VisionLabs
MACHINES CAN SEE

VisionLabs LUNA SDK Mobile Liveness

Инструкция по эксплуатации

(краткое руководство разработчика)

ООО «ВижнЛабс»

123458, г. Москва, ул. Твардовского д. 8, стр. 1

+7 (499) 399 3361

info@visionlabs.ru

www.visionlabs.ru

Содержание

Введение

ПО VisionLabs LUNA SDK Mobile Liveness представляет собой набор средств разработки (SDK – Software Development KIT), включающий библиотеки и нейронные сети для анализа изображений и работы с биометрическими образцами, который позволяет специалистам по разработке программного обеспечения внедрять механизмы определения проверки реален ли человек на изображении или нет.

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

1 Основные понятия

Добро пожаловать в VisionLabs LUNA SDK Mobile Liveness!

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 Liveness;

- Разбирает процесс обработки лица, и демонстрирует, как один процесс влияет на другой. Это поможет настроить VisionLabs LUNA SDK Mobile Liveness 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 Liveness. 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 Liveness 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 Liveness 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 Liveness 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 Liveness 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 Liveness

VisionLabs LUNA SDK Mobile Liveness 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 Liveness:

VisionLabs LUNA SDK Mobile Liveness содержит следующие модули:

- Модуль CORE. Этот Модуль хранит общие низкоуровневые типы и фабрики VisionLabs LUNA SDK Mobile Liveness. Он отвечает за нормальное функционирование всех остальных модулей, предоставляя настройки доступа и общий интерфейс. Модуль Core также содержит корневой объект VisionLabs LUNA SDK Mobile Liveness, который используется для создания экземпляров всех объектов более высокого уровня;

- Модуль обнаружения лиц. Данный модуль предназначен для обнаружения объектов. Он содержит различные реализации детектора объектов и фабрики;

- Модуль определения параметров. Предназначен для определения реален ли человек на изображении или нет.

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 и анализ лиц. Этот конвейер подробно описывается в следующих главах.

3 Модуль Core

3.1 Общие интерфейсы

3.1.1 Объект Face Engine

Объект FaceEngine является корневым объектом всего VisionLabs LUNA SDK Mobile Liveness. 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.

4 Модуль детекции лиц

4.1 Краткое описание

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

4.2 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").

4.3 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).

4.3.1 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.

4.3.2 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.

4.3.3 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.

4.4 Определение положения лица

4.4.1 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 "Нормализация изображений") для использования с остальными алгоритмами FaceEngine.

4.4.2 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.

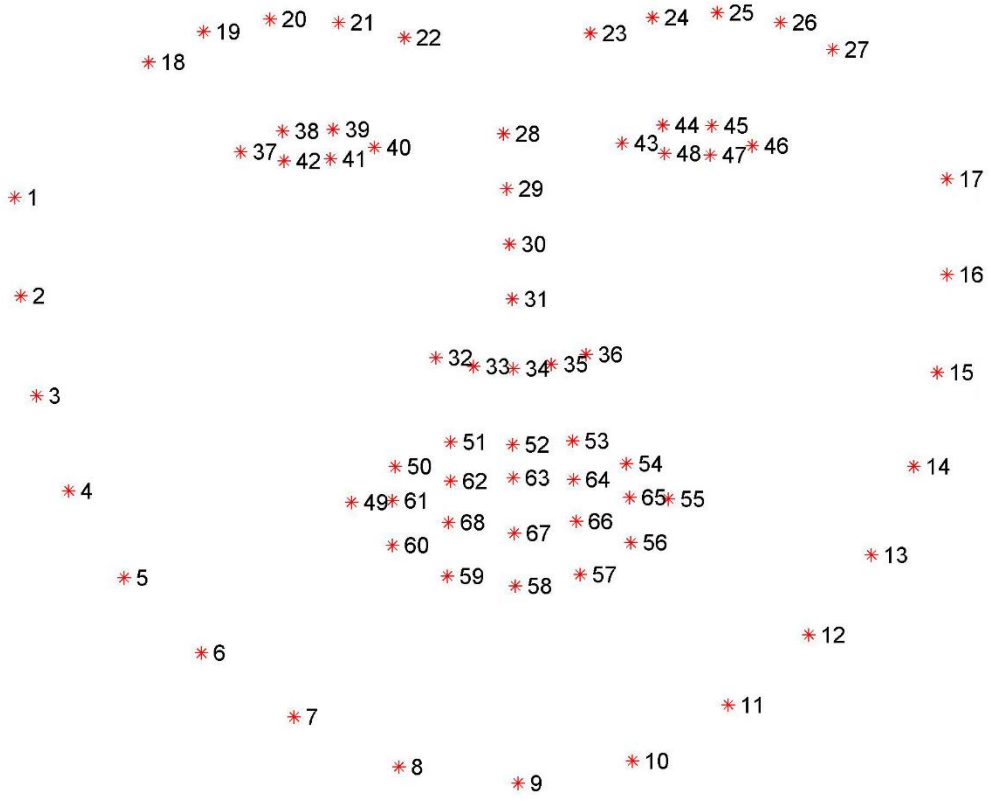

Рисунок 1

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.

5 Нормализация изображений

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.

6 Модуль определения параметров

6.1 Краткое описание

Модуль определения параметров является единственным многофункциональным модулем в VisionLabs LUNA SDK Mobile Liveness. Он разработан как набор инструментов, помогающих оценить, является ли лицо человека реальным или нет. Доступно три алгоритма определения настоящее ли лицо – HeadAndShouldersLiveness, LivenessFlyingFaces и LivenessRGBM.

6.2 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.

6.3 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.

6.4 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.

7 Приложение А. Технические характеристики

7.1 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 2 и 3 содержат истинно положительные оценки (TPR), соответствующие ложноположительным оценкам (FPR) для разных версий нейронных сетей.

Таблица 2. 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 3. 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