  image,
                                              const cv::Rect &  faceLocation
                                            )
```

Computes the faceprint of a face.

Parameters

image A 3-dimensional cv::Mat array representing a BGR image. Must be non empty.

faceLocation The face location (bounding box) calculated by detectFace.

Returns

A unique identifier that can be used to reference the face data in other API functions.

The computed faceprint is stored in memory. A reference to it is returned in the form of an identifier. The identifier can be used by other methods like **fm::FaceMatch::dbSearchFaces** or **fm::FaceMatch::dbSaveFace** to point to the faceprint computed by this method.

◆ dbCompareFaces()

```
float fm::FaceMatch::dbCompareFaces ( const std::string &  uuid1,
                                      const std::string &  uuid2
                                    )                      const
```

Compares two faces stored in the database.

Parameters

uuid1 Unique identifier of the first face to be compared.

uuid2 Unique identifier of the second face to be compared.

Returns

A similarity-score between 0 and 1 computed over the faces identified by uuid1 and

uuid2.

Refer to **Comparison Details** section for an interpretation of the similarity score.

◆ dbDropFace()

bool fm::FaceMatch::dbDropFace (const std::string & uuid)

Removes a faceprint from the database.

Parameters

uuid Unique identifier of the face to be removed from the database.

Returns

true on success, false on failure.

◆ dbGetAllFaces()

std::vector< std::string > fm::FaceMatch::dbGetAllFaces ( ) const

Returns all faceprints contained in the database.

Returns

A vector of face identifiers.

◆ dbSaveFace()

bool fm::FaceMatch::dbSaveFace (const std::string & uuid)

Saves the faceprint previously generated by **fm::FaceMatch::computeFacePrint** in the database.

Parameters

uuid Unique identifier of the face data to transfer from temporary memory.

Returns

true on success, false on failure.

◆ dbSearchFaces()

std::vector< std::pair< std::string, float > > fm::FaceMatch::dbSearchFaces (const std::string & uuid, size_t maxFaces, float minScore) const

Returns all faces whose similarity to the face identified by uuid is greater or equal to minScore.

Parameters

uuid Identifier of the face against which database entries should be compared.

maxFaces The maximum number of faces to be returned when the database contains multiple entries.

minScore The minimum similarity score determining if a database entry is included, should be between 0 and 1.

Returns

A vector of <face identifier, similarity score> pairs.

This method scans the whole face database and compares each entry to the faceprint identified by `uuid`. Entries with a similarity lower than `minScore` are discarded. The faces are sorted in descending order by similarity. At most `maxFaces` entries are returned.

◆ detectFace()

fm::Face fm::FaceMatch::detectFace (const cv::Mat & image)

Analyzes an image and returns the largest face by (bounding box) surface area.

Parameters

image A 3-dimensional cv::Mat array representing a BGR image. Must be non empty.

Returns

A fm::Face object detected in the input image.

◆ detectFaces()

std::vector< fm::Face > fm::FaceMatch::detectFaces (const cv::Mat & image)

Analyzes an image and returns all detected faces.

Parameters

image A 3-dimensional cv::Mat array representing a BGR image. Must be non empty.

Returns

An std::vector of fm::Face objects detected in the input image.

◆ getSettings()

fm::Settings & fm::FaceMatch::getSettings ( ) const

Returns a copy of the settings object in use.

Returns

A copy of the settings object in use.

Example usage:

```
fm::FaceMatch fm;
fm::Settings s = fm.getSettings();
s.neuralNetworksPath = "/path/to/neuralnetworks";
fm.setSettings( s );
```

◆ isDatabaseDirty()

bool fm::FaceMatch::isDatabaseDirty ( ) const

Checks if the database state is unsaved.

Returns

True if the database has been modified since the last fm::FaceMatch::loadDatabase or fm::FaceMatch::saveDatabase.

◆ loadDatabase()

void fm::FaceMatch::loadDatabase ( )

Reads the face database from disk at the path specified in fm::Settings::faceDatabasePath.

An exception is thrown if the database file can not be loaded. Possible reasons for failure include:

- database file does not exist
- database file is not readable (e.g. due to restricted permissions)
- database file is corrupted
- database file version is not supported by the current SDK version

Example usage:

```
fm::Settings s;
fm::FaceMatch fm;
s.neuralNetworksPath = "/path/to/neuralnetworks";
s.faceDatabasePath = "/path/to/facedatabasefile";
s.licenseKey = "license key";
fm.setSettings( s );
fm.authenticate();
fm.loadDatabase();
// use dbSearchFaces to search for similar faceprints
```

◆ operator=()

fm::FaceMatch & fm::FaceMatch::operator= (fm::FaceMatch && fm) noexcept

fm::FaceMatch move assignment operator.

A **fm::FaceMatch** object cannot be copied, but it can be moved. The move assignment operator allows for this.

Example usage:

```
fm::FaceMatch fm;
fm::FaceMatch fm2;
fm2 = std::move( fm );
```

◆ releaseFacePrint()

bool fm::FaceMatch::releaseFacePrint (const std::string & uuid)

Remove faceprint computed by **fm::FaceMatch::computeFacePrint** from memory.

Parameters

uuid The unique identifier of the temporary face data to remove.

Returns

true on success, false on failure.

◆ saveDatabase()

void fm::FaceMatch::saveDatabase ( )

Writes the face database to disk at the path specified in **fm::Settings::faceDatabasePath**.

An exception is thrown if the database can not be saved. Possible reasons for failure include:

- database file is not writeable
- database contains no entries
- database contains too many entries

◆ setSettings()

void fm::FaceMatch::setSettings (const **fm::Settings** & settings)

Set new settings to be used at runtime.

Parameters

settings A **fm::Settings** instance containing the configuration to be used by the **FaceMatch** SDK engine.

May throw `std::runtime_error` exceptions if the **fm::Settings::neuralNetworksPath** specified in the settings does not contain the required networks or the **fm::Settings::faceDatabasePath** is not a valid database file.

Example usage:

```
fm::FaceMatch fm;
fm::Settings s;
s.neuralNetworksPath = "/path/to/neuralnetworks";
fm.setSettings( s );
```