Face Analysis Technology Evaluation (FATE)
MORPH

Performance of Automated Facial Morph Detection and Morph Resistant Face Recognition Algorithms
Concept, Evaluation Plan and API
VERSION 5.0.1

Mei Ngan
Patrick Grother
Kayee Hanaoka
Information Access Division
Information Technology Laboratory

April 5, 2024
## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>July 12, 2019</td>
<td>2.0</td>
<td>Initial document</td>
</tr>
<tr>
<td>September 9, 2020</td>
<td>2.0.1</td>
<td>Update link to General Evaluation Specifications document</td>
</tr>
<tr>
<td>July 7, 2021</td>
<td>2.1</td>
<td>Add optional <code>ageDeltaInDays</code> input argument to function <code>detectMorphDifferentially</code> (see Section 5.3.5)</td>
</tr>
</tbody>
</table>
| May 19, 2022    | 3.0     | - Remove optional `ageDeltaInDays` input argument to differential morph detection function in Section 5.3.5  
- Add new function to support differential morph detection with additional subject metadata in Section 5.3.6 |
| August 18, 2023 | 3.0.1   | Updating project name from FRVT to FATE                                                                                                  |
| February 1, 2024 | 5.0     | Add new functions to perform demorphing (with and without a reference probe photo) in Sections 5.3.8 and 5.3.9. Incrementing version number to 5.0 to align with version of API header file. |
| April 5, 2024   | 5.0.1   | Updating frequency of submissions to one algorithm submission every four calendar months (see Section 2.3).                                   |
Table of Contents

1. MORPH .................................................................................................................. 4
  1.1. SCOPE .............................................................................................................. 4
  1.2. GENERAL EVALUATION SPECIFICATIONS .............................................. 4
  1.3. REPORTING ..................................................................................................... 4
  1.4. ACCURACY METRICS .................................................................................... 4

2. RULES FOR PARTICIPATION ................................................................................. 5
  2.1. IMPLEMENTATION REQUIREMENTS ......................................................... 5
  2.2. PARTICIPATION AGREEMENT ...................................................................... 5
  2.3. NUMBER AND SCHEDULE OF SUBMISSIONS .......................................... 5
  2.4. VALIDATION ..................................................................................................... 5

3. DATA STRUCTURES SUPPORTING THE API .................................................. 6
  3.1. SUBJECT METADATA ..................................................................................... 6
  3.2. REQUIREMENT ................................................................................................. 6

4. IMPLEMENTATION LIBRARY FILENAME ....................................................... 6
  4.1. FILE FORMATS AND DATA STRUCTURES ................................................. 6
  4.1.1. ImageLabel describing the format of an image ........................................ 6

5. API SPECIFICATION ............................................................................................ 7
  5.1. HEADER FILE .................................................................................................. 7
  5.2. NAMESPACE ..................................................................................................... 7
  5.3. API .................................................................................................................... 7
  5.3.1. Implementation Requirements ................................................................. 7
  5.3.2. Interface ...................................................................................................... 7
  5.3.3. Initialization ................................................................................................ 8
  5.3.4. Single-image Morph Detection ................................................................. 9
  5.3.5. Two-image Differential Morph Detection .............................................. 10
  5.3.6. Two-image Differential Morph Detection with Subject Metadata .......... 10
  5.3.7. 1:1 Comparison ......................................................................................... 11
  5.3.8. Single-image Demorphing ....................................................................... 12
  5.3.9. Two-image Differential Demorphing ....................................................... 13

List of Tables

36 Table 1 – Structure for a single image ........................................................................ 6
37 Table 2 – Labels for subject sex ................................................................................ 6
38 Table 3 – Enumeration of image label ...................................................................... 6
39 Table 4 – API Functions .......................................................................................... 7
40 Table 5 – Initialization ............................................................................................. 8
41 Table 6 – Single-image Morph Detection ............................................................... 9
42 Table 7 – Two-image Differential Morph Detection ............................................. 10
43 Table 8 – Two-image Differential Morph Detection with Subject Metadata .......... 11
44 Table 9 – 1:1 Comparison ....................................................................................... 12
45 Table 10 – Single-image Demorphing .................................................................... 12
46 Table 11 – Two-image Differential Demorphing ................................................. 13
47
1. MORPH

1.1. Scope

Facial morphing (and the ability to detect it) is an area of high interest to a number of photo-credential issuance agencies and those employing face recognition for identity verification. The FATE MORPH test will provide ongoing independent testing of prototype facial morph detection technologies. The evaluation is designed to obtain an assessment on morph detection capability to inform developers and current and prospective end-users. This document establishes a concept of operations and an application programming interface (API) for evaluation of different tasks:

1. Algorithmic capability to detect facial morphing (morphed/blended faces) in still photographs
   a. Single-image morph detection of non-scanned photos, printed-and-scanned photos, and images of unknown photo format/origin
   b. Two-image differential morph detection of non-scanned photos, printed-and-scanned photos, and images of unknown photo format/origin

2. Face recognition algorithm resistance against morphing

3. Demorphing
   a. Single-image demorphing - algorithmic ability to recover images of the original identities from a single morphed face
   b. Two-image differential demorphing – algorithmic ability to recover the image of the “other unknown identity” in a morphed image, given the availability of a reference image belonging to one of the contributing subjects

1.2. General Evaluation Specifications

General and common information shared between all Ongoing FRTE/FATE tracks are documented in the General Evaluation Specifications document - https://pages.nist.gov/frvt/api/FRVT_common.pdf. This includes rules for participation, hardware and operating system environment, software requirements, reporting, and common data structures that support the APIs.

1.3. Reporting

For all algorithms that complete the evaluation, NIST will provide performance results back to the participating organizations. NIST may additionally report and share results with partner government agencies and interested parties, in workshops, conferences, conference papers, presentations and technical reports.

Important: This is a test in which NIST will identify the algorithm and the developing organization. Algorithm results will be attributed to the developer. Results will be machine generated (i.e. scripted) and will include timing, accuracy and other performance results. These will be provided alongside results from other implementations. Results will be expanded and modified as additional implementations are tested, and as analyses are implemented. Results may be regenerated on-the-fly, usually whenever additional implementations complete testing, or when new analyses are added.

1.4. Accuracy metrics

This test will evaluate algorithmic ability to detect whether an image is a morphed/blended image of two or more faces and/or to correctly reject 1:1 comparisons of morphed images against other images of the subjects used to create the morph (but similarly, correctly authenticate legitimate non-morphed, mated pairs and correctly reject non-
morphed, non-mated pairs). Per established metrics\textsuperscript{1,2} for assessment of morphing attacks, NIST will compute and report:

- Attack Presentation Classification Error Rate (APCER) – the proportion of morph attack samples incorrectly classified as bona fide presentation
- Bona Fide Presentation Classification Error Rate (BPCER) – the proportion of bona fide samples incorrectly classified as morphed samples
- Mated Morph Presentation Match Rate (MMPMR) - the proportion of comparisons where the morphed image successfully authenticates against all constituents
- True Acceptance Rate (TAR) – the proportion of non-morphed, mated comparisons that correctly authenticate
- False Match Rate (FMR) – the proportion of non-morphed, non-mated comparisons that incorrectly authenticate

We will report the above quantities as a function of alpha (the fraction of each subject that contributed to the morph), image compression ratio, image resolution, image size, and others.

We will also report error tradeoff plots (BPCER vs. APCER, MMPMR vs. FMR, parametric on threshold).

2. Rules for participation

2.1. Implementation Requirements

Developers are not required to implement all functions specified in this API. Developers may choose to implement one or more functions of this API – please refer to Section 5.3.1 for detailed information regarding implementation requirements.

2.2. Participation agreement

A participant must properly follow, complete, and submit the FRTE/FATE MORPH Participation Agreement. This must be done once, either prior or in conjunction with the very first algorithm submission. It is not necessary to do this for each submitted implementation thereafter.

2.3. Number and Schedule of Submissions

Participants may send one submission as often as every four calendar months from the last submission for evaluation. NIST reserves the right to amend this section with submission volume and frequency limits. NIST will evaluate implementations on a first-come-first-served basis and provide results back to the participants as soon as possible.

2.4. Validation

All participants must run their software through the provided FATE MORPH validation package prior to submission. The validation package will be made available at https://github.com/usnistgov/frvt. The purpose of validation is to ensure consistent algorithm output between the participant’s execution and NIST’s execution. Our validation set is not intended to provide training or test data.


3. Data structures supporting the API


3.1. Subject Metadata

Data structure representing information about a subject.

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Structure for a single image</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ code fragment</td>
<td>Remarks</td>
</tr>
<tr>
<td>typedef struct SubjectMetadata</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| | }
| | Sex of the subject |
| | Age of subject (in months) in probe image; -1 indicates an unassigned value |
| | Age/time difference (in months) between probe and reference image; -1 indicates an unassigned value |

<table>
<thead>
<tr>
<th>Table 2</th>
<th>Labels for subject sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label as C++ enumeration</td>
<td>Meaning</td>
</tr>
<tr>
<td>enum class Sex</td>
<td></td>
</tr>
<tr>
<td>Unknown=0, Female, Male,</td>
<td>Either the label is unknown or unassigned</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.2. Requirement

FATE MORPH participants should implement the relevant C++ prototyped interfaces of section 5. C++ was chosen in order to make use of some object-oriented features. Any functions that are not implemented should return ReturnCode::NotImplemented.

4. Implementation Library Filename

The core library shall be named as libfrvt_morph_<provider>_<sequence>.so, with
- provider: single word, non-infringing name of the main provider. Example: acme
- sequence: a three digit decimal identifier to start at 000 and incremented by 1 every time a library is sent to NIST. Example: 007

Example core library names: libfrvt_morph_acme_000.so, libfrvt_morph_mycompany_006.so.

Important: Public results will be attributed with the provider name and the 3-digit sequence number in the submitted library name.

4.1. File formats and data structures

4.1.1. ImageLabel describing the format of an image

<table>
<thead>
<tr>
<th>Table 3</th>
<th>Enumeration of image label</th>
</tr>
</thead>
</table>
Return code as C++ enumeration | Meaning
---|---
enum class ImageLabel {
    Unknown=0,
    NonScanned=1
    Scanned=2,
}; | Image origin is unknown or unassigned
Non-scanned photo
Printed-and-scanned photo

5. API specification

Please note that included with the FATE MORPH validation package (available at https://github.com/usnistgov/frvt) is a “null” implementation of this API. The null implementation has no real functionality but demonstrates mechanically how one could go about implementing this API.

5.1. Header File

The prototypes from this document will be written to a file named frvt_morph.h and will be available to implementers at https://github.com/usnistgov/frvt.

5.2. Namespace

All supporting data structures will be declared in the FRVT namespace. All API interfaces/function calls for this track will be declared in the FRVT_MORPH namespace.

5.3. API

5.3.1. Implementation Requirements

Developers are not required to implement all functions specified in this API. Developers may choose to implement one or more functions of Table 4, but at a minimum, developers must submit a library that implements

1. Interface of Section 5.3.2,
2. initialize() of Section 5.3.3, and
3. AT LEAST one of the functions from Table 4. For any other function that is not implemented, the function shall return ReturnCode::NotImplemented.

Table 4 – API Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Section</th>
</tr>
</thead>
</table>
| detectMorph() – single image morph detection of
  • Non-scanned photo
  • Printed-and-scanned photo
  • Image of unknown format | 5.3.4 |
| detectMorphDifferentially() – two image differential morph detection of
  • Non-scanned photo
  • Printed-and-scanned photo
  • Image of unknown format | 5.3.5 |
| compareImages() – 1:1 comparison | 5.3.6 |

5.3.2. Interface

The software under test must implement the interface Interface by subclassing this class and implementing AT LEAST ONE of the methods specified therein.
There is one class (static) method declared in `Interface::getImplementation()` which must also be implemented. This method returns a shared pointer to the object of the interface type, an instantiation of the implementation class. A typical implementation of this method is also shown below as an example.

```cpp
#include "frvt_morph.h"
using namespace FRVT_MORPH;

class NullImpl;

class Interface
{
    public:
        static std::shared_ptr<Interface> getImplementation();
};

// Other implemented functions

class NullImpl
{
    NullImpl();
    ~NullImpl();
    std::shared_ptr<Interface> Interface::getImplementation()
    {
        return std::make_shared<NullImpl>();
    }
    // Other implemented functions
};
```

### 5.3.3. Initialization

Before any morph detection or matching calls are made, the NIST test harness will call the initialization function of Table 5. This function will be called BEFORE any calls to `fork()` are made. This function **must** be implemented.

<table>
<thead>
<tr>
<th>Prototype</th>
<th>ReturnStatus initialize(const std::string &amp;configDir, const std::string &amp;configValue);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This function initializes the implementation under test and sets all needed parameters in preparation for template creation. This function will be called (N=1) times by the NIST application, prior to parallelizing (M \geq 1) calls to any morph detection or matching functions via <code>fork()</code>. This function will be called from a single process/thread.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Parameters</th>
<th>configDir</th>
<th>A read-only directory containing any developer-supplied configuration parameters or run-time data files.</th>
</tr>
</thead>
<tbody>
<tr>
<td>configValue</td>
<td></td>
<td>An optional string value encoding algorithm-specific configuration parameters. Developers may provide documentation for such configuration parameter(s) in their submission to NIST. Otherwise, the default value for this parameter will be an empty string.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Parameters</th>
<th>None</th>
</tr>
</thead>
</table>

| Return Value | See General Evaluation Specifications document for all valid return code values. This function **must** be implemented. |
5.3.4. Single-image Morph Detection

The function of Table 6 evaluates morph detection on non-scanned photos, scanned photos, and photos of unknown formats. A single image along with an associated image label describing the image format/origin is provided to the function for detection of morphing. Both morphed images and non-morphed images will be used, which will support measurement of a morph attack presentation classification error rate (APCER) with a bona fide presentation classification error rate (BPCER).

**Non-scanned photos**

Non-scanned photos are digital images known to not have been printed and scanned back in. There are a number of operational use-cases for morph detection on such digital images.

**Scanned photos**

While there are existing techniques to detect manipulation of a digital image, once the image has been printed and scanned back in, it leaves virtually no traces of the original image ever being manipulated. So the ability to detect whether a printed-and-scanned image contains a morph warrants investigation.

**Photos of unknown format**

In some cases, the format and/or origin of the image in question is not known, so images with “unknown” labels will also be tested.

Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers.

---

### Table 6 – Single-image Morph Detection

<table>
<thead>
<tr>
<th>Prototypes</th>
<th>ReturnStatus detectMorph(const Image &amp;suspectedMorph, const ImageLabel &amp;label, bool &amp;isMorph, double &amp;score);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This function takes an input image and associated image label describing the image format/origin, and outputs a binary decision on whether the image is a morph and a “morphiness” score on [0, 1] indicating how confident the algorithm thinks the image is a morph, with 0 meaning confidence that the image is not a morph and 1 representing absolute confidence that it is a morph.</td>
</tr>
<tr>
<td>Input Parameters</td>
<td>suspectedMorph Input Image</td>
</tr>
<tr>
<td>label</td>
<td>ImageLabel (Section 4.1.1) describing the format of the input image</td>
</tr>
<tr>
<td>• NonScanned = non-scanned digital photo</td>
<td></td>
</tr>
<tr>
<td>• Scanned = a photo that is printed, then scanned</td>
<td></td>
</tr>
<tr>
<td>• Unknown = unknown photo format/origin</td>
<td></td>
</tr>
<tr>
<td>Output Parameters</td>
<td>isMorph True if image contains a morph; False otherwise</td>
</tr>
<tr>
<td>score</td>
<td>A score on [0, 1] representing how confident the algorithm is that the image contains a morph. 0 means certainty that image does not contain a morph and 1 represents certainty that image contains a morph.</td>
</tr>
<tr>
<td>Return Value</td>
<td>See General Evaluation Specifications document for all valid return code values.</td>
</tr>
</tbody>
</table>

**If this function is not implemented, the return code should be set to ReturnCode::NotImplemented.**

**If this function is not implemented for a certain type of image, for example, the function supports non-scanned photos but not scanned photos, then the function should return ReturnCode::NotImplemented when the function is called with the particular unsupported image type.**
5.3.5. Two-image Differential Morph Detection

Two face samples are provided to the function of Table 7 as input, the first being a suspected morphed facial image and the second image representing a known, non-morphed face image of one of the subjects contributing to the morph (e.g., live capture image from an eGate). This procedure supports measurement of whether algorithms can detect morphed images when additional information (provided as the second supporting known subject image) is provided.

Similar to single-image morph detection, the function of Table 7 will support non-scanned, scanned, and photos of unknown format/origin. The input image type will be specified by the associated ImageLabel input parameter.

Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers.

<table>
<thead>
<tr>
<th>Table 7 – Two-image Differential Morph Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prototypes</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Input Parameters</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Output Parameters</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

5.3.6. Two-image Differential Morph Detection with Subject Metadata

Two face samples are provided to the function of Table 8 as input, the first being a suspected morphed facial image and the second image representing a known, non-morphed face image of one of the subjects contributing to the morph (e.g., live capture image from an eGate). In addition, subject metadata is provided as input to the algorithm, which includes sex, age of the subject (in months) at the time the probe image is taken, and the age/time difference (in months) between the suspected morph and the live probe image. Operationally, this information might be derived from data read from the machine readable zone of a passport for example. This procedure supports measurement of whether algorithms can detect morphed images when additional subject metadata is provided.
Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers.

Table 8 – Two-image Differential Morph Detection with Subject Metadata

<table>
<thead>
<tr>
<th>Prototypes</th>
<th>ReturnStatus detectMorphDifferentially(const Image &amp;suspectedMorph, const ImageLabel &amp;label, const Image &amp;probeFace, const SubjectMetadata &amp;subjectMetadata, bool &amp;isMorph, double &amp;score);</th>
</tr>
</thead>
</table>

**Description**
This function takes two input images - a known unaltered/not morphed image of the subject \(\text{probeFace}\) and an image of the same subject that's in question (may or may not be a morph) \(\text{suspectedMorph}\) with an associated image label describing the image format/origin. Additionally, subject metadata is provided as input to the algorithm, which include sex, age of the subject (in months) at the time the probe image is taken, and the age/time difference (in months) between the suspected morph and the live probe image. This function outputs a binary decision on whether \(\text{suspectedMorph}\) is a morph \(\text{given probeFace as a prior}\) and a "morphiness" score on \([0, 1]\) indicating how confident the algorithm thinks the \(\text{suspectedMorph}\) is a morph, with 0 meaning confidence that the \(\text{suspectedMorph}\) is not a morph and 1 representing absolute confidence that it is a morph.

**Input Parameters**
- **suspectedMorph**: Input Image
- **label**: ImageLabel (Section 4.1.1) describing the format of the suspected morph image
  - NonScanned = non-scanned digital photo
  - Scanned = a photo that is printed, then scanned
  - Unknown = unknown photo format/origin
- **probeFace**: An image of the subject known not to be a morph (e.g., live capture image)
- **subjectMetadata**: SubjectMetadata (Section 3.1) with information about the subject

**Output Parameters**
- **isMorph**: True if image contains a morph; False otherwise
- **score**: A score on \([0, 1]\) representing how confident the algorithm is that the image contains a morph. 0 means certainty that image does not contain a morph and 1 represents certainty that image contains a morph.

**Return Value**
See General Evaluation Specifications document for all valid return code values.

If this function is not implemented, the return code should be set to ReturnCode::NotImplemented.

If this function is not implemented for a certain type of image, for example, the function supports non-scanned photos but not scanned photos, then the function should return ReturnCode::NotImplemented when the function is called with the particular unsupported image type.

5.3.7. 1:1 Comparison

Two face samples are provided to the function of Table 9 for one-to-one comparison of whether the two images are of the same subject. The expected behavior from the algorithm is to be able to correctly reject comparisons of morphed images against constituents that contributed to the morph. The goal is to show algorithm robustness against morphing alterations when morphed images are compared against other images of the subjects used for morphing.

Comparisons of morphed images against constituents should return a low similarity score, indicating rejection of match. Comparisons of unaltered/non-morphed images of the same subject should return a high similarity score, indicating acceptance of match.
Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers.

### Table 9 – 1:1 Comparison

| Prototypes | ReturnStatus compareImages;
| const Image &enrollImage, const Image &verifImage, double &similarity); | Input, Input, Output |
| Description | This function compares two images and outputs a similarity score. In the event the algorithm cannot perform the comparison operation, the similarity score shall be set to -1.0 and the function return code value shall be set appropriately. |
| Input Parameters | enrollImage The enrollment image, verifImage The verification image |
| Output Parameters | similarity A similarity score resulting from comparison of the two images, on the range [0, DBL_MAX]. |
| Return Value | See General Evaluation Specifications document for all valid return code values. |
| | If this function is not implemented, the return code should be set to ReturnCode::NotImplemented. |

#### 5.3.8. Single-image Demorphing

The function of Table 10 evaluates single-image “demorphing” – algorithmic ability to recover images of both identities simultaneously from a single morphed face. The goal is to show algorithm ability to accurately restore the identities of the contributing subjects if the image is a morph. All morphs will be generated with two contributing subjects, and both morphed and non-morphed images will be evaluated. If the input image is a morph, the algorithm should deduce/restore the two individual face images/identities that contributed to the morph. If the input is a bona fide image, the algorithm should produce two images/identities that are essentially the same as the input photo. NIST will report performance by analyzing face recognition outcomes between the original and restored imagery.

Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers.

### Table 10 – Single-image Demorphing

| Prototypes | ReturnStatus demorph( const Image &suspectedMorph, Image &outputSubject1, Image &outputSubject2, bool &isMorph, double &score); |
| Description | This function takes an input image and outputs two images. If the input image is a morph, the algorithm should deduce/restore the two individual face images/identities that contributed to the morph. If the input is a bona fide image, the algorithm should produce two images that are essentially the same as the input photo. |
| | Optionally, the algorithm can also return a binary decision on whether the image is a morph and a "morphiness" score on [0, 1] indicating how confident the algorithm thinks the image is a morph, with 0 meaning confidence that the image is not a morph and 1 representing absolute confidence that it is a morph. A score of -1.0 indicates that the algorithm did not implement morph detection and both “isMorph” and “score” will be ignored. |
| Input Parameters | suspectedMorph Input Image |
| Output Parameters | outputSubject1 outputSubject2 If the input image is a morph, the algorithm should deduce/restore the two individual face images/identities that contributed to the morph. If the input is a bona fide image,
the algorithm should produce two images that are essentially the same as the input photo.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>See General Evaluation Specifications document for all valid return code values.</th>
</tr>
</thead>
</table>

If this function is not implemented, the return code should be set to ReturnCode::NotImplemented.

### 5.3.9. Two-image Differential Demorphing

The function of Table 11 evaluates two-image differential “demorphing” – algorithmic ability to recover the image of the “other unknown identity” in a morphed image, given the availability of a reference image belonging to one of the contributing subjects. The goal is to show algorithm ability to accurately restore the identity of the second subject if the image is a morph. All morphs will be generated with two contributing subjects, and both morphed and non-morphed images will be evaluated. If the input image is a morph, the algorithm should deduce/restore the second/unknown individual face image/identity that contributed to the morph. If the input is a bona fide image, the algorithm should produce an image/identity that is essentially the same as the input photo. NIST will report performance by analyzing face recognition outcomes between the original and restored imagery.

Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers.

<table>
<thead>
<tr>
<th>Prototypes</th>
<th>ReturnStatus demorphDifferentially(const Image &amp;suspectedMorph, const Image &amp;probeFace, Image &amp;outputSubject, bool &amp;isMorph, double &amp;score);</th>
</tr>
</thead>
</table>

| Description | This function takes two input images - a known unaltered/not morphed image of the subject (probeFace) and an image of the same subject that's in question (may or may not be a morph) (suspectedMorph). If the input image is a morph, the algorithm should deduce/restore the other/unknown individual face image/identity that contributed to the morph. If the input is a bona fide image, the algorithm should produce an image that is essentially the same as the input photo. Optionally, the algorithm can also return a binary decision on whether the image is a morph and a "morphiness" score on [0, 1] indicating how confident the algorithm thinks the image is a morph, with 0 meaning confidence that the image is not a morph and 1 representing absolute confidence that it is a morph. A score of -1.0 indicates that the algorithm did not implement morph detection and both “isMorph” and “score” will be ignored. |

<table>
<thead>
<tr>
<th>Input Parameters</th>
<th>suspectedMorph</th>
<th>Input Image</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>probeFace</td>
<td>An image of the subject known not to be a morph (e.g., live capture image)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Parameters</th>
<th>outputSubject</th>
<th>If the input image is a morph, the algorithm should deduce/restore the other/unknown individual face image/identity that contributed to the morph. If the input is a bona fide image, the algorithm should produce an image that is essentially the same as the input photo.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>isMorph (optional)</td>
<td>True if image contains a morph; False otherwise</td>
</tr>
<tr>
<td></td>
<td>score (optional)</td>
<td>A score on [0, 1] representing how confident the algorithm is that the image contains a morph. 0 means certainty that image does not contain a morph and 1 represents certainty that image contains a morph. A score of -1.0 indicates that the algorithm did not implement morph detection and both “isMorph” and “score” will be ignored.</td>
</tr>
</tbody>
</table>
Return Value | See General Evaluation Specifications document for all valid return code values.
| If this function is not implemented, the return code should be set to `ReturnCode::NotImplemented`. |