ИНСТРУКЦИЯ ИСПОЛЬЗОВАНИЯ БИБЛИОТЕКИ

Определения:

Верификация

Верификацию можно рассматривать как процесс принятия бинарного решения: «да» (два изображения принадлежат одному человеку), «нет» (на паре фотографий изображены разные люди). Прежде чем разбираться с метриками верификации, полезно понять, как мы можем классифицировать ошибки в подобных задачах. Учитывая, что есть 2 возможных ответа алгоритма и 2 варианта истинного положения вещей, всего возможно 4 исхода

Из этих исходов два соответствуют правильным ответам алгоритма, а два – ошибкам первого и второго рода соответственно. Ошибки первого рода называют «false accept», «false positive» или «false match» (неверно принято), а ошибки второго рода – «false reject», «false negative» или «false non–match» (неверно отвергнуто).

Просуммировав количество ошибок разного рода среди пар изображений в датасете и поделив их на количество пар, мы получим false accept rate (FAR) и false reject rate (FRR). В случае с системой контроля доступа «false positive» соответствует предоставлению доступа человеку, для которого этот доступ не предусмотрен, в то время как «false negative» означает, что система ошибочно отказала в доступе авторизованной персоне. Эти ошибки имеют разную стоимость с точки зрения бизнеса и поэтому рассматриваются отдельно. В примере с контролем доступа «false negative» приводит к тому, что сотруднику службы безопасности надо перепроверить пропуск сотрудника. Предоставление неавторизованного доступа потенциальному нарушителю (false positive) может привести к гораздо худшим последствиям.

Учитывая, что ошибки разного рода связаны с различными рисками, производители ПО для распознавания лиц зачастую дают возможность настроить алгоритм так, чтобы минимизировать один из типов ошибок. Для этого алгоритм возвращает не бинарное значение, а вещественное число, отражающее уверенность алгоритма в своем решении. В таком случае пользователь может самостоятельно выбрать порог и зафиксировать уровень ошибок на определенных значениях.

Идентификация

Идентификация или поиск человека среди набора изображений. Результаты поиска сортируются по уверенности алгоритма, и наиболее вероятные совпадения попадают в начало списка. В зависимости от того, присутствует или нет искомый человек в поисковой базе, идентификацию разделяют на две подкатегории: closed–set идентификация (известно, что искомый человек есть в базе) и open–set идентификация (искомого человека может не быть в базе).

Точность (accuracy) является надежной и понятной метрикой для closed–set идентификации. По сути, точность измеряет количество раз, когда нужная персона была среди результатов поиска.

Как это работает на практике? Давайте разбираться. Начнем с формулировки бизнес–требований. Допустим, у нас есть веб–страница, которая может разместить десять результатов поиска. Нам нужно измерить количество раз, которое искомый человек попадает в первые десять ответов алгоритма. Такое число называется Top–N точностью (в данном конкретном случае N равно 10).

Для каждого испытания мы определяем изображение человека, которого будем искать, и галерею, в которой будем искать, так, чтобы галерея содержала хотя бы еще одно изображение этого человека. Мы просматриваем первые десять результатов работы алгоритма поиска и проверяем, есть ли среди них искомый человек. Чтобы получить точность, следует просуммировать все испытания, в которых искомый человек был в результатах поиска, и поделить на общее число испытаний.

Биометрический шаблон – Данные, представляющие собой биометрические характеристики зарегистрированной личности.

Степень соответствия между биометрическими шаблонами – показывают степень соответствия между сравниваемыми шаблонами.

Метрики TPR/FPR:
При конвертации вещественного ответа алгоритма (как правило, вероятности принадлежности к классу, отдельно см. SVM) в бинарную метку, мы должны выбрать какой–либо порог, при котором 0 становится 1. Естественным и близким кажется порог, равный 0.5, но он не всегда оказывается оптимальным, например, при вышеупомянутом отсутствии баланса классов.
Одним из способов оценить модель в целом, не привязываясь к конкретному порогу, является AUC–ROC (или ROC AUC) — площадь (Area Under Curve) под кривой ошибок (Receiver Operating Characteristic curve). Данная кривая представляет из себя линию от (0,0) до (1,1) в координатах True Positive Rate (TPR) и False Positive Rate (FPR):

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

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

TPR нам уже известна, это полнота, а FPR показывает, какую долю из объектов negative класса алгоритм предсказал неверно. В идеальном случае, когда классификатор не делает ошибок (FPR = 0, TPR = 1) мы получим площадь под кривой, равную единице; в противном случае, когда классификатор случайно выдает вероятности классов, AUC–ROC будет стремиться к 0.5, так как классификатор будет выдавать одинаковое количество TP и FP.
Каждая точка на графике соответствует выбору некоторого порога. Площадь под кривой в данном случае показывает качество алгоритма (больше — лучше), кроме этого, важной является крутизна самой кривой — мы хотим максимизировать TPR, минимизируя FPR, а значит, наша кривая в идеале должна стремиться к точке (0,1).

Поисковой индекс – структура данных, которая содержит информацию о документах и используется в поисковых системах.

Метрики TPIR/FPIR:
True positive identification rate (TPIR) – вероятность истинно положительной идентификации/вероятность идентификации: ожидаемая доля идентификационных транзакции пользователей, зарегистрированных в системе, в результате которых корректный идентификатор пользователя будет присутствовать среди возвращаемых t идентификаторов. Если выходными данными биометрической системы являются t ближайших кандидатов–совпадений, соответствующая оценка TPIR также известна как t–ранг идентификации.
False positive identification rate (FPIR) – вероятность ложноположительной идентификации: ожидаемая доля идентификационных транзакции пользователей, НЕ зарегистрированных в системе, в результате которых возвращается идентификатор. Это означает, что мошенник, даже не зарегистрированный в системе, отправив свои биометрические данные на обработку, получает от системы положительный ответ–допуск к объекту. Тогда как в данной ситуации ожидаемой реакцией системы не предполагается возвращение никакого ближайшего

кандидата. Это своего рода аналог ошибки второго рода для процедуры верификации. Ошибка FPIR зависит как от числа зарегистрированных пользователей (N), так и от величины порога (n), по которому определяется допустимая величина меры близости между кандидатом и шаблонами базы. Кроме того, на замкнутом множестве невозможно определить FPIR для процедуры идентификации, т.к. все пользователи системы являются зарегистрированными в системе.
False negative identification rate (FNIR) – вероятность ложноотрицательной идентификации: ожидаемая доля идентификационных транзакции пользователей, зарегистрированных в системе, в результате которых корректный идентификатор пользователя НЕ будет присутствовать среди возвращаемых t идентификаторов. FNIR зависит от числа зарегистрированных пользователей (N), от величины порога (n), по которому определяется допустимая величина меры близости между кандидатом и шаблонами базы, а также от количества возвращаемых кандидатов – от ранга идентификации.
FPIR определяется, когда входной образец ошибочно совпадает с одним или несколькими шаблонами из базы. Тогда данная ошибка вычисляется как единица минус вероятность того, что не произошло совпадений ни с одним из шаблонов базы (N – количество зарегистрированных шаблонов в базе). В случае, если FMR является малой величиной, ошибка FPIR может быть аппроксимирована.

Дескриптор – лексическая единица (слово, словосочетание) информационно–поискового языка, служащая для описания основного смыслового содержания документа или формулировки запроса при поиске документа (информации) в информационно–поисковой системе. Дескриптор однозначно ставится в соответствие группе ключевых слов естественного языка, отобранных из текста, относящегося к определённой области знаний.

ROC–кривая – график, позволяющий оценить качество бинарной классификации, отображает соотношение между долей объектов от общего количества носителей признака, верно классифицированных как несущие признак (true positive rate, TPR, называемой чувствительностью алгоритма классификации), и долей объектов от общего количества объектов, не несущих признака, ошибочно классифицированных как несущие признак (false positive rate, FPR, величина 1–FPR называется специфичностью алгоритма классификации) при варьировании порога решающего правила. Также известна как кривая ошибок.

Docker – это популярная программа, в основе которой лежит технология контейнеризации. Docker позволяет запускать Docker–контейнеры с приложениями из заранее заготовленных шаблонов — Docker–образов (или по–другому Docker images).

Контейнеризация — это технология, которая помогает запускать приложения изолированно от операционной системы. Приложение как бы упаковывается в специальную оболочку – контейнер, внутри которого находится среда, необходимая для работы.

ШАГ 0 – Открыть терминал

В Ubuntu консоль запускается при загрузке системы. Терминал – это тоже консоль, но уже в графической оболочке. Его можно запустить, набрав слово Терминал в поисковой строке ОС, или через комбинацию клавиш **Ctrl+Alt+T**.

В общем виде в Ubuntu команды имеют такой вид:

<программа – ключ значение>

Программа – это сам исполняемый файл. Другими словами, это программа, которая будет выполняться по команде.

Ключ – обычно у каждой программы свой набор ключей. Их можно найти в мануале к программе.
Значение – параметры программы: цифры, буквы, символы, переменные.

Напомним, что для выполнения команды нужно ввести её в командную строку – Ubuntu console или эмулирующий работу консоли терминал.

Рассмотрим основные команды консоли Ubuntu:

<sudo>
Промежуточная команда **sudo (SuperUser DO – суперпользователь)** позволяет запускать программы от имени администратора или root–пользователя. Вы можете добавить sudo перед любой командой, чтобы запустить её от имени суперпользователя.

<apt>
Команда **apt** используется для работы с программными пакетами для установки программных пакетов (sudo apt install имя–пакета), обновления репозитория с пакетами (sudo apt update) и обновления пакетов, которые установлены в систему (sudo apt upgrade).

<pwd>
Команда **pwd (print working directory – вывести рабочую директорию)** показывает полное имя рабочей директории, в которой вы находитесь.

<ls>
Команда **ls (list – список)** выводит все файлы во всех папках рабочей директории. С помощью ls –a можно вывести и скрытые файлы.

<cd>
Команда **cd (change directory – изменить директорию)** позволяет перейти в другую директорию. Можно ввести как полный путь до папки, так и её название. Например, чтобы попасть в папку Files, лежащую в директории /user/home/Files, введите cd Files или cd /user/home/Files. Чтобы попасть в корневую директорию, введите cd /.

<cp>
Команда **cp (copy – копировать)** копирует файл. Например, cp file1 file2 скопирует содержимого файла file1 в file2. Команда cp file /home/files скопирует файл с названием file в директорию /home/files.

<mv>
Команда **mv (move – переместить)** помогает перемещать файлы. Также с помощью mv можно переименовывать файлы. Например, у нас есть файл file.txt. С помощью команды mv file.txt new_file.txt мы можем перенести его в ту же директорию, но у файла уже будет новое название new_file.txt.

<rm>
Команда **rm (remove – удалить)** удаляет файлы и каталоги. Так, команда rm file.txt удалит текстовый файл с названием file, а команда rm –r Files удалит директорию Files со всеми содержащимися в ней файлами.

<mkdir>
С помощью **mkdir (make directory – создать директорию)** можно создать новую директорию. Так, команда mkdir directory создаст новую директорию с именем directory в текущей рабочей директории.

<man>
Команда **man (manual – мануал)** открывает справочные страницы с подробной информацией о команде. Введите man, а затем через пробел название команды, о которой вы хотите узнать подробнее. Например, man cp выведет справочную страницу о команде cp.

ШАГ 1 – Установить Docker и запустить Docker–контейнер

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

ШАГ 2 – Установить зависимости

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

ШАГ 3 – Создать и перейти в удобный каталог

mkdir testing
cd testing

ШАГ 4 – Склонировать репозитории OpenCV и перейти на коммит версии 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

ШАГ 5 – Произвести сборку и установку 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 ../..

ШАГ 6 – Склонировать репозиторий библиотеки tclap

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

ШАГ 7 – Склонировать репозиторий проекта

git clone https://github.com/ChervyakovLM/FaceMetric.git
также проект можно скачать и распаковать командами:
wget https://github.com/ChervyakovLM

ШАГ 8 – Произвести сборку

cd FaceMetric
(если проект скачали архивом и распаковали выполнить 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 ..
```

Флаги сборки
CMAKE_BUILD_TYPE – тип сборки;
CMAKE_INSTALL_PREFIX – путь до каталога установки (при выполнении команды «make install» в этот каталог будут помещены исполняемые файлы);
TCLAP_INCLUDE_DIR – путь до каталога библиотеки TCLAP;
FACE_API_ROOT_DIR –путь до каталога с примерами API;
FREEIMAGE_ROOT_DIR – путь до каталога библиотеки FREEIMAGE.

ШАГ 9 – В каталоге проекта FaceMetric появится папка Release, в которой будут находить исполняемые файлы для проверки верификации и идентификации.

```
cd Release
```

ШАГ 10 – Для получения тестовых данных необходимо выполнить команды
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

Дополнительная информация по работе с докер-котейнером
Выход из запущенного контейнера осуществляется выполнением в консоли команды

```
exit
```

Вывести список контейнеров

```
sudo docker ps -a
```

Контейнер имеет название IMAGE “ubuntu” и идентификатор CONTAINER_ID, например, «6b23f3462581», который необходим для запуска и работы с контейнером.
Для запуска контейнера необходимо выполнить команду

```
sudo docker start CONTAINER_ID
```

Вход в командную строку контейнера выполняется командой

```
sudo docker exec -it CONTAINER_ID bash
```

ШАГ 11 – Для запуска верификации необходимо выполнить команду

```
./checkFaceApi_V --split=./verification
```

Верификация
Выполнение этапов верификации:
– извлечение биометрических шаблонов (do_extract)
– вычисление степени соответствия между биометрическими шаблонами (do_match)
– расчет метрик TPR/FPR (do_ROC)
По умолчанию все этапы включены, т.е. флаги этапов do_extract, do_match, do_ROC принимают значения true (или 1). Для того, чтобы отключить выполнение этапа при запуске программы в флаге указать значение false (или 0).

Запуск всех этапов:
./checkFaceApi_V --split=./verification

Запуск этапа извлечения биометрических шаблонов:
./checkFaceApi_V --split=./verification --do_match=0 --do_ROC=0

Запуск этапа вычисление степени соответствия между биометрическими шаблонам:
./checkFaceApi_V --split=./verification --do_extract=0 --do_ROC=0

Запуск этапа расчета метрик TPR/FPR:
./checkFaceApi_V --split=./verification --do_extract=0 --do_match=0

checkFaceApi_V **имеет следующие флаги**:
– **split** – путь до каталога с тестовыми данными, необходимый параметр
– **config** – путь до каталога с конфигурационными файлами FaceEngine, по–умолчанию: input/config
– **extract_list** – путь до списка извлечённых файлов, по–умолчанию: input/extract.txt
– **extract_prefix** – путь до каталога с изображениями, по–умолчанию: input/images
– **grayscale** – открывать изображения как оттенки серого, по–умолчанию: false
– **count_proc** – число используемых ядер процессора, по–умолчанию: thread::hardware_concurrency()
– **extra_timings** – расширенная временная статистика, по–умолчанию: false
– **extract_info** – логирование дополнительных параметров экстракции признаков, по–умолчанию: false
– **debug_info** – вывод отладочной информации, по–умолчанию: false
– **desc_size** – размер дескриптора, по–умолчанию: 512
– **percentile** – параметр управления временной статистики в %, по–умолчанию: 90
– **do_extract** – стадия извлечения признаков из изображений, по–умолчанию: true
– **do_match** – стадия сравнения признаков друг с другом, по–умолчанию: true
– **do_ROC** – стадия расчёта точек ROC–кривой, по–умолчанию: true

ШАГ 12 – Для запуска идентификации необходимо выполнить команду

./checkFaceApi_I --split=./identification

Идентификация
Идентификация
Выполнение этапов идентификации:
– извлечение биометрических шаблонов (do_extract)
– построение поискового индекса (do_graph)
– поиск близких биометрических шаблонов (do_search)
– вставка биометрических шаблонов в поисковый индекс (do_insert)
– удаление биометрических шаблонов из поискового индекса (do_remove)
– расчет метрик TPIR/FPIR (do_tpir)
По умолчанию все этапы включены, т.е. флаги этапов do_extract, do_graph, do_insert, do_remove, do_search, do_tpir принимают значения true (или 1). Для того, чтобы отключить выполнение этапа при запуске программы в флаге указать значение false (или 0).
Для выполнения этапа необходимо завершение предыдущего этапа.
Перед выполнением построения поискового индекса (do_graph) необходимо провести извлечение биометрических шаблонов (do_extract).
Перед выполнением вставки биометрических шаблонов в поисковый индекс (do_insert), удаления биометрических шаблонов из поискового индекса (do_remove) и поиска близких

биометрических шаблонов (do_search) необходимо выполнение построения поискового индекса (do_graph).
Перед выполнением расчета метрик TPIR/FPIR (do_tpir) необходимо выполнение поиска близких биометрических шаблонов (do_search).

Запуск всех этапов:
./checkFaceApi_I --split=./identification

Запуск этапа извлечения биометрических шаблонов:
./checkFaceApi_I --split=./identification --do_graph=0 --do_search=0 --do_insert=0 --do_remove=0 --do_tpir=0

Запуск этапа построения поискового индекса:
./checkFaceApi_I --split=./identification --do_extract=0 --do_search=0 --do_insert=0 --do_remove=0 --do_tpir=0

Запуск этапа поиска близких биометрических шаблонов:
./checkFaceApi_I --split=./identification --do_extract=0 --do_graph=0 --do_insert=0 --do_remove=0 --do_tpir=0

Запуск этапа вставки биометрических шаблонов в поисковый индекс:
./checkFaceApi_I --split=./identification --do_extract=0 --do_graph=0 --do_search=0 --do_remove=0 --do_tpir=0

Запуск этапа удаления биометрических шаблонов из поискового индекса:
./checkFaceApi_I --split=./identification --do_extract=0 --do_graph=0 --do_search=0 --do_insert=0 --do_tpir=0

Запуск этапа расчета метрик TPIR/FPIR:
./checkFaceApi_I --split=./identification --do_extract=0 --do_graph=0 --do_search=0 --do_insert=0 --do_remove=0

checkFaceApi_I **имеет следующие флаги**:
– **split** – путь до каталога с тестовыми данными, необходимый параметр
– **config** – путь до каталога с конфигурационными файлами FaceEngine, по–умолчанию: input/config
– **db_list** – путь до базы данных, списка индексов, по–умолчанию: input/db.txt
– **mate_list** – список запросов для лиц, которые есть в базе, по–умолчанию: input/mate.txt
– **nonmate_list** – список запросов для лиц которых нет в базе, по–умолчанию: input/nonmate.txt
– **insert_list** – список для вставки в базу, по–умолчанию: input/insert.txt
– **remove_list** – список для удалении из базы, по–умолчанию: input/remove.txt
– **extract_prefix** – путь до каталога с изображениями, по–умолчанию: input/images
– **grayscale** – открывать изображения как оттенки серого, по–умолчанию: false
– **count_proc** – число используемых ядер процессора, по–умолчанию: thread::hardware_concurrency()
– **extra_timings** – расширенная временная статистика, по–умолчанию: false
– **extract_info** – логирование дополнительных параметров экстракции признаков, по–умолчанию: false
– **debug_info** – вывод отладочной информации, по–умолчанию: false
– **desc_size** – размер дескриптора, по–умолчанию: 512
– **percentile** – параметр управления временной статистики в %, по–умолчанию: 90

– **nearest_count** – максимальное количество кандидатов для поиска в базе, false, 100
– **search_info** – логирование дополнительных результатов поиска, по–умолчанию: false
– **do_extract** – стадия извлечения признаков из изображений, по–умолчанию: true
– **do_graph** – стадия преобразования признаков изображения в индекс, по–умолчанию: true
– **do_insert** – стадия добавления в индекс, по–умолчанию: true
– **do_remove** – стадия удаления из индекса, по–умолчанию: true
– **do_search** – стадия поиска в индексе, по–умолчанию: true
– **do_tpir** – стадия расчёта метрик идентификации, по–умолчанию: true

Пример №1 FACEAPITEST: Interface
Для проверки заданной библиотеки биометрической верификации необходимо реализовать класс наследник от 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();
};
```

Класс наследник должен содержать реализацию следующих функций:
initialize – инициализация алгоритма вычисления биометрических шаблонов;
createTemplate – вычисление шаблона;
matchTemplates – сравнение шаблонов;
train – донастройка алгоритма вычисления биометрических шаблонов;
getImplementation – получение указателя на реализацию.

Пример реализации класса наследника приведён в файлах face_api_example_V.h и face_api_example_V.cpp, содержащиеся в каталогах include и src соответственно.

Пример №2 FACEAPITEST: IdentInterface
Для проверки заданной библиотеки биометрической идентификации необходимо реализовать класс наследник от 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();
};
```

Класс наследник должен содержать реализацию следующих функций:

initializeTemplateCreation – инициализация алгоритма вычисления биометрических шаблонов;
createTemplate – вычисление шаблона;
finalizeInit – создание индекса из всех шаблонов;
initializeIdentification – инициализация алгоритма поиска по индексу;
identifyTemplate – поиск по индексу;
galleryInsertID – добавление шаблона в индекс;
galleryDeleteID – удаление шаблона из индекса;
getImplementation – получение указателя на реализацию.

Пример реализации класса наследника приведён в файлах face_api_example_I.h и face_api_example_I.cpp, содержащиеся в каталогах include и src соответственно.