Performance of Automated Facial Morph Detection and Morph Resistant Face Recognition Algorithms

Concept, Evaluation Plan and API

VERSION 2.0.1

Mei Ngan
Patrick Grother
Kayee Hanaoka

Information Access Division
Information Technology Laboratory

September 9, 2020
## 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>
</tbody>
</table>
Table of Contents

1. MORPH .................................................................................................................. 4
   1.1. Scope .................................................................................................................. 4
   1.2. GENERAL FRVT 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 ...................................................... 5
   3.1. 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 .......................................................................................... 6
   5.1. HEADER FILE .................................................................................................. 6
   5.2. NAMESPACE ..................................................................................................... 6
   5.3. API .................................................................................................................. 6
       5.3.1. Implementation Requirements ................................................................. 6
       5.3.2. Interface .................................................................................................... 7
       5.3.3. Initialization ............................................................................................... 8
       5.3.4. Single-image Morph Detection ................................................................. 8
       5.3.5. Two-image Differential Morph Detection ................................................ 9
       5.3.6. 1:1 Comparison ....................................................................................... 10

List of Tables

   Table 3 – Enumeration of image label .................................................................... 6
   Table 6 – API Functions ......................................................................................... 7
   Table 7 – Initialization ............................................................................................. 8
   Table 8 – Single-image Morph Detection .............................................................. 8
   Table 9 – Two-image Differential Morph Detection ............................................. 9
   Table 10 – 1:1 Comparison .................................................................................... 10
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 FRVT 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 two separate 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

1.2. **General FRVT Evaluation Specifications**

General and common information shared between all Ongoing FRVT tracks are documented in the FRVT General Evaluation Specifications document - [https://pages.nist.gov/frvt/api/FRVT_common.pdf](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, and 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\(^1\)\(^2\) 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 FRVT 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

Currently, the number and schedule of submissions is not regulated, so participants can send submissions at any time. 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 FRVT 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. Requirement
FRVT 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>Return code as C++ enumeration</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>enum class ImageLabel {</td>
<td></td>
</tr>
<tr>
<td>Unknown=0,</td>
<td>Image origin is unknown or unassigned</td>
</tr>
<tr>
<td>NonScanned=1</td>
<td>Non-scanned photo</td>
</tr>
<tr>
<td>Scanned=2,</td>
<td>Printed-and-scanned photo</td>
</tr>
<tr>
<td>};</td>
<td></td>
</tr>
</tbody>
</table>

5. API specification
Please note that included with the FRVT 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 2, 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 2. For any other function that is not implemented, the function shall return `ReturnCode::NotImplemented`.

### Table 2 – API Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>detectMorph()</code> – single image morph detection of • Non-scanned photo • Printed-and-scanned photo • Image of unknown format</td>
<td>5.3.4</td>
</tr>
<tr>
<td><code>detectMorphDifferentially()</code> – two image differential morph detection of • Non-scanned photo • Printed-and-scanned photo • Image of unknown format</td>
<td>5.3.5</td>
</tr>
<tr>
<td><code>compareImages()</code> – 1:1 comparison</td>
<td>5.3.6</td>
</tr>
<tr>
<td><code>trainMorphDetector()</code> – training for morph detection</td>
<td>Error! Reference source not found.</td>
</tr>
</tbody>
</table>

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.

#### C++ code fragment

```cpp
#include "frvt_morph.h"

using namespace FRVT_MORPH;

NullImpl::NullImpl() {}
NullImpl::~NullImpl() {}

std::shared_ptr<Interface> Interface::getImplementation()
{
    return std::make_shared<NullImpl>();
}

// Other implemented functions
```

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.

#### C++ code fragment

```cpp
#include "frvt_morph.h"

using namespace FRVT_MORPH;

NullImpl::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 3. This function will be called BEFORE any calls to fork() are made. This function must be implemented.

Table 3 – Initialization

| Prototype | ReturnStatus initialize(
|           | const std::string &configDir,
<table>
<thead>
<tr>
<th></th>
<th>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 &gt;= 1 calls to any morph detection or matching functions via fork(). This function will be called from a single process/thread.</td>
</tr>
<tr>
<td>Input Parameters</td>
<td>configDir A read-only directory containing any developer-supplied configuration parameters or run-time data files.</td>
</tr>
<tr>
<td></td>
<td>configValue 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>
<tr>
<td>Output Parameters</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>See General Evaluation Specifications document for all valid return code values. This function must be implemented.</td>
</tr>
</tbody>
</table>

5.3.4. Single-image Morph Detection

The function of Table 4 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 4 – Single-image Morph Detection

| Prototypes | ReturnStatus detectMorph() |
**FRVT MORPH**

```cpp
const Image &suspectedMorph,
const ImageLabel &label,
bool &isMorph,
double &score);
```

**Input Parameters**
- **suspectedMorph**: Input Image
- **label**: ImageLabel (Section 4.1.1) describing the format of the input image
  - NonScanned = non-scanned digital photo
  - Scanned = a photo that is printed, then scanned
  - Unknown = unknown photo format/origin

**Output Parameters**
- **isMorph**: True if image contains a morph; False otherwise
- **score**: A score on [0, 1] representing how confident the algorithm thinks 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.`

### 5.3.5. Two-image Differential Morph Detection

Two face samples are provided to the function of Table 5 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 5 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 5 – Two-image Differential Morph Detection**

<table>
<thead>
<tr>
<th>Prototypes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReturnStatus detectMorphDifferentially(</td>
<td></td>
</tr>
<tr>
<td>const Image &amp;suspectedMorph,</td>
<td></td>
</tr>
<tr>
<td>const ImageLabel &amp;label,</td>
<td></td>
</tr>
<tr>
<td>const Image &amp;probeFace,</td>
<td></td>
</tr>
<tr>
<td>bool &amp;isMorph,</td>
<td></td>
</tr>
<tr>
<td>double &amp;score);</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>Input</td>
</tr>
<tr>
<td>Input</td>
<td>Input</td>
</tr>
<tr>
<td>Output</td>
<td>Output</td>
</tr>
</tbody>
</table>

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) with an associated image label describing the image format/origin. This function outputs a binary decision on whether suspectedMorph is a morph (given probeFace as a prior) and a “morphiness” score on [0, 1] indicating how confident the algorithm thinks the suspectedMorph is a morph, with 0 meaning confidence that the suspectedMorph is not a morph and 1 representing absolute confidence that it is a morph.
### 5.3.6. 1:1 Comparison

Two face samples are provided to the function of Table 6 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 6 – 1:1 Comparison

<table>
<thead>
<tr>
<th>Prototypes</th>
<th>ReturnStatus compareImages(const Image &amp;enrollImage, const Image &amp;verifImage, double &amp;similarity);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
<tr>
<td>Input Parameters</td>
<td>enrollImage The enrollment image</td>
</tr>
<tr>
<td></td>
<td>verifImage The verification image</td>
</tr>
<tr>
<td>Output Parameters</td>
<td>similarity A similarity score resulting from comparison of the two images, on the range [0, DBL_MAX].</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.