VisionLabs
MACHINES CAN SEE

VisionLabs LUNA SDK Mobile Android

Инструкция по эксплуатации

(краткое руководство разработчика)

ООО «ВижнЛабс»

+7 (499) 399 3361

info@visionlabs.ru

Содержание

Введение

ПО VisionLabs LUNA SDK Mobile Android представляет собой набор средств разработки, включающий библиотеки и нейронные сети для анализа изображений и работы с биометрическими образцами, который позволяет специалистам по разработке программного обеспечения внедрять механизмы распознавания лиц и людей в ПО для различных устройств

Эта инструкция по эксплуатации описывает основные понятия набора средств разработки VisionLabs LUNA SDK Mobile Android, демонстрирует основные функции и варианты их использования.

1 Основные понятия

Добро пожаловать в VisionLabs LUNA SDK Mobile Android!

Данный документ:

- Описывает идеи, лежащие в основе управления ресурсами, и дает представление о том, почему было принято то или иное решение. Зная это, вы сможете написать эффективный код с VisionLabs LUNA SDK Mobile Android;

- Разбирает процесс обработки и распознавания лица, и демонстрирует, как один процесс влияет на другой. Это поможет настроить VisionLabs LUNA SDK Mobile Android под ваши требования, что намного продуктивнее слепого следования подсказкам;

- Выделяет важные моменты и пропускает очевидное. Следовательно, вы получаете только необходимую информацию.

1.1 Общие интерфейсы и типы

1.1.1 Интерфейс подсчета ссылок

Интерфейс IRefCounted предоставляет методы для доступа к счетчику ссылок, инкремента и декремента. Все объекты с подсчетом ссылок подразумевают пользовательскую модель управления памятью. Таким образом, объекты поддерживают автоматическое уничтожение, когда счётчик ссылки снижается до нуля, также поддерживаются более сложные стратегии частичного уничтожения и слабые ссылки, необходимые для внутренних необходимостей VisionLabs LUNA SDK Mobile Android. Допустимый минимум этих функций доступен пользователю, позволяя:

- уведомить объект о том, что он необходим, сохраняя ссылку на него;
- уведомить объект, что он больше не требуется, освободив ссылку на него;
- получить фактическое значение счетчика ссылок.

Примечание. Объекты с подсчетом ссылок также требуют особой обработки. Никогда не вызывайте команду delete для любого указателя на объект, полученного из IRefCounting! Это приведёт к повреждению динамической памяти. Команда release уведомляет систему, когда объект должен быть уничтожен, и система уничтожает его должным образом.

Однако не рекомендуется взаимодействовать с механизмом подсчета ссылок вручную, так как это может привести к ошибкам. Вместо этого настоятельно рекомендуется использовать умные указатели, которые предоставляются VisionLabs LUNA SDK Mobile Android и специально созданы для обработки таких объектов. Подробнее см. раздел "Автоматический подсчёт ссылок".

1.1.2 Автоматический подсчёт ссылок

Для Вашего удобства предусмотрен утилитарный класс умного указателя Ref. Он автоматически увеличивает счётчик ссылок при создании новой ссылки и автоматически уменьшает его при уничтожении ссылки. Также этот класс отслеживает, чтобы внутренний обычный указатель оставался не нулевым, тем самым предотвращая ошибки.

Примечание: Ref <> всегда увеличивает счетчик ссылок на 1 во время инициализации. В некоторых случаях первичной инициализации подобное поведение может быть неожиданным. Рассмотрим простой пример:

```
    ISomeObject* createSomeObject();

    {

    /* Здесь функция createSomeObject возвращает объект с начальным количеством ссылок
равным 1 (в противном случае, он будет "мёртвым"). Затем Ref добавляет себе еще одну
ссылку, что в сумме даёт 2!
    */

    Ref<ISomeObject> objref = createSomeObject();
    /* Здесь мы используем объект по своему усмотрению, ожидая, что он будет должным
образом уничтожен, когда данная область будет покинута.
    */

    }

    /* Здесь мы покинули область действия, и Ref был автоматически уничтожен, как и
любой другой объект, созданный в стеке. В то же время Ref уменьшил счёт ссылок своего
внутреннего объекта, снова вернувшись к 1.
    */
```

Однако, объект не уничтожается автоматически. Для этого у него должно быть ровно 0 ссылок. Более того, в этом примере теряется обычный указатель на объект, поэтому исправить его никак нельзя, и возникает утечка памяти.

Имея это в виду, вводится концепция “захвата права владения". Захватывая объект, вы говорите, что его обычный указатель не будет использоваться, и для этого нужен только подходящий Ref. Для приобретения права владения используйте специальную функцию ::acquire(). Исправленная версия приведенного выше примера будет выглядеть следующим образом:

```
    ISomeObject* createSomeObject();

    {

    /* Здесь функция createSomeObject возвращает объект с начальным счётчиком ссылок
равным 1 (в противном случае, он будет "мёртвым"). Затем мы захватываем его, оставляя
счётчик ссылок равным 1.
    */
    Ref<ISomeObject> objref = acquire(createSomeObject());
    /* Здесь мы используем объект по своему усмотрению.
    */

    }

    /* Здесь мы оставили область действия, и Ref был автоматически уничтожен, как и
любой другой объект, созданный в стеке. В то же время Ref уменьшил количество ссылок
внутреннего объекта на 1, приравняв его к 0. Объект должным образом уничтожен
объектной системой.
    */
```

Примечание: не храните и не используйте обычные указатели на объект при использовании функции ::acquire (), так как захват права владения делает их недействительными.

Для создания ссылки на существующий обычный указатель можно использовать функцию ::make_ref(), которая похожа на функцию ::acquire().

Вы можете статически преобразовать тип объекта во время получения права владения или ссылки. Чтобы добиться этого, используйте специальные версии функций ::make_ref_as() и ::acquire_as(). Вы обязаны убедиться, что такое преобразование возможно.

Для подробной информации о доступных методах и функциях, пожалуйста, обратитесь к справочному руководству FaceEngine.

Обратите внимание, что typedefs объявлены для Ref для всех типов с автоматическим подсчётом ссылок. Все они соответствуют следующему соглашению о присвоении имён: InterfaceNamePtr. Например, Ref<IDetector> эквивалентно IDetectorPtr.

1.1.3 Интерфейс сериализуемого объекта

Данный интерфейс представляет объект. Содержимое объекта может быть сериализовано в некоторый поток данных, и также может быть десериализовано. Это можно рассматривать как загрузку и сохранение.

Для взаимодействия с вышеупомянутым потоком данных сериализуемому объекту требуется адаптер, предоставленный пользователю. Такой адаптер называется Archive. Подробное описание см. в разделе "Интерфейс Archive" в главе "Основные модули".

Сериализуемые интерфейсы: Descriptor, IDescriptorBatch.

1.1.4 Вспомогательные типы

1.1.4.1 Тип изображения

Поскольку VisionLabs LUNA SDK Mobile Android является набором библиотек машинного зрения, естественным будет применение некоторой концепции изображения. Таким образом, существует класс Image. Он создан как класс-контейнер с подсчетом ссылок для необработанных данных цвета пикселей.

Подсчет ссылок позволяет нескольким объектам совместно использовать одно изображение. Однако следует понимать, что каждый объект Image содержит ссылку на некоторые данные, поэтому любое изменение данных влияет на все другие объекты, содержащие ту же ссылку. Для детального копирования Image, следует использовать метод clone(), т.к. оператор присвоения просто создаёт ссылку. Также можно вырезать часть изображения и вставить в новое изображение с помощью метода extract().

Данные об элементе изображения можно охарактеризовать расположением цветовых каналов, т. е. количеством цветовых каналов и их порядком. Движок определяет для этого структуру Format. Структура Format определяет:

- Количество цветовых каналов (например, RGB или Grayscale);
- Порядок цветового канала (например, RGB против BGR).

VisionLabs LUNA SDK Mobile Android предполагает 8 бит (то есть 1 байт) для каждого цветового канала и реализует 8-битовые оттенки серого, 24-битные RGB/BGR и планшетные 32-битные форматы.

Для Вашего удобства предусмотрены функции преобразования формата; см. семейство функций convert().

Класс Image поддерживает отображение диапазона данных. Для чтения или записи подмножество байтов можно отобразить в прямоугольной области. Отображенные пиксели представлены структурой SubImage. В отличие от Image, SubImage является просто представлением данных и не считается ссылкой. Вы не должны хранить SubImages дольше, чем это необходимо для завершения изменения данных. Подробности смотрите в документации по семейству функций map().

Поддержка ввода-вывода в форматах OOM, JPEG, PNG и TIFF посредством библиотеки FreeImage library.

Отсутствие ввода-вывода изображения продиктовано тем, что VisionLabs LUNA SDK Mobile Android фокусируется на легкости и минимизирует количество внешних зависимостей. Он не предназначен исключительно для обработки изображений. То есть, можно рассматривать видеокадры как Images и обрабатывать их один за другим. В этом случае требуется внешний видеокодек (возможно, проприетарный).

1.2 Бета-Режим

Некоторые функции в LUNA SDK доступны только в бета-режиме. Данные функции экспериментальные, поэтому могут быть нестабильными. Если вы хотите использовать их, активируйте betaMode в настройках (faceengine.conf).

2 Обзор Структуры VisionLabs LUNA SDK Mobile Android

VisionLabs LUNA SDK Mobile Android подразделяется на несколько модулей. Каждый модуль посвящен одной функции. Ниже приведен список всех модулей с кратким описанием их функциональности. Подробную информацию можно найти в соответствующих главах данного руководства.

2.1 Список модулей VisionLabs LUNA SDK Mobile Android:

VisionLabs LUNA SDK Mobile Android содержит следующие модули:

Модуль		Функция
TrackEngine	Трекинг	Инструмент для обнаружения лица и отслеживания нескольких источников. Позволяет выбрать наиболее подходящие неподвижные изображения для распознавания лиц из последовательности видеокадров.
FaceEngine	Основной модуль (CORE)	Используется для создания экземпляров объектов в других модулях, интерфейс которых открыт для разработчиков.
	Модуль детекции	Используется для обнаружения лиц в кадре, выбора лучшей детекции (bestshot) и определения ключевых точек лица.
	Модуль выравнивания лиц	Используется для нормализации изображения: компенсация поворота плоскости изображения, центрирования и обрезки изображения.
	Модуль определения параметров	Используется для оценки различных свойств изображения или изображаемого объекта. Эти свойства могут использоваться для повышения точности алгоритмов или для выполнения пользовательских задач.
	Модуль работы с биометрическими шаблонами	Используется для извлечения и сопоставления биометрических шаблонов с целью верификации и идентификации лиц.
LivenessEngine	Модуль проверки Liveness	Разработан как набор алгоритмов, призванных по единичному изображению либо последовательности изображений подтвердить витальность (жизнеспособность) человека по одному или нескольким изображениям.

Таким образом, каждый модуль представляет собой набор классов, посвященных какой-то конкретной области. Разделы независимы друг от друга, за несколькими исключениями, например, все разделы более высокого уровня зависят от основного раздела. Зависимости между модулями подробно описаны в соответствующих главах руководства.

Применение функций из этих модулей в указанном порядке позволяет создать конвейер со значительной степенью гибкости, который включает обнаружение, анализ, распознавание и сравнение лиц. Этот конвейер подробно описывается в следующих главах.

3 Основной модуль (CORE)

3.1 Общие интерфейсы

3.1.1 Объект Face Engine

Объект FaceEngine является корневым объектом всего VisionLabs LUNA SDK Mobile Android. С него всё начинается, поэтому необходимо создать хотя бы один его экземпляр. Хотя существует возможность создать несколько экземпляров FaceEngine, это нецелесообразно (см. объяснение в разделе «Автоматический подсчет ссылок» в главе «Ключевые концепции»).

Для создания экземпляра Face Engine вызовите функцию *createFaceEngine*. Кроме того, можно указать *dataPath* и *configPath* по умолчанию в параметрах *createFaceEngine*.

Примечание: *если вы планируете использовать GPU ускорение, вам следует учитывать время инициализации и завершения работы CUDA. В частности, CUDA создает глобальный динамический объект с неявным временем существования (см. http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#initialization).*

Для предотвращения несоответствия времени работы и времени существования FaceEngine, рекомендуется избегать создания статических глобальных экземпляров объектов FaceEngine, так как порядок их уничтожения не определен.

3.1.2 Поставщик параметров

Поставщик параметров — это особая сущность, загружающая параметры из разных расположений. Поскольку параметры могут совместно использоваться несколькими объектами, полезно их кэшировать, чтобы свести к минимуму чтение с диска и предоставить словарный интерфейс для поиска именованных значений.

Объект поставщика действует отдельно от структуры модуля FaceEngine и создается отдельной функцией фабрики createSettingsProvider. Эта функция принимает в качестве параметра путь к файлу конфигурации (см. руководство "Руководство по настройке"). По умолчанию, engine содержит один экземпляр поставщика для всех модулей. Его можно рассматривать как файл конфигурации для подсчета ссылок. Этот поставщик передается объектом Face Engine каждой фабрике, которую он создает. Фабрика, в свою очередь, может считывать данные конфигурации из этого объекта и передавать их в дочерние объекты. В стандартных сценариях не стоит беспокоиться о поставщиках, так как engine делает всё за вас. Однако, когда используются пользовательские параметры создания фабрики (см. Описание CreateFactoryFlags в "Объект Face Engine"), необходимо создавать и добавлять поставщика вручную, когда это требуется.

3.2 Вспомогательные Интерфейсы

3.2.1 Интерфейс Archive

Интерфейс Archive используется для предоставления функций сериализации совместно с источником данных. Он содержит методы преимущественно для чтения и записи данных. Обратите внимание, что *IArchive* не является унаследованным от *IRefCounted*, поэтому не подразумевает каких-либо особых стратегий управления памятью.

Аспекты, о которых следует помнить при применении архива:

- Объекты FaceEngine, использующие *IArchive* для сериализации, вызывают только функцию записи *write()* (во время сохранения) или чтения *read()* (во время загрузки), но никогда не используются одновременно, если иное не указано явно;
- Во время сохранения или загрузки объекты FaceEngine могут свободно записывать или читать свои данные пакетами. Например, может быть выполнено несколько последовательных вызовов *write()* в рамках одного запроса сериализации. Это также верно для *read()*. В основном функции *read()* и *write()* должны вести себя как функции стандартной библиотеки C: *fread()* и *fwrite()*.

Поскольку эти методы интерфейса довольно очевидны и обычно не требуют пояснений, для получения более подробной информации мы советуем ознакомиться со Справочным руководством.

3.3 Путь к данным

3.3.1 Данные модели

Различные модули VisionLabs LUNA SDK Mobile Android могут потребовать файлы данных для работы. Эти файлы содержат различные модели алгоритмов и константы, используемые в процессе работы. Все файлы собраны в единый каталог данных. Предполагается, что каталог данных находится:

- На Linux: /opt/visionlabs/data;
- На Windows: ./ data.

Расположение каталога данных можно переопределить при помощи метода *setDataDirectory()*, доступного в *IFaceEngine*. Текущее расположение данных может быть получено методом *getDataDirectory()*.

3.3.2 Конфигурационные данные

Файл конфигурации называется *faceengine.conf* и по умолчанию хранится в каталоге */data*. Файл *ConfigurationGuide.pdf*, содержащий описания параметров и значения по умолчанию, находится в директории */doc* комплекта поставки.

В процессе работы данные файла конфигурации управляются специальным объектом, который реализует интерфейс *ISettingsProvider* (см. "Поставщик параметров"). Экземпляр поставщика создаётся при помощи функции *createSettingsProvider()*, которая принимает расположение файла конфигурации в качестве параметра или использует вышеупомянутые значения по умолчанию, если расположение файла не указано.

Для любого объекта фабрики можно задать отдельную конфигурацию с помощью метода *setSettingsProvider()*, который доступен в каждом интерфейсе объекта фабрики, включая *IFaceEngine*. На данный момент поставщик связанных параметров может быть получен с помощью метода *getSettingsProvider()*.

4 Трекинг

4.1 Основные понятия

Трек - информация о положении лица одного человека в последовательности кадров.

Трекинг - функция, которая следует за объектом (лицом) в последовательности кадров.

Лучший снимок - изображение, подходящее для распознавания лиц.

4.2 Работа с TrackEngine

TrackEngine основан на методах распознавания и анализа лиц, предоставляемых библиотекой FaceEngine.

Чтобы создать экземпляр TrackEngine, используйте следующие глобальные заводские функции:

ITrackEngine tsdk::createTrackEngine(fsdk::IFaceEngine engine, const char configPath, vsdk::IVehicleEngine vehicleEngine = nullptr, const fsdk::LaunchOptions *launchOptions = nullptr)

- engine - указатель на экземпляр FaceEngine (должен быть уже инициализирован)
- configPath - путь к файлу конфигурации TrackEngine
- vehicleEngine - указатель на объект VehicleEngine (если включена логика транспортного средства)
- launchOptions - параметры запуска функций sdk
- return value - указатель на ITrackEngine

ITrackEngine tsdk::createTrackEngine(fsdk::IFaceEngine engine, const fsdk::ISettingsProviderPtr& provider, vsdk::IVehicleEngine vehicleEngine = nullptr, const fsdk::LaunchOptions launchOptions = nullptr)

- engine - указатель на экземпляр FaceEngine (должен быть уже инициализирован)
- provider - поставщик настроек с конфигурацией TrackEngine
- vehicleEngine - указатель на объект VehicleEngine (если включена логика транспортного средства)
- launchOptions - параметры запуска функций sdk
- return value - указатель на ITrackEngine

Не рекомендуется создавать несколько экземпляров TrackEngine в одном приложении.

Основным интерфейсом для TrackEngine является поток (stream) - объект, в который вы отправляете видеокадры. Для создания потока используйте следующий метод TrackEngine:

IStream* ITrackEngine::createStream()

- return value - указатель на IStream

При необходимости вы можете создать несколько потоков одновременно (в тех случаях, когда вы хотите отслеживать лица с нескольких камер). В каждом потоке engine обнаруживает лица и строит их треки. Каждая трек лица имеет свой собственный уникальный идентификатор. Поэтому можно сгруппировать изображения лиц, принадлежащие одному и тому же человеку, с их идентификаторами треков. Пожалуйста, обратите внимание, что треки могут время от времени прерываться либо из-за людей, покидающих видимую зону, либо из-за сложных условий обнаружения (низкое качество изображения, окклюзии, экстремальные позы головы и т.д.).

Кадры представляются один за другим, и каждый кадр имеет свой собственный уникальный идентификатор.

bool IStream::pushFrame(const fsdk::Image &frame, uint32_t frameId, tsdk::AdditionalFrameData *data) - Помещает одиночный кадр в буфер.

- frame - входное изображение кадра. Формат должен быть R8G8B8 OR R8G8B8X8.
- frameId - уникальный идентификатор последовательности кадров.
- data - - это любые дополнительные данные, которые разработчик хочет получить при реализации callback-функций. Не используйте оператор удаления. Для этого параметра внутри TrackEngine реализован сборщик мусора (gc).
- return value - Значение true, если кадр был добавлен в очередь на обработку, иначе false - кадр был пропущен из-за переполнения очереди.

Также есть несколько разновидностей этого метода: *pushCustomFrame, pushFrameWaitFor, pushCustomFrameWaitFor.*

TrackEngine выдает различные события, чтобы информировать вас о том, что происходит. События происходят в каждом потоке.

Вы можете настроить наблюдателя для приема событий и реагирования на них. Существует два типа наблюдателей: отдельный наблюдатель для каждого потока и пакетный наблюдатель для всех потоков. Наблюдатели для каждого потока установлены устаревшими, теперь они остаются только для совместимости со старыми версиями.

Примечание: Настоятельно рекомендуется использовать новый API пакетных наблюдателей вместо старого для каждого потока.

Пакетные наблюдатели имеют некоторые преимущества перед наблюдателями за потоком:

- уменьшите и установите фиксированное количество потоков, созданных самим TrackEngine.
- устраните накладные расходы на производительность из нескольких одновременно работающих потоков, используемых для обратных вызовов для каждого потока.
- позволяет легко использовать пакетированный API SDK без дополнительной агрегации данных из отдельных обратных вызовов. Как для GPU, так и для CPU пакетный SDK API повышает производительность (для GPU эффект гораздо более значителен).
- предоставьте дополнительную информацию в выходных данных (сигнатуры функций обратного вызова для каждого потока остаются прежними из-за совместимости со старыми версиями)

Примечание: вы должны настроить либо одного наблюдателя для каждого потока, либо пакетное наблюдение для всех потоков, но не для обоих одновременно.

Интерфейсы наблюдателя потока:

Наблюдатели за каждым потоком:

- IBestShotObserver
- IVisualObserver
- IDebugObserver

Пакетные наблюдатели:

- IBatchBestShotObserver
- IBatchVisualObserver
- IBatchDebugObserver

Реализуя один или несколько интерфейсов наблюдателя, можно определить пользовательскую логику обработки в вашем приложении.

Тип IBestShotPredicate определяет критерии пригодности распознавания для обнаружения лиц. Реализуя пользовательский предикат, можно изменить логику выбора наилучшего снимка и, следовательно, указать, какие изображения попадут в фазу распознавания.

Пример настройки API для каждого наблюдателя потока:

void IStream::setBestShotObserver(tsdk::IBestShotObserver* observer) – задает наблюдателя лучшего снимка для потока

- observer - указатель на объект наблюдателя, см. IBestShotObserver. Не устанавливайте значение nullptr, если вы хотите отключить его, затем используйте IStream::setObserverEnabled со значением false.

Пример настройки API пакетного наблюдателя:

void ITrackEngine::setBatchBestShotObserver(tsdk::IBatchBestShotObserver * observer) - задает наблюдателя лучшего снимка для всех потоков

- observer - указатель на объект пакетного наблюдателя, см. IBatchBestShotObserver. Не устанавливайте значение nullptr, если вы хотите отключить его, затем используйте IStream::setObserverEnabled со значением fasle.

4.3 IBestShotObserver

void bestShot(const tsdk::DetectionDescr& descr) вызывается для каждого получившегося лучшего снимка. Он предоставляет информацию о наилучшем снимке, включая номер кадра, координаты обнаружения, обрезанное неподвижное изображение и другие данные.

- descr – описание обнаружение лучшего снимка

```
struct TRACK_ENGINE_API DetectionDescr {
    //! Index of the frame
    tsdk::FrameId frameIndex;

    //! Index of the track
    tsdk::TrackId trackId;

    //! Source image
    fsdk::Image image;

    fsdk::Ref<ICustomFrame> customFrame;

    //! Face landmarks
    fsdk::Landmarks5 landmarks;

#ifndef MOBILE_BUILD
    //! Human landmarks
    fsdk::HumanLandmarks17 humanLandmarks;

    //! NOTE: only for internal usage, don't use this field, it isn't valid ptr
    fsdk::IDescriptorPtr descriptor;
#endif

    //! Is it full detection or redetect step
    bool isFullDetect;

    //! Detections flags
    // needed to determine what detections are valid in extraDetections
    // see EDetectionFlags
    uint32_t detectionsFlags;

    //! Detection
    // always is valid, even when detectionsFlags is combination type
    // useful for one detector case
    // see detectionObject
    fsdk::Detection detection;

    //! extra detections
    // needed when detectionsFlags has combination type,
    // e.g. for EDetection_Body_Face extraDetections[EDetection_Face],
extraDetections[EDetection_Body] are valid
    // note: for simple detection type extra detection with corresponding index is
valid too
    fsdk::Detection extraDetections[EDetectionObject::EDetection_Simple_Count];

    bool hasDetectionFlag(EDetectionObject obj) {
        return (detectionsFlags & (1 << obj)) ? true : false;
    }

    void setDetectionFlag(EDetectionObject obj, bool enable) {
        if (enable) {
            detectionsFlags |= (1 << obj);
        }
        else {
            detectionsFlags &= ~(1 << obj);
        }
    }

    void setExtraDetection(EDetectionObject obj, const fsdk::Detection &detection) {
        extraDetections[obj] = detection;
    }
};
```

void trackEnd(const tsdk::TrackId& trackId) рассказывает, что трек с trackId закончился, и от него больше не следует ожидать лучших кадров.

- trackId – идентификатор трека

void trackStatusUpdate(tsdk::FrameId frameId, tsdk::TrackId trackId, tsdk::TrackStatus status) сообщает, что статус трека обновлен.

- frameId – идентификатор кадра
- trackId - идентификатор трека
- status – новый статус трека

```
/** @brief Track status enum. (see human tracking algorithm section in docs for
details)
*/
enum class TrackStatus : uint8_t {
    ACTIVE = 0,
    NONACTIVE
};
```

void trackReIdentificate(tsdk::FrameId frameId, tsdk::TrackId trackId, tsdk::TrackId reidTrackId) tells that the track with id = trackId was matched to one of the old non-active tracks with id = reidTrackId. See section ReIdentification for details.

- frameId - идентификатор кадра
- trackId - идентификатор трека, который был сопоставлен с одним из старых неактивных треков
- reidTrackId - идентификатор неактивного трека, который успешно удалось отследить с помощью id = trackId

4.4 IVisualObserver

void visual(const tsdk::FrameId &frameId, const fsdk::Image &image, const tsdk::TrackInfo * trackInfo, const int nTrack) позволяет визуализировать текущее состояние потока. Он предназначен в основном для целей отладки. Функция должна быть перегружена.

- frameId – текущий id кадра
- image – изображение кадра
- trackInfo – массив текущих активных треков

```
struct TRACK_ENGINE_API TrackInfo {
    //! Face landmarks
    fsdk::Landmarks5 landmarks;

    //! Last detection for track
    fsdk::Rect rect;

    //! Id of track
    TrackId trackId;

    //! Score for last detection in track
    float lastDetectionScore;

    //! Is it detected or tracked bounding box
    bool isDetector;
};
```

- nTrack – номер треков

4.5 IDebugObserver

void debugDetection(const tsdk::DetectionDebugInfo& descr) обратный вызов отладки детектора. В обычной реализации ничего не делает.

- descr - описание отладки обнаружения

```
struct DetectionDebugInfo {
    //! Detection description
    DetectionDescr descr;

    //! Is it detected or tracked bounding box
    bool isDetector;

    //! Filtered by user bestShotPredicate or not.
    bool isFiltered;

    //! Best detection for current moment or not
    bool isBestDetection;
};
```

void debugForegroundSubtraction(const tsdk::FrameId& frameId, const fsdk::Image& firstMask, const fsdk::Image& secondMask, fsdk::Rect * regions, int nRegions) обратный вызов отладки фонового вычитания. В обычной реализации ничего не делает.

- frameId - идентификатор кадра переднего плана
- firstMask - результат операции вычитания фона
- secondMask - результат операции вычитания фона после процедур эрозии и расширения
- regions - области, полученные после операции вычитания фона
- nRegions - количество возвращенных регионов

4.6 BestShotPredicate

bool checkBestShot(const tsdk::DetectionDescr& descr) Прогноз для обнаружения лучшего снимка. Это место для выполнения любых необходимых проверок качества (например, с помощью оценщиков FaceEngine). Эта функция должна быть перегружена.

- descr – описание детекции

возвращает значение - true, если descr прошел проверку, в противном случае false

4.7 VisualPredicate

bool needRGBImage(const tsdk::FrameId frameId, const tsdk::AdditionalFrameData *data) Прогноз для визуального обратного вызова. Он служит для принятия решения о том, выводить ли исходное изображение в визуальном обратном вызове или нет. Эта функция может быть перегружена. Реализация по умолчанию возвращает true.

- frameId - идентификатор кадра
- data – дополнительные данные кадра, переданные пользователем
- return value - true, если исходное изображение (или изображение rgb для пользовательского кадра) требуется для вывода в визуальном обратном вызове, false иначе

4.8 IBatchBestShotObserver

void bestShot(const fsdk::Span &streamIDs, const fsdk::Span &data) Пакетная версия обратного вызова bestShot.

- streamIDs – массив идентификаторов потока
- data - массив данных обратного вызова для каждого потока

```
struct TRACK_ENGINE_API BestShotCallbackData {
    //! detection description. see 'DetectionDescr' for details
    tsdk::DetectionDescr descr;

    //! additional frame data, passed by user in 'pushFrame'. see
'AdditionalFrameData' for details
    tsdk::AdditionalFrameData *frameData;
};
```

void trackEnd(const fsdk::Span &streamIDs, const fsdk::Span &data) Пакетная версия обратного вызова bestShot.

- streamIDs - массив идентификаторов потока
- data - массив данных обратного вызова для каждого потока

```
/**
* @brief Track end reason. See 'TrackEndCallbackData' for details.
*/
enum class TrackEndReason : uint8_t {
    //! non-active track ends because of lifetime expired
    NONACTIVE_TIMEOUT,
    //! active track ends because of reidentification with old non-active track
    ACTIVE_REID,
    //! non-active track ends because of reidentification with older non-active track
    //  (that means, that current track couldn't been updated and was matched to old
non-active at the same time)
    NONACTIVE_REID,
    //! all alive tracks are finished on TrackEngine stop call
    TRACKENGINE_STOP
};

struct TRACK_ENGINE_API TrackEndCallbackData {
    //! frame id
    tsdk::FrameId frameId;

    //! track id
    tsdk::TrackId trackId;

    //! parameter implies reason of track ending
    // NOTE: now it's using only for human tracking, don't use this for other
detectors
    TrackEndReason reason;
};
```

void trackStatusUpdate(const fsdk::Span &streamIDs, const fsdk::Span &data) Пакетная версия обратного вызова trackStatusUpdate.

- streamIDs - массив идентификаторов потока
- data - массив данных обратного вызова для каждого потока

```
struct TRACK_ENGINE_API TrackStatusUpdateCallbackData {
    //! frame id
    tsdk::FrameId frameId;

    //! track id
    tsdk::TrackId trackId;

    //! new track status
    tsdk::TrackStatus status;
};
```

void trackReIdentificate(const fsdk::Span &streamIDs, const fsdk::Span &data) Пакетная версия обратного вызова trackReIdentificate.

- streamIDs - массив идентификаторов потока
- data - массив данных обратного вызова для каждого потока

```
struct TRACK_ENGINE_API TrackReIdentificateCallbackData {
    //! id of frame
    tsdk::FrameId frameId;

    //! id of track, that was matched to one of the old non-active tracks
    tsdk::TrackId trackId;

    //! id of the non-active track, that successfully mathed to track with id =
'trackId'
    // see human tracking algorithm section in docs for details
    tsdk::TrackId reidTrackId;

    //! similarity from matching of tracks descriptors
    float similarity;
};
```

4.9 IBatchVisualObserver

void visual(const fsdk::Span &streamIDs, const fsdk::Span &data) Пакетная версия обратного вызова visual.

- streamIDs - массив идентификаторов потока
- data - массив данных обратного вызова для каждого потока

```
struct TRACK_ENGINE_API VisualCallbackData {
    //! frame id
    tsdk::FrameId frameId;

    //! this is either original image (if 'pushFrame' used) or RGB image got from
custom frame convert (is 'pushCustomFrame' used)
    fsdk::Image image;

    //! tracks array raw ptr
    tsdk::TrackInfo *trackInfo;

    //! number of tracks
    int nTrack;

    //! additional frame data, passed by user in 'pushFrame'. See
'AdditionalFrameData' for details.
    tsdk::AdditionalFrameData *frameData;
};
```

4.10 IBatchDebugObserver

void debugForegroundSubtraction(const fsdk::Span &streamIDs, const fsdk::Span &data) Пакетная версия обратного вызова debugForegroundSubtraction.

- streamIDs - массив идентификаторов потока
- data - массив данных обратного вызова для каждого потока

void debugDetection(const fsdk::Span &streamIDs, const fsdk::Span &data) Пакетная версия обратного вызова debugDetection.

- streamIDs - массив идентификаторов потока
- data - массив данных обратного вызова для каждого потока

```
struct TRACK_ENGINE_API DebugForegroundSubtractionCallbackData {
    //! frame id
    tsdk::FrameId frameId;

    //! first mask of  the foreground subtraction
    fsdk::Image firstMask;

    //! second mask of  the foreground subtraction
    fsdk::Image secondMask;

    //! regions array raw ptr
    fsdk::Rect *regions;

    //! number of regions
    int nRegions;
};

/** @brief Detection data for debug callback.
    */
struct TRACK_ENGINE_API DebugDetectionCallbackData {
    //! Detection description
    DetectionDescr descr;

    //! Is it detected or tracked bounding box
    bool isDetector;

    //! Filtered by user bestShotPredicate or not.
    bool isFiltered;

    //! Best detection for current moment or not
    bool isBestDetection;
};
```

void IStream::setObserverEnabled(tsdk::StreamObserverType type, bool enabled) Включает или выключает наблюдателя

- type – тип наблюдателя
- enabled – флаг включения/выключения

void IStream::join() Блокирует текущий поток до тех пор, пока все кадры в этом потоке не будут обработаны и все обратные вызовы не будут выполнены (поток нельзя использовать после соединения).

5 Модуль детекции

5.1 Краткое описание

Модуль детекции объектов отвечает за быстрые и грубые задачи детекции, такие как поиск лица на изображении.

5.2 Структура детекции

Структура детекции представляет ограничивающий пространство изображения прямоугольник обнаруженного объекта, а также даёт оценку обнаружения.

Оценка обнаружения является мерой уверенности в результате верной классификации конкретного объекта и используется для выбора наиболее подходящего лица из всех.

Оценка обнаружения является мерой достоверности классификации, а не мерой качества исходного изображения. Хотя оценка и связана с качеством (данные низкого качества обычно дают более низкую оценку), она не является допустимой мерой для оценки визуального качества изображения.

Для выполнения этой задачи существуют специальные оценки (Подробнее см. "Оценка качества изображения" в главе "Модуль оценки параметров").

5.3 Обнаружение лиц

Обнаружение объекта выполняется объектом IDetector. Среди всех его функций наибольший интерес представляет detect(). Для работы функции требуется изображение и область интереса (происходит обрезка изображения и поиск лица выполняется только в заданной области).

5.3.1 Типы детекторов

Поддерживаемые типы детекторов:

- FaceDetV1
- FaceDetV2
- FaceDetV3

Существует два основных семейства детекторов. Первое из них включает два варианта детекторов: FaceDetV1 и FaceDetV2. Второе семейство на данный момент включает только один вариант детектора - FaceDetV3.

FaceDetV3 является самым новым и наиболее точным детектором. В плане производительности он аналогичен детектору FaceDetV1.

В пользовательском коде можно указать необходимый тип детектора с помощью параметра во время создания объекта IDetector.

Детектор лиц реализует метод redetect(), который предназначен для оптимизации обнаружения лиц на последовательности видеокадров. Вместо проведения полномасштабного детектирования на каждом кадре, имеется возможность детектировать detect() новые лица на более низкой частоте (например, каждом пятом кадре) и подтверждать их с помощью функции redetect(). Это значительно повышает производительность за счет уменьшения вызовов detect(). Обратите внимание, что redetect () также обновляет характерные точки лица.

5.3.2 Конфигурирование FaceDetV1 и FaceDetV2

Детектор FaceDetV1 является более надёжным, а FaceDetV2 работает в два раза быстрее (Подробнее см. Приложение A глава "Спецификации").

Производительность детекторов FaceDetV1 и FaceDetV2 зависит от количества лиц на изображении. Детектор FaceDetV3 не зависит от количества лиц, поэтому он может быть медленнее чем FaceDetV1 на изображениях с одним лицом и значительно быстрее на изображениях со множеством лиц.

5.3.3 Конфигурирование FaceDetV3

FaceDetV3 находит лица размером от minFaceSize пикселей до maxFaceSize пикселей (помните, что maxFaceSize <=minFaceSize * 32). Вы можете изменить минимальный и максимальный размер лиц, которые будут искаться на изображении, в конфигурации faceengine.conf.

Например, config->setValue("FaceDetV3::Settings", "minFaceSize", 20);

Чем меньше размер лица, тем больше нам нужно времени.

Рекомендуем использовать для minFaceSize значения 20, 40 и 90. Для распознавания рекомендуется размер 90 пикселей. Если вы хотите найти лица со значением нестандартного размера, вам нужно указать характерную точку лица со значением: 95% * value. Например, мы хотим найти лица размером 50 пикселей, это означает, что в конфигурации мы должны установить: 50 * 0,95 ~ 47 пикселей.

Примечание. FaceDetV3 может обеспечивать точное определение 5 характерных точек только для лиц с размером большим чем 40x40, для лиц меньшего размера точность определения характерных точек уменьшается.

Если на целевом изображении мало лиц и размер лиц после изменения размера изображения будет меньше 40x40, то мы рекомендуем использовать 68 характерных точек.

Если на целевом изображении много лиц (больше семи), будет быстрее увеличить minFaceSize, чтобы получить достаточно большие лица для точного определения характерных точек.

5.4 Определение положения лица

5.4.1 Пять характерных точек (landmark5)

Определение положения лица — это процесс обнаружения специальных характерных точек (так называемых “landmarks”) на лице. FaceEngine выполняет обнаружение характерных точек одновременно с детекцией лиц, так как некоторые характерные точки являются побочными продуктами детекции.

Минимум требуется 5 характерных точек: две для глаз, одна для кончика носа и две для уголков рта. Используя эти координаты, можно нормализовать исходное изображение (см. Главу "**Ошибка! Источник ссылки не найден.**") для использования с остальными алгоритмами FaceEngine.

5.4.2 Шестьдесят восемь характерных точек (landmark68)

Также реализовано более продвинутое определение положения лица по 68 точкам. Его следует использовать для получения точной информации о лице и его отдельных элементов. Характерные точки представлены на Рис. 1.

При использовании 68 характерных точек требуется дополнительное время на вычисления, поэтому не стоит использовать их, если не требуется точная информация о лице. Если Вы используете 68 характерных точек, 5 характерных точек будут переопределены в более точное подмножество из 68 характерных точек.

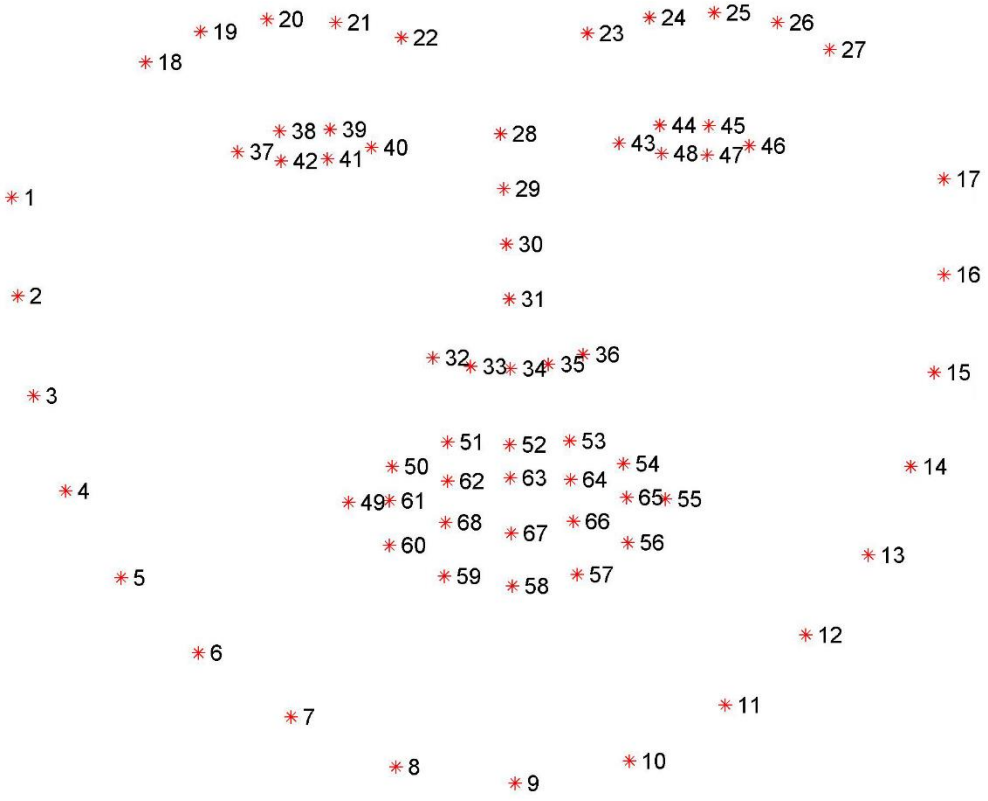

Рисунок 1

Средняя погрешность определения характерной точки на нормализованном изображении (см. Главу "Нормализация изображений") приведена в таблице 1

Таблица 1. Средняя погрешность определения характерной точки

Точка	Ошибка (пиксели)	Точка	Ошибка (пиксели)	Точка	Ошибка (пиксели)	Точка	Ошибка (пиксели)
1	±3,88	18	±3,77	35	±1,62	52	±1,65
2	±3,53	19	±2,83	36	±1,90	53	±2,01
3	±3,88	20	±2,70	37	±1,78	54	±2,00
4	±4,30	21	±3,06	38	±1,69	55	±1,93
5	±4,67	22	±3,92	39	±1,63	56	±2,18
6	±4,87	23	±3,46	40	±1,52	57	±2,17
7	±4,67	24	±2,59	41	±1,54	58	±1,99
8	±4,01	25	±2,53	42	±1,60	59	±2,32
9	±3,46	26	±2,95	43	±1,55	60	±2,33
10	±3,87	27	±3,84	44	±1,60	61	±2,06
11	±4,56	28	±1,88	45	±1,74	62	±1,97
12	±4,94	29	±1,75	46	±1,72	63	±1,56
13	±4,55	30	±1,92	47	±1,68	64	±1,86
14	±4,45	31	±2,20	48	±1,65	65	±1,94
15	±4,13	32	±1,97	49	±1,99	66	±2,00
16	±3,68	33	±1,70	50	±1,99	67	±1,70
17	±4,09	34	±1,73	51	±1,95	68	±2,12

Обычные 5 характерных точек должны примерно соответствовать следующим позициям:

- Среднему значению из позиций 37, 40 для левого глаза;
- Среднему значению из позиций 43, 46 для правого глаза;
- Позиции 31 для кончика носа;
- Позиции 49 и 55 для уголков рта.

Ориентиры для обоих случаев выводятся детектором лиц через структуры Landmarks5 и Landmarks68. Обратите внимание, что в отношении характеристик производительности, результат определения 5 характерных точек будет получен совместно с детекцией лица без дополнительных затрат, чего нельзя сказать для случая с 68 точками. Таким образом, старайтесь запрашивать наименьшее количество точек для выполнения задачи.

Типичные случаи использования 68 характерных точек:

- Сегментация;
- Определение положения головы.

6 Модуль выравнивания лиц

Нормализация (warping) — это процесс приведения изображения лица к определённому стандарту. Для работы требуются ключевые точки и детекция лица (см. Главу «Модуль детекции лица»).

- компенсация двумерного вращения изображения (крена головы);
- центрирование изображение на основе положения глаз;
- обрезка изображения.

Таким образом, все нормализированные изображения имеют единый характерный стандарт, например, левый глаз всегда находится в зоне, обозначенной определенными координатами. Это обеспечивает определенную схожесть входных данных, что улучшает работу различных алгоритмов.

Рисунок 2

Имейте в виду, что нормализация изображения не является потокобезопасной, поэтому необходимо создать объект *warper* для каждого рабочего потока.

7 Модуль определения параметров

7.1 Краткое описание

Модуль определения параметров является единственным многофункциональным модулем в VisionLabs LUNA SDK Mobile Android. Он разработан как набор инструментов, помогающих оценить различные свойства изображения или изображенного объекта. Эти свойства могут использоваться для повышения точности алгоритмов, реализуемых другими средствами VisionLabs LUNA SDK Mobile Android, или для выполнения пользовательских задач.

7.2 Определение положения головы

Этот алгоритм оценки определяет положение головы в пространстве камеры, а именно по углам рыскания, тангажа и крена.

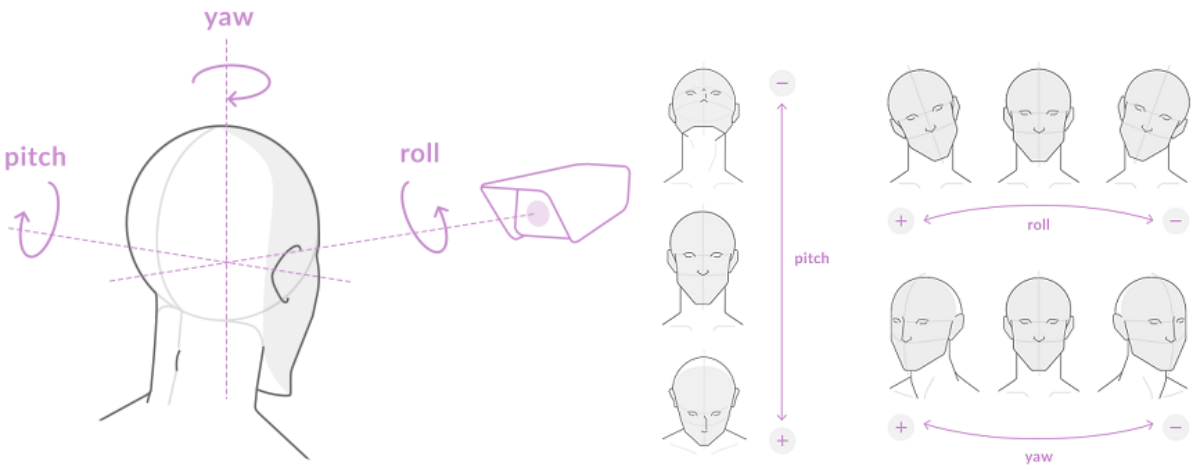

Рисунок 3

Поскольку сложно достоверно определить перемещение головы по трём осям, не зная точные настройки камеры, оценивается только вращение головы по трём осям измерения.

Характеристики определения:

- Единицы измерения (Градусы);
- Обозначения (Эйлеровы углы);
- Точность (см. Таблицу).

Примечание. Точность расчёта уменьшается при увеличении угла поворота головы. Типичные погрешности для различных диапазонов углов приведены в таблице 2.

Таблица 2. Точность расчёта положения головы

	Диапазон	-45°…+45°	<-45° or >+45°

Средняя ошибка расчёта (на ось)	Угол рыскания (Yaw)	±2.7°	±4.6°
Средняя ошибка расчёта (на ось)	Угол тангажа (Pitch)	±3.0°	±4.8°
Средняя ошибка расчёта (на ось)	Угол крена (Roll)	±3.0°	±4.6°

Начальное положение соответствует лицу, расположенному перпендикулярно к направлению камеры, и параллельно вертикальной оси камеры. См. Рис. 3.

7.3 Определение качества изображения

Примечание. Алгоритм оценки обучен работе с нормализованными изображениями (подробности см. в главе "Нормализация изображений").

Главное правило для определения качества:

- Выполните детекцию лица и проверьте, достаточно ли высока точность детекции. Если нет, отклоните детекцию;
- Создайте нормализованное изображение лица (см. главу "Модуль обработки дескрипторов") с помощью детекции лица и характерных точек лица;
- С помощью алгоритма оценки оцените визуальное качество, окончательно исключите некачественные изображения.

Хотя приведенная выше схема может показаться несколько сложной, она является наиболее эффективной с точки зрения производительности, так как в случае исключения изображения на каждом шаге уменьшается рабочая нагрузка для следующего шага.

Этот алгоритм оценки предназначен для прогнозирования визуального качества изображения. Он обучен работе с предварительно нормализованными изображениями лиц и возвращает меньшее значение коэффициента качества, если:

- Изображение размыто;
- Изображение недоэкспонированно (т. е. слишком темное);
- Изображение переэкспонированно (т. е. слишком светлое);
- Лицо на изображении освещено неравномерно (большая разница между светлыми и тёмными участками на лице);
- Изображение содержит блики (слишком зеркально).

Механизм оценки определяет качество фотоизображения исходя из вышеуказанных параметров. Для каждого параметра эстиматор возвращает два значения: коэффициент качества и вердикт.

Коэффициент качества - это значение в диапазоне [0..1], где 0 соответствует низкому качеству, а 1 - высокому. Эстиматор возвращает низкий коэффициент качества для параметра Blur, если изображение слишком размыто.

Вердикт даёт оценку качеству в соответствии с оцениваемым параметром. Например, если изображение слишком размыто, то эстиматор вернёт значение "isBlurred = true".

Для каждого из оцениваемых параметров можно задать порог. Коэффициент качества и вердикт связаны через этот порог. Если полученное значение коэффициента качества ниже, чем порог, тогда качество изображения является низким и эстиматор возвращает значение «true». Например, если коэффициент качества изображения в плане размытия выше порога, тогда вердикт будет «false».

Если полученное значение для любого из параметров ниже указанного порога, тогда изображение имеет низкое качество. Если вердикты для всех параметров имеют значение «false», тогда у изображения высокое качество.

Примеры приведены на изображениях ниже. Изображения хорошего качества показаны справа.

Рисунок 4 Размытое изображение (слева), не размытое изображение (справа)

Рисунок 5 Тёмное изображение (слева), изображение высокого качества (справа)

Рисунок 6 Засвеченное изображение (слева), изображение высокого качества (справа)

Рисунок 7 Изображение с неравномерным освещением (слева), изображение с равномерным освещением (Справа)

Рисунок 8 Изображение содержит блики на лице (слева), изображение высокого качества (справа)

Примечание. Равномерность освещения относится к освещению лица на изображении. Чем ниже разница между светлыми и тёмными зонами на лице, тем выше оцениваемое значение. Когда освещение равномерно распределено по лицу, значение близко к «1».

Примечание. Зеркальность является способностью лица отражать свет. Чем выше оцененное значение, тем выше качество изображения. Если оцененное значение является низким, то на лице есть блики.

Таблица 3. Параметры качества изображения и их пороги

Порог	Оцениваемое значение	Рекомендуемый диапазон значений	Значение по умолчанию
blurThreshold	Размытие	[0.57..0.65]	0.61
darknessThreshold	Темнота изображения	[0.45..0.52]	0.50
lightThreshold	Сет на изображении	[0.44..0.61]	0.57
illuminationThreshold	Равномерность освещения	[0..0.3]	0.1
specularityThreshold	Зеркальность	[0..0.3]	0.1

Наиболее важными параметрами для распознавания лиц являются "blurThreshold", "darknessThreshold"и "lightThreshold". Их следует подбирать тщательно.

Вы можете выбирать изображения более высокого визуального качества путём задания высоких значений "illuminationThreshold" и "specularityThreshold". Распознавание лиц слабо зависит от этих двух параметров.

7.4 Определение атрибутов лица

Примечание. Алгоритм оценки обучен работе с нормализованными изображениями (подробности см. в главе "Нормализация изображений").

Алгоритм оценки атрибутов лица определяет атрибуты лица. В настоящее время доступно определение следующих атрибутов лица;

- Возраст: определяется возраст человека;
- Пол: определяется пол человека;
- Этническая принадлежность: определяется Этническая принадлежность человека.

Перед тем, как использовать алгоритм оценки атрибутов лица, пользователю следует решить, оценивать или нет перечисленные выше особые характеристики с помощью структуры IAttributeEstimator::EstimationRequests. Атрибуты впоследствии передаются в главный метод estimate().

Алгоритм оценки переопределяет выходную структуру AttributeEstimationResults, которая состоит из необязательных полей, описывающих результаты запрашиваемых пользователем признаков.

- Возраст указан в годах:
  - Для кооперативных условий (см. Приложение Б): средняя ошибка зависит от возраста человека, дополнительную информацию см. в таблице 4. Точность определения равна 2.3.
- При определении пола 0 - женщина, 1 - мужчина.
  - Точность определения в кооперативном режиме составляет 99,81% с порогом 0,5;
  - Точность определения в некооперативном режиме составляет 92,5%.
- Определение расовой принадлежности возвращает 4 нормализованных float значения, каждое из которых описывает вероятность принадлежности человека к одной из рас.
  - Точность оценки этнической принадлежности в некооперативном режиме составляет 90,7%.
  - В настоящее время алгоритм оценки может различить 4 типа рас: Афроамериканец, Индиец, Азиат, Европеец.
  - Оценивает этническую принадлежность субъекта по внешности субъекта на заданном изображении;
  - Выводит структуру EmotionsEstimation с вышеупомянутыми данными.

EthnicityEstimation отображает вероятность принадлежности к определённой расе в оценках, которые представлены как нормализованные значения с плавающей точкой в диапазоне [0..1]. Сумма всех оценок всегда равна 1. Каждая оценка обозначает сходство с соответствующей расой.

Таблица 4. Средняя погрешность при определении возраста для указанных возрастных групп при кооперативных условиях

Возраст (лет)	Средняя погрешность (лет)
0-3	±3.3
4-7	±2.97

8-12	±3.06
13-17	±4.05
17-20	±3.89
20-25	±1.89
25-30	±1.88
30-35	±2.42
35-40	±2.65
40-45	±2.78
45-50	±2.88
50-55	±2.85
55-60	±2.86
60-65	±3.24
65-70	±3.85
70-75	±4.38
75-80	±6.79

Примечание. В более ранних версиях Luna SDK алгоритм оценки атрибутов лица имел низкие показатели точности в некооперативном режиме (точность определения пола составляла только 56%) и не определял возраст ребенка. После решения этой проблемы средняя ошибка определения возраста для возрастной группы стала немного выше из-за расширения функциональных возможностей сети.

7.5 Определение цветности/монохромности

Этот алгоритм оценки определяет, является входное изображение монохромным или цветным. Он реализует функцию estimate(), которая принимает исходное изображение image и выводит Булево значение, указывающее, является ли изображение монохромным (true) или нет (false).

7.6 Определение атрибутов глаз

Примечание. Алгоритм оценки обучен работе с нормализованными изображениями (подробности см. в главе “Нормализация изображений”).

Этот алгоритм оценки направлен на определение:

- Состояния глаз: Открыты, Закрыты, Перекрыты;
- Точного расположения зрачков как массива характерных точек;
- Точного положение век как массива характерных точек.

В интерфейс алгоритма оценки можно передать только нормализованное изображение с обнаруженным лицом. Чем лучше качество изображения, тем лучше результат.

Классификатор атрибутов глаз поддерживает три категории: “Открыты”, “Закрыты”, “Перекрыты”. Изображения низкого качества или те, на которых глаза чем-то перекрыты (например, очками, волосами, жестом), попадают в категорию “Перекрыты”. Всегда полезно проверить состояние глаз перед использованием результата сегментации.

Точное положение позволяет провести сегментацию зрачков и век. Алгоритм оценки способен выводить формы зрачков и век как совокупность точек, образующих эллипсис. Используйте результаты сегментации только в том случае, если глаза в состоянии «Открыты».

Алгоритм оценки:

- Реализует функцию estimate(), которая принимает нормализованное исходное изображение (см. главу “Нормализация изображений”) и нормализованные характерные точки типа Landmarks5 или Landmarks68. Нормализованное изображение и характерные точки получены от нормализатора (warper) (см. IWarper::warp());
- Классифицирует состояние глаз и определяет характерные точки зрачков и век;
- Выдаёт структуры EyesEstimation на выходе.

Примечание. Обозначения “правый” и “левый” относится к тому, как вы видите изображение на экране. Это означает, что левый глаз у человека на изображении не обязательно левый, а расположен на левой стороне экрана. Следовательно, правый глаз находится в правой части экрана. Более формально, метка “левый” относится к левому глазу субъекта (и аналогично для правого глаза), таким образом, что xright< xleft.

EyesEstimation::EyeAttributes представляет состояние глаз как перечисление EyeState с возможными значениями: Открыты, Закрыты, Перекрыты.

Характерные точки зрачков представлены шаблонной структурой *Landmarks*, которая специализирована для 32 точек. Характерные точки век представлены шаблонной структурой Landmarks, которая специализирована для 6 точек.

7.7 Определение направления взгляда

Алгоритм определяет направление взгляда относительно положения головы. Поскольку сложно достоверно определить перемещение головы по трём осям, не зная точные настройки камеры, оценивается только вращение головы по трём осям измерения.

Характеристики определения:

- Единицы измерения (Градусы);
- Обозначения (Эйлеровы углы);
- Точность (см. Таблицу).

Примечание. Угол крена (roll) не определяется, точность прогноза направления взгляда уменьшается при увеличении угла поворота головы. Типичные погрешности для различных диапазонов углов приведены в таблице 5.

Таблица 5. Точность прогноза направления взгляда

	Диапазон	- 25°...+25°	-25°... -45 °или 25 °... +45°
Средняя ошибка расчёта (на ось)	Угол рыскания (Yaw)	±2.7°	±4.6°
Средняя ошибка расчёта (на ось)	Угол тангажа (Pitch)	±3.0°	±4.8°

Начальное положение соответствует направлению взгляда перпендикулярному к плоскости лица, и осью симметрии параллельной вертикальной оси камеры. См. Рис. 3.

7.8 Определение наличия очков

Алгоритм оценки наличия очков определяет наличие или отсутствие очков на субъекте. В настоящее время существует 3 типа состояний, которые механизм оценки может определить:

- NoGlasses определяет, есть ли вообще какие-либо очки на субъекте;
- EyeGlasses определяет, надеты ли на субъект оптические очки;
- SunGlasses определяет, надеты ли на субъект солнечные очки.

Примечание. Исходное входное изображение должно быть нормализовано для правильной работы алгоритма оценки (см. Главу "Нормализация изображений"). Качество оценки зависит от пороговых значений, расположенных в файле настройки (faceengine.conf) в разделе GlassesEstimator::Settings. По умолчанию, эти пороговые значения установлены на оптимальные.

Таблица 6 содержит истинно-положительные оценки (TPR), соответствующие выбранным ложноположительным оценкам (FPR).

Таблица 6. Коэффициенты TPR/FPR для определения наличия очков

Состояние	TPR	FPR
NoGlasses	0,997	0,00234
EyeGlasses	0,9768	0,000783
SunGlasses	0,9712	0,000383

7.9 Определение эмоций

Примечание. Алгоритм оценки обучен работе с нормализованными изображениями (подробности см. в главе "Нормализация изображений").

Этот алгоритм оценки определяет, выражает ли лицо на фотоизображении перечисленные эмоции:

- гнев,
- отвращение,
- страх,
- радость,

- удивление,
- грусть,
- нейтрально.

В интерфейс алгоритма оценки можно передать только нормализованные изображения с детектированными лицами. Чем выше качество изображения, тем лучше результат.

Алгоритм оценки (см. IEmotionsEstimator in IEstimator.h):

- Выполняет функцию estimate(), которая принимает нормализованное исходное изображение (См. главу “Нормализация изображений”). Нормализованное изображение получено от нормализатора (warper) (см. IWarper::warp()).;
- Определяет эмоции, выраженные субъектом на заданном изображении;
- Выводит структуру EmotionsEstimation с вышеупомянутыми данными.

EmotionsEstimation представляет эмоции как нормализованные значения с плавающей точкой в диапазоне [0..1], где 0 – отсутствие конкретной эмоции и 1 – эмоция наиболее ярко выражена.

8 Модуль работы с биометрическими шаблонами (дескрипторами)

8.1 Краткий обзор

Данная секция описывает дескрипторы, все процессы и объекты, относящиеся к ним.

Дескриптор представляет собой набор параметров объекта, закодированных специальным образом. Обычно дескрипторы независимы от различных аффинных преобразований объектов и небольших цветовых вариаций. Это свойство позволяет эффективно использовать дескрипторы для идентификации, поиска и сравнения изображений реальных объектов.

Для получения дескриптора следует выполнить операцию, которая называется *извлечением* дескриптора *(extraction).*

Основной сценарий применения дескрипторов заключается в сравнении двух дескрипторов и нахождении их порога схожести. Таким образом можно идентифицировать людей путём сравнения их дескрипторов с существующей базой дескрипторов.

Все операции сравнения дескрипторов называются *матчингом (matching).* Результатом матчинга двух дескрипторов является расстояние между составляющими соответствующих наборов, упомянутых выше. Таким образом, по величине этого расстояния мы можем сказать, являются ли два объекта примерно одинаковыми. Это позволяет выполнять сравнение, поиск и другие операции более высокого уровня.

8.2 Задача идентификации

Решается задача распознавания лиц по фото или кадрам видео при их сравнении с существующей базой данных лиц. Вначале проводится детекция – отделение лиц от остальных объектов в кадре. Затем проводится идентификация этих лиц.

Для этих целей используется дескриптор, который извлекается из изображения лица. Лицо человека не изменяется в течение его жизни.

Извлечение дескриптора проводится из области изображения объекта вблизи ранее обнаруженных характерных точек лица, поэтому качество дескриптора сильно зависит от этих точек и исходного изображения (из которого получен дескриптор).

Процесс распознавания состоит из четырёх основных этапов:

- детекция лица на фотоизображении;
- нормализация детекции лица – центрирование и компенсация аффинных углов;
- извлечение дескриптора;
- сравнение дескрипторов.

Можно извлечь также свойства лица (пол, возраст, эмоции и т. п.) или оценку атрибутов фотоизображения (размытие, равномерность освещения и т. п.).

8.3 Дескриптор

Объект дескриптор хранит компактный набор свойств, а также некоторые вспомогательные параметры, которые использовались для извлечения этих свойств из исходного изображения.

Вместе эти параметры определяют совместимость дескрипторов. Не все дескрипторы совместимы друг с другом. Несовместимые дескрипторы невозможно добавить в пакет и сопоставить, поэтому следует обратить внимание, какие настройки используются при их извлечении. См. раздел "Извлечение дескриптора" для получения дополнительной информации об извлечении дескриптора.

8.3.1 Версии дескрипторов

Алгоритм обработки дескриптора лица со временем развивается, поэтому новые версии VisionLabs LUNA SDK Mobile Android содержат улучшенные модели алгоритма.

В настоящее время доступны следующие версии: 54, 56, 57, 58 и 59.

Дескрипторы имеют реализации backend и mobilenet. Backend версии более точны, а mobilenet быстрее и их файлы модели меньше (см. Приложение А глава "Спецификации").

Примечание. Версия 57, 58, 59 поддерживает только реализацию backend.

Версия 59 является наиболее точной.

Дополнительные сведения о производительности и точности различных версий дескрипторов см. в секции "Производительность классификации" приложении А глава "Спецификации" .

Примечание. Дескрипторы разных версий несовместимы! Это означает, что нельзя выполнять матчинг дескрипторов разных версий. Это не относится к backend и mobilenet версиям одной модели: они совместимы.

Версия дескриптора указывается в конфигурационном файле (см. раздел "Конфигурационные данные" в главе "Модуль Core").

8.4 Пакет дескрипторов

При сопоставлении значительных объемов дескрипторов желательно, чтобы они постоянно находились в памяти для повышения производительности (подразумевается дружелюбное к кэшу расположение данных и согласованность данных). В этом случае полезны *пакеты дескрипторов* (*descriptor batches*). Если дескрипторы оптимизированы для более быстрого создания и уничтожения, то пакеты оптимизированы для длительного использования и лучшего представления данных дескрипторов для аппаратной части.

Как и любой другой объект, пакет создается фабрикой. Кроме типа, должен быть определен размер пакета. Размер — это память, которую пакет резервирует для своих данных. Невозможно добавить больше данных, чем зарезервировано.

Пакет должен быть заполнен данными. Существуют следующие варианты:

- добавление существующего дескриптора в пакет;
- загрузка содержимого пакета из архива.

Следует иметь в виду следующее:

- При добавлении существующего дескриптора его данные копируются в пакет. Это означает, что объект дескриптора может быть безопасно освобожден;
- При добавлении первого дескриптора в пустой пакет происходит начальное выделение памяти. Пакет не выделяет память до этого момента. В этот же момент в пакет копируются внутренние вспомогательные параметры дескриптора (если таковые имеются). Это обуславливает совместимость дескрипторов внутри пакета. При инициализации пакет не принимает несовместимые дескрипторы.

После инициализации можно выполнять операции матчинга с пакетом практически так же, как и для простого дескриптора.

Как и любой другой объект хранения данных, пакет дескрипторов реализует метод ::clear(). Результатом этого метода является преобразование пакета в неинициализированное состояние, за исключением освобождения памяти. Другими словами, ёмкость пакета остается неизменной, и память не перераспределяется. Однако фактическое число дескрипторов в пакете и их параметры сбрасываются. Это позволяет повторно заполнить пакет.

Освобождение памяти происходит при освобождении пакета.

Следует соблюдать осторожность при сериализации и десериализации пакетов. При создании пакета ему назначается буфер памяти фиксированного размера. Размер буфера включается в большой двоичный объект пакета (BLOB) при его сохранении. Поэтому при выделении пакетного объекта для чтения в BLOB убедитесь, что его размер не меньше размера пакета, сохраненного в BLOB (даже если он в данный момент не заполнен целиком). В противном случае загрузка завершится неудачно. Естественно, таким образом, можно десериализовать меньший пакет в больший.

8.5 Извлечение дескрипторов

Экстрактор дескрипторов (*Descriptor extractor*) — это объект, ответственный за извлечение дескрипторов. Как и любой другой объект, он создается фабрикой. Для извлечения дескриптора, помимо исходного изображения, необходимы:

- область детекции лица на изображении (см. главу "Модуль детекции лиц");
- предварительно выделенный дескриптор (см. раздел "Дескриптор");
- предварительно рассчитанные характерные точки (см. главу "Нормализация изображений").

За это действие отвечает объект извлечения дескрипторов. Он представлен простым интерфейсом IDescriptorExtractor и только одним методом extract(). Обратите внимание, что объект дескриптора должен быть создан до вызова extract() путем вызова соответствующего метода фабрики.

Характерные точки используются в качестве набора координат точек интереса, которые в свою очередь определяют области исходного изображения, из которых извлекается дескриптор. Это позволяет извлекать наиболее важные данные для конкретного типа объекта. Например, для сравнения двух лиц необходимо знать, по крайней мере, определенные параметры глаз, носа и рта. Таким образом, мы должны сначала вызвать экстрактор характеристик лица для определения расположения глаз, носа и рта, и перенести эти координаты в характерные точки. Затем экстрактор дескрипторов берет эти координаты и строит по ним дескриптор.

Извлечение дескрипторов является одной из наиболее ресурсоёмких операций. По этой причине можно рассмотреть потоковую обработку. Имейте в виду, что извлечение дескрипторов не является потокобезопасным, поэтому необходимо создать объект extractor для каждого рабочего потока.

Следует отметить, что область детекции лица и характерные точки необходимы только для нормализации изображения - стадии подготовки дескриптора к извлечению (см. Раздел "Нормализация изображений"). Если исходное изображение уже нормализировано, эти параметры можно пропустить. Для этой цели интерфейс IDescriptorExtractor предоставляет специальный метод extractFromWarpedImage().

Реализация извлечения дескрипторов поддерживает выполнение на графических процессорах (GPU).

Интерфейс IDescriptorExtractor предоставляет метод extractFromWarpedImageBatch(), который позволяет извлечь пакет дескрипторов из массива изображений за одну команду. Этот метод обеспечивает более высокое использование GPU и лучшую производительность (см. таблицу Ошибка! Источник ссылки не найден. в приложении A).

Также IDescriptorExtractor возвращает descriptor score для каждого извлеченного дескриптора. Descriptor score это нормализованное значение в диапазоне [0,1], где 1 - нормализованное лицо, 0 - не нормализованное лицо. Это значение позволяет фильтровать дескрипторы, извлеченные из ложноположительных детекций.

8.6 Матчинг дескрипторов

Имеется возможность сравнить пару (или более) ранее извлеченных дескрипторов, чтобы узнать степень их схожести. С помощью этой информации можно реализовать поиск лица и другие аналитические функции.

С помощью функции match, определенной интерфейсом IDescriptorMatcher, представляется возможным сопоставить пару дескрипторов или один дескриптор с пакетом дескрипторов (подробнее о пакетах см. Раздел "Пакет дескрипторов").

Простое правило, которое поможет в выборе хранилища:

- используйте отдельные объекты IDescriptor при поиске среди менее чем ста дескрипторов;
- используйте пакет при поиске среди большего количества дескрипторов.

Распространенной практикой в работе с большими объёмами данных является организация дескрипторов в несколько пакетов, используя один пакет на один рабочий поток для обработки.

Имейте в виду, что матчинг дескрипторов не является потокобезопасным, поэтому необходимо создавать один объект matcher на один рабочий поток.

8.7 Индексация Дескрипторов

8.7.1 Применение HNSW

Для ускорения процесса матчинга дескрипторов для пакета дескрипторов может быть создан специальный индекс. При использовании индекса матчинг проводится в два этапа:

- Во-первых, необходимо построить индексированную структуру данных - index- используя IIndexBuilder. Это довольно медленный процесс, поэтому его не следует выполнять часто. Вы строите индекс, добавляя объекты IDescriptor или IDescriptorBatch и в конце используете метод сборки - IIndexBuilder::buildIndex;
- Как только индекс будет готов, Вы можете использовать его для быстрого поиска ближайших соседей по переданному дескриптору.

Существует два типа индексов: IDenseIndex и IDynamicIndex. Разница в этих интерфейсах: плотный индекс (IDenseIndex) доступен только для чтения, а динамический индекс (IDynamicIndex) - для редактирования: вы можете добавлять или удалять дескрипторы.

Можно построить только динамический индекс. Так как же получить плотный индекс? Ответ - посредством десериализации. Представьте, у вас есть несколько процессов, которые могут потребоваться для поиска в индексе. Один из вариантов - для каждого из процессов построить индекс отдельно, но, как упоминалось ранее, построение индекса очень медленный процесс, и, вы не захотите делать это чаще, чем требуется. Поэтому второй вариант - построить его один раз и сериализовать в файл. Здесь и проявляется разница между плотным и динамическим индексом: форматы, используемые для хранения этих двух типов индексов различны. С точки зрения пользователя разница в том, что плотный индекс загружается быстрее, но он доступен только для чтения. После загрузки нет никакой разницы в производительности с точки зрения поиска на этих двух типах индексов.

Для сериализации индекса используйте методы IDynamicIndex::saveToDenseIndex или IDynamicIndex::saveToDynamicInd. Для десериализации используйте методы IFaceEngine::loadDenseIndex или IFaceEngine::loadDynamicIndex.

Примечание. Индекированные файлы не являются кроссплатформенными. Если вы сериализуете индекс на какой-то платформе, его можно использовать только на этой платформе. Не только операционная система, но и другая архитектура процессора нарушают совместимость.

Примечание. Индекс HNSW (Hierarchical Navigable Small World) не поддерживается на встраиваемых платформах и 32-разрядных десктопных платформах.

9 Модуль проверки Liveness

Существует три алгоритма оценки Liveness – HeadAndShouldersLiveness, LivenessFlyingFaces и LivenessRGBM.

9.1 Определение области головы и плеч (HeadAndShouldersLiveness)

Этот алгоритм оценки определяет, является ли лицо человека реальным или происходит подлог лица (фотография, напечатанное изображение), и подтверждает наличие тела человека в кадре. Лицо должно быть в центре кадра, а расстояние между лицом и границами кадра должно быть в три раза больше, чем пространство, занимаемое лицом в кадре. Лицо человека и область груди должны быть в кадре. Камера должна быть размещена на уровне талии и направлена снизу вверх. Эстиматор проверяет наличие границ мобильного устройства для выявления мошенничества. Поэтому в кадр не должны попадать прямоугольные области (окна, картины и т. д.).

Алгоритм оценки (см. IHeadAndShouldersLiveness в IEstimator.h):

- Реализует функцию *estimateHeadLiveness()*, которая принимает исходное изображение в формате R8G8B8 и структуру *fsdk::Detection* соответствующего исходного изображения (см. раздел “Структура детекции” в главе “Модуль детекции лиц”).

  Оценивает, является человек реальным или это подлог.
  Выводит нормализованную оценку с плавающей точкой в диапазоне [0..1], 1 - реальный человек, 0 - подлог.
- Реализует функцию *estimateShouldersLiveness()*, которая принимает исходное изображение в формате R8G8B8 и структуру *fsdk::Detection* соответствующего исходного изображения (см. раздел “Структура детекции” в главе “Модуль детекции лиц”).

  Оценивает, является человек реальным или это подлог. Выводит нормализованную оценку с плавающей точкой в диапазоне [0..1], 1 - реальный человек, 0 – подлог.

9.2 LivenessFlyingFaces

Эстиматор определяет, является лицо настоящим или это подлог (напечатанное изображение лица).

Алгоритм оценки (см. ILivenessFlyingFacesEstimator в IEstimator.h):

- Реализует функцию estimate(), которая принимает fsdk::Image с допустимым исходным изображением в формате R8G8B8 и структуру fsdk::Detection соответствующего исходного изображения (см. раздел “Структура детекции” в главе “Модуль детекции лиц”).
- Реализует функцию *estimate()*, которая принимает fsdk::Image с допустимым исходным изображением в формате R8G8B8 и структуру *fsdk::Detection*

соответствующего исходного изображения (см. раздел “Структура детекции” в главе “Модуль детекции лиц”).

Эти методы оценивают, являются разные люди реальными или это подлог. Выводит нормализованную оценку с плавающей точкой в диапазоне [0..1], 1 - реальный человек, 0 – подлог.

9.3 LivenessRGBM

Эстиматор определяет, является лицо настоящим или это подлог (напечатанное изображение лица).

Алгоритм оценки (см. ILivenessRGBMEstimator в IEstimator.h):

- Реализует функцию estimate(), которая принимает fsdk::Image с допустимым исходным изображением в формате R8G8B8, структуру детекции соответствующего исходного изображения, fsdk::Image с накопленным фоном (см. раздел “Структура детекции” в главе “Модуль детекции лиц”).
  Оценивает, является человек реальным или это подлог.
  Выводит нормализованную оценку с плавающей точкой в диапазоне [0..1], 1 – реальный человек, 0 – подлог.
- Реализует функцию *update()*, которая требует fsdk::Image с этим кадром, номер этого изображения и ранее накопленный фон. Накопленный фон будет перезаписан после этого вызова.

10 Приложение А. Технические характеристики

10.1 Классификация производительности

Классификация производительности измерялась на двух наборах данных:

- Набор данных, полученный в кооперативном режиме (содержащий 20 тысяч изображений из различных источников);
- Набор данных, полученный в некооперативном режиме (содержащий 20 тысяч изображений).

Таблицы 7 и 8 содержат истинно положительные оценки (TPR), соответствующие ложноположительным оценкам (FPR) для разных версий нейронных сетей.

Таблица 7. Производительность классификации на низких FPR для данных в кооперативном режиме

FPR	TPR CNN 54	TPR CNN 56	TPR CNN 57	TPR CNN 58	TPR CNN 59	TPR CNN 54m	TPR CNN 56m	TPR CNN 59m
10^-7^	0.9765	0.9907	0.9906	0.9910	0.9911	0.9699	0.9652	0.9876
10^-6^	0.9849	0.9914	0.9915	0.9916	0.9915	0.9829	0.9814	0.9904
10^-5^	0.9892	0.9916	0.9917	0.9918	0.9919	0.9887	0.9886	0.9915
10^-4^	0.9909	0.9917	0.9918	0.9919	0.9921	0.9910	0.9910	0.9919

Таблица 8. Производительность классификации на низких FPR для данных в некооперативном режиме

FPR	TPR CNN 54	TPR CNN 56	TPR CNN 57	TPR CNN 58	TPR CNN 59	TPR CNN 54m	TPR CNN 56m	TPR CNN 59m
10^-7^	0.9638	0.9698	0.9723	0.9767	0.9832	0.8813	0.8844	0.9377
10^-6^	0.9773	0.9809	0.9817	0.9839	0.9880	0.9233	0.9229	0.9629
10^-5^	0.9852	0.9871	0.9873	0.9880	0.9908	0.9538	0.9561	0.9794

FPR	TPR CNN 54	TPR CNN 56	TPR CNN 57	TPR CNN 58	TPR CNN 59	TPR CNN 54m	TPR CNN 56m	TPR CNN 59m
10^-4^	0.9896	0.9902	0.9905	0.9909	0.9924	0.9752	0.9757	0.9880

10.2 Размер дескриптора

Таблица 9 показывает размер сериализованных дескрипторов для оценки требований к памяти.

Таблица 9. Размер дескриптора

Версия нейронной сети	Объём данных, байты	Объём метаданных, байты	Общий объём
CNN 54	512	8	520
CNN 56	512	8	520
CNN 57	512	8	520
CNN 58	512	8	520
CNN 59	512	8	520

Метаданные включают сведения о подписи и версии, которые могут быть опущены во время сериализации, если указан флаг NoSignature.

При оценке размера отдельного дескриптора в памяти или требований к хранилищу сериализации с параметрами по умолчанию рекомендуется использовать значения из столбца "Общий объём".

При оценке требований к памяти для пакетов дескрипторов используйте значения из столбца "Объём данных", так как пакет дескрипторов не дублирует метаданные для каждого дескриптора и, таким образом, более эффективно использует память.

Примечание: *эти числа предназначены только для приблизительных вычислений, так как они не включают в себя накладные расходы ресурсов, такие как выравнивание данных в памяти для ускоренной обработки SIMD и т. д.*

11 Приложение В. Словарь терминов

11.1 Дескриптор

Набор характеристик, предназначенных для описания объекта реального мира (например, лица человека). Характеристики вычисляются с помощью алгоритмов компьютерного зрения. Характеристики сопоставляются друг с другом для определения сходства представленных объектов.

11.1.1 Кооперативный режим съёмки и распознавания

Фотосъёмка лица субъекта при условии, что субъект знает о проведении съёмки и содействует получению качественного фотоизображения.

Основные аспекты:

- - Положение головы близко к фронтальному;
- - Нейтральное выражение лица;
- - Отсутствие перекрытий лица (т. е., не мешают волосы, нет шляпы, нет тёмных очков, ничто не скрывает лицо);
- - Отсутствие экстремальных условий освещения (т. е. умеренное освещение, отсутствие прямых солнечных лучей);
- - Устойчивая и хорошо настроенная оптика (то есть отсутствие размытия при движении, глубины резкости и цифровой постобработки, за исключением шумоподавления).

Кооперативная фотосъемка противоположна так называемой съёмке “в диких условиях”, которую еще называют некооперативной съемкой.

11.2 Матчинг

Процесс сравнения дескрипторов. Матчинг обычно реализуется как функция расстояния, применяемая к наборам характеристик и используемая для сравнения расстояний. Чем меньше расстояние, тем ближе дескрипторы, следовательно, тем больше похожи объекты.

Для удобства существуют вспомогательные функции для преобразования расстояния в нормализованный показатель подобия, где 100% означает полную идентичность, а 0% означает полное отличие.