МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет Механико-математический

Кафедра Программирования

1. Направление подготовки Математика и компьютерные науки

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

Сухомлинов Дмитрий Игоревич

(Фамилия, Имя, Отчество автора)

Тема работы: Разработка библиотеки поиска по фотографии в базе данных видеоматериалов

**«К защите допущена» Научный руководитель**

Заведующий кафедрой, доцент каф. АФТИ ФФ НГУ,

д.ф.-м.н., профессор лаб. 13 ИАИЭ СОРАН

Марчук А.Г /\_\_\_\_\_\_\_\_ Таранцев И. Г. /\_\_\_\_\_\_\_\_\_\_

(фамилия , И., О.) (подпись, МП) (фамилия , И., О.) (подпись, МП)

«…»………………20…г. «…»………………20…г.

Дата защиты: «…» ……………20…г.

Новосибирск, 2019

**Введение**

В настоящее время видеокамеры стали неотъемлемой частью нашей жизни. Каждый день записывается и сохраняется огромное количество видеозаписей – от любительских, снятых с помощью мобильных телефонов и домашних видеокамер, до профессиональных видеорепортажей, созданных с помощью специализированного видеооборудования. В архиве телекомпании за несколько лет накапливается огромное количество видеозаписей. Задача поиска нужной записи является очень непростой задачей. Традиционный способ решения задачи поиска нужной видеозаписи - дополнение каждой записи рядом текстовых атрибутов (комментариев), по которым и производится поиск.

В последнее время активное развитие получили алгоритмов распознавания и поиска лиц на изображении. Эти алгоритмы уже могут работать достаточно быстро и достаточно надежно. Если в базу данных телекомпании встроить алгоритм поиска лиц по фотографии, то это добавит качественно новую возможность поиска - достаточно предъявить алгоритму поиска лиц фото искомого человека и он найдет все видеозаписи, в которых встречался этот человек.

Очевидно, что прямое сравнение фотографии со всеми видеозаписями телекомпании займет очень много времени. Но если в каждой видеозаписи заранее найти все лица и их описание сохранить в базе данных, то сравнение одного описания лица из предъявленной фотографии со всеми лицами в базе данных займет не так много времени.

**Постановка задачи**

Целью работы является разработка библиотеки индексации всех лиц во всех видеозаписях в некоторой базе данных, включающей сотни и тысячи видеозаписей с целью быстрого поиска всех лиц, похожих на лицо, предъявленное для поиска. Предполагается создать кроссплатформенную библиотеку, легко встраиваемую в любой пользовательский продукт. Данная библиотека должна работать на стандартном компьютере и должна осуществлять следующие операции:

* Обработку видеозаписи из любого числа кадров.
* Обнаружение лиц людей в каждом кадре каждой видеозаписи.
* Объединение лиц, принадлежащих одному и тому же человеку, в отдельную группу с указанием того, на каких кадрах видеозаписи данное лицо встречается.
* Выделение «описание лица» в виде отдельного блока данных, способного сохраняться в стандартной базе данных.
* Поиск конкретного лица в наборе описаний лиц.

Стоит учесть тот факт, что лица мы храним в виде «описаний», сделанных по некоторому шаблону. Качество данного шаблона напрямую зависит от качества изображения. Поскольку, качество изображения может быть разным – от «размытого», снятого на камеру скрытого видеонаблюдения, до очень четкого, сделанного на профессиональную аппаратуру – необходимо сделать возможность настройки порогов сравнения «описаний лиц», хранящихся в нашей базе.

Реализация собственной библиотеки распознавания и сравнения лиц является трудоемкой задачей. В рамках данной работы рассматривается задача использования готовых реализаций библиотек распознавания и сравнения лиц для решения поставленной задачи.

Разрабатываемая библиотека должна обеспечивать:

* Минимальную погрешность при составлении базы «описаний лиц» и поиску по базе данных (ошибки допустимы только при неправильно подобранных порогах сравнения и прочих не консистентных настройках).
* Поиск лица человека по базе данных за минимально возможное время.
* Вывод результирующих данных в наиболее удобном для поиска формате.
* Простоту настройки и обслуживания.

Для тестирования библиотеки необходимо разработать специальное приложение и сравнить результат работы библиотеки поиска лиц с поиском лиц человеком.

**Описание существующих библиотек распознавания лиц**

В ходе работы был рассмотрен ряд встраиваемых библиотек, предназначенных для распознавания лиц:

* OpenCV (Open Source Computer Vision Library) – распространенная многофункциональная библиотека компьютерного зрения. Библиотека с открытым исходным кодом и полностью бесплатна как для академических, так и коммерческих целей, распространяется по лицензии BSD. Поддерживает все наиболее используемы операционные системы (Windows, Linux, macOS, iOS, Android и др.). Включает в себя широкий спектр различных функций, в том числе линейную алгебру, функциональные вычисления, машинное обучение, работу с потоковым видео и фотографиями, в том числе и возможность распознавания лиц методом Виолы-Джонса [1] и сравнения их друг с другом. Из недостатков существенно выделяется ее «перегруженность» (большое количество вспомогательных и второстепенных функций, что делает ее использование затруднительным в определенных случаях), недостаточно полная документация (решение различных проблем намного чаще находится в сторонних источниках, чем в официальных), неудобство отладки и факт общей сложности библиотеки в плане изучения.
* Microsoft Azure Face SDK – технология распознавания лиц от Microsoft. SDK, поддерживается только операционная система Windows. Позволяет обнаруживать лица; выделять их атрибуты на изображении; идентифицировать личность, путем нахождения совпадений в частном репозитории, хранящем до 1 млн. человек; распознавать эмоции, в т.ч. страх, злость и т.п. Библиотека является платной и работает только в облаке. Для работы требуется либо использовать публичное облако Microsoft Azure (что неприемлемо для большинства компаний из-за политики защиты собственного контента), либо развернуть собственное частное облако (что очень дорого).
* FindFace – технология распознавания лиц, основанная на нейронных сетях. Библиотека распространяется на платной основе в двух вариантах:

1. SDK, работающее на локальной машине. Поддерживаются операционные системы Windows и Linux. Требует порядка 3.5 Гб RAM и процессор с поддержкой AVX либо Nvidia GPU с 6, либо 3 Гб памяти соответственно.
2. Модуль распознавания лиц, работающий в облаке, так же с возможность интеграции в любые сервисы и приложения.

Библиотеку отличает возможность распознавания лиц в не самых благоприятных условиях отображения лица (плохое освещение, частично закрытое лицо, очки, борода или усы), возможность распознавания выражения лица, возраста, различных состояний (трезвый/опьяненный), различных углов поворота лица (вдоль горизонтальной оси).

* Face Recognition SDK by id3 – алгоритм распознавания лиц, основанный на нейронных сетях и машинном обучении. Распространяемая на платной основе библиотека, поддерживаемая на операционных системах Windows, Linux и macOS. Библиотеку отличает небольшой размер шаблона «описания лица», позволяющий удобно встраивать его в базу данных и модульный дизайн. Библиотека распространяется в виде SDK, устанавливаемого на компьютер конечного пользователя и не требует подключения к интернету (все вычисления происходят offline на компьютере пользователя).
* FaceX – платформа для распознавания, обнаружения шаблона («описания лица) и сравнения и отслеживания лиц. Распространяется на платной основе и предназначена исключительно для мобильных платформ (Android и iOS). Работа с платформой осуществляется путем отправки HTTP POST запросов, то есть модуль распознавания лиц находится на сервере владельцев сервиса и работает только online. Библиотека не имеет шаблона распознавания лиц, ответом на запрос является json-файл, который указывает определенный набор точек лица (губы, глаза, брови и т.п.) и не может использоваться для однозначной идентификации человека.
* SAFR Recognition SDK – встраиваемая библиотека распознавания лиц. Распространяется на платной основе в двух версиях:

1. SDK для работы offline. Существует для всех наиболее используемых операционных систем (Windows, Linux, macOS, Android, iOS). Минимальные требования компьютера для работы составляют процессор Intel Core i5 или AMD Ryzen 7, NVIDIA GTX 1030, 1Гб RAM для каждой подключенной камеры (для отслеживания в режиме реального времени).
2. Модуль распознавания лиц для работы online.

Предназначена для распознавания и сравнения лиц на фотографиях, видеозаписях и в режиме реального времени. Минусом является отсутствие какого-либо шаблона «описания лица» для сохранения в базу данных.

* VeriLook SDK – технология распознавания лиц, спроектированная для разработчиков биометрических систем. Распространяется на платной основе на операционных системах Windows, Linux, macOS, Android, iOS, требует минимально процессор Intel Core i7-4771 и существует только в режиме SDK для работы offline. Библиотека распознавать несколько лиц на одном кадре в один момент времени, частично закрытые лица, определять пол, возраст, эмоции, повороты головы, имеет шаблон «описания лица» для сохранения в базе данных.
* Luna SDK – встраиваемое SDK распознавания лиц, предоставляемое как в виде готовой программы, так и встраиваемого SDK. Распространяется на платной основе на всех основных операционных системах таких как Windows, Linux и macOS. Библиотека распространяется в двух версиях: front-end и полная версия (обе версии работают offline, на локальной машине). Front-end версия предназначена для простых задач, не требующих извлечения дескрипторов лиц и их сравнения. Полная версия содержит все функции front-end версии и включает в себя извлечение дескрипторов лиц и их сравнение. Библиотека умеет отслеживать набор определенных точек на лице, способна определять пол, возраст, угол поворота лица, также содержит «описание лица» (в рамках данной библиотеки называемое «дескриптором») представляющее из себя набор характеристик, описывающих лицо независимо от посторонних факторов (возраст, прическа, макияж и т.п.).
* 3DiVi Face SDK – библиотека распознавания, сравнения, идентификации и верификации лиц. Распространяется на платной основе, представляет из себя SDK для работы offline на компьютере пользователя и предназначена для операционных систем Windows, Linux, Android и iOS. Минимальный требуемый процессор Intel Core i5-2400. Достоинствами библиотеки является широкий спектр функций, таких как идентификация и верификация лиц, определение возраста, пола, качества фотографий.
* Luxand FaceSDK – еще одна библиотека, предназначенная для распознавания лиц. Распространяется на платной основе в виде SDK, работающего offline, однако, требуется подключение к интернету для активации лицензионного ключа. Библиотека кроссплатформенна (Windows, Linux, macOS), для работы требует процессор с частотой минимум 1.6 ГГц и 256 Мб RAM. Поддерживает функции поиска и сравнения лиц, определения пола, возраста, эмоций. Так же поддерживает возможность создания «описания лица», которое можно хранить в базе данных и использовать для различных операций библиотеки.

Ниже представлена таблица сравнения вышеперечисленных библиотек. В ней наглядно отражены важные факторы, упомянутые во время описания, такие как:

* Каким образом распространяется библиотека – платно или бесплатно;
* На каких операционных системах существует;
* Имеет ли библиотека возможность создавать «описание лица», которое мы могли бы положить в базу данных для дальнейшей работы;
* Каким образом распространяется модуль распознавания лиц – в качестве SDK, предназначенного для работы offline или же в качестве online-сервиса, предназначенного исключительно для работы в облаке.



Рисунок 1. Таблица сравнения основных положений вышеперечисленных библиотек.

Единственная бесплатная библиотека, подходящая под наши цели – это библиотека OpenCV. Однако стоит обратить внимание на механизм, называемый в библиотеке Face Detection [2], - нахождение заданного лица в наборе изображений. Этот механизм представляет из себя нейросеть, которую необходимо предварительно обучить на наборе изображений, содержащих лица, а затем по этому набору изображений осуществлять поиск. Обучение приложения – процесс, требующий понимания работы нейронных сетей. Заставлять неискушенного пользователя заниматься подготовкой тестового набора данных для обучения нейронной сети нецелесообразно, поэтому от использования библиотеки OpenCV было решено отказаться.

В связи с этим, было решено использовать платную библиотеку. Как было показано ранее, количество библиотек распознавания лиц на рынке очень велико. Некоторые из них имеют определенные преимущества перед другими, однако, в целом, эти преимущества нивелируются недостатками. Как следствие – библиотеки распознавания лиц, которая по абсолютно всем параметрам была бы как минимум не хуже всех остальных не существует. Следовательно, невозможно выбрать «правильную» библиотеку, идеально подходящую для наших целей. Таким образом было принято решение создать собственную программную оболочку – некий интерфейс, не зависящий от конкретной библиотеки и позволяющий встроить любую библиотеку минимальными изменениями.

В итоге, для работы была выбрана библиотека Luxand FaceSDK, поскольку ее создатели согласились предоставить бесплатную версию для исследований в рамках дипломной работы.

**Библиотека Luxand FaceSDK**

В ходе реализации данного проекта использовалась сторонний продукт Luxand FaceSDK. Он представляет из себя стороннюю кроссплатформенную статическую библиотеку, позволяющую решать задачи обнаружения и сравнения лиц. Библиотека предлагает API (Application Programming Interface) для обнаружения и отслеживания лиц и «описаний лица», распознавания пола, выражений лица (улыбка, закрытые глаза и т.д.) и распознавания лиц на фотографиях.

Библиотека предоставляет два способа реализаций «описаний лица»:

1. Представляет из себя 70 точек, указывающих на различные места на лице – кончик носа, середина правого и левого глаза, острие подбородка и т.д. В библиотеке представляет из себя массив из 70-ти экземпляров структур, содержащих две переменные типа “int” – координата Х и координата Y каждой из семидесяти точек. Следовательно, размер данной реализации «описания лица» равен “70\*sizeof(int)”. Данная реализация является более «декоративной», т.е. предназначенной для наглядного изображения конкретных точек на фотографии. Использовать данную реализацию «описания лица» для сохранения в базу данных или верификации двух лиц не консистентно, поскольку она (реализация) не заявлена создателями библиотеки как однозначно определяющая то или иное лицо.
2. Представляет из себя структуру, состоящую из 1040 элементов типа char, соответственно, размер экземпляра данной структуры равен 1040 байт. Внутреннее представление этой структуры и каким образом с данной структурой работает библиотека нам не известно. Мы знаем только тот факт, что экземпляр данной структуры является «описанием лица», однозначно сопоставляющим себе некое определенное лицо конкретного человека. Данная реализация «описания лица» используется в данной работе как объект для сохранения в базе данных.

Библиотека доступна на 32-х и 64-х разрядных версиях операционной системы на языках программирования C++, Java, C#, Basic и других. Для реализации программы в ходе данной работы используется 64-битная версия библиотеки, предназначенная для языка C++.

В рамках реализации данной работы используются функции библиотеки, описанные ниже.

Активация библиотеки по ключу:

int FSDK\_ ActivateLibrary (const char \* LicenseKey);

Инициализация библиотеки, аргумент – путь к .dll файлу:

int FSDK\_Initialize(char \* DataFilesPath);

Завершение работы библиотеки:

int FSDK\_Finalize();

Загрузить изображение из буфера данных. В качестве аргументов передается указатель на изображение (куда оно и будет загружено), буфер данных, размеры картинки в пикселях, размер строки картинки в байтах и формат пикселя (глубина цвета изображения):

int FSDK\_LoadImageFromBuffer(HImage \* Image, const unsigned char \* Buffer, int Width, int Height, int ScanLine, FSDK\_IMAGEMODE ImageMode);

При загрузке, изображение загружается в память. Данная функция очищает эту память, если изображение больше не нужно:

int FSDK\_FreeImage(HImage Image);

Функция обнаружения набора лиц на фотографии, указывающая количество и положение лиц в найденном наборе. В качестве аргументов передается изображение, указатель на int, в который будет сохранено количество лиц на изображении, указатель на массив структур, в который будут сохранены положения этих лиц и ограничение по размеру на этот массив в байтах:

int FSDK\_DetectMultipleFaces(HImage Image, int \* DetectedCount, TFacePosition \* FaceArray, int MaxSizeInBytes);

Взять «описание лица» по изображению и положению лица в изображении:

int FSDK\_GetFaceTemplateInRegion(HImage Image, const TFacePosition \* FacePosition, FSDK\_FaceTemplate \* FaceTemplate);

Функция сравнения двух лиц по их «описаниям», возвращающая вероятностную меру их совпадения:

int FSDK\_MatchFaces(const FSDK\_FaceTemplate \* FaceTemplate1, const FSDK\_FaceTemplate \* FaceTemplate2, float \* Similarity);

Возвращаемым значением функций является код ошибки в диапазоне от -28 до 0, где от -28 до -1 – это разного рода ошибки, а 0 – выполнение прошло успешно.

К недостатку библиотеки можно отнести тот факт, что при угле наклона лица больше 30 градусов относительно горизонтали, процент обнаружений лиц резко падает.

**Эксперименты с библиотекой**

Для того, чтобы определить пригодность библиотеки для использования в работе, был проведен ряд экспериментов со сравнением лиц.

Для первого эксперимента была взята выборка из 20 фотографий с лицами людей. Первые 10 фотографий выборки принадлежали одному и тому же человеку, остальные 10 принадлежали другим людям. Для чистоты эксперимента фотографии были преобразованы к одинаковому размеру и примерно одинаковому шаблону (т.е. лицо человека находится примерно в середине, занимает примерно одинаковую площадь на каждой фотографии и т.п.). Было произведено попарное сравнение лиц (каждое с каждым) с различными настройками библиотеки (библиотека позволяет настроить предполагаемое значение угла отклонения лица от горизонтали и размер, к которому входное изображение будет приведено) и небольшими вариациями входных данных (цветные и черно-белые фотографии). Было обнаружено, что получаемые данные в достаточной мере коррелируют друг друга, то есть изменение входных данных, в плане настройки библиотеки или вариации глубины цвета, равномерно изменяет значение результата. Ниже представлены изображения всех лиц в формате 250х250 пикселей и таблица с результатами сравнения этих изображений.



Рисунок 2. Выборка из 20-ти фотографий, используемых для сравнения.

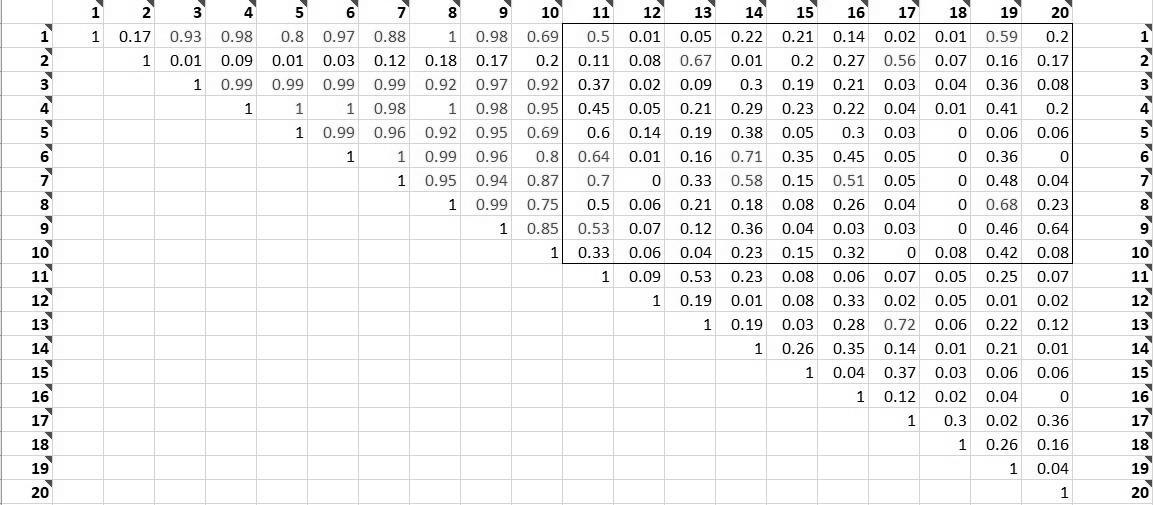
**

Рисунок 3. Как можно заметить, вероятностная мера сравнения одинаковых лиц (от 1 до 10, верхняя левая четверть) достаточно велика, в то время как для разных лиц (правая половина) значение заметно уменьшается.

Можно обратить внимание на то, что значения меры сравнения с рисунком №2 достаточно низкое, хотя на рисунках №1-10 изображен один и тот же человек. Если обратить внимание на сам рисунок, то можно увидеть, что человек на нем изображен в очках. Из этого можно сделать вывод, что посторонние предметы на лице очень сильно уменьшают качество работы библиотеки.

Следственно, выводом является факт того, что данная библиотека является более чем приемлемым SDK для работы над данным проектом.

**Сравнение лиц на изображении**

Одним из наиболее важных моментов является сравнение двух лиц и определение степени их схожести. Назовем эту задачу верификацией [3]. «Описание лица» представляет из себя некую внутреннюю структуру, определенную в конкретной библиотеке распознавания лиц. Верификация двух лиц происходит посредством сравнения этих двух структур. Результатом сравнения двух лиц является вероятностная мера [4] – численное значение   
µ(), принадлежащее отрезку [0; 1], где событие обозначает принадлежность этих двух лиц одному человеку. В зависимости от значения µ() принимается одно из двух бинарных решений:

1. да, лица принадлежат одному и тому же человеку,
2. нет, лица принадлежат разным людям.

Итого мы имеем 2 варианта решения и 2 варианта того, как оно есть на самом деле. Мы получаем 4 возможных положения вещей:Рис. 1 Цветом фона обозначено реальное положение вещей (темный – принять, светлый – отвергнуть), а цветом рамки – решение, принятое алгоритмом (темный – принять, светлый – отвергнуть).

В соответствии с возможными исходами имеется 2 корректных решения и 2 ошибочных:



Рис. 2 Левый столбец представляет из себя описание реального положения вещей, а верхняя стока – выбор алгоритма. Возможны 2 корректных результата, ошибка 1 рода («ложная тревога») и ошибка 2 рода («пропуск цели»).

Лица с двух фотографий принимаются одинаковыми при µ() больше, либо равному некоторому заданному пороговому значению . Соответственно, при уменьшении значения увеличивается вероятность ошибки 1 рода, а вероятность ошибки 2 рода уменьшается, и наоборот, при увеличении значения вероятность ошибки 1 рода уменьшается, а вероятность ошибки 2 рода увеличивается. В соответствии различными задачами есть смысл либо искусственно занизить значение, чтобы получить гораздо большую выборку лиц, пусть и с высокой вероятностью получения ошибочных данных, либо завысить значение, благодаря чему выборка будет меньше объемом, однако точность совпадения заметно возрастет. Помимо этих двух вариантов существует так же вопрос нахождения некоего среднего оптимального значения – такого, при котором вероятности ошибок 1 и 2 рода окажутся примерно равны. Выбор необходимого значения зависит от конкретной задачи и подбирается экспериментально для каждой библиотеки распознавания лиц.

**Сравнение лиц в последовательности изображений**

Приведем в качестве примера 3 изображения лица и определим значение µ (степень совпадения этих лиц) для каждой пары из этого набора лиц.

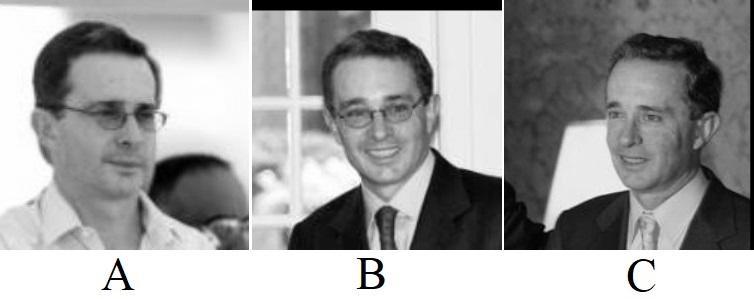


Рис. 4 Выборка из 3-х лиц для попарного сравнения.

При сравнении мы получаем следующие параметры µ:

µ() = 0.99 µ() = 0.92 µ() = 0.69

Отсюда мы видим следующую картину – несмотря на то, что лица, очевидно, принадлежат одному и тому же человеку, значение µ для разных пар сильно отличается. Библиотека с высокой вероятностью (больше 0.9) определяет пару *А* *В* и пару *В С* как лица, принадлежащие одному и тому же человеку. Однако для пары *А С* значение µ резко падает (меньше 0.7), что, при определенных обстоятельствах, заставит нас сделать вывод о том, что лица с изображений *А* и *С* принадлежат разным людям, хотя, судя по результатам двух других сравнений (и нашему реальному взгляду) на всех трех фотографиях изображен один и тот же человек. Нам необходимо, чтобы наша библиотека исключала подобные ошибки.

При поиске изображений в каждом кадре видеозаписи, на разных кадрах будут обнаруживаться очень много похожих друг на друга (практически идентичных) лиц одного и того же человека. При этом, с определенной вероятностью некоторые из этих лиц будут считаться разными (как лица *А* и *C* из примера выше). Например, при съемке интервью двумя камерами после монтажа в итоговой видеозаписи будет происходить регулярное переключение между камерами с разным положением лиц. При этом легко может получиться так, что в начале записи будут чередоваться лица типа *А* и *C* из примера выше, а затем появится лицо типа *В*. Следовательно, возникает необходимость двухэтапной (или двойной) верификации. То есть, сначала на этапе первичной верификации мы ищем максимально похожие с точки зрения алгоритма лица (например, со значением µ ≥ 0.9) и можем с высокой вероятностью считать, что эти лица действительно принадлежат одному и тому же человеку. Для того, чтобы обеспечит оптимальную скорость выполнения, предлагается не сохранять все найденные «описания лица» для одинаковых лиц, а выбирать наиболее качественное описание лица и сохранять только его. Это позволит сильно сократить объем хранимых данных и очень сильно ускорит процесс первичной верификации, поскольку сравнение нужно выполнять только с небольшим числом лиц. Понятие «качество» в данном случае означает то, насколько хорошо данное описание лица передает отличительные особенности данного лица. Это значение определяется каждой библиотекой индивидуально. Например, как уже говорилось в одной из предыдущих глав, «описание лица» библиотеки Luxand FaceSDK представляет из себя структуру размеров 1040 байт, а значение качества – это переменная размером 1 байт, записанная со смещением в 12 байт и принимающая значения от -128 до 127, причем если значение меньше 70, то «описание лица» считается плохим.

В результате первичной верификации мы имеем набор наиболее качественных лиц, в каждом элементе которого с большой долей вероятности (с пороговым значением µ ≥ 0.9) собраны все наиболее похожие лица. Однако, вспоминая ситуацию выше, нам следует помнить, что в получившемся наборе могут быть разные описания одного и того же лица. Для того, чтобы слить разные элементы набора в один, мы проведем процедуру вторичной верификации с меньшим порогом сравнения (например, с пороговым значением µ ≥ 0.65), сравнивая каждый элемент получившегося набора с каждым. То есть на первом этапе мы выбираем очень высокий порог – порог, который существенно выше требуемого для минимизации ошибок 1-го и 2-го рода, а на втором этапе выбираемый порог уже должен минимизировать эти самые ошибки первого и второго рода. Двойная верификация позволяет решить следующие задачи:

1. Высокий порог на этапе первичной верификации позволяет различать лица, определенно принадлежащие разным людям;
2. Выбор наиболее качественного лица на этапе первичной верификации позволяет на этапе повторной верификации при сравнении с меньшим значением µ отсеять действительно разные лица и объединить одинаковые, но не прошедшие верификацию на первом этапе.

Таким образом, первый, более высокий порог позволяет не смешивать лица, действительно принадлежащие разным людям, а второй порог позволяет минимизировать ошибки 1-го и 2-го рода.

**Библиотека распознавания лиц**

Для реализации описанной задачи предлагается создать абстрактный класс для реализации задачи поиска и распознавания лиц. Данный абстрактный класс включает в себя набор чисто виртуальных функций, требующих реализации в классе-наследнике. Под «наследником» в данном случае подразумевается реализация интерфейса с использованием конкретной библиотеки распознавания лиц. Данная концепция позволяет абстрагироваться от конкретного формата входных данных и от определенной библиотеки, предназначенной для распознавания лиц. Это позволяет один раз внедрить разрабатываемую библиотеку в код конкретной базы данных, и ничего не менять в клиентском коде при переходе на новую, более эффективную библиотеку распознавания лиц.

Для начала работы нам необходимо создать объект «Искатель лиц», занимающийся поиском лиц на отдельных изображения и формированием описаний лиц. Дальнейшая работа выполняется только с этим объектом. В первую очередь объект должен быть проинициализирован. Инициализация в данном случае – некая предварительная обработка, которая не относится к непосредственно созданию объекта. Она реализуется индивидуально, различные реализации «искателей лиц» могут иметь разные требования.

Основная работа по обнаружению лиц выполняется в цикле. Из видеозаписи декодируется очередной кадр и извлекается информация об этом кадре – номер кадра; массив данных, представляющих из себя последовательность пикселей, считанных слева-направо снизу-вверх; длина и ширина изображения, глубину цвета. Все эти данные передаются в соответствующую функцию «Искателя лиц», где он уже делает свою работу – проводит первичную верификацию с использованием более высокого порога µ и создает локальную базу «описаний лиц» (т.е. «описания лиц» в некотором предварительном формате – в виде указателей на объекты в памяти). Внутри цикла мы обладаем возможностями внести дополнительные параметры обработки. Например, мы можем обрабатывать только каждый 10-й кадр и т.д.

После окончания обработки видеозаписи необходимо завершить работу обработчика лиц. В это время проводится вторичная верификация с использованием меньшего порога µ, в результате которой сокращается число описаний лиц. Эти описания должны быть занесены в базу данных для долговременного хранения (они будут использоваться при поиске лица по фотографии).

Немаловажным является вопрос самого создания экземпляра класса «обработчика лиц». Поскольку реализация библиотеки является задачей разработчика библиотеки, а вопрос использования этой реализации в конечном приложении базы данных интересует разработчика этого приложения, то предполагается поставлять последнему реализацию в виде, во-первых, динамической библиотеки, закрытой для просмотра извне, и, во-вторых, интерфейса, описывающего функции данной библиотеки; как, когда и в каком порядке они должны быть вызваны. Очевидно, что первым должен быть вызван конструктор той или иной реализации абстрактного интерфейса. Поскольку, как уже упоминалось, сама реализация закрыта для конечного пользователя, а конструктор интерфейса не позволит нам сконструировать объект наследника, мы инкапсулируем конструктор интерфейса, сделав его недоступным, а для создания объекта будем использовать статическую функцию, принадлежащую абстрактному интерфейсу, создающую экземпляр класса наследника.

Интерфейс обработчика лиц носит название **IFaceFinder**. Функции данного интерфейса описаны ниже.

Создание объекта путем вызова статической функции (по своей сути – фабрики объектов):   
IFaceFinder\* faceFinder = IFaceFinder::createFaceFinder();

Инициализация обработчика лиц:  
int result = faceFinder->init();   
// 0 = OK

Функция, обрабатывающая конкретный кадр:  
unsigned char\* data = nullptr;  
int nDX, nDY, nScanLine, nFrameIndex;  
int result = faceFinder->addImage(nFrameIndex, data, nDX, nDY, nScanLine, IFaceFinder::ColorDepth::Bit24);

Функция, генерирующая конечную базу данных, являющуюся результатом работы библиотеки и завершающая работу обработчика лиц:  
faceFinder->finish();

Это не все - нужны методы извлечения описаний лиц из библиотеки.

**Декодирование видеозаписи**

Для тестирования работы библиотеки необходимо иметь возможность декодирования видеозаписи и преобразования ее в формат, совместимый с методом addImage библиотеки распознавания лиц. Для этого, в рамках данной работы было решено реализовать обобщенный интерфейс для декодирования последовательности изображения. Интерфейс получил название **ILoader**. Реализацией так же является наследник данного интерфейса, предназначенный для загрузки и декодирования определенного типа видео (например, видео определенного формата). Ниже представлены функции интерфейса, требующие реализации в наследнике:

Создание объекта путем вызова статической функции (по своей сути – фабрики объектов):   
ILoader\* loader = ILoader::createLoader();   
// result = 0 = OK (не совсем понятно, куда пойдет переменная result?)

Инициализация загрузчика (предварительные действия, подготовка к работе):  
int result = loader->init();   
// result = 0 = OK

Загрузка видеозаписи с диска через путь к файлу:  
int result = loader->loadFile("path/to/file");   
//

Функция, возвращающая информацию о том, имеются ли необработанные кадры:  
bool bHasPicture = loader->hasFrameToRead();   
// result = 0 = OK (куда идет result?)

Функция, возвращающая данные следующего кадра. Переменные для данных передаются в функцию по ссылке:  
unsigned char\* data = nullptr;   
int nDX, nDY, nScanLine, nFrameIndex;   
int result = loader->readNextFrame(&data);   
//

Функция, очищающая память: //Не уверен, что вы имели ввиду это  
loader->ReleaseImage(data);   
//

Функция, пропускающая следующий кадр:  
int result = loader->SkipNextFrame();   
//

Функция, возвращающая количество кадров в видеозаписи:  
int loader->getFramesNumber();   
//

Функция, завершающая работу разгрузчика:  
void loader->finish();   
//

В рамках данной работы был реализован класс **AviLoader**, предназначенный для загрузки и декодирования видеозаписей формата .avi.

**Запись результатов в базу**

Для тестирования работы библиотеки была использована встраиваемая СУБД SQLite. Библиотека была встроена в тестовое приложение на уровне кода. Для использования библиотеки был реализован класс SQLWorker, содержащий 3 публичные функции.

Записать новое видео в базу в базу:  
int writeNewFace(const std::string& pathToVideo, const int framesNumber);  
// result = 0 = OK (не совсем понятно, куда пойдет переменная result?)

Записать «описание лица» в базу:  
int writeFaceData(IFaceFinder\* faceFinder);  
// result = 0 = OK (не совсем понятно, куда пойдет переменная result?)

Содержится ли информация по данному видео в базе данных:я  
bool isDatabaseContainsVideo(const std::string& path) const;;   
// result = 0 = OK

Вернуть все записи «описания лица» faceTemplate с порогом схожести не менее threshold:  
std:: map<double, std::pair<std::string, std::string>> getFacesFromDb(FSDK\_FaceTemplate\* faceTemplate, const double threshold);  
//

**Реализация**

В ходе данной работы было разработано консольное приложение. Приложение было реализовано на платформе Microsoft Visual Studio 2019 на языке С++ стандарта С++17 с использованием библиотек Luxand FaceSDK и Component Object Model библиотеки GetAVIInfo.

Приложение тестировалось на вычислительной машине: *Intel® Core™ i7-7700HQ CPU @ 2.80 GHz, 8 GB RAM, Windows 10 Pro x64.*

Финальное приложение запускается в двух режимах - дополнение базы данных и поиск в существующей базе данных конкретного лица. Во время дополнения базы данных происходят поиск всех лиц в видеозаписи, формирование наборов «описаний лиц» с последующей записью результатов в базу данных. Для работы в данном режиме программа должна быть запущена командной строкой со следующими аргументами:

1. --update – ключ, объявляющий режим обновления существующей базы данных;
2. C:/путь/к/видеофайлу.avi – путь к видеозаписи, данные которой должны быть записаны в базу данных;
3. Число от 0 до 100 – первичный порог сравнения (в процентах);
4. Число от 0 до 100 – вторичный порог сравнения (в процентах);
5. Число от 2 до ∞ (опционально) – проверка только кадров номерами, кратными данному числу.

Результатом работы программы в данном режиме является обновление (создание, если не существует) базы данных с данными указанного файла. База данных содержит в себе 3 таблицы:

1. Videos. Таблица, хранящая обработанные видеозаписи. Имеет три колонки – index (INTEGER, уникальный автоматически присваиваемый ключ), frames\_number (INTEGER, количество кадров в данной видеозаписи), path\_to\_a\_video (STRING, путь к файлу на диске);
2. Faces. Таблица, содержащая «описания лиц». Имеет 3 колонки – index (INTEGER, уникальный автоматически присваиваемый ключ), video\_index (INTEGER, index видеозаписи из таблицы “Videos”, которой принадлежит данное «описание лица»), face\_description (BLOB, «описание лица», сохраненное в бинарном виде);
3. Regions. Таблица, содержащая кадры, в которое конкретное «описание лица» появляется на конкретной видеозаписи. Имеет 4 колонки - video\_index (INTEGER, index видеозаписи из таблицы “Videos”, на которой находятся данные кадры), face\_index (INTEGER, index лица из таблицы “Faces, которое находится на данных кадрах), first\_frame (INTEGER, порядковый номер кадра, на котором появляется данное лицо, начиная с нуля), duration (INTEGER, количество кадров подряд, в течении которых лицо видимо на видеозаписи).

После запуска программы в режиме обновления программа декодирует из указанного файла все кадры, обработает каждый кадр по отдельности, т.е. найдет все лица на каждом кадре по отдельности, сравнит их с учетом заданных порогов сравнения, разобьет на группы по принадлежностям к одному человеку и запишет результаты в базу данных.

Второй режим работы предназначен для поиска лица с фотографии в существующей базе данных. Для работы в данном режиме программа должна быть запущена со следующими аргументами:

1. --find – ключ, объявляющий режим поиска в базе данных;
2. C:/путь/к/фотографии.bmp – путь к фотографии с лицом, которое должно быть найдено в базе данных. Для упрощения работы введено обязательное ограничение – на фотографии должно находиться только одно лицо;
3. Число от 0 до 100 – порог сравнения (в процентах), при котором лицо в базе и лицо с фотографии считаются принадлежащими одному и тому же человеку;

После запуска программы в режиме поиска, программа составит «описание лица» по лицу человека со входной фотографии, сравнит это «описание лица» со всеми «описаниями лиц» в базе данных и выдаст в качестве результата текстовый файл, в котором будут записаны все найденные лица, вероятность совпадения «описаний лиц» которых с «описанием лица» со входной фотографии больше либо равно заданного порога. При этом, найденные лица отсортированы в порядке убывания вероятностной меры µ (от большей к меньшей).

Результат работы:

Итогом работы программы является выходной текстовый файл и набор изображений формата “.bmp”. Выходной файл представляет из себя перечисление следующей информации:

1. Порядковый номер в списке выходного файла;
2. Вероятностная мера µ совпадения данного «описания лица» с «описанием лица», изображенным на входном изображении;
3. Путь к видеозаписи, на которой находится данное лицо;
4. Порядковый номер кадра, на котором встречается данное лицо, изображенное в наилучшем и наиболее четком виде;
5. Полный набор регионов видеозаписи, где данное лицо встречается.

Набор кадров видеозаписи с порядковыми номерами из списка выходного файла в качестве имен представляют из себя вышеупомянутый набор изображений формата “.bmp”.

**Тестирование**

Для тестирования приложения были подобраны 6 видеозаписей. Каждая из этих видеозаписей содержит 2 и более различных человеческих лиц. Так же на каждой из этих видеозаписей присутствует один и тот же человек.

Вставь сюда “раскадровку” видеозаписи - несколько кадров с разными лицами. Можно вставить по 10 картинок каждого ролика с максимально разными лицами. Обязательно указать длительность видеозаписи.

Помимо этого, есть тестовое изображение, не являющееся кадром какой-либо из тестовых видеозаписей, но изображающее человека, присутствующего во всех вышеперечисленных видеозаписях. 

Рис. 5 В базе производился поиск человека с данного изображения. Обозначим за №0.

Задача – во-первых, составить базу данных по набору тестовых видеозаписей, а, во-вторых, по составленной базе данных найти все вхождения человека с тестового изображения.

Во время создания базы данных программа запускалась с первичным порогом и вторичным порогом .

Во время поиска по созданной базе данных использовался порог сравнения .

Сюда нужно добавить время работы - время индексации каждого ролика + сколько лиц было выделено. Затем время поиска по фотографии.

Еще здесь должна быть таблица с вероятностями и фото кадров с наилучшим качеством.

а далее нужно сравнить с “ручным” поиском - найти процент совпадений и различий, число ошибок 1-го и 2-го уровня.

Обращаю внимание, что мы не пытаемся объединять лица из разных роликов в одно лицо!!! Это сделано специально, чтобы при удалении ролика из базы не было проблем переиндексацией и из-за того, что на практике не будет очень много роликов про одно и то же лицо. Поскольку поиск не занимает много времени, то пусть одно и то же лицо найдется в нескольких роликах.

В результате был обнаружен набор следующих изображений, содержащий человека со входного изображения:

**вфывфы



|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**Заключение**

В рамках данной работы были получены следующие результаты:

* Проведено исследование и сравнение существующих библиотек распознавания лиц; была выбрана библиотека Luxand Facesdk;
* Проведено тестирование библиотеки на наборах входных данных с целью определения того, насколько данная библиотека удовлетворяет требованиям, предъявленным к данной работе;
* Разработана встраиваемая библиотека, позволяющая обрабатывать входную видеозапись, искать в ней все встречающиеся лица и объединять в группы лица, принадлежащие одному и тому же человеку;
* Разработано консольное приложение для тестирования разработанной библиотеки, позволяющее создавать базу данных лиц по набору видеозаписей и искать по созданной базе данных определенное лицо;
* Приложение протестировано на наборе видеозаписей с целью определения его работоспособности.

**Литература**

1. [Электронный ресурс] Оценка качества алгоритмов распознавания лиц <https://habr.com/ru/company/ntechlab/blog/329412/>
2. [Электронный ресурс] Face Recognition with OpenCV, документация к библиотеке OpenCV

<https://docs.opencv.org/3.4/da/d60/tutorial_face_main.html>

1. Чернова Н. И. Теория вероятностей: Учеб. пособие / Новосиб. гос. ун-т. Новосибирск, 2007
2. [Электронный ресурс] Paul Viola, Michael Jones: Rapid Object Detection using a Boosted Cascade of Simple features, 2001

<https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf>