LIBRARY OPERATING INSTRUCTIONS

Definitions:

Verification

Verification can be viewed as a process of making a binary decision: “yes” (two images belong to the same person), “no” (two photos show different people). Before dealing with verification metrics, it is useful to understand how we can classify errors in such tasks. Considering that there are 2 possible answers of the algorithm and 2 variants of the true state of affairs, there are 4 possible outcomes in total.

Of these outcomes, two correspond to the correct answers of the algorithm, and two correspond to errors of the first and second kind, respectively. Errors of the first kind are called "false accept", "false positive" or "false match" (incorrectly accepted), and errors of the second kind are called "false reject", "false negative" or "false non-match" (incorrectly rejected).

Summing up the number of errors of various kinds among pairs of images in the dataset and dividing them by the number of pairs, we get a false accept rate (FAR) and a false reject rate (FRR). In the case of an access control system, "false positive" corresponds to granting access to a person for whom this access is not provided, while "false negative" means that the system erroneously denied access to an authorized person. These errors have different business costs and are therefore treated separately. In the access control example, "false negative" causes the security officer to double-check the employee's badge. Giving unauthorized access to a potential intruder (false positive) can lead to much worse consequences.

Given that different types of errors are associated with different risks, facial recognition software vendors often provide an opportunity to tweak the algorithm to minimize one type of error. To do this, the algorithm returns not a binary value, but a real number that reflects the confidence of the algorithm in its decision. In this case, the user can independently choose the threshold and fix the error level at certain values.

Identification

Identification or search for a person among a set of images. The search results are sorted by the confidence of the algorithm, and the most likely matches are placed at the top of the list. Depending on whether or not the person being sought is present in the search database, identification is divided into two subcategories: closed-set identification (it is known that the person being sought is in the database) and open-set identification (the person being sought may not be in the database).

Accuracy is a reliable and understandable metric for closed-set identification. In essence, accuracy measures the number of times the person of interest was among the search results.

How does it work in practice? Let's figure it out. Let's start with the formulation of business requirements. Let's say we have a web page that can host ten search results. We need to measure the number of times the person we are looking for appears in the first ten responses of the algorithm. Such a number is called Top-N precision (in this particular case, N is 10).

For each trial, we define an image of the person we will be looking for and a gallery in which we will search, so that the gallery contains at least one other image of that person. We look at the first ten results of the search algorithm and check if the person we are looking for is among them. To get accuracy, sum all the trials in which the person you are looking for was in the search results and divide by the total number of trials.

Biometric template - Data representing the biometric characteristics of a registered identity.

The degree of correspondence between biometric templates - shows the degree of correspondence between the compared templates.

TPR/FPR Metrics:
When converting the real response of the algorithm (usually the probability of belonging to a class, see SVM separately) into a binary label, we must choose some threshold at which 0 becomes 1. A threshold of 0.5 seems natural and close, but it is not always turns out to be optimal, for example, in the aforementioned lack of class balance.
One way to evaluate the model as a whole, without being tied to a specific threshold, is AUC–ROC (or ROC AUC) - the area (Area Under Curve) under the error curve (Receiver Operating Characteristic curve). This curve is a line from (0.0) to (1.1) in True Positive Rate (TPR) and False Positive Rate (FPR) coordinates:

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

We already know TPR, this is completeness, and FPR shows what proportion of objects of the negative class the algorithm predicted incorrectly. In the ideal case, when the classifier makes no errors (FPR = 0, TPR = 1), we will get the area under the curve equal to one; otherwise, when the classifier randomly outputs class probabilities, the AUC–ROC will tend to 0.5, since the classifier will output the same amount of TP and FP.
Each point on the graph corresponds to the choice of some threshold. The area under the curve in this case shows the quality of the algorithm (more is better), in addition, the steepness of the curve itself is important - we want to maximize TPR while minimizing FPR, which means that our curve should ideally tend to the point (0,1).

Search index is a data structure that contains information about documents and is used by search engines.

TPIR/FPIR Metrics:
True positive identification rate (TPIR): The expected proportion of user identification transactions registered in the system that would result in a valid user ID being present among the returned t IDs. If the output of a biometric system is the t closest candidate matches, the corresponding TPIR score is also known as the t-rank of identification.
False positive identification rate (FPIR): the expected proportion of user identification transactions that are NOT registered in the system, as a result of which an identifier is returned. This means that a fraudster, even if not registered in the system, by sending his biometric data for processing, receives a positive response from the system – admission to the object. Whereas in this situation, the expected reaction of the system does not assume the return of any nearest

candidate. This is a kind of analogue of the second kind error for the verification procedure. The FPIR error depends both on the number of registered users (N) and on the value of the threshold (n), which determines the allowable proximity measure between the candidate and the database templates. In addition, it is impossible to determine FPIR for the identification procedure on a closed set, because all users of the system are registered in the system.
False negative identification rate (FNIR) - the expected proportion of user identification transactions registered in the system, as a result of which the correct user ID will NOT be present among the returned t IDs. FNIR depends on the number of registered users (N), on the threshold value (n), which determines the allowable proximity measure between the candidate and base templates, and also on the number of returned candidates - on the identification rank.
FPIR is defined when an input pattern erroneously matches one or more patterns from the base. Then this error is calculated as one minus the probability that there were no matches with any of the database templates (N is the number of registered templates in the database). In case the FMR is small, the FPIR error can be approximated.

Descriptor is a lexical unit (word, phrase) of an information retrieval language that serves to describe the main semantic content of a document or formulate a query when searching for a document (information) in an information retrieval system. The descriptor is uniquely assigned to a group of natural language keywords selected from a text related to a particular field of knowledge.

ROC-curve - a graph that allows you to evaluate the quality of a binary classification, displays the ratio between the proportion of objects from the total number of feature carriers that are correctly classified as carriers of the feature (true positive rate, TPR, called the sensitivity of the classification algorithm), and the proportion of objects from the total number of objects that do not carriers of a feature erroneously classified as carriers of a feature (false positive rate, FPR, the value of 1–FPR is called the specificity of the classification algorithm) when the threshold of the decision rule is varied. Also known as the error curve.

Docker is a popular program based on containerization technology. Docker allows you to run Docker containers with applications from pre-prepared templates - Docker images (or in other words Docker images).

Containerization is a technology that helps run applications in isolation from the operating system. The application, as it were, is packaged in a special shell - a container, inside which is the environment necessary for work.

Step 0 – Open terminal

In Ubuntu, the console starts when the system boots. The terminal is also a console, but already in a graphical shell. It can be launched by typing the word Terminal in the OS search bar, or through the key combination **Ctrl + Alt + T**.

In general, in Ubuntu, the commands are as follows:

<program – key value>

The program is the executable itself. In other words, a program will be executed on command.

Key – usually each program has its own set of keys. They can be found in the manual for the program.
Value – program parameters: digits, letters, symbols, variables.

Recall that to execute a command, you need to enter it into the command line – Ubuntu console or a terminal that emulates the operation of the console.

Consider the basic Ubuntu console commands:

<sudo>
The intermediate command **sudo** (SuperUser DO – superuser) allows you to run programs as an administrator or root user. You can add **sudo** before any command to run it as root.

<apt>
Command **apt** is used to work with software packages to install software packages (sudo apt install package–name), update a package repository (sudo apt update), and upgrade packages that are installed on the system (<sudo apt upgrade>).

<pwd>
Command **pwd** (print working directory) shows the full name of the working directory you are in.

<ls>
Command **ls** (list) displays all files in all folders of the working directory. You can also list hidden files with ls –a.

<cd>
Command **cd** (change directory) allows you to change directory. You can enter both the full path to the folder and its name. For example, to get to the Files folder in the /user/home/Files directory, type cd Files or cd /user/home/Files. To get into the root directory, type cd /.

<cp>
Command **cp** (copy) copies the file.
For example, cp file1 file2 will copy the contents of file1 to file2.
The cp file /home/files command will copy a file named file to the /home/files directory.

<mv>
Command **mv** (move) helps to move files. You can also rename files with mv. For example, we have a file file.txt. With the command mv file.txt new_file.txt we can move it to the same directory, but the file will already have a new name new_file.txt.

<rm>
Command **rm** (remove) deletes files and directories. For example, the rm file.txt command will delete the text file named file, and the rm -r Files command will delete the Files directory with all the files it contains.

<mkdir>

With **mkdir** (make directory) you can create a new directory. Thus, the mkdir directory command will create a new directory named directory in the current working directory.

<man>
Command **man** (manual) opens man pages with detailed information about the command. Type man followed by a space followed by the name of the command you want to learn more about. For example, man cp will display a man page for the cp command.

Step 1 – Install Docker and run Docker container

```
sudo apt install docker.io
sudo docker run -e LD_LIBRARY_PATH=/usr/local/lib -it ubuntu bash
```

Step 2 – Install Dependencies

```
apt update
apt install git
apt install cmake
apt install build-essential
apt install libgoogle-glog-dev
apt install wget
apt install unzip
```

Step 3 – Create and change to a convenient directory

```
mkdir testing
cd testing
```

Step 4 – Clone OpenCV repositories and commit version 4.6.0

```
git clone https://github.com/opencv/opencv.git
git clone https://github.com/opencv/opencv_contrib.git
cd opencv_contrib
git checkout 4.6.0
cd ../opencv
git checkout 4.6.0
```

Step 5 – Build and install OpenCV

```
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=RELEASE
-DCMAKE_INSTALL_PREFIX=/usr/local -
DOPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules ..
make
make install
cd ../..
```

Step 6 – Clone tclap library repository

```
git clone https://github.com/mirror/tclap.git
```

Step 7 – Clone project repository

git clone https://github.com/ChervyakovLM/FaceMetric.git
the project can also be downloaded and unpacked with the commands:
wget https://github.com/ChervyakovLM

Step 8 – Make an assembly

cd FaceMetric
(if the project was downloaded as an archive and unpacked, execute cd FaceMetric-1.0)
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=../Release
-DTCLAP_INCLUDE_DIR=/testing/tclap/include
-DFACE_API_ROOT_DIR=/testing/FaceMetric/CI/face_api_test -
DFREEIMAGE_ROOT_DIR=/testing/FaceMetric/CI/FreeImage ..
make
make install
cd ..

Build Flags
CMAKE_BUILD_TYPE - build type;
CMAKE_INSTALL_PREFIX – path to the installation directory (when executing the “make install” command, executable files will be placed in this directory);
TCLAP_INCLUDE_DIR - path to the TCLAP library directory;
FACE_API_ROOT_DIR - path to the directory with API examples;
FREEIMAGE_ROOT_DIR - path to the FREEIMAGE library directory.

Step 9 – The Release folder will appear in the FaceMetric project directory, in which executable files will be found for checking verification and identification.

cd Release

Step 10 – To obtain test data, you need to run the commands

wget https://github.com/ChervyakovLM/FaceMetric/releases/download/v1.0/identification.zip
wget https://github.com/ChervyakovLM/FaceMetric/releases/download/v1.0/verification.zip
unzip identification.zip
unzip verification.zip

Additional information on working with a docker container

Exiting a running container is done by executing the command in the console

exit

List containers

sudo docker ps -a

The container has the name IMAGE “ubuntu” and a CONTAINER_ID, for example, “6b23f3462581”, which is necessary to launch and work with the container.
To start the container you need to run the command

sudo docker start CONTAINER_ID

Logging into the container command line is done with the command

sudo docker exec -it CONTAINER_ID bash

Step 11 – To start verification, you need to execute the command

./checkFaceApi_V --split=./verification

Verification

Performing verification steps:
– extraction of biometric templates (do_extract)
– calculation of the degree of matching between biometric templates (do_match)
– calculation of TPR/FPR metrics (do_ROC)
By default, all stages are enabled, i.e. do_extract, do_match, do_ROC stage flags take the values true (or 1). In order to disable the execution of the stage when the program starts, set the flag to false (or 0).

Running all stages:
./checkFaceApi_V --split=./verification

Starting the biometric template extraction step:
./checkFaceApi_V --split=./verification --do_match=0 --do_ROC=0

Starting the stage of calculating the degree of correspondence between biometric templates:
./checkFaceApi_V --split=./verification --do_extract=0 --do_ROC=0

Starting the TPR/FPR metrics calculation step:
./checkFaceApi_V --split=./verification --do_extract=0 --do_match=0

checkFaceApi_V **has the following flags**:
– **split** – path to the directory with test data, required parameter
– **config** – path to the directory with FaceEngine configuration files, by default: input/config
– **extract_list** – path to the list of extracted files, by default: input/extract.txt
– **extract_prefix** – path to the directory with images, by default: input/images
– **grayscale** – open images as grayscale, default: false
– **count_proc** – number of used processor cores, by default: thread: hardware_concurrency()
– **extra_timings** – extended timing statistics, default: false
– **extract_info** – logging additional parameters of feature extraction, default: false
– **debug_info** – display debug information, default: false
– **desc_size** – descriptor size, default: 512
– **percentile** – time statistics control parameter in %, default: 90
– **do_extract** – stage of feature extraction from images, default: true
– **do_match** – stage of feature comparison with each other, default: true
– **do_ROC** – stage of calculation of ROC–curve points, by default: true

Step 12 – To start identification, you need to execute the command

./checkFaceApi_I --split=./identification

Identification

Performing identification steps:
– extraction of biometric templates (do_extract)
– building a search index (do_graph)
– search for close biometric templates (do_search)
– inserting biometric templates into the search index (do_insert)
– removal of biometric templates from the search index (do_remove)
– calculation of TPIR/FPIR metrics (do_tpir)
By default, all stages are enabled, i.e. do_extract, do_graph, do_insert, do_remove, do_search, do_tpir stage flags are set to true (or 1). In order to disable the execution of the stage when the program starts, set the flag to false (or 0).

Running all stages:
./checkFaceApi_I --split=./identification

Starting the biometric template extraction step:
./checkFaceApi_I --split=./identification --do_graph=0 --do_search=0 --do_insert=0 --do_remove=0 --do_tpir=0

Starting the stage of building a search index:
./checkFaceApi_I --split=./identification --do_extract=0 --do_search=0 --do_insert=0 --do_remove=0 --do_tpir=0

Launching the stage of searching for similar biometric templates:
./checkFaceApi_I --split=./identification --do_extract=0 --do_graph=0 --do_insert=0 --do_remove=0 --do_tpir=0

Launching the stage of inserting biometric templates into the search index:
./checkFaceApi_I --split=./identification --do_extract=0 --do_graph=0 --do_search=0 --do_remove=0 --do_tpir=0

Starting the step of removing biometric templates from the search index:
./checkFaceApi_I --split=./identification --do_extract=0 --do_graph=0 --do_search=0 --do_insert=0 --do_tpir=0

Starting the TPIR/FPIR metrics calculation step:
./checkFaceApi_I --split=./identification --do_extract=0 --do_graph=0 --do_search=0 --do_insert=0 --do_remove=0

checkFaceApi_I **has the following flags**:
– **split** – path to the directory with test data, required parameter
– **config** – path to the directory with FaceEngine configuration files, by default: input/config
– **db_list** – path to the database, list of indexes, by default: input/db.txt

– **mate_list** – a list of requests for persons that are in the database, by default: input/mate.txt
– **nonmate_list** – list of requests for persons who are not in the database, by default: input/nonmate.txt
– **insert_list** – list to be inserted into the database, by default: input/insert.txt
– **remove_list** – list to be removed from the database, by default: input/remove.txt
– **extract_prefix** – path to the directory with images, by default: input/images
– **grayscale** – open images as grayscale, default: false
– **count_proc** – number of processor cores used, by default: thread::hardware_concurrency()
– **extra_timings** – extended timing statistics, default: false
– **extract_info** – logging additional parameters of feature extraction, default: false
– **debug_info** – display debug information, default: false
– **desc_size** – descriptor size, default: 512
– **percentile** – time statistics control parameter in %, default: 90
– **nearest_count** – maximum number of candidates to search in the database, false, 100
– **search_info** – logging additional search results, default: false
– **do_extract** – stage of feature extraction from images, default: true
– **do_graph** – stage of converting image features into an index, by default: true
– **do_insert** – stage of adding to the index, by default: true
– **do_remove** – stage of removal from the index, by default: true
– **do_search** – index search stage, default: true
– **do_tpir** – identification metrics calculation stage, default: true

Example #1 FACEAPITEST: Interface.
To check the given biometric verification library, it is necessary to implement a class that inherits from FACEAPITEST: Interface.

```
class Interface {
public:
virtual ~Interface() {}
virtual ReturnStatus
initialize(const std::string &configDir) = 0;
virtual ReturnStatus
createTemplate(
const Multiface &faces,
TemplateRole role,
std::vector<uint8_t> &templ,std::vector<EyePair> &eyeCoordinates,
std::vector<double> &quality) = 0;
virtual ReturnStatus
matchTemplates(
const std::vector<uint8_t> &verifTemplate,
const std::vector<uint8_t> &initTemplate,
double &similarity) = 0;
virtual ReturnStatus
train(
const std::string &configDir,
const std::string &trainedConfigDir) = 0;
static std::shared_ptr<Interface>
```

```
getImplementation();
};
```

The inheritor class must contain the implementation of the following functions:

initialize – initialization of the algorithm for calculating biometric templates;
createTemplate – template calculation;
matchTemplates – template comparison;
train – additional adjustment of the algorithm for calculating biometric templates;
getImplementation – get a pointer to the implementation.

An example of the implementation of the successor class is given in the face_api_example_V.h and face_api_example_V.cpp files contained in the include and src directories, respectively.

Example #2 FACEAPITEST: IdentInterface.

To check the given biometric identification library, it is necessary to implement the class inherited from FACEAPITEST: IdentInterface.

```
class IdentInterface {
public:
virtual ~IdentInterface() {}
virtual ReturnStatus
initializeTemplateCreation(
const std::string &configDir,
TemplateRole role) = 0;
virtual ReturnStatus
createTemplate(
const Multiface &faces,
TemplateRole role,
std::vector<uint8_t> &templ,
std::vector<EyePair> &eyeCoordinates) = 0;
virtual ReturnStatus
finalizeInit(
const std::string &configDir,
const std::string &initDir,
const std::string &edbName,
const std::string &edbManifestName) = 0;virtual ReturnStatus
initializeIdentification(
const std::string &configDir,
const std::string &initDir) = 0;
virtual ReturnStatus
identifyTemplate(
const std::vector<uint8_t> &idTemplate,
const uint32_t candidateListLength,
std::vector<Candidate> &candidateList,
bool &decision) = 0;
virtual ReturnStatus
galleryInsertID(
```

```
const std::vector<uint8_t> &templ,
const std::string &id) = 0;
virtual ReturnStatus
galleryDeleteID(
const std::string &id) = 0;
static std::shared_ptr<IdentInterface>
getImplementation();
};
```

The inheritor class must contain the implementation of the following functions:
initializeTemplateCreation – initialization of the algorithm for calculating biometric templates;
createTemplate – template calculation;
finalizeInit – create an index from all templates;
initializeIdentification – initialization of the index search algorithm;
identifyTemplate – search by index;
galleryInsertID – adding a template to the index;
galleryDeleteID – removal of the template from the index;
getImplementation – get a pointer to the implementation.

An example of the implementation of the successor class is given in the face_api_example_I.h and face_api_example_I.cpp files contained in the include and src directories, respectively.