# なりすまし判定技術

## SDK/インターフェース仕様書(ver.2.2.1)

ディープラーニング対応版

株式会社スワローインキュベート

2023年11月27日

# ■はじめに

なりすまし判定SDKは、株式会社スワローインキュベートが提供しています。

本書に基づき、当SDKをご利用いただく前に、以下のご注意事項を十分に読んだ上で、

ご利用いただきますようお願いします。

# ■ご注意事項

・本書は、予告なしに変更されることがあります。

・本書を無断で、複製、転用、公衆送信、貸与等を行わないようお願いします。

・SDKをご利用いただくには、あらかじめ当社利用規約に同意いただく必要があります。

詳しくは営業担当までお問い合わせください。

## お問い合わせ

**株式会社スワローインキュベート**

**なりすまし判定技術　テクニカルサポート窓口**

**TEL: 029-886-9912　MAIL: support@swallow-incubate.com**

# ■SDK更新履歴

| バージョン | 日時 | 変更内容サマリー |
|---|---|---|
| ver.1.0.0 | 2019年10月29日 | 初版 |
| ver.2.0.0 | 2021年09月30日 | ・ディープラーニングモデルの導入<br>・くちびる判定機能の削除<br>・サングラス判定機能の追加<br>・マスク検出機能の追加 |
| ver.2.1.0 | 2022年01月26日 | ・フェイク判定モデルの精度向上<br>・入力画像の顔面積判定パラメータの追加<br>・WEBカメラの汎用対応(推奨カメラ不要) |
| ver.2.1.1 | 2022年02月10日 | ・フェイク判定モデルの精度向上<br>・その他微修正 |
| ver.2.2.0（本版） | 2022年08月04日 | ・目位置検出ディープラーニングモデルの導入<br>・まばたき検出の精度向上<br>・フェイク判定モデルの精度向上<br>・その他微修正 |
| ver.2.2.1（本版） | 2023年11月27日 | ・OpenCV4.7.0へのリンク<br>・その他微修正 |

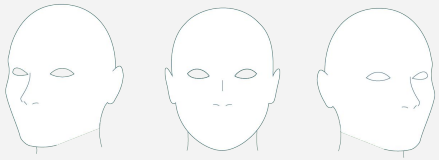
# このSDKでできること

# ■このSDKでできること

なりすまし判定技術SDKでは、ディープラーニングや機械学習を用いて、カメラ入力によるフレームをもとに、所定のアクションの有無による「アクション判定」モードと、フレームごとに生体顔かスマホ・タブレットなどに投影されたフェイク顔かどうか判定する「フェイク判定」モードを体感することができます。

## アクション判定

顔向き
アクション

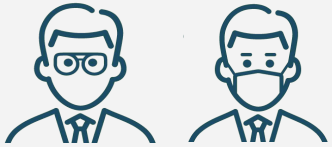


目線向き
アクション

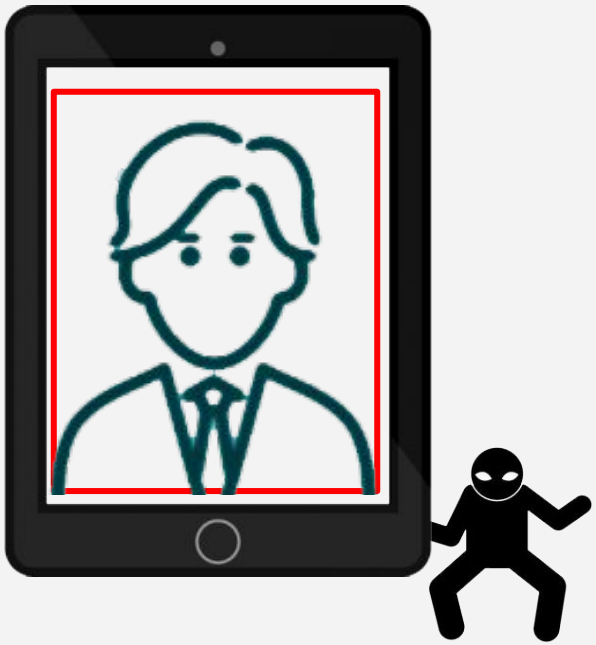
まばたき
アクション

メガネ脱着
アクション

マスク脱着
アクション



## フェイク判定



# ご利用にあたって

# ■ご利用環境

現在のバージョンでは、以下のご利用環境に対応しています。

| 項目 | 内容 | |
|---|---|---|
| **インターフェース** | C++言語 (C++11以降) | |
| **対応OS** | Windows OS / Linux OS / macOS / Raspberry Pi OS<br>iOS / Android OS | |
| **CPUアーキテクチャ** | 64bit系（x86_64 / Arm64） ※一部32bit系にも対応します | |
| **推奨メモリ** | 2GB以上を推奨 | |
| **依存ライブラリ** | OpenCV 4.7.0 以上 / OpenCV Contrib 4.7.0 以上 | |
| **実行環境 / ビルド環境** | Linux kernel 4.9以降 | gcc/g++/clangを推奨 |
| | macOS 10.11以降 | |
| | Raspberry Pi OS 以降 推奨 | |
| | Windows 10 64bit系 以降 | MicroSoft Visual C++コンパイラを推奨 |
| | iOS 15以降 | 最新のXcodeの利用を推奨 |
| | Android OS 7.0以降 | 最新のAndroid Studio / Gradle / NDK の利用を推奨 |

※その他の環境でのご利用を希望される場合は、お問い合わせください。

# ■推奨入力画像

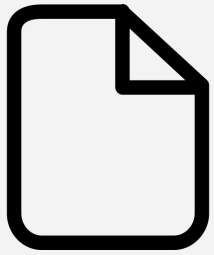
現在のバージョンでは、以下の入力画像を推奨しています。

| 項目 | 内容 |
| --- | --- |
| 画像カラー仕様 | **RGBカラー画像** (8bit3ch または 8bit4ch) ※RGB565は非対応です |
| センサ種別 | **可視光センサ画像のみ**(赤外線センサ画像には対応していません) |
| 推奨フレームサイズ | **VGA（640 x 480）**サイズ以上 |
| 入力画像解像度 | <検出される顔領域のピクセル数 ><br>**最低 width 80px 以上 推奨** |
| 顔領域の明るさ | 顔領域 平均輝度値 **50.0**以上（8bit256階調） |
| 撮影距離 | <カメラから顔までの距離 ><br>**～1m程度まで**<br>※入力画像解像度を上げることで距離を伸ばすことも可能です。顔領域最低サイズを参考にしてください。 |

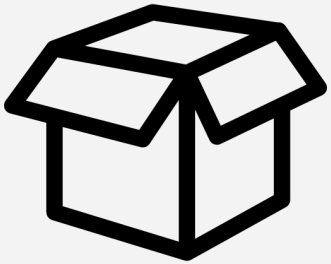
※その他の入力画像でのご利用を希望される場合は、お問い合わせください。

# SDK構成

# ■SDK構成



interfaceファイル
(hpp)



動的リンクライブラリ
(dll/so/dylibなど)

なりすまし判定ライブラリ


C++コード / ビルド手順

なりすまし判定サンプルアプリ

期限付きトークン


USBドングル

アクティベーションツール
(いずれか1つ)


ご利用マニュアル
(本書)

# ■SDK構成

本SDKは動的リンクライブラリとそのインターフェースであるヘッダーファイルで構成されています。C++インターフェースとなっていますが、
スマホOSには、C++言語向けのラッパーサンプルコードを用意しています。

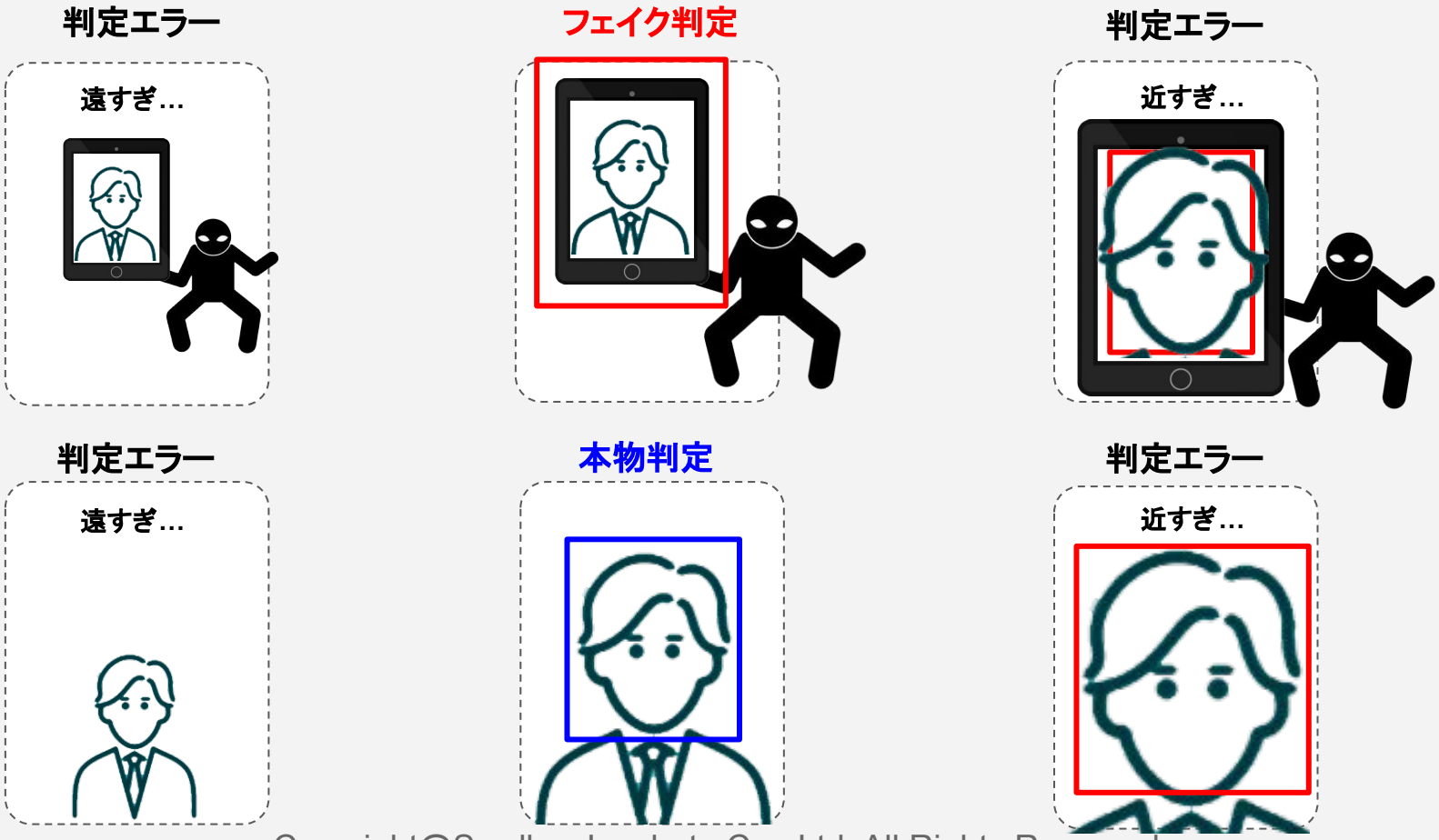
| OS | 提供物 |
|---|---|
| **macOS** | LivenessCheck.hpp （インターフェースファイル）<br>libLivenessCheck.dylib<br>サンプルアプリ（C++） |
| **Windows OS** | LivenessCheck.hpp<br>LivenessCheck.dll (実行時参照ライブラリ )<br>LivenessCheck.lib (ビルド時取込ライブラリ )<br>サンプルアプリ（C++） |
| **各種Linux OS**<br>**(Raspberry Pi OS含む)** | LivenessCheck.hpp（インターフェースファイル）<br>libLivenessCheck.so<br>サンプルアプリ（C++） |
| **iOS** | LivenessCheck.Framework（LivenessCheck.hpp含む）<br>サンプルアプリ（Objective-C ラッパーサンプルコード込み）<br>iOS向けOpenCV + OpenCV Contribビルド済ライブラリ |
| **Android OS** | LivenessCheck.hpp<br>libLivenessCheck.so（arm64-v8a / armeabi-v7a / x86 / x86_64）<br>Android向けOpenCV + OpenCV Contribビルド済ライブラリ<br>サンプルアプリ（JNI ラッパーサンプルコード込み） |

# なりすまし判定 – フェイク顔判定

# ■なりすまし判定 - フェイク顔判定

ディープラーニング版

単一フレームの画像から、顔が検出された場合、スマホ・タブレット等に映し出されたフェイク顔（偽造顔）か、生体顔かどうかをディープラーニングを用いて判定します。
この検出は、所定サイズ (初期値80px)以上の顔が検出できた場合のみ、フェイク顔かどうかを判定するため、所定サイズ以下の場合は判定されません。また、スマホ・タブレットの外枠が隠れるほどカメラに接近させてくる場合は、顔領域の検出サイズ超過を設定するか、フレームに占める顔サイズとの比率でエラーを返すことが可能です。



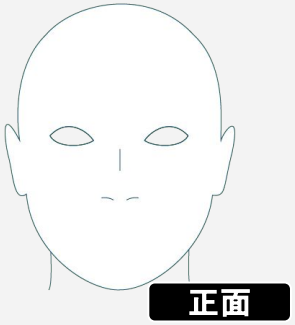
# なりすまし判定 – 顔向き判定

# ■なりすまし判定 - 顔向き判定

ディープラーニング版

単一フレームの画像から、顔が検出された場合、顔向きをディープラーニングを用いて判定します。
処理結果が格納されるBioDataクラスの、faceDirStatusH、faceDirStatusVメンバより0～2のいずれかの値が出力されることにより顔向きの判定を行うことが可能です。
動画などの時系列データを用いることにより、写真などのフェイク顔にはできない顔向きの動き・変化を検出できるかどうかによってライブネスチェックに用いることができます。

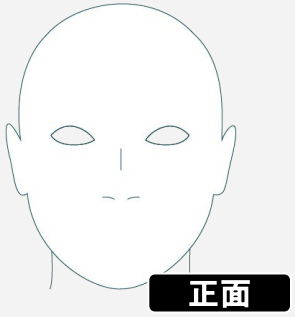

**faceDirStatusH = 1**




**faceDirStatusH = 0**


**faceDirStatusH = 2**


**faceDirStatusV = 1**




**faceDirStatusV = 0**


**faceDirStatusV = 2**

# なりすまし判定 – メガネ判定

# ■なりすまし判定 - メガネ装着判定

ディープラーニング版

単一フレームの画像から、顔が検出された場合、メガネ装着の有無をディープラーニングを用いて判定します。処理結果が格納されるBioDataクラスの、eyeGlassStatusメンバより0～2のいずれかの値が出力されることによりメガネ装着判定を行うことが可能です。
動画などの時系列データを用いて、写真などのフェイク顔にはできないメガネの着脱動作を検出できるかどうかによってライブネスチェックに用いることができます。

裸眼

eyeGlassStatus = 0

クリアレンズ
メガネ

eyeGlassStatus = 1

サングラス

eyeGlassStatus = 2

# なりすまし判定 – マスク判定

# ■なりすまし判定 - マスク装着判定

ディープラーニング版

単一フレームの画像から、顔が検出された場合、マスク装着の有無をディープラーニングを用いて判定します。処理結果が格納されるBioDataクラスの、maskStatusメンバより真偽値(true/false)が出力されることによりマスク装着判定を行うことが可能です。
動画などの時系列データを用いて、写真などのフェイク顔にはできないマスクの着脱動作を検出できるかどうかによってライブネスチェックに用いることができます。

マスク
なし

**maskStatus = false**

マスク
あり

**maskStatus = true**

# なりすまし判定 – 目向き判定

# ■なりすまし判定 - 目向き判定

ディープラーニング版

単一フレームの画像から、目が検出された場合、左右の目の向きをディープラーニングを用いて判定します。
処理結果が格納されるBioDataクラスの、eyeDirStatusLeftH/eyeDirStatusRightHメンバより0～2のいずれかの値が出力されることにより目向きの判定を行うことが可能です。
動画などの時系列データを用いて、写真などのフェイク顔にはできない目向きの動き・変化を検出できるかどうかによってライブネスチェックに用いることができます。

左向き

eyeDirStatusH = 1

正面

eyeDirStatusH = 0

右向き

eyeDirStatusH = 2

# なりすまし判定 – 目の開閉判定

# ■なりすまし判定 - 目の開閉判定

ディープラーニング版

単一フレームの画像から、目が検出された場合、目の開閉状態をディープラーニングを用いて判定します。処理結果が格納されるBioDataクラスの、eyeStatusLeft/eyeStatusRightメンバより0～2のいずれかの値が出力されることにより目の開閉状態の判定を行うことが可能です。
動画などの時系列データを用いて、写真などのフェイク顔にはできない目の開閉・まばたきを検出できるかどうかによってライブネスチェックに用いることができます。

**eyeStatus = 0**

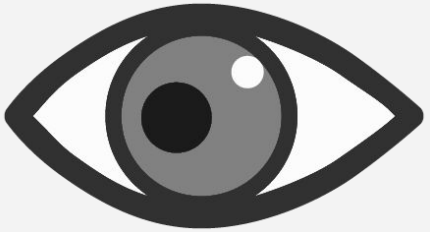
Error

**判定エラー**

**eyeStatus = 1**

**目が閉じている**

**eyeStatus = 2**

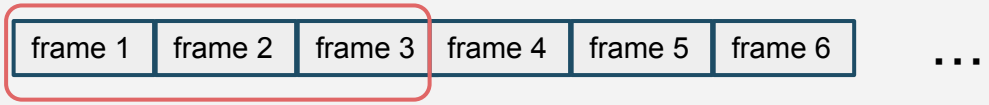


**目が開いている**

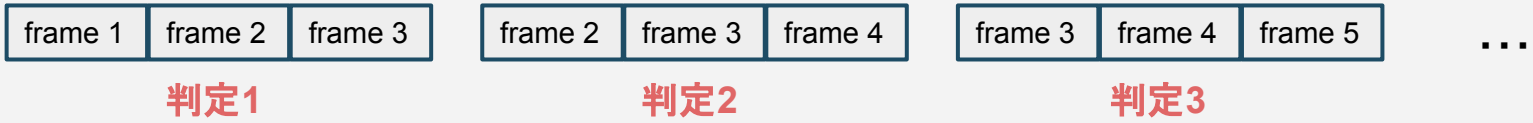
# ■なりすまし判定(動画) - まばたき判定例

なりすまし判定ライブラリは、1フレームの検出結果を返すのみとなりますので、まばたき判定は、検出結果の時間変化をもとに独自で判定していただく必要があります。以下にまばたき判定の例を掲載いたしますが、独自に判定アルゴリズムを策定いただくことも可能です。詳しくはサンプルアプリを参照してください。

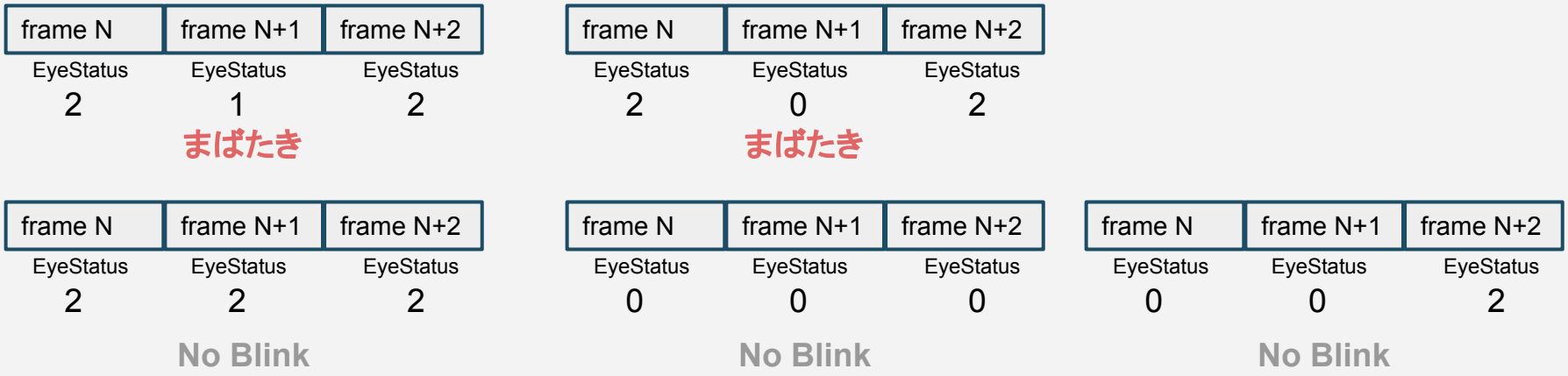
**<まばたき判定に用いるフレーム数 >** - 例）3つずつ




**<まばたき判定に使う時間フレーム >** 例）連続する3フレーム



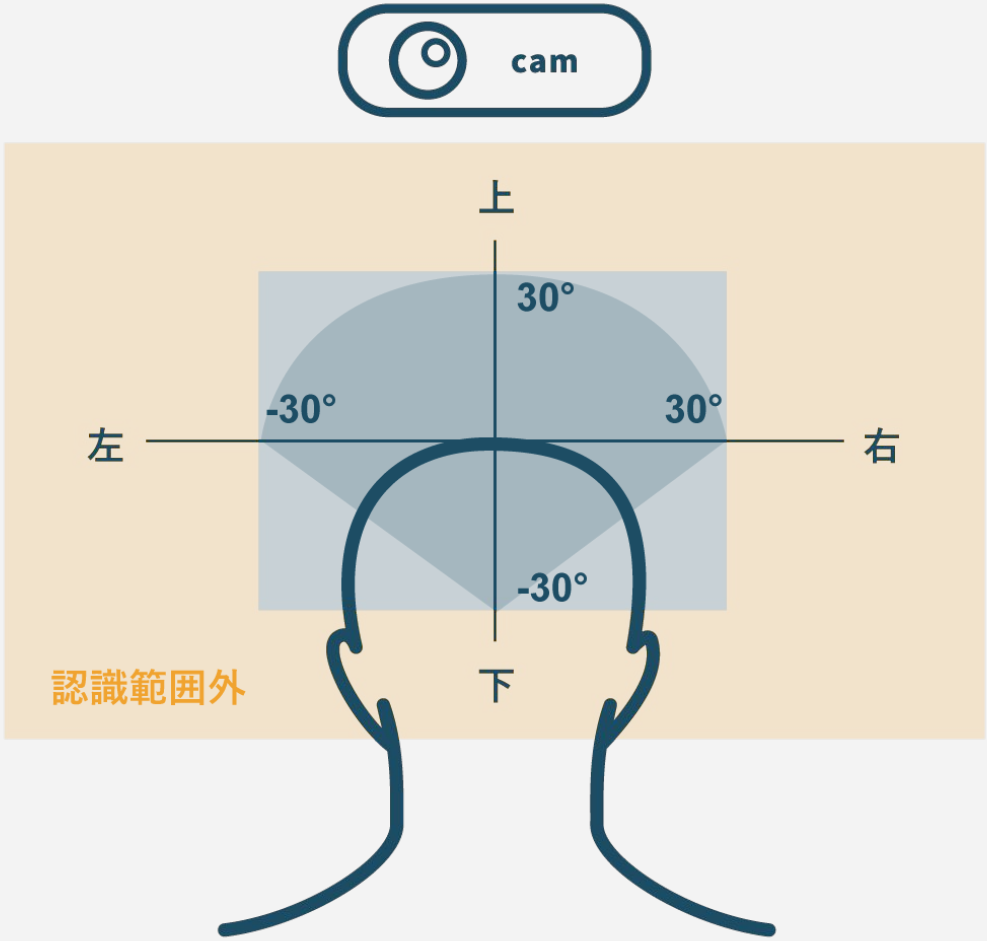

**<まばたき判定アルゴリズム >（例）**



# ライブラリ仕様

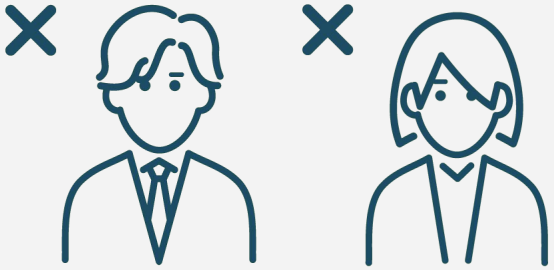
# ■ライブラリ仕様 - 検出可能画角

現在のバージョンでは、検出可能な画角は、上下左右ともに30°程度ですが
顔の形状などによるため、個人差があります。

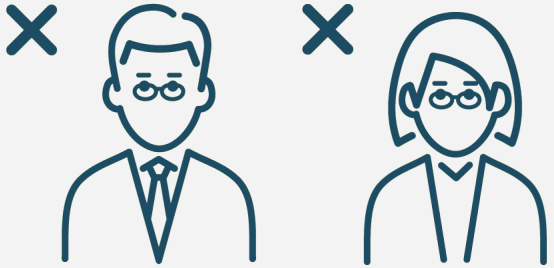


# ■ライブラリ仕様 - ✕検出不可となるケース

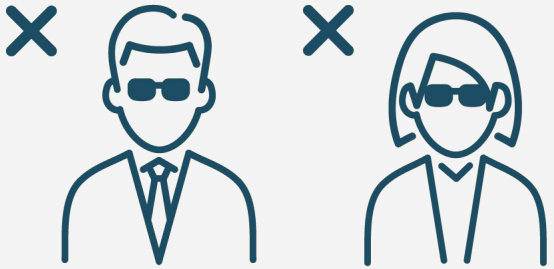
**目の位置検出においては**、以下のケースで検出エラーになりやすくなりますのでご注意ください。



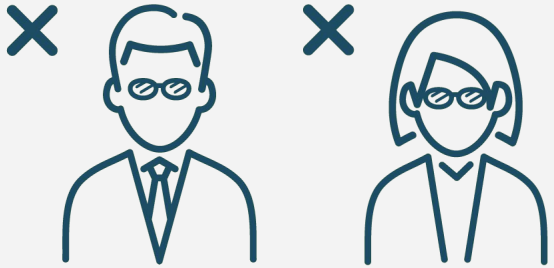
髪の毛が目にかかっている



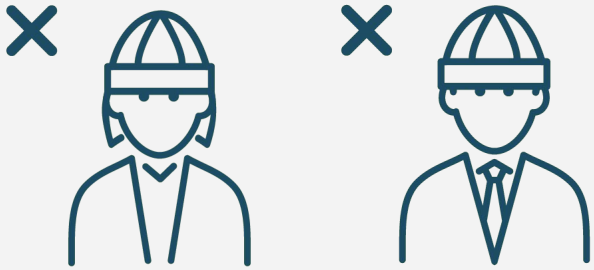
メガネフレームが目にかかっている



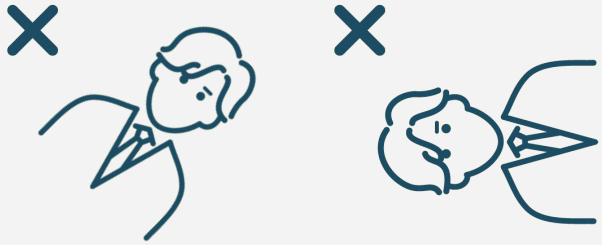
レンズが黒いサングラス



外光の映り込みが激しい



帽子を深くかぶっている



顔が傾いて撮影されている

# ■ライブラリ仕様 - △ 検出に影響を与える場合があるケース

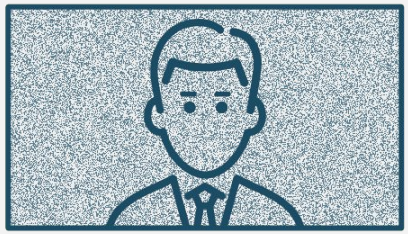
**顔の位置検出においては**、以下のケースで検出に影響を与えやすくなりますのでご注意ください。

**マスク**
**(顔位置検出エラーとなる場合あり)**

# ■ライブラリ仕様 - △ 検出に影響を与える場合がある撮影状況

各種検出・判定においては、以下の撮影の状況では検出に影響を与えやすくなりますのでご注意ください。

△



**×露光が不十分**
(環境光の差に影響を受けます)

△

**×オートフォーカス制御**
(フォーカシング中検出エラーになりやすい)

△

**×カメラの設置が**
**まっすぐではない**

△

**×画角から外れている**
(鼻より上しか写っていない)

△

**×画角から外れている**
(片目しか写っていない)

# ■ライブラリ仕様 - 機能一覧

| 項目 | 機能項目 | 要件 |
|---|---|---|
| 基本機能 | アクティベーション機能 | LivenessCheckオブジェクトのプロセス立ち上げごとに、<br>USBまたはトークンによるアクティベーションを行います。 |
| | モデルファイルの読み込み | 弊社で用意した学習モデルファイルを読み込みます。 |
| | パラメータ調整 | LivenessCheckオブジェクトのinit()に構造体を読み込ませることで所定のパラメータ調整を行うことができます。 |
| | 画像データの読み込み | LivenessCheckクラスにて、画像データを読み込ませることができます。 |
| 各種情報<br>センシング機能 | 顔位置検出機能 | 入力された画像から、顔領域候補位置を検出します。 |
| | 複数顔検出機能 | 入力された画像に複数の顔が検出された場合は以降の処理を中止します。 |
| | フェイク顔判定機能 | 検出された顔を含む入力画像をもとに、深層学習モデルを用いた推論を行い、スマホ・タブレット等に投影されたフェイク顔か、生体顔かの判定を行います。 |
| | 顔向き判定機能 | 検出された顔領域画像から、深層学習モデルを用いた推論を行い、水平方向・垂直方向それぞれの顔向きを判定します。 |
| | メガネ装着判定機能 | 検出された顔領域画像から、深層学習モデルを用いた推論を行い、メガネの装着の有無及びメガネありの場合は、クリアレンズメガネかサングラスかの区別も可能です。 |
| | マスク装着判定機能 | 検出された顔領域画像から、深層学習モデルを用いた推論を行い、マスクの装着の有無を判定します。 |
| | 目の位置検出機能 | 検出された顔領域画像から、深層学習モデルを用いた推論を行い、両目の検出を行います。 |
| | 目の開閉判定機能 | 検出された目画像から、深層学習モデルを用いた推論を行い、目の開閉状態を判定します。 |
| | 目の向き判定機能 | 検出された目画像から、深層学習モデルを用いた推論を行い、水平方向の目の向きを判定します。 |

# ■ライブラリ仕様 - 機能別処理速度目安

現在のバージョンでは、ライブラリ各機能ごとの処理時間は以下の通りとなります。ライブラリは NonGPU環境でも動作するため、CPUでの動作での参考値となります。なおライブラリに画像を読み込ませると以下の順にすべての機能を実行しますが、機能毎に処理を行うか行わないかの On/Offを行うことも可能です。また、顔位置検出においては、本ライブラリ以外のツールで検出した結果を指定して処理を進めることも可能です。

| 機能 | On/Off | アルゴリズム | 出力 | 処理時間※ |
|---|---|---|---|---|
| 顔位置検出 | 必須<br>(位置指定可) | CNN系物体検出 | cv::Rect型（x, y, width, height） | 約15ms程度 |
| フェイク判定 | On/Off可 | CNN系二値分類 | 真偽値(true = フェイク / false = 生体顔) | 約20ms程度 |
| 顔向き判定(水平) | On/Off可 | CNN系多値分類 | 0 = 正面 / 1 = 左向き / 2 = 右向き | 約20ms程度 |
| 顔向き判定(垂直) | | | 0 = 正面 / 1 = 上向き / 2 = 下向き | |
| メガネ判定 | On/Off可 | | 0 = なし / 1 = クリアレンズ / 2 = サングラス | 約7ms程度 |
| マスク判定 | On/Off可 | CNN系二値分類 | 真偽値(true = マスクあり / false = マスクなし) | 約7ms程度 |
| 目の位置検出 | On/Off可 | CNN系物体検出 | cv::Rect型（x, y, width, height） | 約15ms程度 |
| 目の開閉判定 | On/Off可 | CNN系多値分類 | 0 = エラー / 1 = 閉じ目 / 2 = 開き目 | 約5ms程度 |
| 目向き判定 | On/Off可 | | 0 = 正面 / 1 = 左向き / 2 = 右向き | 約5ms程度 |

※処理時間は、macBookPro 2017年モデル(Intel Core i7 第8世代 CPU)をもとに算出しています

# ■ライブラリ仕様 - 機能別処理順序

なりすまし判定ライブラリを実行時、1フレームに対する各機能別処理順序は以下の通りとなります。
機能毎に処理を行うか行わないかの On/Offを行うことも可能ですが、順序を入れ替えることはできません。また各機能において検出エラーが出た場合は、それ以降の処理を行いません。

## ●処理順序

| 順序 | 機能 | On/Off |
|---|---|---|
| 1 | 顔位置検出 | 必須<br>(位置指定可) |
| 2 | フェイク判定 | On/Off可 |
| 3 | 顔向き判定(水平) | On/Off可 |
| 4 | メガネ判定 | On/Off可 |
| 5 | マスク判定 | On/Off可 |
| 6 | 目の位置検出 | On/Off可 |
| 7 | 目の開閉判定 | On/Off可 |
| 8 | 目向き判定 | On/Off可 |

## ●例1 - フェイク判定のみを行う場合

| 順序 | 機能 |
|---|---|
| 1 | 顔位置検出 |
| 2 | フェイク判定 |

フェイク判定のみを行う場合の処理は、顔位置検出とフェイク判定のみです。顔位置検出は必須ですが、別エンジンで顔位置を特定している場合、顔位置を指定することにより顔位置検出をスキップすることも可能です。

## ●例2 - 顔向き判定と目向き判定のみを行う場合

| 順序 | 機能 |
|---|---|
| 1 | 顔位置検出 |
| 2 | 顔向き判定 |
| 3 | 目の位置検出 |
| 4 | 目の開閉判定 |
| 5 | 目向き判定 |

顔向き判定を行う場合の処理は、顔位置検出と顔向き判定のみです。顔位置検出は必須ですが、別エンジンで顔位置を特定している場合、顔位置を指定することにより顔位置検出をスキップすることも可能です。

目向き判定を行う場合の処理は、目向き判定を行うために必要な処理として、目の位置検出および目の開閉判定処理が必須となります。

# ■ライブラリ仕様 - クラス構成

| クラス名 | | データ型 | メンバ名 | 説明 |
|---|---|---|---|---|
| struct **ExternalFilePath** | この構造体について | | | なりすまし判定を行うためのモデルファイルを読み込ませるために、LivenessCheckクラスのオブジェクト化直後に、本構造体をinit()関数の引数として与えます。 |
| | 顔位置検出用 | std::string | faceDetectModel | 顔位置検出に用いるモデルファイルを読み込む変数です。 |
| | | std::string | faceDetectParam | |
| | フェイク判定用 | std::string | fakeModelH | フェイク判定に用いるモデルファイルを読み込む変数です。 |
| | | std::string | fakeModelV | |
| | 顔向き判定用 | std::string | faceDirHModel | 顔向き判定に用いるモデルファイルを読み込む変数です。 |
| | | std::string | faceDirVModel | |
| | メガネ判定用 | std::string | glassesModel | メガネ判定に用いるモデルファイルを読み込む変数です。 |
| | マスク判定用 | std::string | maskModel | マスク判定に用いるモデルファイルを読み込む変数です。 |
| | 目位置検出用 | std::string | eyeDetectModel | 目位置検出に用いるモデルファイルを読み込む変数です。 |
| | 目の開閉判定用 | std::string | eyeStatusModel | 目の開閉判定に用いるモデルファイルを読み込む変数です。 |
| | 目向き判定用 | std::string | eyeDirModel | 目の向き判定に用いるモデルファイルを読み込む変数です。 |

# ■ライブラリ仕様 - クラス構成

| クラス名 | 機能 | データ型 | メンバ名 | 説明 |
|---|---|---|---|---|
| struct **Params** | この構造体について | | | 各検出パラメータの値を読み込ませるために、LivenessCheckクラスのオブジェクト化後、本構造体をinit()関数の第二引数として与えます。 |
| | 顔位置検出用 | int | minFaceWidth | 顔位置検出を行う最小顔横幅サイズ(px)を指定します。 |
| | | int | maxFaceWidth | 顔位置検出を行う最大顔横幅サイズ(px)を指定します。 |
| | | float | faceAreaMinRatio | 顔の接近度を判定するため、「顔位置検出サイズ面積÷入力画像サイズ面積」により算出される顔サイズの許容する下限・上限割合を指定します0.05～1.00の値の範囲で指定します。 |
| | | float | faceAreaMaxRatio | |
| | | bool | edgePosErrMode | 検出された顔位置が上下左右端に位置する場合に、以降の処理を停止するかどうかを真偽値で指定します。 |
| | フェイク判定用 | bool | isFakeMode | フェイク判定を行うかどうかを真偽値で指定します。 |
| | | float | fakeJudgeTh | Biodata::isFakeLikelihoodの値から、フェイク顔と判定するための閾値となります。0.0～1.0の値の範囲で指定します。 |
| | 顔向き判定用 | bool | isFaceDirMode | 顔向き判定を行うかどうかを真偽値で指定します。 |
| | メガネ判定用 | bool | isGlassesMode | メガネ判定を行うかどうかを真偽値で指定します。 |
| | マスク判定用 | bool | isMaskMode | マスク判定を行うかどうかを真偽値で指定します。 |
| | | float | maskJudgeTh | Biodata::isMaskLikelihoodの値から、マスク装着状態と判定するための閾値となります。0.0～1.0の値の範囲で指定します。 |
| | 目の位置検出用 | bool | isEyeMode | 目の位置検出・目の開閉判定を行うかどうかを真偽値で指定します。 |
| | 目の向き判定用 | bool | isEyeDirMode | 目の向き判定を行うかどうかを真偽値で指定します。 |

# ■ライブラリ仕様 - クラス構成

## ●BioDataクラス – 基本機能メンバ

| クラス名 | データ型 | メンバ名 | 説明 |
|---|---|---|---|
| class **BioData** | このクラスについて | | ライブネス情報センシング結果格納クラスです。処理途中での検出エラー時でも、検出できた部分までのメンバは取得可能です。 |
| | cv::Mat | originalImg | 入力した元画像を取得可能です。 |
| | cv::Mat | checkImg | 入力した元画像に、検出結果を矩形や丸点などで描画プロットした画像です。 |
| | cv::Mat | faceRectImg | 入力した元画像から、検出された顔画像のみを切り出した画像を取得可能です。 |
| | cv::Rect | faceRectArea | 検出された顔矩形領域情報を取得可能です。(x座標,y座標,width,height) |
| | float | faceBrightness | 検出された顔矩形領域の平均輝度値を8bit256階調で取得可能です。 |
| | bool | isValidFace | 検出された顔において、検出位置、顔の明るさ、顔のサイズ、フレーム面積と顔面積の割合が適正かどうかを判定します。 |
| | std::vector<cv::Rect> | vFaceRect | 検出したすべての複数の顔領域情報を取得可能です。(x座標,y座標,width,height) |
| | std::string | msg | 処理結果メッセージを取得することができます。 |
| | void | clear() | BioDataクラスの全メンバ変数の値を初期化します。 |

# ■ライブラリ仕様 - クラス構成

## ●BioDataクラス – フェイク判定機能メンバ

| クラス名 | データ型 | メンバ名 | 説明 |
|---|---|---|---|
| class **BioData** | bool | isFakeFace | 検出された顔が、スマホやタブレットなどの端末に映されたフェイク顔か生体顔かどうかの真偽値です。スマホやタブレットに映された顔であると推定された場合はtrueが返り、以降の検出処理を中止します。 |
| | float | isFakeLikelihood | 深層学習モデルにより推論されたフェイク顔であることの確からしさを表す値です。<br>0.0～1.0の値の範囲で出力され、1.0に近づくほどフェイク顔であると推定され、0に近づくほど生体顔であると推定されます。 |

## ●BioDataクラス – 顔向き判定機能メンバ

| クラス名 | データ型 | メンバ名 | 説明 |
|---|---|---|---|
| class **BioData** | short | faceDirStatusH | 検出された顔をもとにした、深層学習モデルによる水平方向の顔向きの推論結果です。<br>初期値は-1となっており、判定が行われると出力は0～2の整数値となります。<br>0は正面、1は左向き、2は右向きであると推定されます。 |
| | short | faceDirStatusV | 検出された顔をもとにした、深層学習モデルによる垂直方向の顔向きの推論結果です。<br>初期値は-1となっており、判定が行われると出力は0～2の整数値となります。<br>0は正面、1は上向き、2は下向きであると推定されます。 |

## ●BioDataクラス – メガネ判定機能メンバ

| クラス名 | データ型 | メンバ名 | 説明 |
|---|---|---|---|
| class **BioData** | short | eyeGlassStatus | 検出された顔をもとにした、深層学習モデルによるメガネ装着の有無の推論結果です。<br>初期値は-1となっており、判定が行われると出力は0～2の整数値となります。<br>0は裸眼、1はクリアレンズメガネ、2はサングラスであると推定されます。 |

# ■ライブラリ仕様 - クラス構成

## ●BioDataクラス – マスク判定機能メンバ

| クラス名 | データ型 | メンバ名 | 説明 |
|---|---|---|---|
| class **BioData** | bool | maskStatus | 検出された顔をもとにした、深層学習モデルによるマスク装着の有無の推論結果です。<br>マスクを装着していると推定された場合はtrueが返ります。 |
| | float | isMaskLikelihood | 深層学習モデルにより推論されたマスク装着状態であることの確からしさを表す値です。0.0～1.0の値の範囲で出力され、1.0に近づくほどマスク装着状態であると推定され、0に近づくほどマスクなしであると推定されます。 |

## ●BioDataクラス – 目の位置検出機能メンバ

| クラス名 | データ型 | メンバ名 | 説明 |
|---|---|---|---|
| class **BioData** | cv::Mat | eyeRectImgLeft | 入力した元画像から、検出された左目画像を切り出した画像を取得可能です。 |
| | cv::Mat | eyeRectImgRight | 入力した元画像から、検出された右目画像を切り出した画像を取得可能です。 |
| | cv::Rect | eyeRectAreaLeft | 検出された左目矩形領域情報を取得可能です。(x座標,y座標,width,height) |
| | cv::Rect | eyeRectAreaRight | 検出された右目矩形領域情報を取得可能です。(x座標,y座標,width,height) |
| | float | eyeRectAvgBrightLeft | 検出された左目矩形領域の平均輝度値を8bit256階調で取得可能です。 |
| | float | eyeRectAvgBrightRight | 検出された右目矩形領域の平均輝度値を8bit256階調で取得可能です。 |
| | short | eyeStatusLeft | 検出された顔をもとにした、深層学習モデルによる目の開閉状態の推論結果です。<br>初期値は-1となっており、判定が行われると出力は0～2の整数値となります。<br>0は判定エラー、1は閉じ目、2は開き目であると推定されます。 |
| | short | eyeStatusRight | |

# ■ライブラリ仕様 - クラス構成

## ●BioDataクラス – 目向き判定機能メンバ

| クラス名 | データ型 | メンバ名 | 説明 |
|---|---|---|---|
| class **BioData** | short | eyeDirStatusLeftH | 検出された顔をもとにした、深層学習モデルによる左右の水平方向の目向きの推論結果です。初期値は-1となっており、判定が行われると出力は0～2の整数値となります。<br>0は正面、1は左向き、2は右向きであると推定されます。 |
| | short | eyeDirStatusRightH | |

# ■ライブラリ仕様 - クラス構成

| クラス名 | データ型 | メンバ名 | 説明 |
|---|---|---|---|
| class **LivenessCheck** | このクラスについて | | なりすまし判定に必要な顔および目の情報を検出するための実行クラスとなります。 |
| | bool | init | 初期化を行うメンバ関数です。第一引数にExternalFilePath構造体、第二引数にParams構造体をとります。 |
| | bool | registerBioData | メモリ確保したBioDataクラスを本クラスに登録するための関数です。 |
| | bool | process | cv::Matを引数とし、顔位置検出、目位置検出、各種判定処理を行うクラスです。<br>検出結果はすべてBioDataクラスに格納されます。 |

# ■ライブラリ仕様 - クラス構成

## ◆ USBアクティベーション方式

| クラス名 | データ型 | メンバ名 | 説明 |
|---|---|---|---|
| class **ActivationChallenge** | このクラスについて | | LivenessCheckクラスを実行する前にアクティベーションを行うための<br>クラスとなります。 |
| | std::string | config() | アクティベーション情報および本ライブラリの情報を取得可能です。 |
| | bool | checkUSB() | USBドングルがマシンに接続されているかをチェックするための関数です。 |
| | bool | validUSB() | USBドングルとライブラリに埋め込まれたIDとが一致するかをチェックするための関数です。本関数からtrueが返ることによりLivenessCheckクラスの初期化処理であるinit()関数が成功するようになります。 |

## ◆ トークンアクティベーション方式

| クラス名 | データ型 | メンバ名 | 説明 |
|---|---|---|---|
| class **ActivationChallenge** | std::string | config() | アクティベーション情報および本ライブラリの情報を取得可能です。 |
| | bool | checkToken() | 本関数の引数にとるトークン文字列が、ライブラリに埋め込まれた対応トークンとマッチするかどうかのみを判定します。<br>**本関数からtrueが返るだけではアクティベーションは完了しません。** |
| | bool | validToken() | 本関数の引数にとるトークン文字列が、ライブラリに埋め込まれた対応するトークン情報を読み取り、有効期限内かどうかをチェックするための関数です。有効期限内であれば本関数からtrueが返り、LivenessCheckクラスの初期化処理であるinit()関数が成功するようになります。 |
| | long | getExpDate() | 本関数の引数にとるトークン文字列が、ライブラリに埋め込まれた対応するトークン情報を読み取り、有効期限をlong型の数値で返します。例えば返り値が「20211231」であれば、2021年12月31日まで有効なトークンです。 |

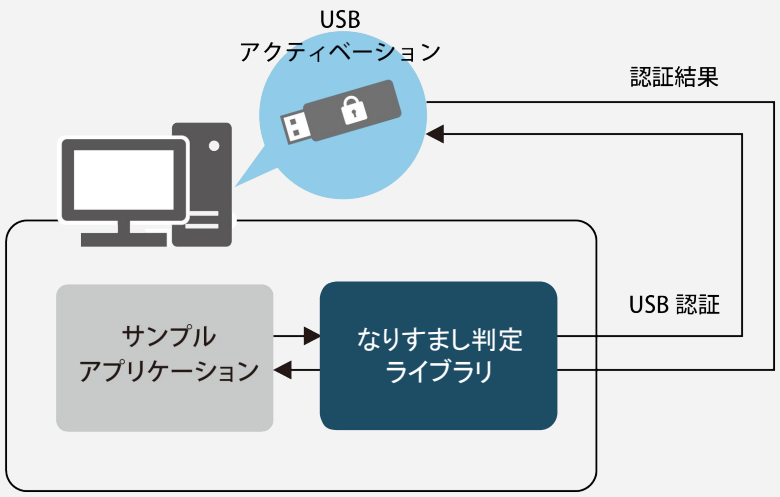
# ■ライブラリ仕様 - ファクトリ関数

| 返り値 | 関数名 | 説明 |
|---|---|---|
| LivenessCheck* | newLivenessCheck() | LivenessCheckクラスをオブジェクト化させるための関数です。<br>内部で派生クラスの生成などが行われるためLivenessCheckクラスのオブジェクト化は、必ずこの関数を用いて行ってください。 |
| void | releaseLivenessCheck() | LivenessCheckオブジェクトをメモリ開放させるための関数です。<br>LivenessCheckオブジェクトが不使用になった場合は、必ずこの関数を実行してください。 |
| ActivationChallenge* | newActivationChallenge() | ActivationChallengeクラスをオブジェクト化させるための関数です。<br>内部で派生クラスの生成などが行われるため、ActivationChallengeクラスのオブジェクト化は、必ずこの関数を用いて行ってください。 |
| void | releaseActivationChallenge() | ActivationChallengeオブジェクトをメモリ開放させるための関数です。<br>ActivationChallengeオブジェクトが不使用になった場合は、必ずこの関数を実行してください。 |

# ■ライブラリ仕様 - アクティベーション

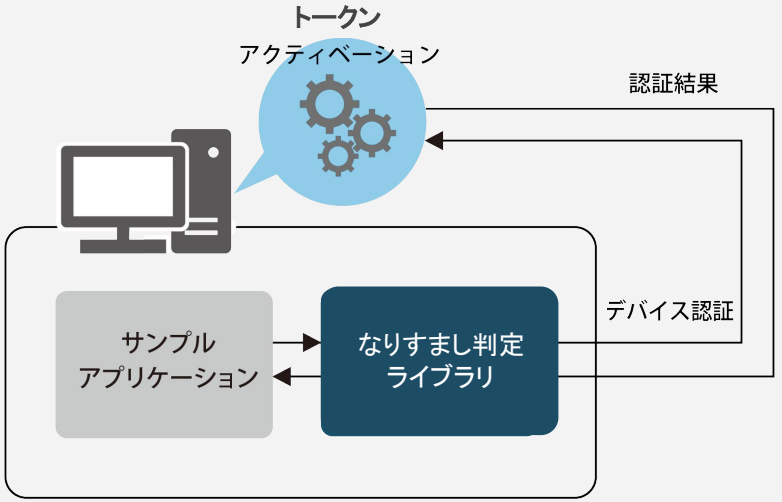
開発版ライセンスでは、本ライブラリをご利用いただくにあたって、アクティベーションが必要となります。アクティベーションには、USB方式と有効期限を利用したトークン方式の2タイプがあります。

## ◆ USBアクティベーション




**有償（USBドングル5本まで無料）**
※PC版は原則こちらでお願いしております

## ◆ トークンアクティベーション




**無償**
※主にスマホ/タブレット端末での利用企業様向けとなります

# ■ライブラリ仕様 - アクティベーション情報

ActivationChallengeオブジェクトのconfig()を使用することで、返り値に、
アクティベーション情報やライブラリ基本情報を取得することができます。
弊社にお問い合わせいただく場合に取得をお願いする場合があります。

### ◆ USBアクティベーション版 config() 実行結果例

```
[toshikazuohno@sample]$ ./check_SDK
Activation   : USB
Client ID    : 2349799210
Client Name : Swallow Incubate
SDK Version : LivenessCheckSDK - ver.1.0.1
Start-Date   : 2020-05-11
[toshikazuohno@sample]$
```

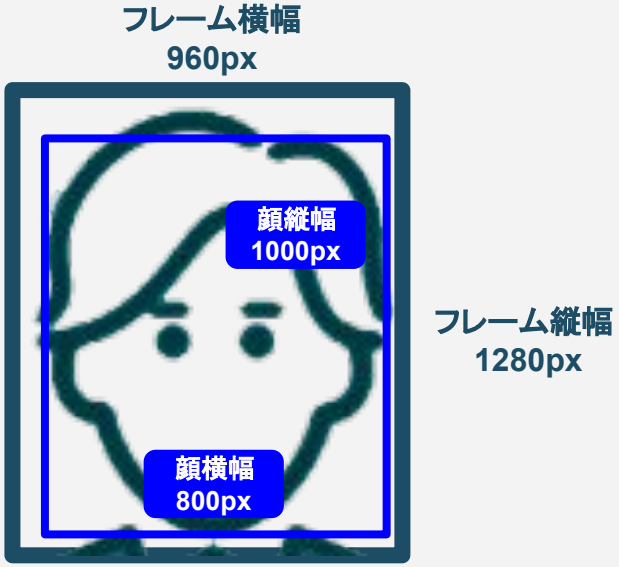
### ◆ トークンアクティベーション版 config() 実行結果例

```
[toshikazuohno@sample]$ ./check_SDK
Activation   : Token
Client ID    : 2349799210
Client Name : Swallow Incubate
SDK Version : LivenessCheckSDK - ver.1.0.1
Start-Date   : 2020-05-11
[toshikazuohno@sample]$
```

# 顔面積 許容率について

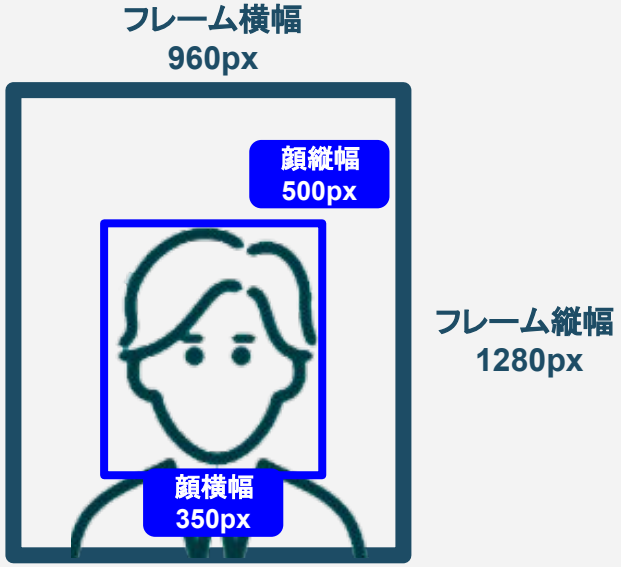
# ■顔面積 許容率について

検出された顔の面積が、画像フレームの面積のどの程度の割合を占めるかによって、簡易な接近度の判定を行うことが可能です。
Params構造体**faceAreaMinRatio/faceAreaMaxRatio**メンバにて顔面積の最小割合及び最大割合を指定することで、被写体の顔が所定以下または所定以上に接近している場合に、エラーを返すことが可能です。0.05～1.00の値の範囲で設定可能となっており、1.00にするとすべての顔を受け入れることが可能です。なお、上限下限値の範囲は入力画像タイプによって異なり、横向き画像の場合は 0.05～1.00、縦向き画像の場合は0.12～0.50となります。
主にフェイク判定において、顔の過度な接近は、フェイク顔と生体顔との判定に影響が出る場合があるため、特段の理由がない場合は本モードを設定して、以降の処理を行わないことを推奨します。




顔面積割合
0.651

<計算式>
(800px × 1000px) ÷ (960px × 1280px)
検出顔面積 ÷ フレーム面積



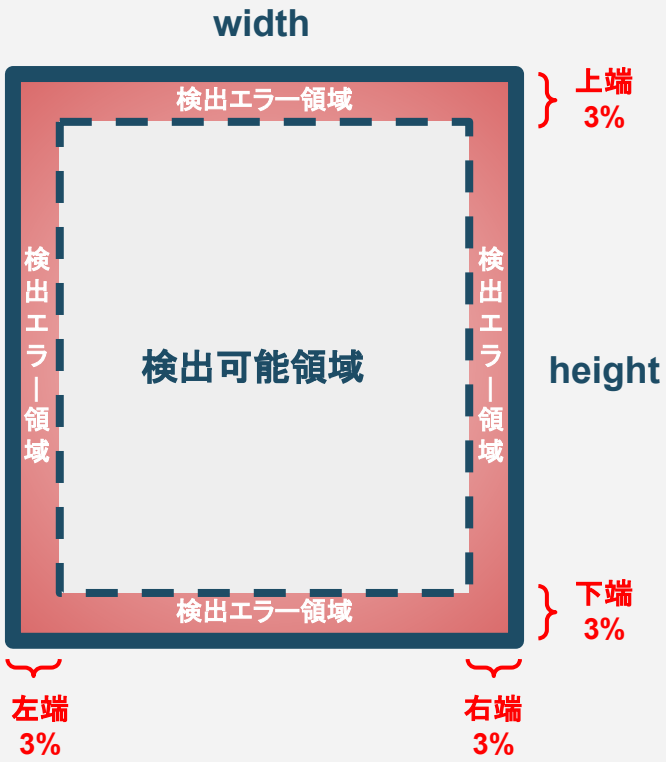

顔面積割合
0.142

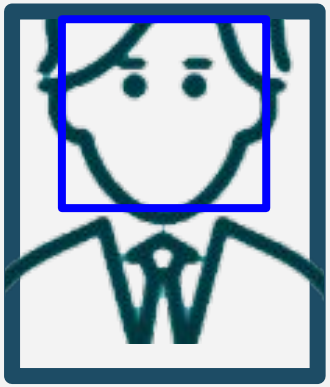
<計算式>
(350px × 500px) ÷ (960px × 1280px)
検出顔面積 ÷ フレーム面積

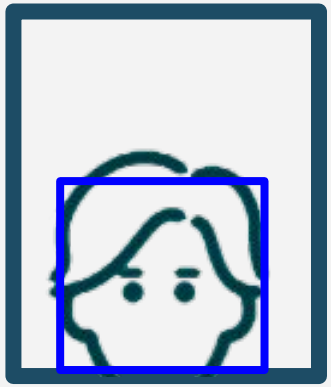
# 上下左右端位置の顔について

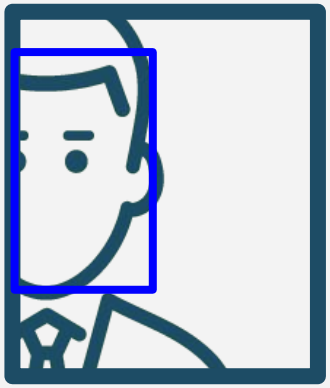
# ■上下左右端位置の顔について

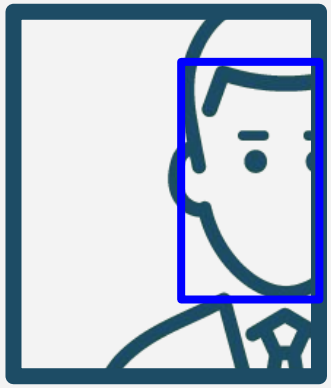
検出された顔の位置が、画像フレームの上下左右端に位置する場合に、以降の検出を行うかどうかを、Params構造体**edgePosErrMode**メンバにて指定可能です。なお、本モードを設定すると、画像フレームの**横幅px/縦幅pxの各3%以内の位置を上下左右端の領域**として、顔検出位置の上下左右端いずれかが位置する場合にはエラーとなります。
上下左右端に顔がある場合は、各種判定に影響が出る場合があるため、特段の理由がない場合は本モードを設定して、以降の処理を行わないことを推奨します。





上端エラーの例


下端エラーの例


左端エラーの例


右端エラーの例

# 出力データ詳細

# ■message一覧①

BioData型のメンバ変数であるmsgにて取得できる主なメッセージは以下のいずれかとなります。その他のエラーが発生した場合はお問合せください。

| メッセージ内容 | 内容 |
|---|---|
| Process Successful | 全ての処理が完了しました |
| Error: init() first | process()実行前に、init()を実行してください |
| Error: registerBioData() first | process()実行前に、registerBioData()を実行してください |
| Error: Empty input image | 入力画像が存在しません |
| Error: Input RGB image(24bit) | RGB画像(24bit)を入力してください |
| Error: Failed to detect face(1001) | 顔位置検出に失敗しました（顔が存在しない） |
| Error: Failed to detect face(1002) | 顔位置検出に失敗しました（横幅サイズ不足） |
| Error: Failed to detect face(1003) | 顔位置検出に失敗しました（横幅サイズ超過） |

# ■message一覧②

BioData型のメンバ変数であるmsgにて取得できる主なメッセージは以下のいずれかとなります。その他のエラーが発生した場合はお問合せください。

| メッセージ内容 | 内容 |
|---|---|
| Error: Face position error (1004) | 顔がカメラから設定値より離れすぎています（接近度判定エラー） |
| Error: Face position error (1005) | 顔がカメラに設定値より接近しすぎています（接近度判定エラー） |
| Error: Face position error (1006) | 顔がカメラから規定値より離れすぎています（接近度判定エラー） |
| Error: Face position error (1007) | 顔がカメラに規定値より接近しすぎています（接近度判定エラー） |
| Error: Face position error (1008) | 顔がカメラから規定値より離れすぎています（接近度判定エラー） |
| Error: Face position error(1011) | 顔が上下左右端位置にあります（左端） |
| Error: Face position error(1012) | 顔が上下左右端位置にあります（上端） |
| Error: Face position error(1013) | 顔が上下左右端位置にあります（右端） |
| Error: Face position error(1014) | 顔が上下左右端位置にあります（下端） |
| Error: Lack of face area brightness(1021) | 顔領域の明るさ不足です（顔領域平均輝度値50以上 – 8bit256階調） |

# ■message一覧②

BioData型のメンバ変数であるmsgにて取得できる主なメッセージは以下のいずれかとなります。その他のエラーが発生した場合はお問合せください。

| メッセージ内容 | 内容 |
|---|---|
| Error: Multiple faces detected | 【警告】複数の顔が検出されました |
| Error: Detected fake face | 【警告】デバイス内の偽造顔が検出されました |
| Error: Failed to detect eyes( wearing sunglasses) | 目の位置検出に失敗しました（サングラス装着のため） |
| Error: Failed to detect eyes | 目の位置検出に失敗しました |
| Error: Failed to detect open eye | 閉じ目のみを検出しました（開き目なし） |
| Error: Face direction is Lower | 顔向きが下向きのため目向き判定・目の開閉判定ができません |

# フェイク顔判定 – 精度エラー対策

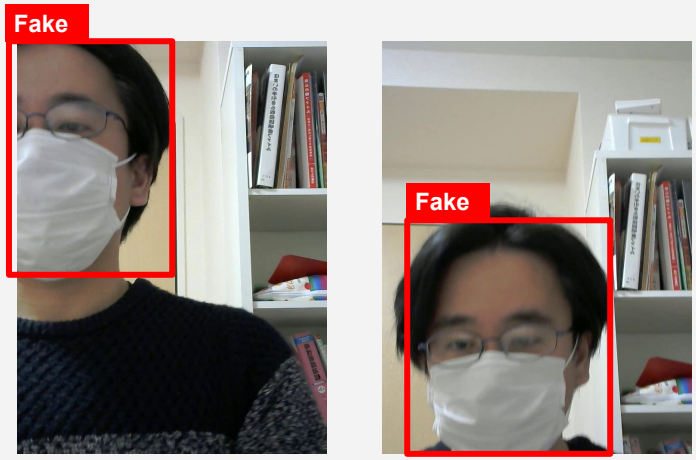
# ■フェイク判定 - 精度エラー対策

学習モデルの関係上、精度エラーが出やすい画像タイプがありますので、以下の通り混同行列として列挙します。精度エラーには、**偽陽性**と**偽陰性**の2タイプがあります。
具体的な対策については、次のページ以降でそれぞれ説明していきます。

| 種別 | | 定義 | エラーとなりやすいケース |
|---|---|---|---|
| OK | 真陽性 | フレーム内の顔がフェイク顔である場合にフェイク判定がtrueとなること | |
| | 真陰性 | フレーム内の顔がリアル顔である場合にフェイク判定がfalseとなること | |
| NG | 偽陽性 | フレーム内の顔がリアル顔であるにも関わらず、フェイク判定がtrueとなること | ●上下左右端に顔が位置している場合<br>●フレーム全体または顔周辺が暗い画像の場合<br>●背景にベゼル（枠）のようなものがある場合 |
| | 偽陰性 | フレーム内の顔がフェイク顔であるにも関わらず、フェイク判定がfalseとなること | ●上下左右端に顔が位置している場合<br>●フレーム全体または顔周辺が暗い画像の場合<br>●ベゼル(枠)が見えなくなるほど近接した顔の場合 |

# ■フェイク判定 - 精度エラー対策①上下左右端

上下左右端に顔が位置するフレームの場合、顔の形状が不自然になるため、
偽陽性や偽陰性が出る頻度が上がる場合があります。
この場合、ユーザ様には極力フレーム中央に顔を移動していただくか、
上下左右端エラーをOnにしていただくことで、エラーを回避することができます。





**対策案①:**
**アプリ上でガイドを提示する**

顔が中央に位置したことが確認できてからフェイク判定を行うなど

**対策案②:**
**Params::edgePosErrModeをtrueにする**

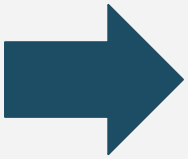
フェイク判定を行わず、BioData::isValidFaceがfalseとなります。

# ■フェイク判定 - 精度エラー対策②暗い画像

フレーム全体または顔周辺が暗い画像の場合、偽陽性や偽陰性が出る頻度が上がる場合があります。
この場合、入力画像に対して 画像処理によるコントラスト補正 を行っていただくことでエラーを回避することができます。（光補正機能付きのカメラを選択していただいても構いません）

**対策案：**
**コントラスト補正**

画像処理によるコントラスト補正を行う。

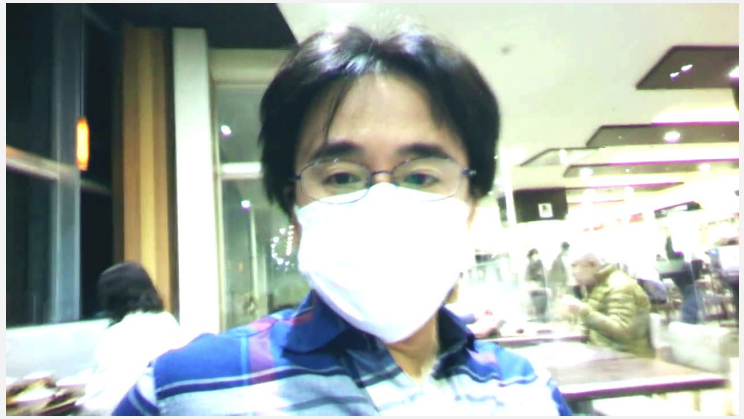


OpenCVのcv::convertScaleAbs()を使用。
（alpha=1.5 / beta=20の場合）

# ■フェイク判定 - 精度エラー対策③背景に枠がある

顔を含むフレームの背景にベゼル（枠）に類似したものが写り込んでいる画像の場合、
偽陽性が出る頻度が上がる場合があります。
この場合、撮影する場所（背景）を変更していただくことを推奨します。

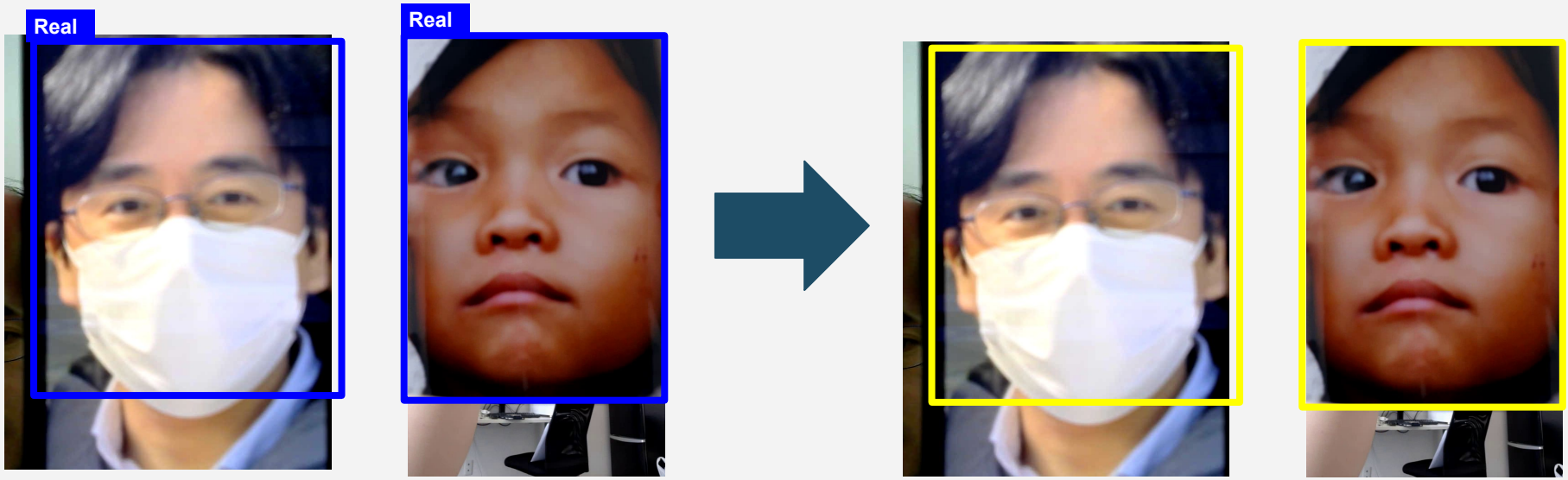
**対策案:**
**背景を変更する**



# ■フェイク判定 - 精度エラー対策④接近顔

スマホやタブレットなどのベゼル（枠）が隠れるほどに近接した顔が投影された場合は、
フェイク判定がfalseとなる頻度が上がる場合があります。
この場合、顔面積許容率の上限を 50%未満にしていただくことで、偽陰性を回避することができます。
または、アクション判定を組み合わせることで、セキュリティを強化することも可能です。

**対策案:**
**Params::faceAreaMaxRatioを0.50未満にする**



# ■フェイク判定 - 精度エラー対策⑤テロップ

実際の利用シーンではないと思われますが、
検出された顔の付近に、テロップや字幕、文字ガイドなどの編集された要素が映り込んでいる場合は、
フェイク判定がtrueとなる頻度が上がる場合があります。
そのため、そのような処理が施された YouTube動画などを用いて検証されることは推奨できません。


例）顔の付近にテロップが入っている参考画像①


例）顔の付近にテロップが入っている参考画像②

# サービス構築例

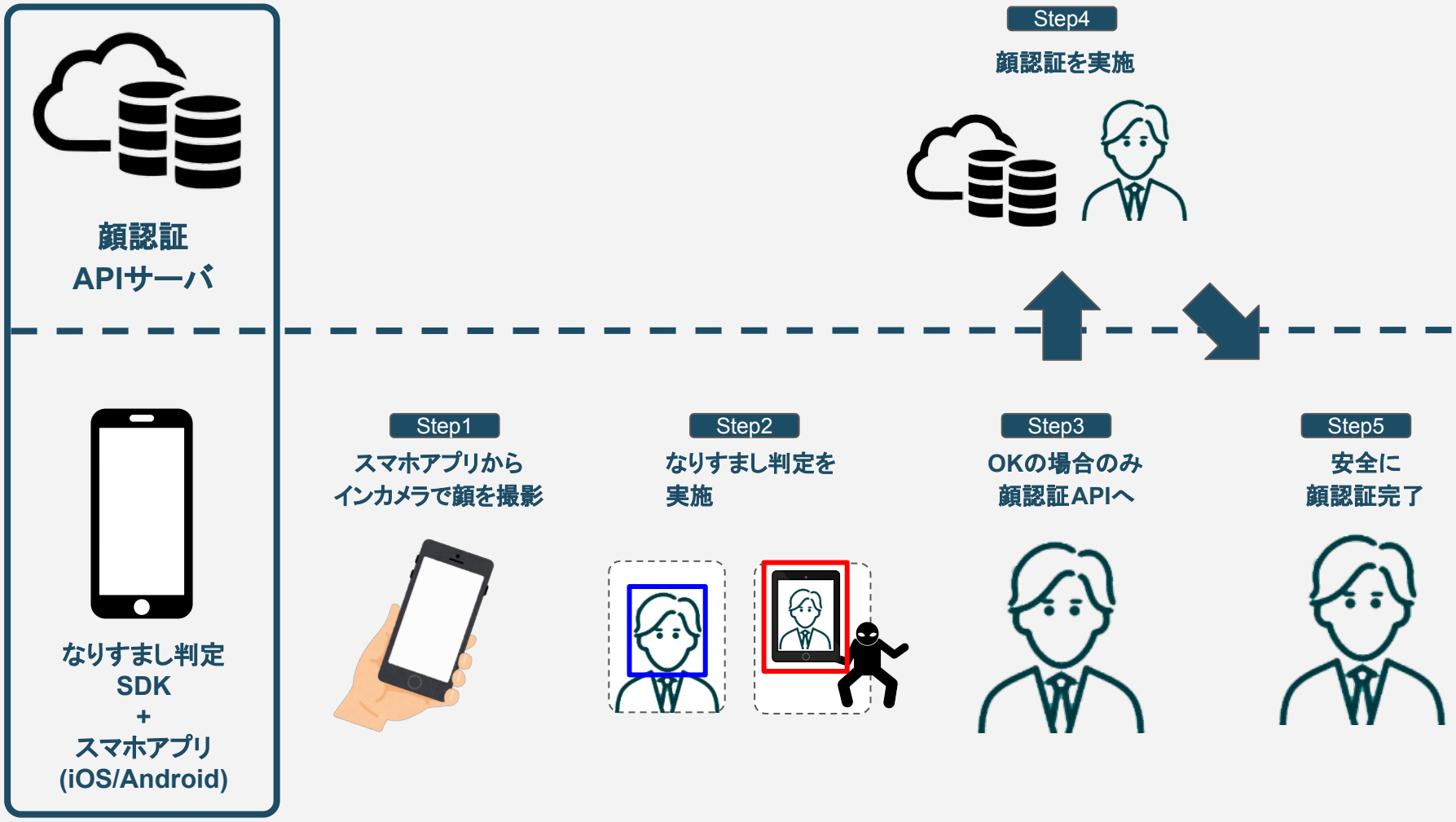
# ■なりすまし判定方法の例

顔の向き検出、目の動き検出、目の開閉状態判定、マスク/メガネ装着判定を、自由に組み合わせてユーザーに指示することにより、なりすましかどうかの判定を行うことが可能です。また、スマホやタブレットに映された顔かどうかも判定します。

| 項目 | なりすまし判定手段 | 詳細 |
|---|---|---|
| アクション判定（動画） | 首ふり動作検出 | ユーザーに顔の向きを上下左右に動かす指示を行うことでなりすましかどうかを判定させることが可能です。 |
| | 左右の目の動き検出 | ユーザーに目の向きを上下左右に動かす指示させることでなりすましかどうかを判定させることが可能です。 |
| | まばたき検出 | 所定の回数まばたきを指示させることでなりすましかどうかを判定させることが可能です。処理FPSを超える速度でのまばたきは認識しない場合があります。 |
| | メガネ脱着検出 | メガネ装着ユーザーに対して、メガネの脱着を指示することで、なりすましかどうかを判定させることが可能です。 |
| | マスク脱着検出 | マスク装着ユーザーに対して、マスクの脱着を指示することで、なりすましかどうかを判定させることが可能です。 |
| | ランダムアクション | ユーザーに、顔の動き、目の動き、まばたき、目の開閉などを認証のたびに、毎回ランダムな順番で行わせることで、ビデオなりすましから保護します。 |
| フェイク判定(静止画) | フェイク顔判定 | 顔位置検出後に、スマホやタブレット等に映された顔かどうかを判定し、端末を利用したなりすましから保護します。 |

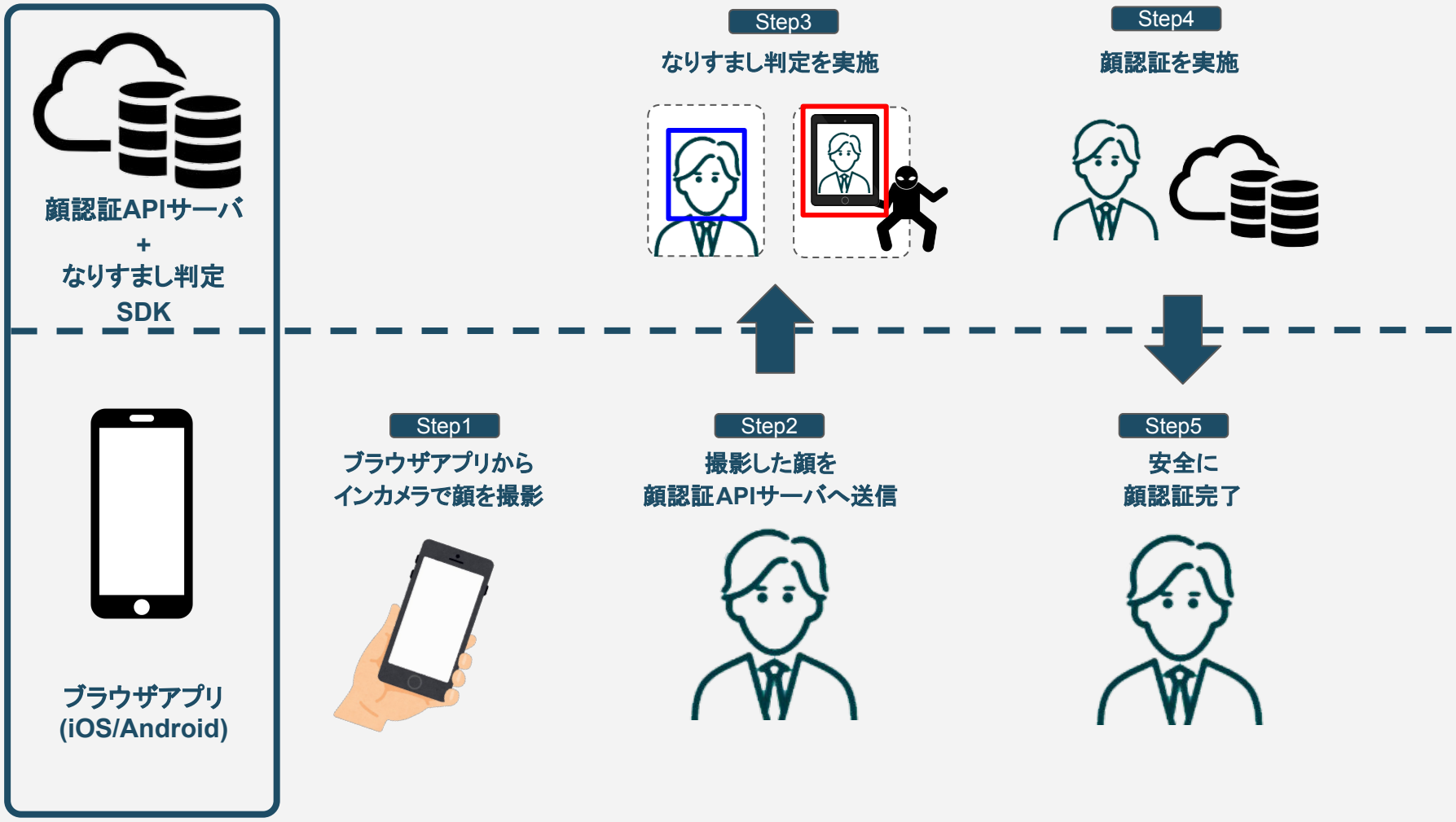
# ■サービス構成例①

下記の例は、スマホアプリになりすまし判定SDKを組み込み、クライアント側でなりすまし判定を実施してから、顔認証APIとの通信で顔認証を実施するアーキテクチャです。
なりすまし判定SDKをクライアントに組み込むことで、単一フレームのみでのなりすまし判定以外にも、動画を用いたアクション判定によるなりすまし判定を行うことも可能です。



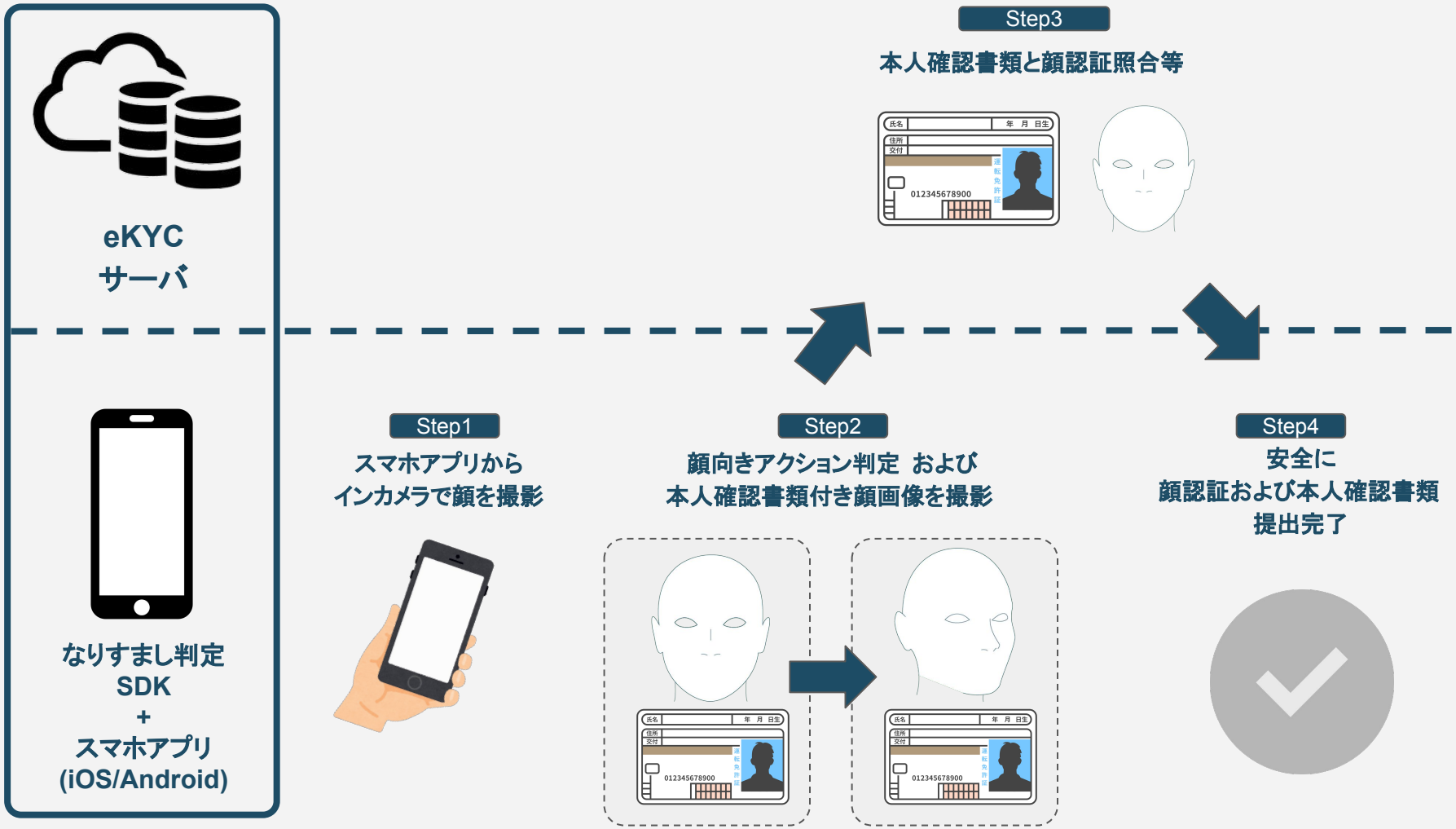
# ■サービス構成例②

下記の例は、スマホではブラウザアプリなどから顔を撮影するのみとし、なりすまし判定SDKはサーバに設置しておき、クライアント側から送られてきた顔画像を、サーバ上でなりすまし判定および顔認証を実施するアーキテクチャです。新たに顧客にアプリをインストールしてもらう手間なく、なりすまし判定の仕組みを導入した顔認証システムを構築することが可能です。



# ■サービス構成例③

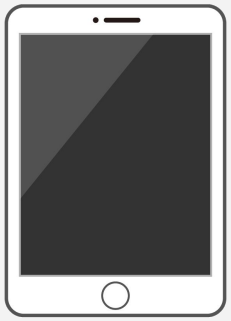
下記の例は、スマホアプリになりすまし判定SDKを組み込み、アプリでアクション判定を実施しながら本人確認書類を撮影し、所定のアクションを認識できた場合のみ、本人確認書類付き本人顔画像をサーバへ送信するアーキテクチャです。
写真なりすまし等を防止可能なため、eKYCサービスなどにも応用可能です。



# ■サービス構成例④

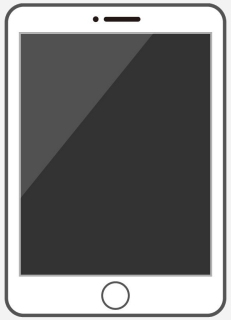
下記の例は、スマホアプリになりすまし判定SDKおよび顔認証SDKを組み込み、クライアント側でなりすまし判定から顔認証まで完結させるアーキテクチャです。
なりすまし判定SDKおよび顔認証SDKの双方をクライアントに組み込むことで、不要な通信を削減し、処理時間を短縮することが可能です。また動画を用いたアクション判定によるなりすまし判定にも適したアーキテクチャとなっています。

サーバなし

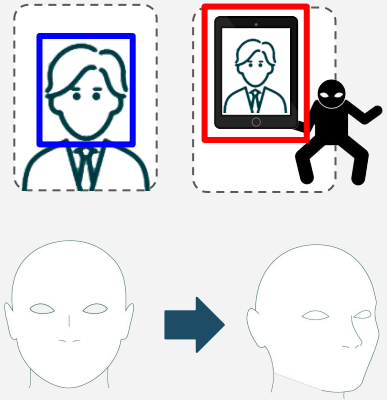


顔認証SDK
+
なりすまし判定
SDK
+
タブレットアプリ
(iOS/Android)

Step1
タブレットアプリから
インカメラで顔を撮影



Step2
なりすまし判定を
実施



Step3
OKの場合のみ
顔認証を実施

Step4
安全に
顔認証完了