Face Recognition Vendor Test
Ongoing

Still Face 1:1 Verification

Application Programming Interface (API)

VERSION 4.0.3

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

Contact via frvt@nist.gov

March 22, 2021

National Institute of
Standards and Technology
U.S. Department of Commerce

Revision History

Date	Version	Description
April 1, 2019	4.0	Initial document
June 24, 2020	4.0.1	Update feature extraction times in Table 1.3 from 1000ms to 1500ms
September 9, 2020	4.0.2	Update link to General Evaluation Specifications document Adjust the legal similarity score range to [0, 1000000]
March 22, 2021	4.0.3	Update 1:1 matching time limit in Table 1.3 from 5 milliseconds to 0.1 milliseconds (or 100 microseconds)

Table of Contents

List of Tables

List of Figures

1. FRVT 1:1

1.1. Scope

This document establishes a concept of operations and an application programming interface (API) for evaluation of face recognition (FR) implementations submitted to NIST's ongoing Face Recognition Vendor Test. This API is for the 1:1 identity verification track. Separate API documents will be published for future additional tracks to FRVT. All images include exactly one face.

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. This includes rules for participation, hardware and operating system environment, software requirements, reporting, and common data structures that support the APIs.

1.3. Time limits

The elemental functions of the implementations shall execute under the time constraints of Table 1. These time limits apply to the function call invocations defined in section 3. Assuming the times are random variables, NIST cannot regulate the maximum value, so the time limits are median values. This means that the median of all operations should take less than the identified duration.

The time limits apply per image. When K images of a person are present, the time limits shall be increased by a factor K.

NOTE: For developers that cannot meet the required time limit for matching two templates, please contact frvt@nist.gov.

Table 1 – Processing time limits in milliseconds, per 640 x 480 image

Function	1:1 verification
Feature extraction enrollment	1500 (1 core) 640x480 pixels
Feature extraction for verification	1500 (1 core) 640x480 pixels
Matching	0.1 (1 core)

2. Data structures supporting the API

The data structures supporting this API are documented in the FRVT - General Evaluation Specifications document available at https://pages.nist.gov/frvt/api/FRVT_common.pdf with corresponding header file named *frvt_structs.h* published at https://github.com/usnistgov/frvt.

3. Implementation Library Filename

The core library shall be named as libfrvt_11_<***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_11_acme_000.so, libfrvt_11_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. API Specification

FRVT 1:1 participants shall implement the relevant C++ prototyped interfaces in Section 4.4. C++ was chosen in order to make use of some object-oriented features.

4.1. Header File

The prototypes from this document will be written to a file named **frvt11.h** and will be available to implementers at https://github.com/usnistgov/frvt.

4.2. 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_11` namespace.

4.3. Overview

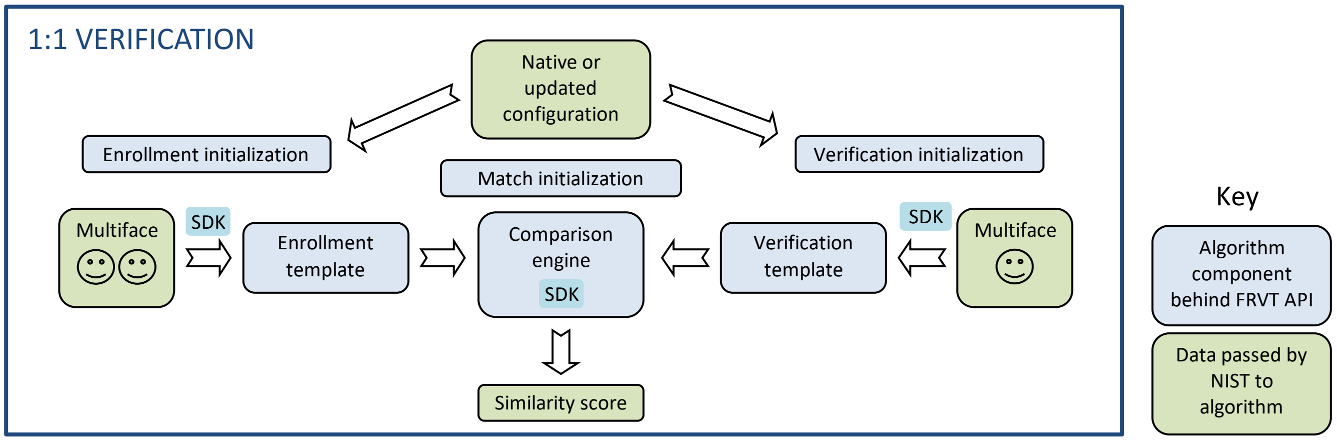
1:1 VERIFICATION
Native or updated configuration
Enrollment initialization
Verification initialization
Match initialization
Multiface
SDK
Enrollment template
Comparison engine
SDK
Verification template
SDK
Multiface
Similarity score
Key
Algorithm component behind FRVT API
Data passed by NIST to algorithm

Figure 1 – Schematic of 1:1 verification

The 1:1 testing will proceed in the following phases: optional offline training; preparation of enrollment templates; preparation of verification templates; and matching. Note that training, template creation, and matching may all be performed as separate processes. These are detailed in Table 2.

Table 2 – Functional summary of the 1:1 application

Phase	Description	Performance Metrics to be reported by NIST
Initialization	Function to read configuration data, if any.	None
Enrollment	Given K ≥ 1 input images of an individual, the implementation will create a proprietary enrollment template. That is, createTemplate(role=FRVT::TemplateRole::Enrollment_11) will be called. NIST will manage storage of these templates.	Statistics of the time needed to produce a template. Statistics of template size. Rate of failure to produce a template
Verification	Given K ≥ 1 input images of an individual, the implementation will create a proprietary verification template. That is, createTemplate(role=FRVT::TemplateRole::Verification_11) will be called. NIST will manage storage of these templates.	Statistics of the time needed to produce a template. Statistics of template size. Rate of failure to produce a template.
Matching (i.e. comparison)	Given a proprietary enrollment and a proprietary verification template, compare them to produce a similarity score.	Statistics of the time taken to compare two templates. Accuracy measures, primarily reported as DETs, including for partitions of the input datasets.

NIST requires that these operations may be executed in a loop in a single process invocation, or as a sequence of independent process invocations, or a mixture of both.

4.4. API

4.4.1. Interface

The software under test must implement the interface `Interface` by subclassing this class and implementing each method specified therein.

	C++ code fragment	Remarks
1.	`class Interface`	

2.	`{`
	`public:`	

3.	`virtual ReturnStatus initialize(`
	`const std::string &configDir ) = 0;`	

4.	`virtual ReturnStatus createTemplate(`
	`const Multiface &faces,`
	`TemplateRole role,`
	`std::vector<uint8_t> &templ,`
	`std::vector<EyePair> &eyeCoordinates) = 0;`	

5.	`virtual ReturnStatus matchTemplates(`
	`const std::vector<uint8_t> &verifTemplate,`
	`const std::vector<uint8_t> &enrollTemplate,`
	`double &similarity) = 0;`	

6.	`static std::shared_ptr<Interface> getImplementation();`	Factory method to return a managed pointer to the `Interface` object. This function is implemented by the submitted library and must return a managed pointer to the `Interface` object.
8.	`};`	

There is one class (static) method declared in `Interface.getImplementation()` which must also be implemented by the implementation. 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	Remarks

	`#include "frvt11.h"`
	
	`using namespace FRVT_11;`
	
	`NullImpl:: NullImpl () { }`
	
	`NullImpl::~ NullImpl () { }`
	
	`std::shared_ptr<Interface>`
	`Interface::getImplementation()`
	`{`
	`    return std::make_shared<NullImpl>();`
	`}`
	
	`// Other implemented functions`	

4.4.2. Initialization

The NIST test harness will call the initialization function in Table 3 before calling template generation or matching. This function will be called BEFORE any calls to `fork()`[1] are made.

Table 3 – Initialization

Prototype	ReturnStatus initialize(	
	const string &configDir);	Input
Description	This function initializes the implementation under test. It will be called by the NIST application before any call to `createTemplate()` or `matchTemplates()`. The implementation under test should set all parameters.	

[1] http://man7.org/linux/man-pages/man2/fork.2.html

	This function will be called N=1 times by the NIST application, prior to parallelizing M >= 1 calls to `createTemplate()` via `fork()`.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files. The name of this directory is assigned by NIST, not hardwired by the provider. The names of the files in this directory are hardwired in the implementation and are unrestricted.
Output Parameters	none	
Return Value	See General Evaluation Specifications document for all valid return code values.	

4.4.3. Template generation

The function of Table 4 supports role-specific generation of template data. Template format is entirely proprietary. Some of the proposed datasets include K > 2 image per person for some persons. This affords the possibility to model a recognition scenario in which a new image of a person is compared against all prior images. Use of multiple images per person has been shown to elevate accuracy over a single image.

For this test, NIST will enroll K >= 1 images under each identity. The method by which the face recognition implementation exploits multiple images is not regulated. The test seeks to evaluate developer provided technology for multi-presentation fusion.

This document defines a template to be the result of applying feature extraction to a set of K >= 1 images. An algorithm might internally fuse K feature sets into a single model or maintain them separately. In any case, the resulting proprietary template is contained in a contiguous block of data. All verification functions operate on such multi-image templates.

Table 4 – Template generation

Prototypes	ReturnStatus createTemplate(	
	const Multiface &faces,	Input
	TemplateRole role,	Input
	std::vector<uint8_t> &templ,	Output
	std::vector<EyePair> &eyeCoordinates);	Output
Description	Takes a Multiface and outputs a proprietary template and associated eye coordinates. The vectors to store the template and eye coordinates will be initially empty, and it is up to the implementation to populate them with the appropriate data. In all cases, even when unable to extract features, the output shall be a template that may be passed to the matchTemplates() function without error. That is, this routine must internally encode "template creation failed" and the matcher must transparently handle this. Note: In the rare event that more than one face is detected in an image, features should be extracted from the foreground face, that is, the largest face in the image.	
Input Parameters	faces	Implementations must alter their behavior according to the number of images contained in the structure and the TemplateRole type.
	role	Label describing the type/role of the template to be generated. Valid values are FRVT::TemplateRole::Enrollment_11 or FRVT::TemplateRole::Verification_11.
Output Parameters	templ	The output template. The format is entirely unregulated. This will be an empty vector when passed into the function, and the implementation can resize and populate it with the appropriate data.
	eyeCoordinates	For each input image in the Multiface, the function shall return the estimated eye centers. This will be an empty vector when passed into the function, and the implementation shall populate it with the appropriate number of entries. Values in eyeCoordinates[i] shall correspond to faces[i].
Return Value	See General Evaluation Specifications document for all valid return code values.	

4.4.4. Matching

Matching of one enrollment against one verification template shall be implemented by the function of Table 5.

Table 5 – Template matching

Prototype	ReturnStatus matchTemplates(	
	const std::vector<uint8_t> &verifTemplate,	Input
	const std::vector<uint8_t> &enrollTemplate,	Input
	double &similarity);	Output
Description	Compare two proprietary templates and output a similarity score, which need not satisfy the metric properties. When either or both of the input templates are the result of a failed template generation (see Table 4), the similarity score shall be -1 and the function return value shall be `VerifTemplateError`.	
Input Parameters	verifTemplate	A verification template from createTemplate(role=Verification_11). The underlying data can be accessed via verifTemplate.data(). The size, in bytes, of the template could be retrieved as verifTemplate.size().
	enrollTemplate	An enrollment template from createTemplate(role=Enrollment_11). The underlying data can be accessed via enrollTemplate.data(). The size, in bytes, of the template could be retrieved as enrollTemplate.size().
Output Parameters	similarity	A similarity score resulting from comparison of the templates, on the range [0, 1000000].
Return Value	See General Evaluation Specifications document for all valid return code values.	