МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

Институт (факультет) Институт радиоэлектроники и информационных технологий (ИРИТ)
Направление подготовки (специальность) 09.03.01 Информатика и вычислительная техника
Направленность (профиль) образовательной программы Вычислительные машины, комплексы, системы и сети
Кафедра Вычислительные системы и технологии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

бакалавра
Студента Кузнецовой Полины Викторовны группы 14-В-1

на тему Программная система поиска похожих людей по фотографии

СТУДЕНТ

Кузнецова П.В.
(подпись) (фамилия, и., о.)

РУКОВОДИТЕЛЬ

Гай В.Е.
(подпись) (фамилия, и., о.)

ЗАВЕДУЮЩИЙ КАФЕДРОЙ

Кондратьев В.В.
(подпись) (фамилия, и., о.)

КОНСУЛЬТАНТЫ:

1. По _____
_____ _____
(подпись) (фамилия, и., о.)

2. По _____
_____ _____
(подпись) (фамилия, и., о.)

3. По _____
_____ _____
(подпись) (фамилия, и., о.)

ВКР защищена _____ _
(дата)

протокол №_____

с оценкой_____

Оглавление

Введение

На данный момент существует множество программ, способных идентифицировать человека, но нахождением похожих лиц по фотографии занимается лишь относительно небольшое число разработчиков, хотя в индустрии развлечений существует большая потребность и актуальность нахождения группы похожих лиц из больших баз данных фотографий. Так, большое распространение находят сервисы, занимающиеся поиском похожих лиц в социальных сетях. Многие люди пытаются найти своего двойника или человека, походящего на их идеал. Но в данных сервисах ограничен функционал и имеется платный контент, что уменьшает аудиторию пользователей.

Так же нахождение похожих лиц является актуальным в системе безопасности. На основании накопленных баз данных лиц, попавших в поле зрения камер, производится поиск похожих лиц по этой базе. Система поиска похожих лиц даёт возможность значительно ускорить процесс поиска конкретного человека в видеоархиве и получить статистику его появлений в поле зрения различных камер.

Область информационного поиска не ограничивается задачей кластеризации фотографий людей, с помощью данной программной системы так же можно выполнять поиск похожих объектов.

Система кластеризации фотографий должна максимально быстро и точно подбирать совокупность фотографий, похожих на заданную. Но стоит учесть, что степень похожести людей для каждого человека разная и точность решения данной задачи оценивается экспертом.

1. Требования к продукту

1.1. Назначение разработки и область применения

Разрабатываемая система предназначена для нахождения группы похожих лиц из больших баз данных фотографий. Она может использоваться как отдельное приложение или как составная часть комплекса программ для решения поставленной задачи. Данная система может применяться как в сфере развлечений, так и в целях обеспечения безопасности. К сфере развлечений относятся такие задачи, как поиск двойников, нахождение людей, соответствующих представлениям об идеалах человека, делающего запрос, в некоторых случаях это может быть поиск родственников, поиск каскадёров, заменяющих актёров в опасных сценах и многое другое.

К сфере безопасности можно отнести такие задачи, как выделение группы преступников для опознавания, отслеживание преступников по камерам наблюдения, борьба с терроризмом.

1.2. Технические требования

Требования, предъявляемые разрабатываемой системой к ЭВМ:

1. операционная система с графическим интерфейсом;
2. требования к аппаратному обеспечению определяются операционной системой;
3. мышь, дисплей.

Требования к функционалу разрабатываемой системы поиска похожих лиц по фотографии:

1. система должна предоставить возможность пользователю выбрать базу данных, содержащую фотографии, из которых будут выбираться похожие лица;
2. система должна предоставить возможность пользователю выбрать изображение, по которому будет вестись поиск;
3. система должна произвести предварительную обработку изображения, поступающего на вход;
4. система должна сформировать совокупность признаков, по которым будет осуществляться поиск похожих лиц;
5. система должна выполнить кластеризацию изображений в базе данных фотографий и вывести на экран в качестве результата группу изображений, похожих на загруженное фото.

2. Анализ требований к разрабатываемому продукту

2.1. Выбор операционной системы

В данный момент самыми распространенными операционными системами являются Windows, Linux, MAC OS. Рассмотрим каждую из них.

Windows – это семейство операционных систем, разработанное корпорацией Microsoft. Windows обладает дружественным графическим интерфейсом, в связи с чем пользуется большой популярностью среди пользователей. Данная линейка операционных систем имеет широкий спектр применений. Она используется как на персональных компьютерах, так и на смартфонах. Так же имеется возможность создания серверных операционных систем.

Linux – это семейство Unix-подобных операционных систем, разрабатываемое программистами по всему миру. Оно является свободным программным обеспечением и распространяется бесплатно, что является большим плюсом данной операционной системы. Так же Linux является хорошо расширяемой системой и готовой к установке обновлений и сопровождения. Но Linux является гораздо менее защищенной системой, чем Windows, а так же обладает худшей поддержкой компьютерного оборудования, чем Windows, такого как сканеры, принтеры и т.д.

MAC OS – операционная система, разработанная фирмой Apple, известная раньше, как Macintosh Operating System. Она является Unix-подобной операционной системой. По популярности занимает второе место после ОС Windows.

Создание программного обеспечения для каждой из перечисленных операционных систем слишком затратно и не удобно, поэтому разрабатываемое программное обеспечение не должно быть привязано к конкретной операционной системе и иметь возможность запуска на любой платформе.

2.2. Выбор языка программирования

Для разрабатываемой системы поиска похожих лиц по фотографии могут быть применены следующие языки программирования: C++, R, Python. Рассмотрим подробнее каждый из них.

Python – высокоуровневый язык программирования, который хорошо интегрируется с существующими программами и обладает повышенной читаемостью кода. Но в то же время для его использования необходим интерпретатор, что не всегда удобно.

R – язык программирования, который используется для статистической обработки данных, а так же для работы с графикой. Данный язык специализирован на работе с матрицами, но доступ к отдельным элементам данных матриц является долгим. Так же существенным недостатком является сложность в использовании для пользователя, что не приемлемо для разрабатываемой системы.

C++ – язык программирования, обладающий высокой скоростью выполнения программного кода, а так же поддерживающий работу с графикой. Язык программирования, имеющий в своем составе богатую стандартную библиотеку. Данный язык является популярным для разработки прикладных приложений, вследствие чего доступно много ресурсов для обучения и решения проблем. Так же множество прикладных интегрированных сред разработки поддерживают данный язык программирования.

Проанализировав плюсы и минусы рассматриваемых языков программирования, можно сделать вывод, что наилучшим вариантом будет использование языка C++.

2.3. Выбор среды разработки

В данном проекте будет использоваться интегрированная среда разработки (IDE). Данная среда представляет собой совокупность программных средств, обеспечивающих поддержку всех этапов разработки проекта, начиная от написания кода и заканчивая отладкой и компиляцией.

Сейчас существует множество интегрированных сред разработки, как платных, так и свободных. Большинство из них предназначены для написания программ на каком-то одном языке программирования. Однако, существуют и такие IDE, которые поддерживают сразу несколько языков программирования. Именно такие среды разработки и будут рассматриваться для написания данной работы.

Microsoft Visual Studio Express – одна из самых распространенных сред разработки. Данная IDE распространяется в открытом доступе для отдельных разработчиков и доступна только для проектов с открытым исходным кодом. Хотя Visual Studio является платформозависимой средой разработки, но поддерживает разработку программ под системы Windows, iOS и Android.

Eclipse – свободная среда разработки, изначально создаваемая для Java-приложений, сейчас же поддерживает и такие языки программирования, как C, C++, Python и другие. В этой IDE имеется возможность создания графических интерфейсов, построения диаграмм, составления отчетов, тестирования и многое другое. Но всё-таки Eclipse больше нацелен на разработку Java-приложений и при создании программного обеспечения на языке, выбранном для разрабатываемой системы, а именно на языке C++, возникает множество сложностей при установке и дальнейшей работе с этой средой разработки.

Qt Creator – свободная кроссплатформенная IDE, созданная для упрощения разработки приложений на разных платформах, в своей основе содержит библиотеку Qt. Данная среда поддерживает такие языки программирования, как C, C++ и использует QML в качестве языка описания интерфейсов. Так же она допускает возможность использования визуальных средств разработки графического интерфейса, используя QtWidgets или QML, то есть имеется поддержка встроенного дизайнера интерфейсов. В Qt существует большой функционал для работы с изображениями и файлами, что очень важно для разрабатываемой системы. Так как Qt Creator является очень распространенной средой разработки, в нем присутствует постоянная поддержка и разработка. Qt Creator поддерживает работу с несколькими компиляторами и отладчиками.

Учитывая все достоинства и недостатки рассмотренных IDE, в качестве среды разработки был выбран кроссплатформенный фреймворк Qt Creator.

2.4. Обзор существующих систем поиска похожих лиц по фотографии

Все существующие системы поиска похожих лиц по фотографии используют для распознавания и кластеризации изображений методы, основанные на использовании нейронных сетей. Существенным недостатком программ, представленных пользователям на данный момент, является то, что их возможно использовать только в режиме онлайн. Так же большинство популярных сервисов требуют внесения денежных средств, хотя результат поиска не гарантируют.

Рассмотрим несколько наиболее распространенных сервисов поиска похожих людей по фотографии.

Find Face – веб-сервис, разработанный в России. Осуществляет поиск людей в социальной сети «ВКонтакте» по фотографии. В основе лежит алгоритм распознавания лиц FaceN, основанный на нейронных сетях.

Результат работы данного сервиса представлен на рисунке 1.

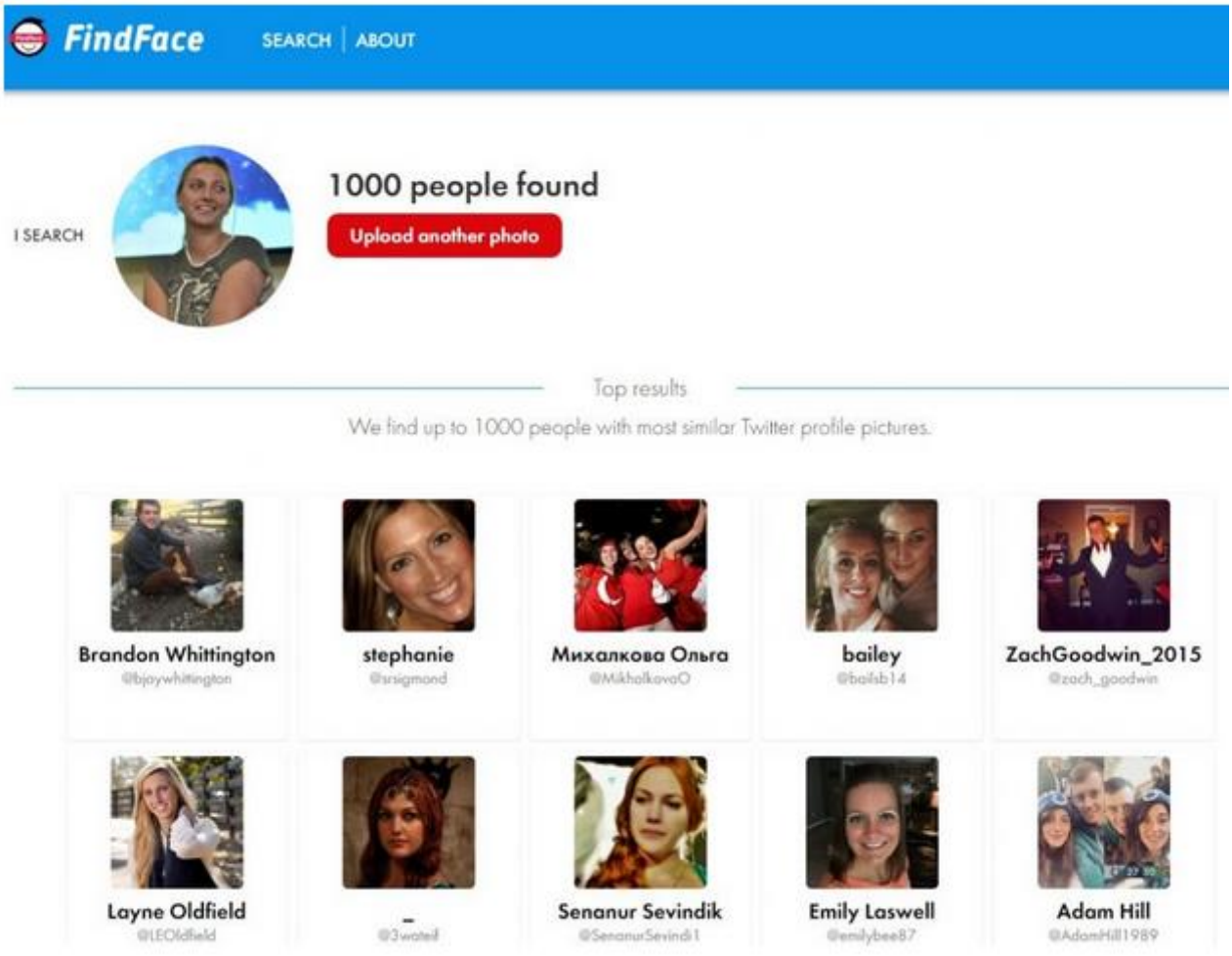


Рисунок 1 - Результат работы сервиса Find Face

Существенными недостатками данного сервиса являются:

- необходимость регистрации в социальной сети «ВКонтакте»;

- наличие платного контента (после 30 попыток поиска взимается сумма от 150 до 450 рублей в месяц);

- поиск не всегда корректно срабатывает.

Twin Strangers – сайт для нахождения двойников, созданный разработчиками из Дублина. За время работы данного сайта было обнаружено несколько пар невероятно похожих людей. Пример приведен на рисунке 2.

Рисунок 2 – Результат работы сервиса Twin Strangers

Но так же у него есть недостатки:

- сервис полностью платный (350 рублей в месяц);
- поиск осуществляется по списку зарегистрированных пользователей, то есть если искомый человек не посещал данный ресурс, найти его не удастся.

Так же большое распространение для поиска похожих людей получили поисковые системы, такие как Google и Яндекс. Технология распознавания изображений, использующаяся в данных сервисах, базируется на компьютерном распознавании и машинном обучении.

На рисунках 3 и 4 изображен результат работы поиска с помощью Google.

Рисунок 3 – Изображение поданное на вход Google поиска

Рисунок 4 - Результат работы поиска с помощью Google

В целом, использование Google и Яндекс поиска удобно и доступно, но имеются и недостатки:

- в основе лежат алгоритмы нахождения похожих изображений, а не конкретно лиц, вследствие чего поиск чаще всего срабатывает некорректно.

Таким образом, все рассмотренные системы возможно использовать только в режиме онлайн, а так же они либо требуют внесения денежных средств либо осуществляют не всегда корректный поиск.

2.5. Выбор подхода к решению задачи поиска похожих лиц по фотографии

Алгоритм нахождения похожих лиц по фотографии можно разделить на 2 части. Первая часть связана с нахождением признаков изображения, вторая – непосредственно с принятием решения, то есть кластеризацией. Для каждой из частей требуется подобрать свой подход к решению задачи. Формирование признаков изображения может осуществляться несколькими способами. Рассмотрим самые распространенные из них.

- с помощью преобразования Фурье;
- с помощью вейвлет-преобразования;
- с помощью алгоритмов, основанных на теории активного восприятия.

1) Преобразование Фурье любой функции вещественной переменной ставит в сопоставление другую функцию вещественной переменной. Функция, поставленная в сопоставление, определяет коэффициенты, которые можно найти при разложении функции, поданной на вход, на элементарные составляющие.

Преобразование Фурье задается в интегральном виде по формуле (1).

$$f(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-ix\omega} dx \qquad (1)$$

Этот метод подходит для вычисления признаков изображения, так как любое изображение можно представить в виде двумерного сигнала. Но он является очень сложным, так как, во-первых, при вычислении используются интегральные функции, а во-вторых, для того, чтобы получить спектр изображения нужно применить одномерное преобразование Фурье для всех строк изображения, а затем для всех столбцов. Пример преобразования Фурье приведен на рисунке 5.

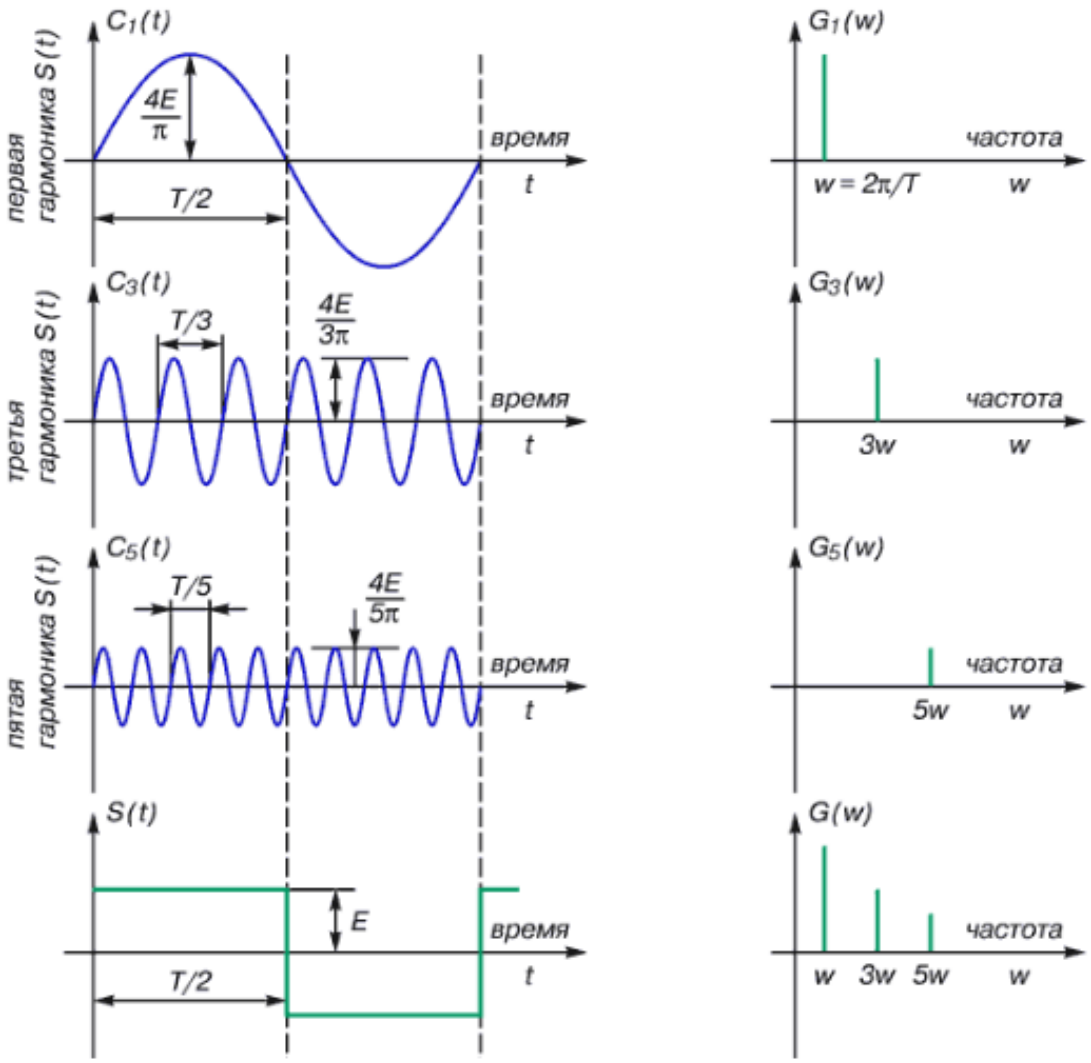


Рисунок 5 –Преобразование Фурье

2) Вейвлет – это математическая функция, с помощью которой можно проанализировать частотные компоненты данных. Графически функцию можно представить в виде волнообразных колебаний, амплитуда которых вдали от начала координат уменьшается до нуля. В общем случае сигнал анализируется в плоскости вейвлет-коэффициентов, а именно масштаб – время – уровень. Они определяются с помощью интегрального преобразования сигнала. В отличие от спектров Фурье, полученные в результате преобразований вейвлет-спектограммы четко определяют привязку спектра особенностей сигнала ко времени.

Вейвлет-преобразование является интегральным и представляет собой обобщение спектрального анализа. Оно представляет сигнал в частотно-временном виде, получая его из временного сигнала.

Вейвлеты – это математические функции, локальные во времени и по частоте. Данные функции получаются в результате изменений одной базовой функции, таких как сдвиг или растяжение. Пример вейвлет-преобразования приведен на рисунке 6.

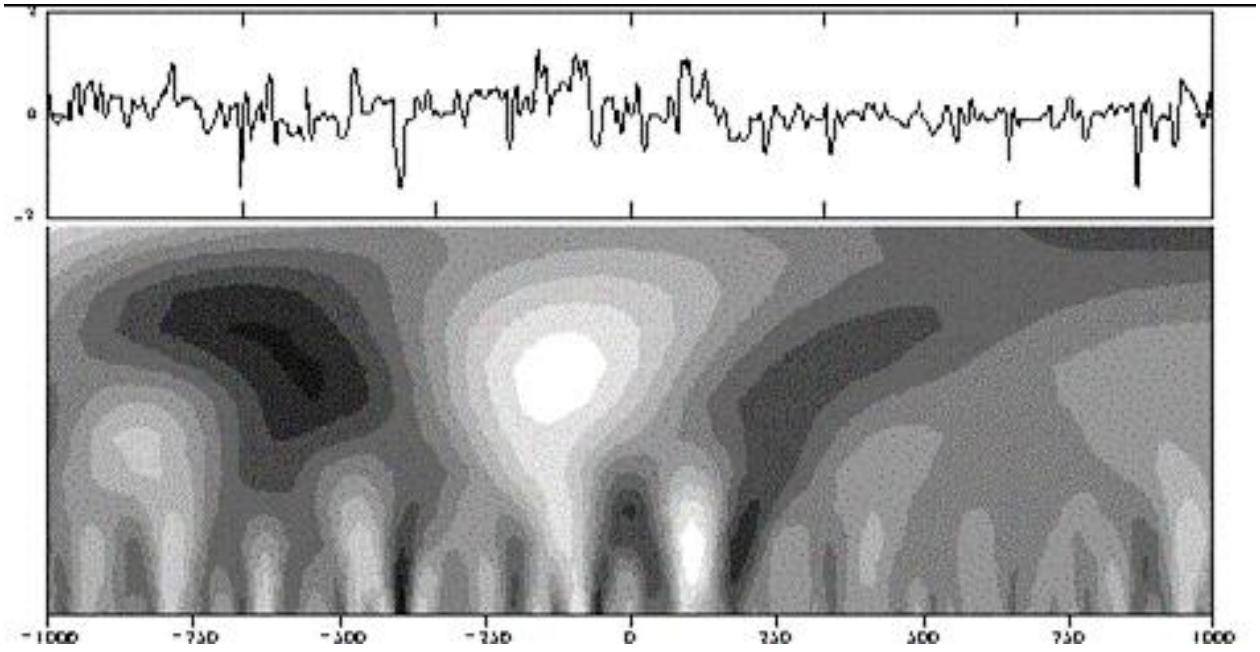


Рисунок 6 – Вейвлет-преобразование

3) Метод теории активного восприятия нацелен на этап формирования исходного описания, то есть подготовки данных и этап формирования признаков, то есть этап анализа.

Обобщенная схема обработки изображения с помощью теории активного восприятия приведена на рисунке 7.

Рисунок 7 – Обобщенная схема вычисления признаков на основании теории активного восприятия

Рассмотрим подробнее каждый шаг алгоритма.

Q-преобразование реализуется на этапе формирования исходного описания. Оно заключается в приведении всех яркостей пикселей изображения к интервалу [0;1] и в последующем сложении сегментов исходного изображения для облегчения вычисления

признаков изображения. В результате *Q*-преобразования формируется матрица «визуальных» масс.

«Визуальная» масса изображения – это мера, ставящаяся в соответствие изображению или подобласти этого изображения.

Матрица «визуальных» масс строится на основании масс каждой из частей изображения по формуле 2.

$$g(t) = \sum_{k=(t-1)*L+1}^{t-L} f(k), t = \overline{1,N}\ , (2)$$

где L – количество отсчетов, входящих в сегмент, N – число сегментов, на которые разбивается изображение, g – результат применения *Q*-преобразования к сигналу f, в роли которого выступает изображение, $g(t)$ – t-ый отсчет сигнала g.

После этого к каждой матрице применяются фильтры, описанные в теории активного восприятия. Эти фильтры схожи с фильтрами Уолша, разница между ними лишь в применении к изображению, так как фильтры из теории активного восприятия применяются к изображению после *Q*-преобразования.

В создаваемой системе используется алгоритм, основанный на теории активного восприятия. Выбран именно этот алгоритм, так как в отличие от методов с использованием преобразования Фурье и вейвлет-преобразования, алгоритм, основанный на теории активного восприятия, не требует вычислений большой сложности, а значит, повышается скорость работы программы.

Теперь рассмотрим вторую часть алгоритма нахождения похожих лиц по фотографии, а именно – кластеризацию. Она может осуществляться несколькими способами:

- с использованием нейронных сетей;
- с использованием метода ближайших соседей (k-means)

Существует множество методов кластеризации, основанных на нейронных сетях. Их объединяет то, что все они основаны на способе машинного обучения, под названием обучение с учителем. В ходе этого метода система обучается на основании пар «стимул-реакция», по которым впоследствии она сама сможет самостоятельно прогнозировать ответ для любого объекта.

При использовании классических нейросетевых архитектур был выявлен ряд недостатков:

- большая размерность изображений порождает сложные нейронные сети;

- из-за большого количества параметров увеличивается время и вычислительная сложность процесса обучения, а следовательно замедляется работа всей системы;

- при повышении эффективности системы так же увеличивается и вычислительная сложность решения задачи, а следовательно и время её выполнения;

- отсутствие инвариантности входного сигнала.

Данные недостатки могут оказаться критическими при решении поставленной задачи.

Метод ближайших соседей считается наиболее популярным методом кластеризации. Он основан на обучении без учителя, что значительно упрощает его применение.

Обучение без учителя – это способ машинного обучения, в ходе которого система выполняет поставленную задачу без вмешательства экспериментатора. Этот способ широко применяется задач обнаружения взаимосвязи между объектами, к которым также относится задача кластеризации.

Сам алгоритм k-means построен таким образом, что при его использовании достигается минимизация суммарного квадратичного отклонения точек кластеров от центров этих кластеров. То есть используется формула 3.

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2, (3)$$

где k— число кластеров, S_i— полученные кластеры, i = 1, 2, … , k и μ_i — центры масс векторов $x_j \in S_i$.

В разрабатываемой системе в качестве способа кластеризации был выбран метод ближайших соседей. Этот выбор обусловлен тем, что для метода поиска похожих лиц методом k-means не требуется время на обучение, а затраты памяти минимальны, так как нет необходимости хранить сложную нейронную сеть.

3. Разработка структуры системы поиска похожих лиц по фотографии

3.1. Разработка общей структуры системы

Задача нахождения похожих лиц по фотографии является задачей кластеризации. Она заключается в разбиении заданной выборки объектов на непересекающиеся подмножества, называемыми кластерами, так, чтобы каждый кластер состоял из схожих объектов, а объекты разных кластеров существенно отличались.

Выполнение данного алгоритма осуществляется в три этапа:

1. нормализация изображения;
2. формирование системы признаков;
3. кластеризация (принятие решения).

Для реализации первых двух этапов используется теория активного восприятия, для третьего – метод ближайших соседей. Обобщенная структура системы представлена на рисунке 8.

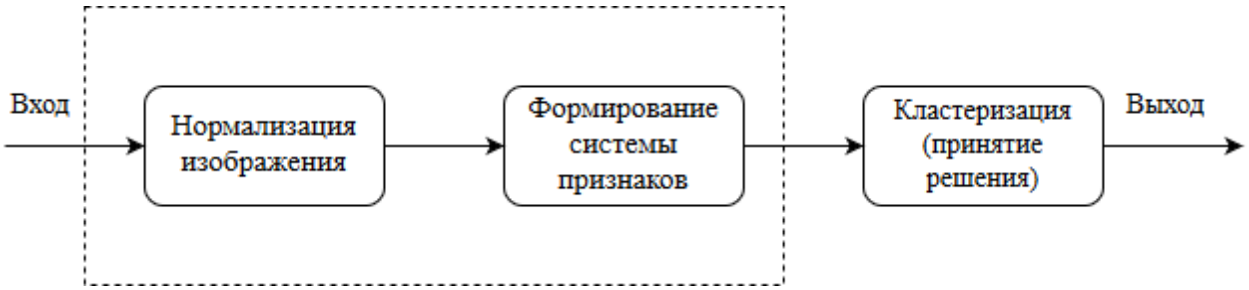


Рисунок 8 – Обобщенная структурная схема

На вход системы подается предварительно обрезанное, повернутое, масштабированное и приведенное в градации серого изображение. После чего оно проходит три этапа обработки и на выходе получаем группу изображений, отобранных по определенному признаку из сформированной заранее базы данных.

В разрабатываемой системе база данных имеет собственный формат. Она представлена в виде текстового файла, в котором содержится информация о каждом изображении базы. Описание любой фотографии содержит следующие атрибуты: имя файла, количество блоков, на которые делится изображение, список признаков. Каждый атрибут располагается в новой строке. Признаки описаны в виде вещественных чисел, в которых целая и дробная части разделяются точкой. Типовое количество знаков после десятичной точки равно пяти, согласно стандартному количеству цифр в дробной части при преобразовании числа в строку в Qt.

Общая структура базы данных приведена на рисунке 9.

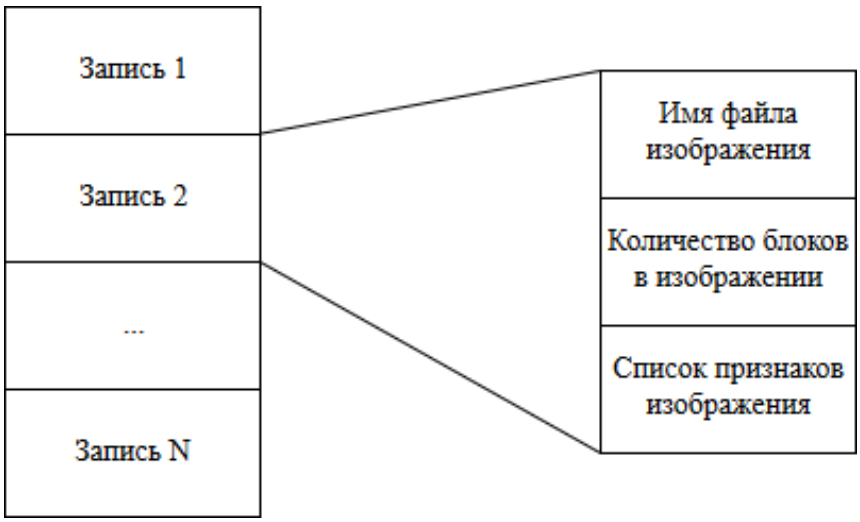


Рисунок 9 – Структура базы данных

Расширенная структурная схема системы нахождения похожих лиц по фотографии приведена на рисунке 10.

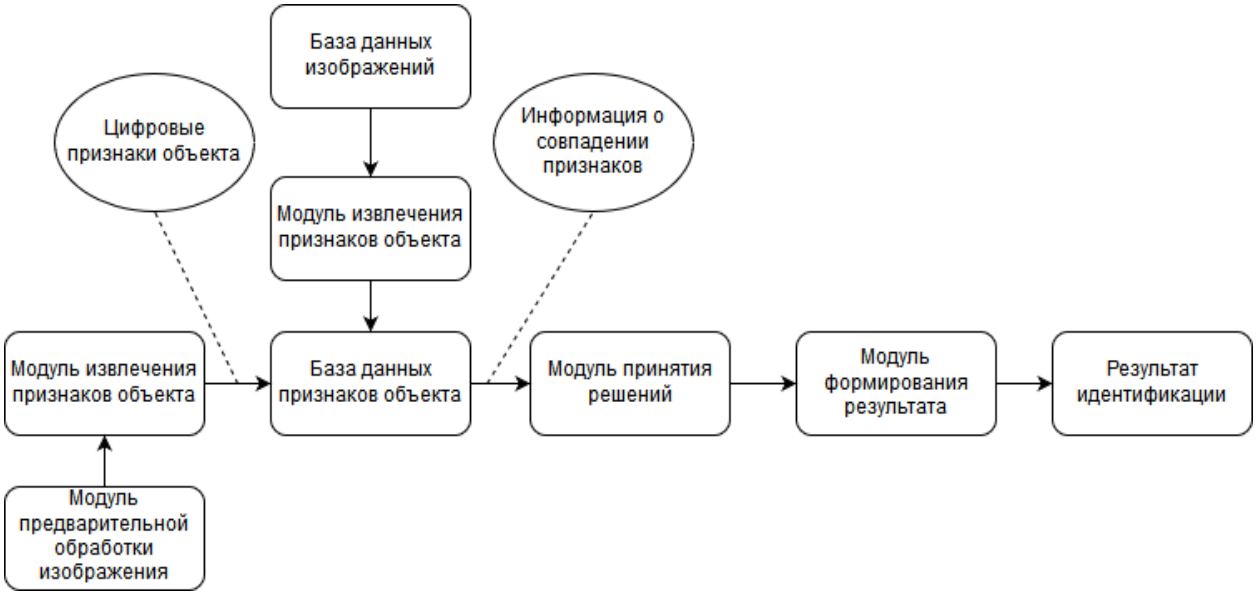


Рисунок 10 – Расширенная структурная схема

3.2. Разработка алгоритма предварительной обработки данных

Перед преобразованием изображения в программе с графическим интерфейсом пользователю предоставляется возможность задать положение глаз на фотографии. Исходя из предоставленных пользователем данных, изображение поворачивается, масштабируется и обрезается. В результате данных преобразований входное изображение приобретает формат, понижающий воздействие фоновых цветовых и яркостных составляющих входного изображения на процесс распознавания. Таким образом, глаза человека на входном изображении располагаются на одной горизонтальной прямой, а расстояние между ними равно 1/3 ширины получившегося изображения. Само изображение приводится к размеру *N*x*M* пикселей.

Исходя из того, что цветовые характеристики изображения не оказывают значительного влияния на результат по сравнению с яркостными, для упрощения дальнейших расчётов фотография преобразуется в градации серого.

В процессе данного преобразования происходит окрашивание целевого пикселя на основе цветов пикселей исходного изображения. Такой порядок окрашивания выбран для того, чтобы избежать возникновения пробельных линий на целевом изображении.

3.3. Разработка алгоритма нормализации изображения

Нормализация изображения производится путем *Q*-преобразования и заключается в приведении яркостей пикселей к интервалу [0;1]. Исходя из того, что размер преобразованного изображения *N*x*M*, получаем нормализованное изображение такого же размера.

Нормирование изображения реализуется с помощью следующего алгоритма:

1. определяется минимальный элемент матрицы яркостей изображения;
2. минимальный элемент вычитается из каждого элемента матрицы;
3. определяется максимальный элемент из получившихся в результате расчётов;
4. каждый элемент матрицы делится на найденный максимальный элемент.

Для увеличения быстродействия алгоритма, минимум и максимум ищутся одновременно, то есть в одном цикле. В таком случает, для соответствия работе исходного алгоритма, после выполнения цикла поиска минимума и максимума, требуется из максимума вычесть минимум.

Пример выполнения нормирования для изображения, размером 8x8 пикселей, приведен на рисунке 11.

0.1	0.1	0	0.3	0.1	0.1	0	0
0.2	0.2	0	0	0.2	0.2	0	0.7
0	0	0.8	0.7	0	0.2	0.8	0.7
0.1	0	0.9	1	0	0	0.9	1
0.1	0.1	0	0	0.1	0.1	0	0
0.2	0.2	0	0	0.2	0.2	0	0
0	0	0.8	0.7	0	0	0.8	0.7
0	0.9	0.9	1	0	0	0.9	1

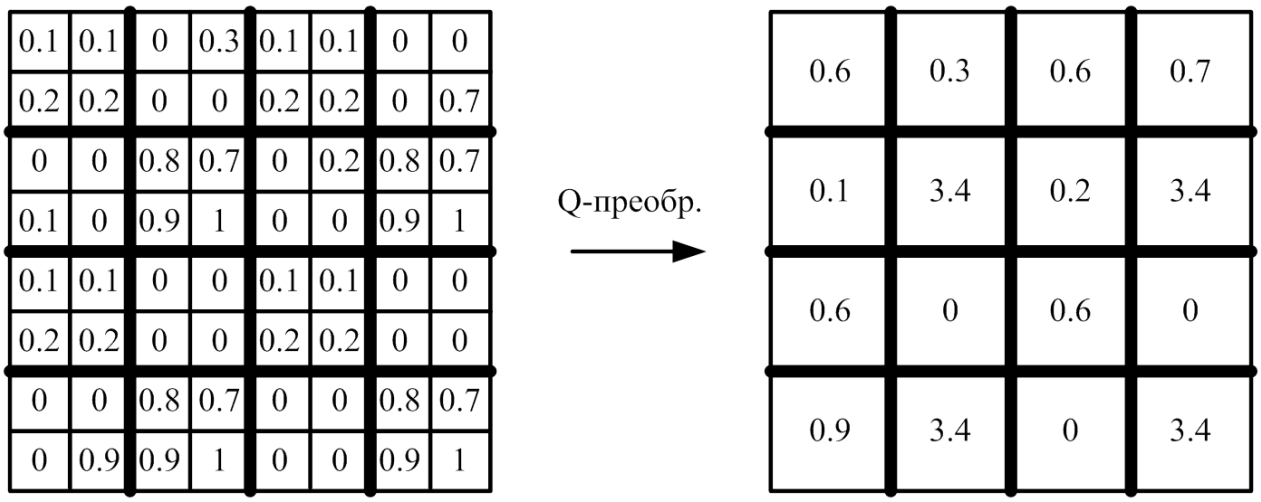
Матрица I
размером 8×8 $(N \times M)$ пикселей

Рисунок 11 – Результат выполнения нормирования

3.4. Разработка алгоритма формирования системы признаков

По нормализованным изображениям формируется система признаков, для чего используется матрица «визуальных масс». Опытным путем было выявлено, что наиболее эффективная работа алгоритма достигается при делении изображения на 16 частей по вертикали и 16 частей по горизонтали. Исходя из того, что размер преобразованного изображения NxM пикселей, получаем x частей, размером kxm пикселей. Каждая часть делится на блоки таким образом, чтобы в итоге получилась матрица размером 4x4. В данном случае, каждая часть делится на блоки, размером axb пикселей. Для каждого из блоков вычисляется сумма нормализованных яркостей пикселей, входящих в блок. Данное преобразование называется Q-преобразованием, в результате которого формируется матрица визуальных масс, размером 4x4 для каждой из частей обрабатываемого изображения.

Пример Q-преобразования для изображения размером 8x8 представлен на рисунке 12.



Матрица I
размером 8×8 $(N\times M)$ пикселей

Матрица «визуальных масс»
размером 4×4 пикселей

Рисунок 12 – Схема Q-преобразования

Далее для каждой матрицы «визуальных масс» применяются фильтры, описанные в теории активного восприятия. Все 16 фильтров приведены на рисунке 13.

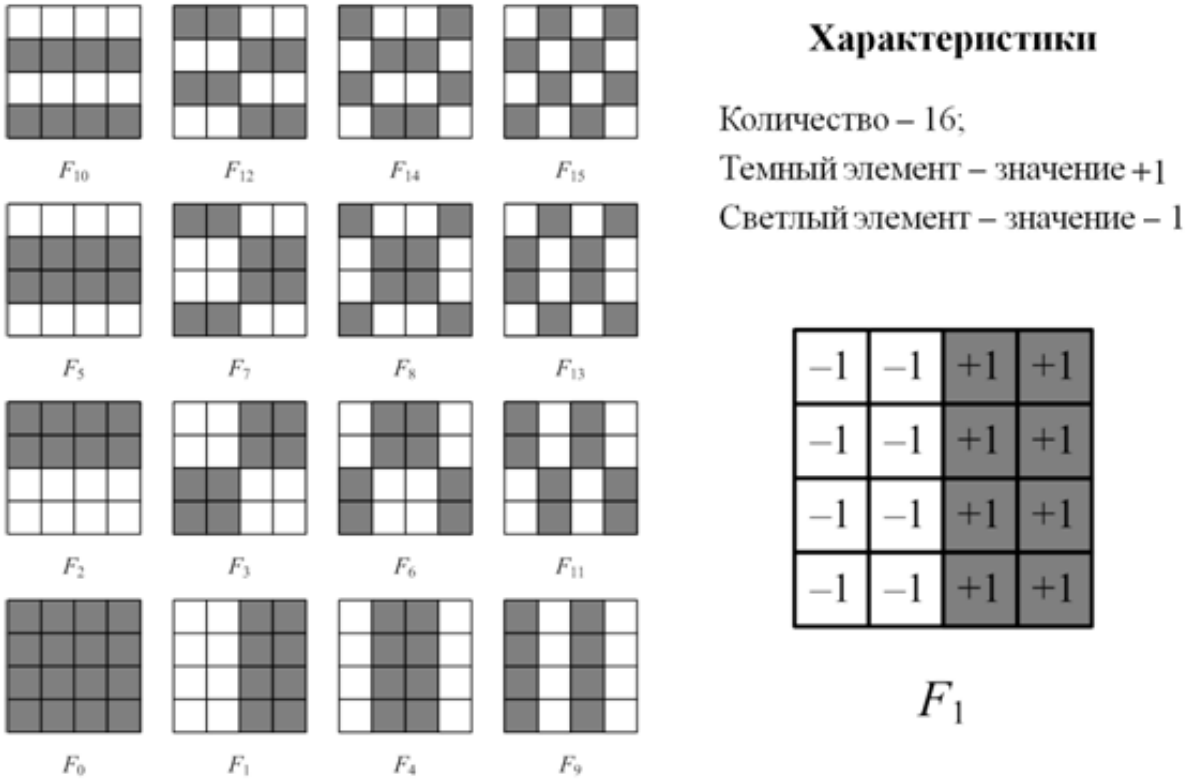


Рисунок 13 - Фильтры, описанные в теории активного восприятия

После этого формируется вектор спектральных коэффициентов за счёт перемножения матрицы визуальных масс на каждый из фильтров.

Таким образом, базовым в теории активного восприятия является *U*-преобразование, которое представляет собой последовательное применение к изображению *Q*-преобразования и системы фильтров.

В исследовательской работе используется именно *U*-преобразование, так как оно имеет минимальную вычислительную сложность за счет того, что в его составе преобладают элементарные вычислительные операции, такие как сложение и вычитание.

3.5. Разработка алгоритма принятия решений

Алгоритм принятия решения заключается в кластеризации изображений. На этапе кластеризации происходит отбор группы фотографий по признакам, которыми являются элементы вектора спектральных коэффициентов.

Сначала сравниваются знаки значений векторов. При рассмотрении отбрасываются значения, которые по абсолютному значению менее, чем 20% от максимального значения элемента в текущем векторе спектральных коэффициентов. Эта операция производится для уменьшения влияния нестабильных (маленьких) значений на результат. Далее рассчитывается процент совпадений и в результат включаются те фотографии, чей процент оказался больше порогового значения, выбранного экспертом.

4. Разработка программных средств

Программная система нахождения похожих лиц по фотографии в целом разделяется на 2 больших части. Первая часть представляет собой графический интерфейс пользователя, с помощью которого пользователь программы осуществляет взаимодействие с системой. Вторая часть является скрытой и осуществляет все внутренние расчеты и преобразования.

Рассмотрим подробнее интерфейсную часть разрабатываемой программной системы. Она состоит из двух окон. С помощью первого окна пользователь формирует базу данных, по которой будет вестись поиск. Внешний вид этого окна представлен на рисунке 14.

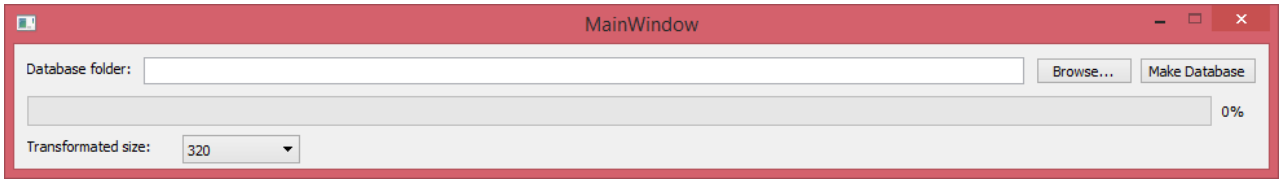


Рисунок 14 – Окно формирования базы данных фотографий

Обработчики событий окна (нажатие на кнопки, взаимодействие с окном и т.д.) описываются в файлах исходного кода makedatabasewindow.cpp, makedatabasewindow.h, представленных в приложении. В файле makedatabasewindow.cpp описаны следующие функции системы: по нажатию кнопки «Browse…» пользователю предоставляется выбрать путь к папке, в которой хранятся фотографии, из которых буде формироваться база данных изображений. По нажатию кнопки «Make Database» осуществляется проверка, указана ли папка с базой данных. Если она указана, то с помощью перебора всех файлов в данной папке по их именам задается положение глаз человека на данной фотографии. По этим данных вычисляются признаки каждого изображения и результаты заносятся в созданный программой файл Database.txt в соответствии с форматом, описанным ранее. Данная реализация создания базы данных написана для тестирования основной программы. При дальнейшем применении разрабатываемой системы нахождения похожих лиц по фотографии возможно использование базы данных, сформированной другим способом. В таком случае обязательным является лишь наличие текстового файла, оформленного по определенным правилам, указанным выше.

С помощью второго окна осуществляется непосредственно поиск похожих людей по загруженной пользователем фотографии. Внешний вид этого окна представлен на рисунке 15.

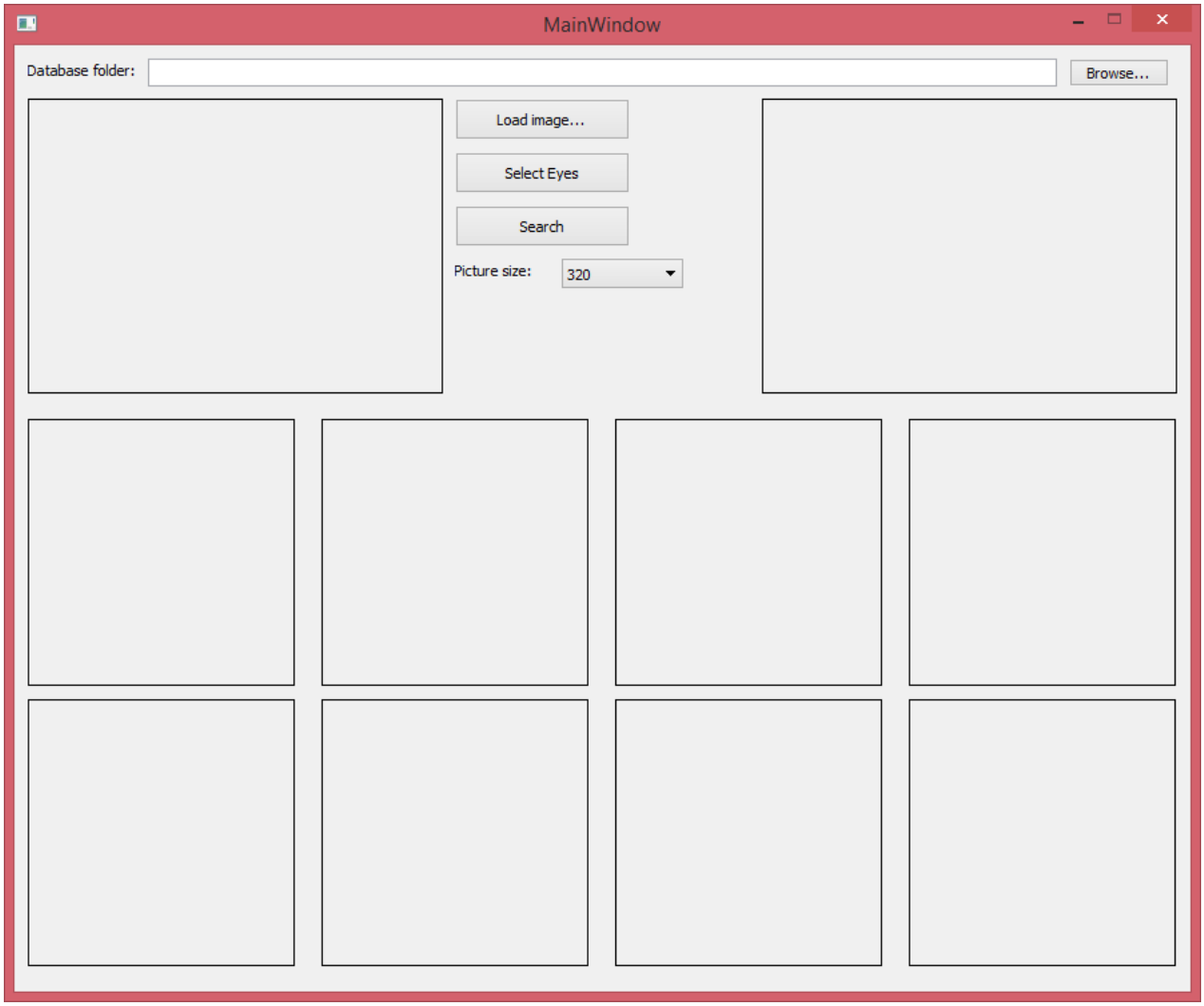


Рисунок 15 – Окно поиска похожих людей

Обработчики событий этого окна описываются в файлах исходного кода mainwindow.cpp, mainwindow.h. Эти файлы так же представлены в приложении. В файле mainwindow.cpp реализуются следующие функции: по нажатию кнопки «Browse…» пользователь должен выбрать папку, где располагается база данных фотографий с уже сформированным файлом Database.txt. По нажатию кнопки «Load image…» пользователю предоставляется возможность выбрать файл с фотографией, по которой будет вестись поиск. После нажатия на кнопку «Select Eyes» пользователь должен с помощью мыши выбрать центры глаз, по которым будет осуществлена трансформация изображения и расчет признаков. По кнопке «Search» производится нахождение похожих лиц из базы данных, указанной пользователем.

Перейдем к рассмотрению второй части программной системы нахождения похожих лиц по фотографии. Она состоит из файлов main.cpp, imageattributes.h, imageattributes.cpp,

imagevector.h. В файле main.cpp описывается применение ключа «–s» при запуске программы. Этот ключ необходим для переключения окон, описанных выше.

В файлах imageattributes.h и imageattributes.cpp описан класс, с помощью которого можно удобно хранить признаки изображения, а так же в случае необходимости получить список этих признаков и количество блоков. В качестве единственного параметра конструктора на вход передается количество блоков.

Файл imagevector.h содержит в себе класс «ImageVector», в котором описаны функции трансформации и вычисления признаков изображения. Первая функция «prepareImage» принимает на вход ссылку на изображение и координаты глаз, отмеченных на фотографии. Трансформация заключается в повороте, масштабировании и обрезании изображения таким образом, чтобы удобно было рассчитывать признаки. Результат данных преобразований передается на вход второй функции «calculateAttributes». В данной функции реализуется сразу несколько ключевых шагов основного алгоритма нахождения похожих лиц по фотографии. Во-первых, в ней происходит преобразование изображения в оттенки серого и в этом же цикле находятся минимальное и максимальное значения элементов матрицы яркостей изображения. Так же производятся остальные шаги *Q*-преобразования над получившимся изображением. Эта функция содержит в себе все 16 фильтров из теории активного восприятия и при прохождении каждого пикселя в блоке эти фильтры применяются, таким образом, ко всему изображению. В результате применения фильтров происходит вычисление признаков изображения.

5. Тестирование системы

5.1. Описание набора данных

В качестве тестовой базы данных были взяты фотографии известных личностей, причем по несколько снимков одного и того же человека. Был выбран именно такой набор исходных данных для того, чтобы можно было лучше выявить работоспособность программы и протестировать, насколько точно она находит похожие фотографии. Так же среди этих популярных людей имеются похожие личности, что дает дополнительную возможность проверить правильность работы программы.

Все фотографии в базе данных имеют формат JPEG. Данный формат был выбран, так как он является самым популярным для распространения изображений. Декодированием формата занимается фреймворк Qt. В тестовой выборке было 40 фотографий, средний размер которых 1200x1200. Количество горизонтально и вертикально ориентированных фотографий примерно совпадает. Часть изображений, удовлетворяющих требованиям выше, приведена на рисунке 16.

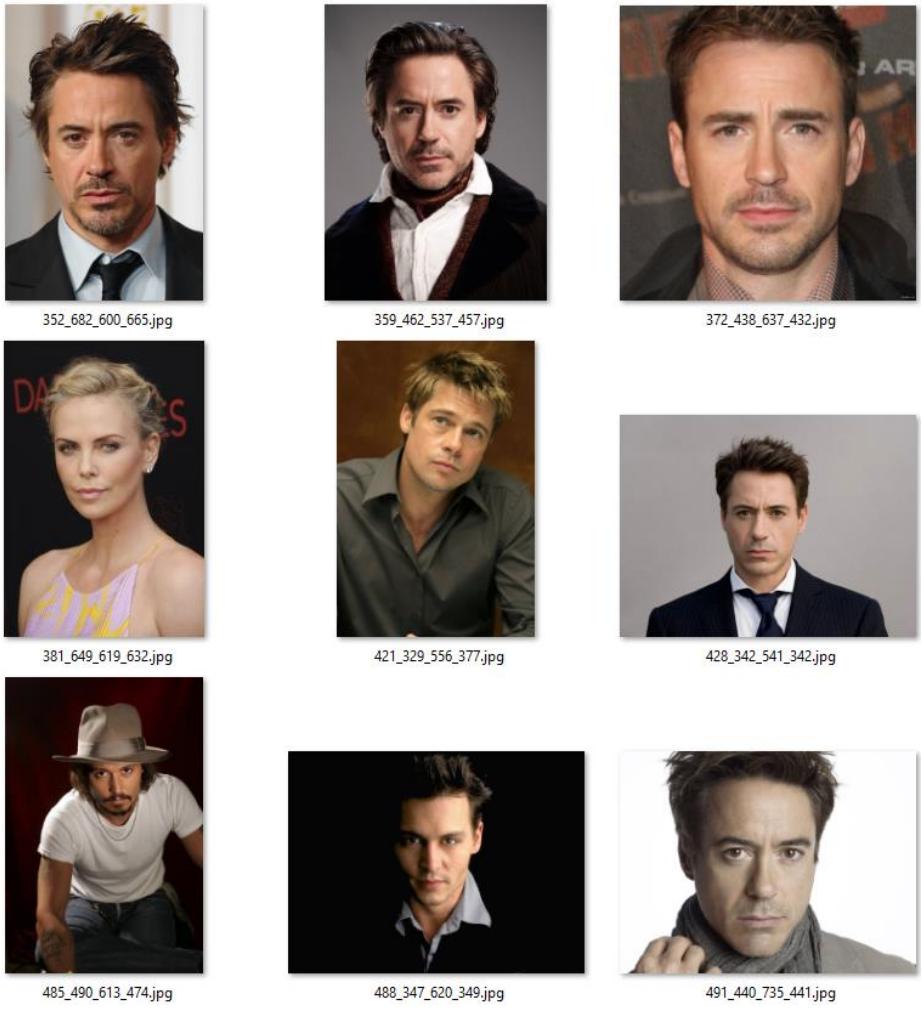


Рисунок 16 – Часть изображений в тестовой выборке

5.2. Описание методики тестирования

Для удобства реализации алгоритма и для сохранения качества фотографии был выбран квадратный формат преобразованного изображения. Было подобрано несколько возможных значений количества блоков, на которые будет поделено изображение: 8, 16, 32.

В главном окне программы можно выбрать размер стороны квадратного изображения, к которому будет приводиться исходная фотография. Надо учитывать то, что выбираемый размер желательно подобрать таким образом, чтобы он примерно соответствовал размеру оригинального изображения или был чуть меньше его. В случае, если размер изображений совпадает, то при трансформации изображения информация практически не теряется. Если же выбранный размер больше оригинальной фотографии, то при увеличении последней до требуемого размера будут появляться дополнительные пиксели, которые будут вносить неточности в процесс распознавания. Так как на большинстве фотографий тестовой выборки, пример которой приведен на рисунке 16, лицо занимает не всё изображение, а примерно половину, то в качестве размера стороны изображения, к которому будет приводиться фотография, был выбран размер равный 640 пикселям.

На исходном наборе данных была протестирована программа с различными значениями деления изображения на блоки.

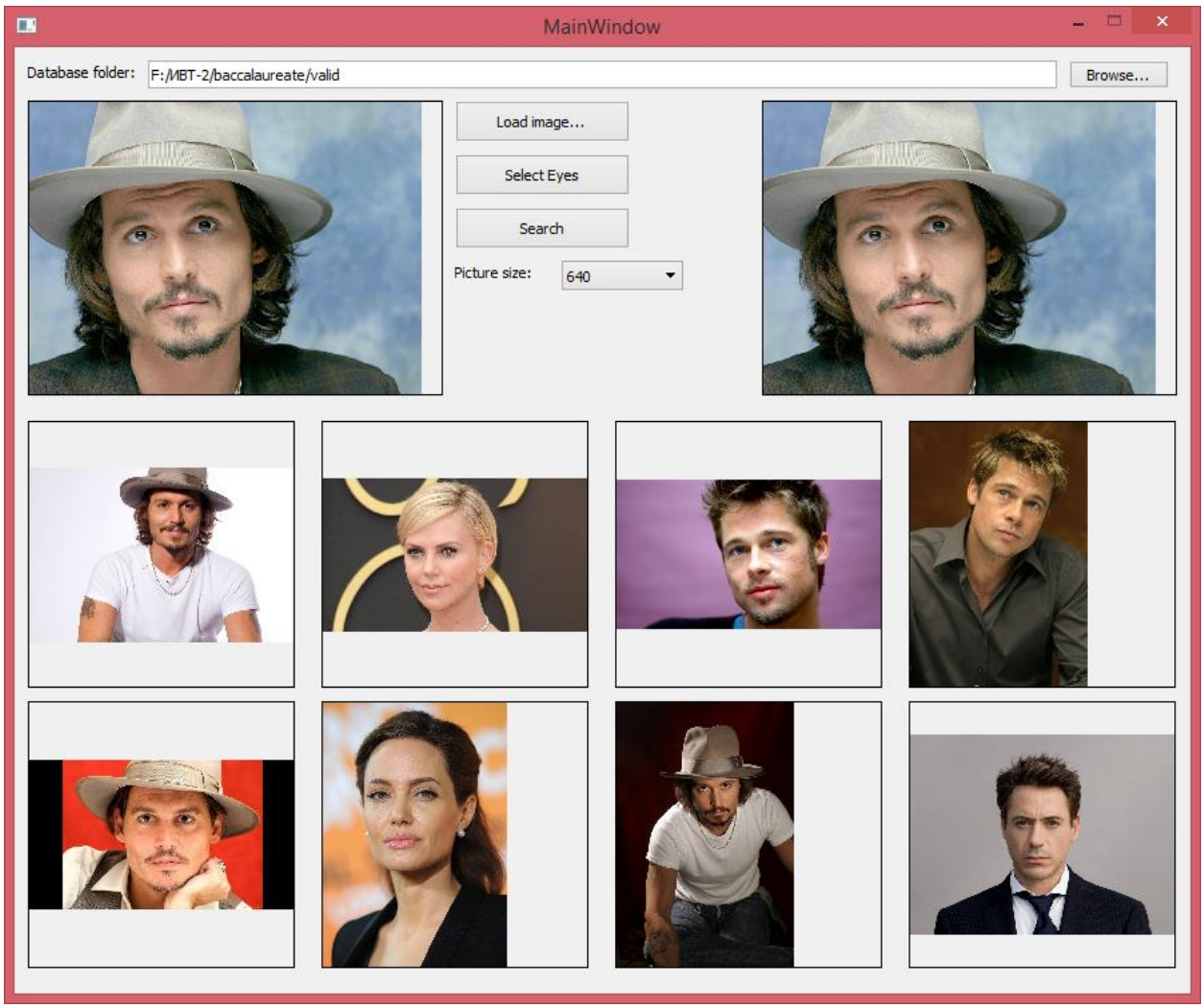


Рисунок 17 – Результат работы программы при делении на 8 блоков

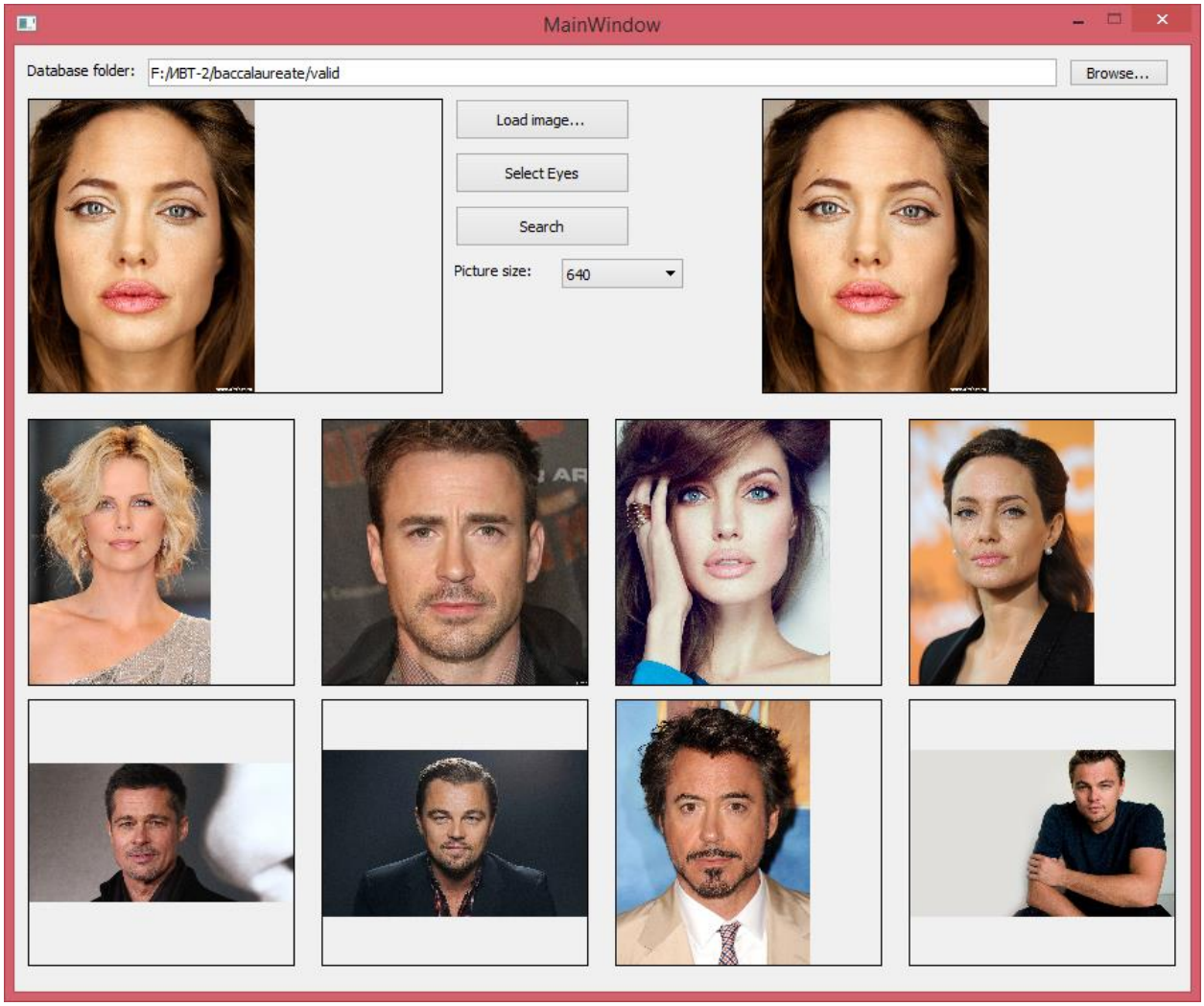


Рисунок 18 – Результат работы программы при делении на 8 блоков

На рисунках 17 и 18 поиск производится при делении преобразованного изображения на 8 блоков по вертикали и по горизонтали.

На рисунке 17 осуществлен поиск мужчины, американского актера, Джонни Деппа. На рисунке 18 произведен поиск женщины, американской актрисы, Анджелины Джоли. Данные изображения были взяты из базы данных. В результате выполнения поиска по библиотеке наиболее вероятным совпадением являются те же самые фотографии, что и подавались на вход. Так же в выборке похожих в первом случае присутствует 3 фотографии этого человека, во втором случае – 2 фотографии. Но они не все являются первыми в списке похожих изображений. Это объясняется тем, что деление выполняется на крупные части и из-за этого значения признаков получаются грубыми.

Данный режим работы предоставляет максимальное быстродействие, но выдает грубые результаты.

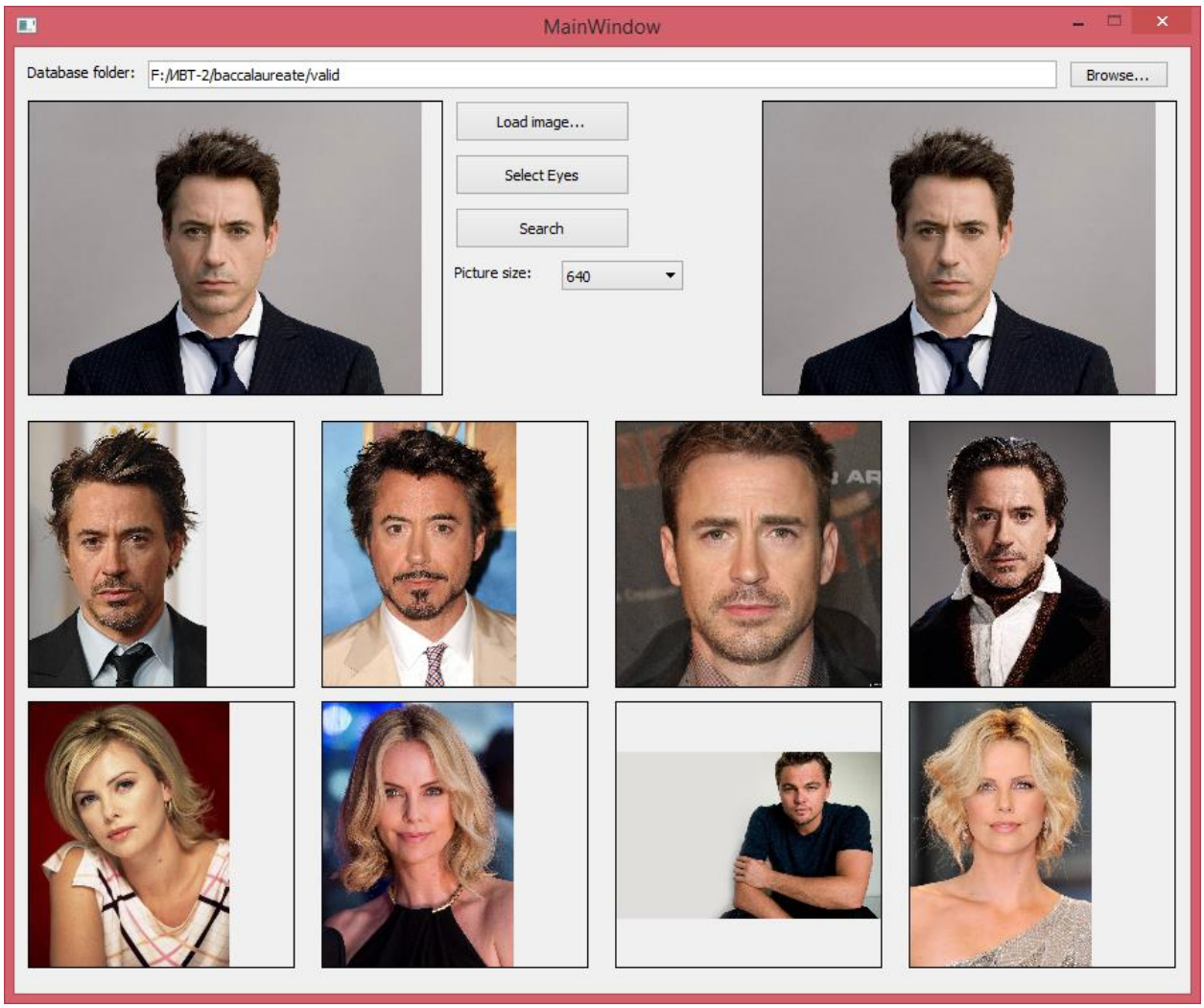


Рисунок 19 – Результат работы программы при делении на 16 блоков

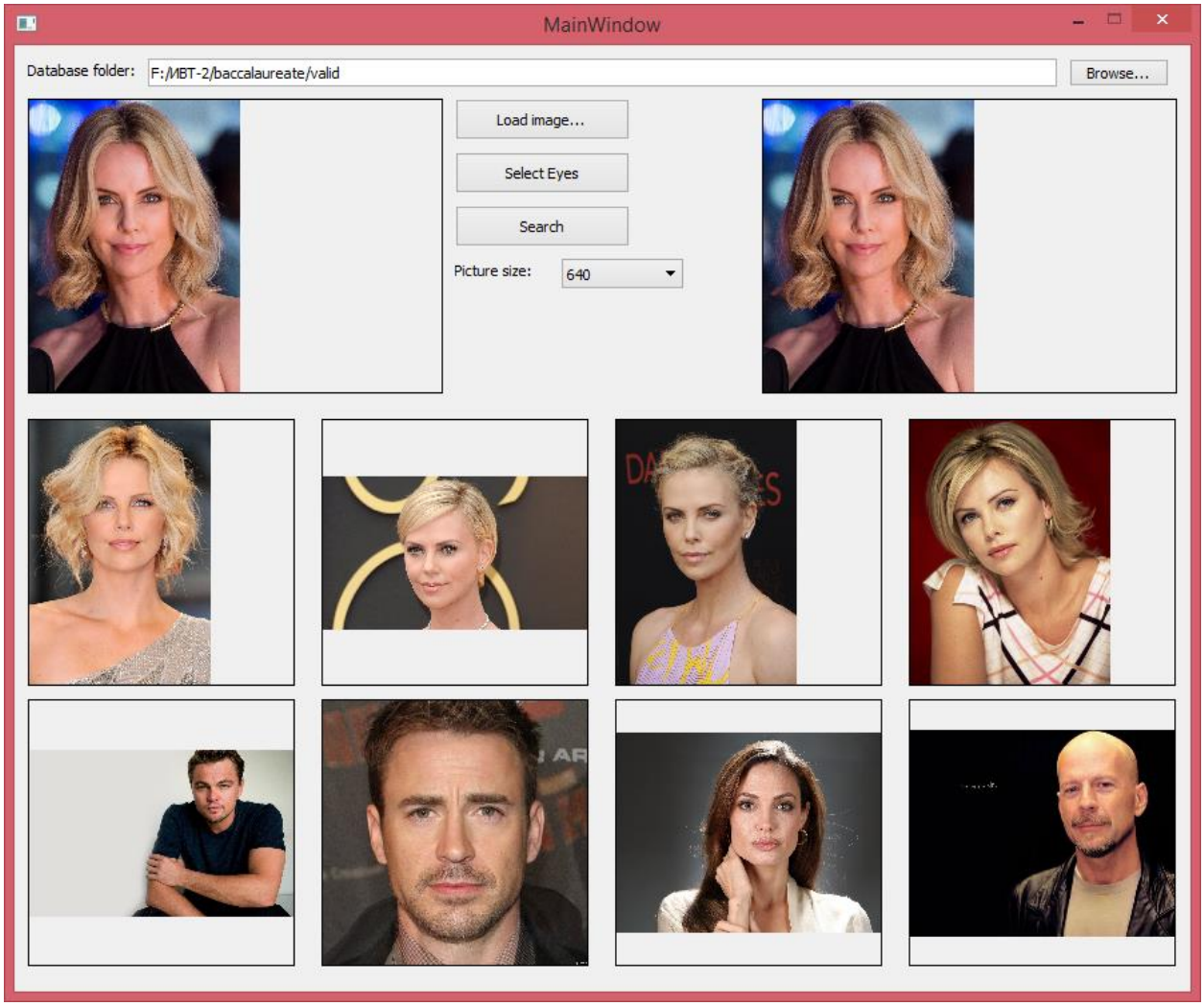


Рисунок 20 – Результат работы программы при делении на 16 блоков

На рисунках 19 и 20 приведен результат работы программы при делении изображения в ходе преобразования на 16 блоков по вертикали и по горизонтали.

На рисунке 19 показан результат поиска по фотографии мужчины, американского актера, Роберта Дауни. На рисунке 20 поиск производился по американской актрисе, Шарлиз Терон. Как видно из результатов поиска, программа довольно точно определила фотографии того и другого актера. Наиболее похожими явились те же фотографии, что и подавались на вход. Так же менее похожими, но всё же набравшими высокий процент совпадения, в обоих случаях явились фотографии этого же человека. В последнем же ряду на обоих рисунках представлены люди, у которых низкий процент совпадения с исходным изображением, но всё же имеются общие черты.

Таким образом, данный режим выдает наиболее точные результаты и является приемлемым по быстродействию поиска.

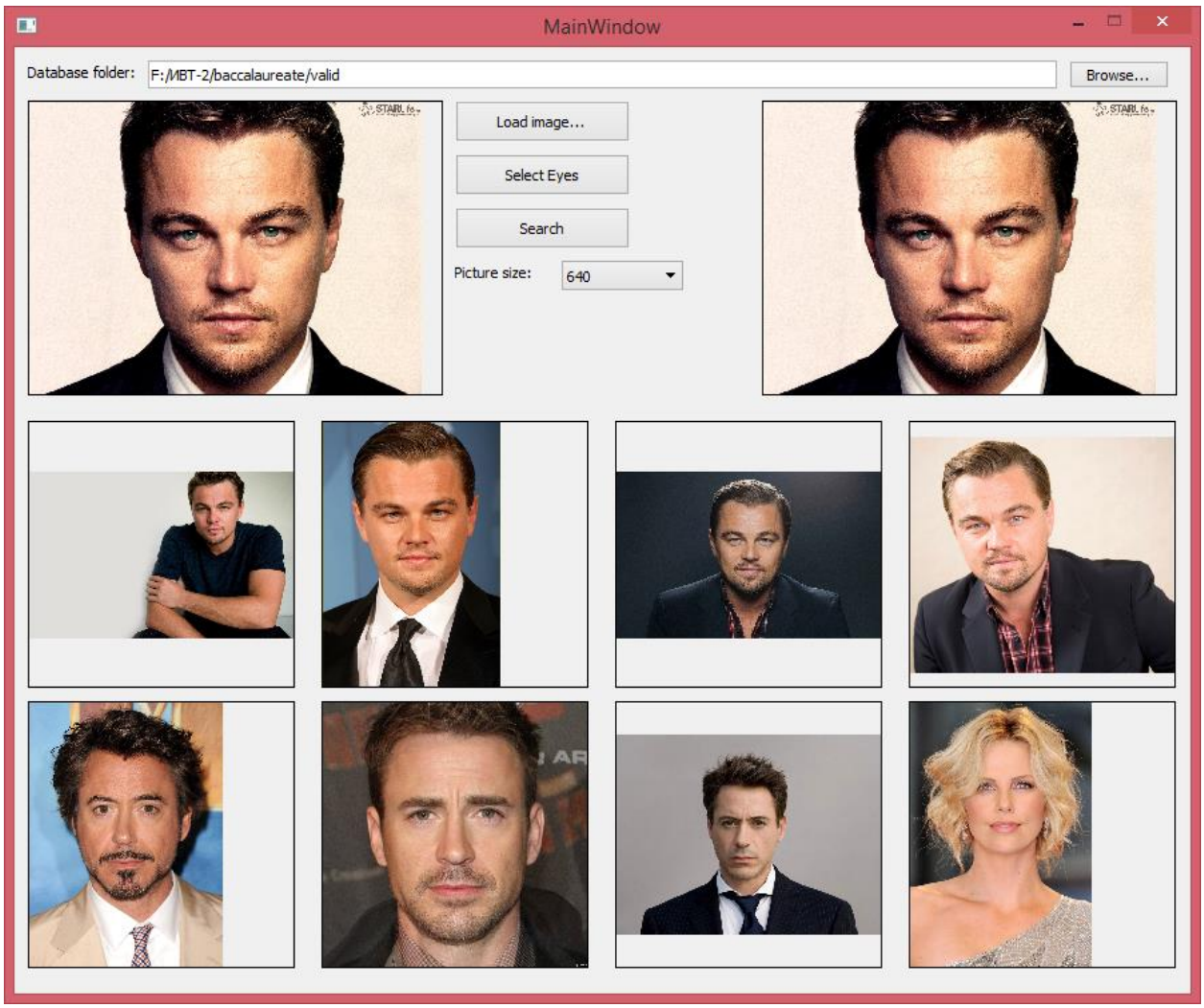


Рисунок 21 – Результат работы программы при делении на 32 блоков

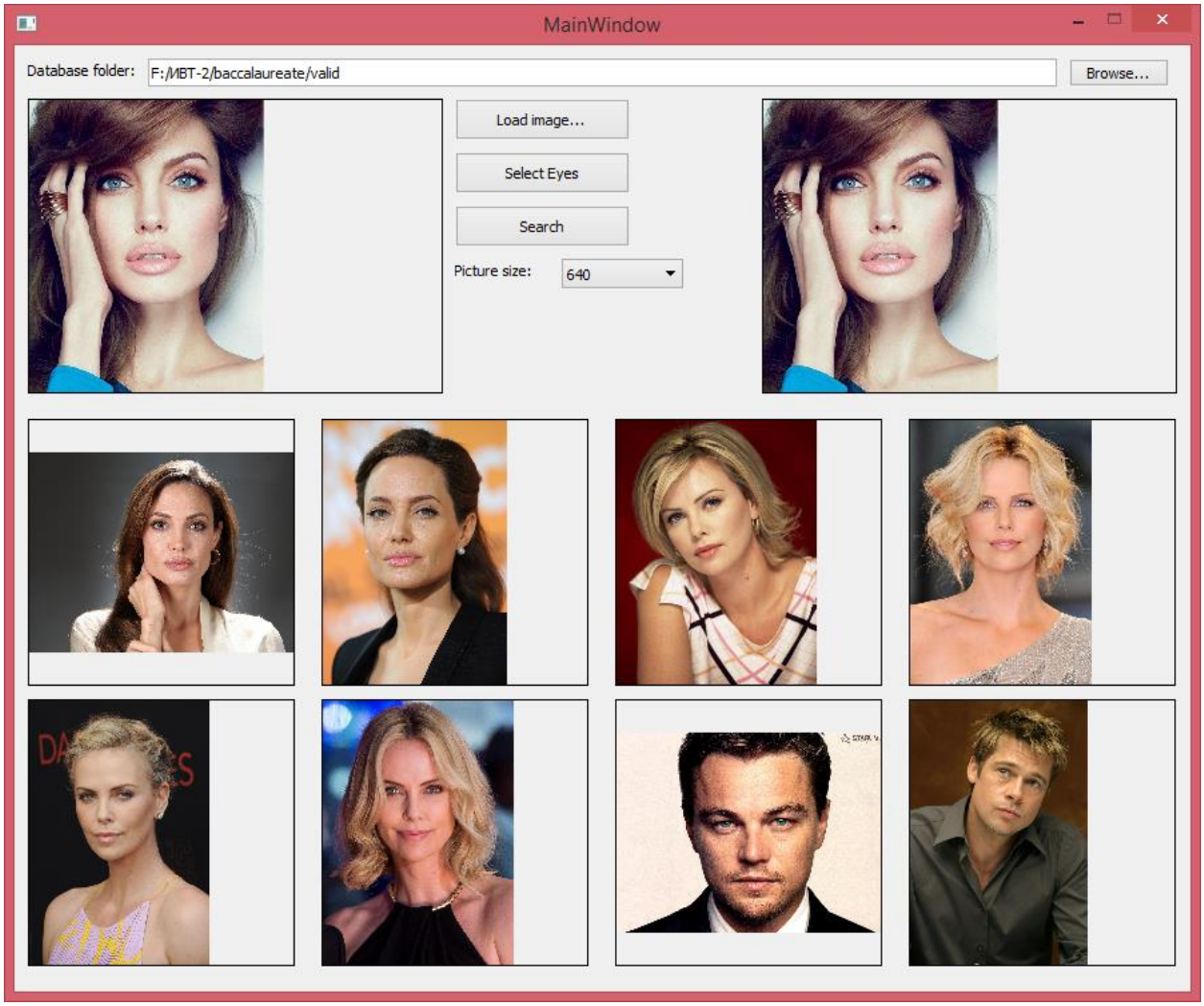


Рисунок 22 – Результат работы программы при делении на 32 блоков

На рисунках 21 и 22 показаны результаты работы программы в случае, когда преобразованное изображение делится на 32 блока по вертикали и 32 – по горизонтали.

На рисунке 21 поиск осуществлен по американскому актеру, Леонардо Ди Каприо. На рисунке 22 представлен результат поиска по фотографии американской актрисы, Анджелины Джоли. Результаты поиска показывают, что в обоих случаях в качестве наиболее вероятной была найдена та же фотография, что и подавалась на вход. Последующие снимки тоже очень похожи на исходное изображение. В данном эксперименте присутствует большая схожесть, чем при делении на 16 блоков, но расчет признаков и нахождение результата занимают больше времени, чем в остальных случаях.

Данный метод деления является наиболее точным, но требует больших временных затрат на обнаружение похожих людей по загруженной фотографии.

Таким образом, в ходе тестирования программы было рассмотрено три варианта деления преобразованного изображения на блоки. Было выявлено, что оптимальным является второй вариант деления – на 16 блоков по вертикали и 16 блоков по горизонтали. Такой вывод можно сделать из того, что в данном случае результат поиска является достаточно точным, выдавая изображения лишь немного грубее, чем в последнем варианте, когда деление производится на 32 блока. Но это компенсируется быстротой подсчета признаков и поиска похожих людей. Первый же вариант является очень грубым и не достаточно хорошо справляется с задачей нахождения похожих лиц по фотографии.

Заключение

В результате выполнения выпускной квалификационной работы была спроектирована и программно реализована система поиска похожих людей по фотографии.

На этапах нормализации и формирования признаков изображения была применена теория активного восприятия. При кластеризации изображений по признакам использовался метод ближайших соседей.

Созданная программная система предназначена для поиска группы похожих людей по загруженной пользователем фотографии. Данная система пригодна для кластеризации не только фотографий людей, но и объектов. Был проведён анализ программы в рамках поставленного эксперимента. Проведенное тестирование подтвердило работоспособность созданного программного обеспечения. На основе полученных результатов тестирования можно сделать вывод, что данный метод решения поставленной задачи имеет большой потенциал и дальнейшее развитие системы будет направлено на улучшение результатов поиска и увеличение быстродействия данного программного продукта.

Список литературы

1. Утробин, В.А. Элементы теории активного восприятия изображений. НГТУ. – Нижний Новгород, 2010. – 9 с.
2. Гай, В.Е. Алгоритмы формирования спектрального представления звукового сигнала на основе *U* - преобразования. Бизнес-информатика №1 (23). НИУ ВШЭ. – Москва, 2013. – 44-49 с.
3. Иерархическая кластеризация // WiKi ru-wiki.org URL: http://ru-wiki.org/wiki/Иерархическая_кластеризация
4. K-means // WiKi ru-wiki.org URL: http://ru-wiki.org/wiki/K-means
5. Шлее Макс. Профессиональное программирование на C++. +CD. Qt 4.8. БХВ – Петербург, 2012. – 912.
6. Обучение с учителем // Википедия — свободная энциклопедия ru.wikipedia.org URL: https://ru.wikipedia.org/wiki/Обучение_с_учителем

Приложение

imageattributes.h

```
#ifndef IMAGEATTRIBUTES_H
#define IMAGEATTRIBUTES_H

//класс хранит признаки одной фото
class ImageAttributes
{
public:
    ImageAttributes(int blocks); //конструктор
    ~ImageAttributes(); //деструктор
    void setAttribute(int blockNumber, int filterNumber, double filter);
//установка признака
    double getAttribute(int blockNumber, int filterNumber); //удаление
признака
    int getBlocksCount(); //получение кол-ва блоков

private:
    double** attributes; //двумерный массив хранения признаков
    int blocksCount; //кол-во блоков
};

#endif
```

---

imageattributes.cpp

```
#include "imageattributes.h"
//конструктор
ImageAttributes::ImageAttributes(int blocks)
{
    blocksCount = blocks;

    attributes = new double*[blocksCount];
    for (int i = 0; i < blocksCount; ++i)
    {
        attributes[i] = new double[16];
    }
}
//деструктор
ImageAttributes::~ImageAttributes()
{
    //удаление двумерного массива
    for (int i = 0; i < blocksCount; ++i)
    {
        delete [] attributes[i];
    }
    delete [] attributes;
}
//установка признаков
void ImageAttributes::setAttribute(int blockNumber, int filterNumber, double
filter)
{

    attributes[blockNumber][filterNumber] = filter;
}

double ImageAttributes::getAttribute(int blockNumber, int filterNumber)
{
    return attributes[blockNumber][filterNumber];
```

```
}
int ImageAttributes::getBlocksCount()
{
    return blocksCount;
}
```

---

imagevector.h

```
#ifndef IMAGEVECTOR_H
#define IMAGEVECTOR_H

#include <QImage>
#include <QPoint>
#include <QDebug>
#include "imageattributes.h"

#define PIC_SIZEs 640 //размер загружаемого фото в памяти, по которому
определяются признаки
#define PIC_PART 32//кол-во блоков в ряду и столбце

class ImageVector
{
public:
    //трансформация изображения
    static QImage prepareImage(const QImage& loadedImage, const QPoint& eye1,
const QPoint& eye2, unsigned int picSize)
    {
        QImage imageTransformated = QImage(picSize, picSize,
QImage::Format_RGB32); //создается в памяти изображение указанного размера
        QPoint p(eye2.x() - eye1.x(), eye2.y() - eye1.y()); //расстояние
между глазами

        double angle = -1 * atan2(p.y(), p.x()); //угол между глазами
        double scale = (picSize / 3) / sqrt(p.x() * p.x() + p.y() * p.y());
//отношение расстояния между глазам к идеальному расстоянию (1/3 фото)
        //заливка пустого изображения черным
        imageTransformated.fill(Qt::black);
        //перебор пикселей в результирующем изображении и построение проекции
на реальное изображение
        for (int x = 0; x < imageTransformated.size().width(); ++x)
        {
            for (int y = 0; y < imageTransformated.size().height(); ++y)
            {
                double x1 = (x - imageTransformated.size().width() / 4); //
сдвигаем итоговое изображения так, чтобы глаз оказался в нуле
                double y1 = (y - imageTransformated.size().height() / 3);

                double x2 = x1 / scale; // масштабируем в обратную сторону
                double y2 = y1 / scale;

                double x3 = x2 * cos(-angle) - y2 * sin(-angle); //
поворачиваем в обратную сторону
                double y3 = x2 * sin(-angle) + y2 * cos(-angle);

                double x4 = x3 + eye1.x(); // ставим глаз в исходное
положение
                double y4 = y3 + eye1.y();

                int xFin = (int)round(x4); //округление координат
```

```
                int yFin = (int)round(y4);
                //пропускаем пиксели, которые не влазят в оригинал
                if (((xFin >= 0) && (xFin < loadedImage.size().width())) &&
((yFin >= 0) && (yFin < loadedImage.size().height())))
                    imageTransformated.setPixel(x, y, loadedImage.pixel(xFin,
yFin));

            }

        }
        return imageTransformated;
    }


    static ImageAttributes* calculateAttributes(const QImage& image)
    {

        int min = 256;
        int max = -1;
        QImage imageGrey = QImage(image.size().width(),
image.size().height(), QImage::Format_Grayscale8); //создается в памяти
изображение для сохранеия серого
        //пробегаем писксели оригинального изображения
        for (int x = 0; x < image.size().width(); ++x)
        {
            for (int y = 0; y < image.size().height(); ++y)
            {

               QRgb qRgb = image.pixel(x,y); //получаем значения цвета
пикселя
                int newRgb = qGray(qRed(qRgb), qGreen(qRgb), qBlue(qRgb));
//устанавливаем цвет пикселя в сером изображении
                QColor c = QColor(newRgb, newRgb, newRgb);
                imageGrey.setPixelColor(x, y, c);

                if (newRgb < min)
                    min = newRgb;

                if (newRgb > max)
                    max = newRgb;
            }
        }
        max -= min;
        //делим изображение на части (16x16)
        QVector<QImage> parts;
        for(int i = 0; i < PIC_PART; ++i)
        {
            for(int j = 0; j < PIC_PART; ++j)
            {
                //копирование участка с координатами (верхнего левого угла и
размера)
                QImage image =
imageGrey.copy(i*(imageGrey.size().width()/PIC_PART),
j*(imageGrey.size().height()/PIC_PART), imageGrey.size().width()/PIC_PART,
imageGrey.size().height()/PIC_PART);
                parts.push_back(image); //сохранение в массив
            }
        }
        //создание класса атрибутов
        ImageAttributes* resultAttributes = new
ImageAttributes(PIC_PART*PIC_PART);
```

```
        double matrix16[4][4]; //текущий блок
        unsigned short filters [16]; //фильтры
        filters[0] = 0xFFFF;
        filters[1] = 0x3333;
        filters[4] = 0x6666;
        filters[9] = 0xAAAA;
        filters[2] = 0xFF00;
        filters[3] = 0x33CC;
        filters[6] = 0x6699;
        filters[11] = 0xAA55;
        filters[5] = 0x0FF0;
        filters[7] = 0xC33C;
        filters[8] = 0x9669;
        filters[13] = 0x5AA5;
        filters[10] = 0x0F0F;
        filters[12] = 0xC3C3;
        filters[14] = 0x9696;
        filters[15] = 0x5A5A;

        //для каждого блока от 0 до размера массива
        for(int block = 0; block < parts.size(); ++block)
        {
            //очищение матрицы
            for (int i = 0; i < 4; ++i)
                for(int j = 0; j < 4; ++j)
                    matrix16[i][j] = 0;
            //проход по пикселям внутри блока
            for (int x = 0; x < parts[block].size().width(); ++x)
            {
                for (int y = 0; y < parts[block].size().height(); ++y)
                {
                    QRgb qrgb =  parts[block].pixel(x,y); //получение цвета
пикселя
                    //суммированное нормальзованное значение (Q-
преобразование)

matrix16[x/(parts[block].size().width()/4)][y/(parts[block].size().height()/4
)] += (double)(qRed(qrgb) - min)/(double)max;
                }
            }

            double sumF = 0; //переменная хранит значение признака
            //применение фильтров
            for (int f = 0; f < 16; ++f)
            {
                sumF = 0;
                for (int i = 0; i < 4; ++i)
                {
                    for (int j = 0; j < 4; ++j)
                    {
                        if (((filters[f] >> (j*4+i)) & 0x01) == 0x01)
                            sumF += matrix16[i][j];
                        else
                            sumF -= matrix16[i][j];

                    }
                }
                //установка атрибута
                resultAttributes->setAttribute(block, f, sumF);
```

```
            }
        }

        return resultAttributes;
    }

};

#endif // IMAGEVECTOR_H
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QFileDialog>
#include <QMessageBox>
#include <QMouseEvent>
#include "imageattributes.h"
#include "imagevector.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    void mouseReleaseEvent(QMouseEvent* event); //ункция обработки отпускания
мыши в окне

private slots:
    void on_pushButtonBrowseDatabase_clicked();

    void on_buttonLoadImage_clicked();

    void on_buttonSearch_clicked();

    void on_buttonSelectEyes_clicked();

private:
    Ui::MainWindow *ui; //пременная для доступа к элементам окна
    QString dir; //строка содержащая путь до БД в системе
    QImage* givenImage; //указатель на загружаемое изображение

    double scaleCoeffX; //коэффициенты масштабирования для правильного
указания глаз
    double scaleCoeffY; //определяют соотношения между изображением внутри
виджета и реальным

    ImageAttributes* attributes; //хранит признаки искомой фотографии

    bool isEyePointing; //включен ли режим указания глаз
    QPoint* eye1; //указатели на точки глаз
    QPoint* eye2;
```

```
};

#endif // MAINWINDOW_H
```

---

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
//конструктор окна
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui(new
Ui::MainWindow)
{
    ui->setupUi(this); //инициализируется графический интерфейс
    dir = "";
    this->setFixedSize(this->size());
    givenImage = NULL;

    scaleCoeffX = 0;
    scaleCoeffY = 0;

    isEyePointing = false;
    eye1 = NULL;
    eye2 = NULL;

    attributes = NULL;
}
//деструктор
MainWindow::~MainWindow()
{
    if (givenImage != NULL)//удаление загруженной фотографии из памяти
        delete givenImage;

    delete ui; //удаляется графический интерфейс
}

void MainWindow::mouseReleaseEvent(QMouseEvent* event)
{
    if (!isEyePointing)
        return;

    QPoint point = this->ui->mainImage->pos(); //верхний левый угол виджета
фотографии
    QSize labelSize = this->ui->mainImage->size(); //размер виджета
    //произошло нажатие на виджет или нет
    bool isPictureClick = true;
    isPictureClick &= (event->x() > point.x());
    isPictureClick &= (event->y() > point.y());
    isPictureClick &= (event->x() < point.x() + labelSize.width());
    isPictureClick &= (event->y() < point.y() + labelSize.height());

    if (isPictureClick)
    {
        if (eye1 == NULL)
        {
           eye1 = new QPoint();
           eye1->setX(event->x() - point.x());
           eye1->setY(event->y() - point.y());
        }
        else if (eye2 == NULL)
        {
```

```
            eye2 = new QPoint();
            eye2->setX(event->x() - point.x());
            eye2->setY(event->y() - point.y());
            isEyePointing = false;

            QPoint realEye1; //глаза на фотографии в памяти
            QPoint realEye2;


            realEye1.setX(scaleCoeffX * eye1->x());
            realEye1.setY(scaleCoeffY * eye1->y());

            realEye2.setX(scaleCoeffX * eye2->x());
            realEye2.setY(scaleCoeffY * eye2->y());


            unsigned int picSize = this->ui->comboBox->currentText().toInt();
            //вызов функции трансформации
            QImage processed = ImageVector::prepareImage(*givenImage,
realEye1, realEye2, picSize);
            //если коэффициенты были созданны, удалить
            if (attributes != NULL)
                delete attributes;
            //вызов функции рассчета признаков
            attributes = NULL;
            attributes = ImageVector::calculateAttributes(processed);
            //удаление первого глаза
            if (eye1 != NULL)
                delete eye1;
            eye1 = NULL;
            //удаление второго глаза
            if (eye2 != NULL)
                delete eye2;
            eye2 = NULL;
        }

    }

}
//функция выбора БД
void MainWindow::on_pushButtonBrowseDatabase_clicked()
{
    dir = QFileDialog::getExistingDirectory ( this, "Select Database Folder",
QString(), QFileDialog::ShowDirsOnly); //открытие диалога с папками системы
    ui->lineEditDatabasePath->setText(dir); //строчку пути выводит на виджет
}
//функция нажатия на кнопку загрузить изображение
void MainWindow::on_buttonLoadImage_clicked()
{
    if (givenImage != NULL)
        delete givenImage;

    QString file = QFileDialog::getOpenFileName(this, "Select image", "",
"*.jpg *.JPG"); //выбор файла
    givenImage = new QImage(file); //загружаем фотографию в память
    QImage scaledGrey = givenImage->scaled(this->ui->mainImage->size(),
Qt::KeepAspectRatio); //масштабируем фото по размеру виджета
    //устанавливаем коэффициенты масштабирования
    scaleCoeffX = ((double)givenImage->size().width() /
(double)scaledGrey.size().width());
```

```
    scaleCoeffY = ((double)givenImage->size().height() /
(double)scaledGrey.size().height());

    ui->mainImage->setPixmap(QPixmap::fromImage(scaledGrey)); //вывод
преобразованной фото в виджет

    if (attributes != NULL)
        delete attributes;

    attributes = NULL;
}

void MainWindow::on_buttonSearch_clicked()
{
    //проверка указана ли БД
    if (dir.compare("") == 0)
    {
        QMessageBox messageBox(this);
        messageBox.setText("Specify database path!");
        messageBox.setWindowTitle("Error.");
        messageBox.exec();
        return;
    }
    //указаны ли глаза
    if (attributes == NULL)
    {
        QMessageBox messageBox(this);
        messageBox.setText("Specify eyes first!");
        messageBox.setWindowTitle("Error.");
        messageBox.exec();
        return;
    }

    QFile databaseFile(dir + "/Database.txt"); //открытие файла БД
    //удалось ли открыть файл БД
    if(!databaseFile.open(QIODevice::ReadWrite))
    {
        QMessageBox messageBox(this);
        messageBox.setText("Can't open file!");
        messageBox.setWindowTitle("Error.");
        messageBox.exec();
        return;
    }
    //открытие текстового потока
    QTextStream in(&databaseFile);
    //контейнер сортированнй по ключу для хранения путей похожих фотографий
    QMap<double, QString> files; //дельта; путь до фотографии в системе
    double currentDelta = 0; //хранит дельту между загруженным фото и текущей
фото из БД

    QString blocksNumber; //колличество блоков на 1 фото
    QString temp; //временная переменная
    QString fileName; //путь до текущей фото в БД
    //пока не кончилась БД
    while(!databaseFile.atEnd() || !in.atEnd())
    {
       // qDebug() << "Pos: " << databaseFile.pos();
        fileName = in.readLine(); //считывание имени текущей фото в БД
       // qDebug() << "Pos: " << databaseFile.pos();
       // qDebug() << "File name: " << fileName;
        blocksNumber = in.readLine(); //считывание колличества блоков
```

```
        // qDebug() << "Filename: " << fileName;
         currentDelta = 0; //обнуляем дельту
        // qDebug() << "Current blocks: " << blocksNumber.toInt();
        // qDebug() << "Attributes blocks: " << (attributes-
>getBlocksCount()*16);
         //qDebug() << "Pos: " << databaseFile.pos();
         //совпадает ли кол-во блоков в БД с кол-вом блоков в загруженном
изображении
         if (blocksNumber.toInt() == (attributes->getBlocksCount()*16))
         {
             //перебирает номера блоков
             for(int i = 0; i < attributes->getBlocksCount(); ++i)
             {
                 //перебирает номера ячеек фильтров
                 for(int j = 0; j < 16; ++j)
                 {
                     //считывает из БД значение ячейки
                     temp = in.readLine();
                     //qDebug() << "Pos: " << databaseFile.pos();
                     //if ((temp.toDouble()*attributes->getAttribute(i, j)) >
0)
                        // currentDelta += abs(temp.toDouble() - attributes-
>getAttribute(i, j));
                     currentDelta += ((temp.toDouble()*attributes-
>getAttribute(i,j)>0) ? 0 : 1); //увеличивает или уменьшает дельту в
зависимости от значений признаков
                 }
             }

             files.insert(currentDelta, fileName); //вставляет каждую фото в
контейнер путей похожих фото
             //даляет фото из контейнера, если его размер больше 9
             if (files.size() > 9)
                 files.remove(files.lastKey());
         }
         else
         {
             qDebug() << "Skip: " << fileName << " because filters count not
euqal";
             for(int i = 0; i < blocksNumber.toInt(); ++i)
                 temp = in.readLine(); //читает в холостую
         }
         qDebug() << "Pos: " << databaseFile.pos();
     }

     qDebug() << "Similar images count: " << files.size();
     QList<QString> similarImages = files.values(); //массив путей похожих
фотографий
     QList<double> deltas = files.keys(); //массив дельт похожих фото
     qDebug() << deltas;
     //если кол-во похожих больше 0
     if (similarImages.size() > 0)
     {
         //вставляет фото в соответствущий виджет
         QImage test(dir + "/" + similarImages.at(0)); //открытие фото
         QImage scaledGrey = test.scaled(this->ui->mainImage_2->size(),
Qt::KeepAspectRatio); //масштабирование
         ui->mainImage_2->setPixmap(QPixmap::fromImage(scaledGrey));
//отображение на виджете
```

```
        if (similarImages.size() > 1)
        {
            QImage test(dir + "/" + similarImages.at(1));
            QImage scaledGrey = test.scaled(this->ui->similarImage1->size(),
Qt::KeepAspectRatio);
            ui->similarImage1->setPixmap(QPixmap::fromImage(scaledGrey));
        }

        if (similarImages.size() > 2)
        {
            QImage test(dir + "/" + similarImages.at(2));
            QImage scaledGrey = test.scaled(this->ui->similarImage2->size(),
Qt::KeepAspectRatio);
            ui->similarImage2->setPixmap(QPixmap::fromImage(scaledGrey));
        }

        if (similarImages.size() > 3)
        {
            QImage test(dir + "/" + similarImages.at(3));
            QImage scaledGrey = test.scaled(this->ui->similarImage3->size(),
Qt::KeepAspectRatio);
            ui->similarImage3->setPixmap(QPixmap::fromImage(scaledGrey));
        }

        if (similarImages.size() > 4)
        {
            QImage test(dir + "/" + similarImages.at(4));
            QImage scaledGrey = test.scaled(this->ui->similarImage4->size(),
Qt::KeepAspectRatio);
            ui->similarImage4->setPixmap(QPixmap::fromImage(scaledGrey));
        }

        if (similarImages.size() > 5)
        {
            QImage test(dir + "/" + similarImages.at(5));
            QImage scaledGrey = test.scaled(this->ui->similarImage5->size(),
Qt::KeepAspectRatio);
            ui->similarImage5->setPixmap(QPixmap::fromImage(scaledGrey));
        }

        if (similarImages.size() > 6)
        {
            QImage test(dir + "/" + similarImages.at(6));
            QImage scaledGrey = test.scaled(this->ui->similarImage6->size(),
Qt::KeepAspectRatio);
            ui->similarImage6->setPixmap(QPixmap::fromImage(scaledGrey));
        }

        if (similarImages.size() > 7)
        {
            QImage test(dir + "/" + similarImages.at(7));
            QImage scaledGrey = test.scaled(this->ui->similarImage7->size(),
Qt::KeepAspectRatio);
            ui->similarImage7->setPixmap(QPixmap::fromImage(scaledGrey));
        }

        if (similarImages.size() > 8)
        {
            QImage test(dir + "/" + similarImages.at(8));
            QImage scaledGrey = test.scaled(this->ui->similarImage8->size(),
Qt::KeepAspectRatio);
```

```
            ui->similarImage8->setPixmap(QPixmap::fromImage(scaledGrey));
        }
    }
    else
    {
        qDebug() << "No in similar list";
    }

    databaseFile.close(); //закрытие файла БД
}
//включает режим указания глаз
void MainWindow::on_buttonSelectEyes_clicked()
{
    isEyePointing = true;
}
```

---

makedatabasewindow.h

```
#ifndef MAKEDATABASEWINDOW_H
#define MAKEDATABASEWINDOW_H

#include <QMainWindow>
#include <QFileDialog>
#include <QMessageBox>
#include "imageattributes.h"
#include "imagevector.h"

namespace Ui {
class MakeDatabaseWindow;
}

class MakeDatabaseWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MakeDatabaseWindow(QWidget *parent = 0);
    ~MakeDatabaseWindow();

private slots:
    void on_pushButtonBrowseDatabase_clicked();

    void on_pushButtonMakeDatabase_clicked();

private:
    Ui::MakeDatabaseWindow *ui;
    QString dir;
};

#endif // MAKEDATABASEWINDOW_H
```

---

makedatabasewindow.cpp

```
#include "makedatabasewindow.h"
#include "ui_makedatabasewindow.h"

MakeDatabaseWindow::MakeDatabaseWindow(QWidget *parent) :
QMainWindow(parent), ui(new Ui::MakeDatabaseWindow)
```

```
{
    ui->setupUi(this);
    dir = "";
    this->setFixedSize(this->size());
}

MakeDatabaseWindow::~MakeDatabaseWindow()
{
    delete ui;
}

void MakeDatabaseWindow::on_pushButtonBrowseDatabase_clicked()
{
    dir = QFileDialog::getExistingDirectory ( this, "Select Database Folder",
QString(), QFileDialog::ShowDirsOnly);
    ui->lineEditDatabasePath->setText(dir);
}
//создание БД
void MakeDatabaseWindow::on_pushButtonMakeDatabase_clicked()
{
    //если не указана БД выдает ошибку
    if (dir.compare("") == 0)
    {
        QMessageBox messageBox(this);
        messageBox.setText("Specify database path!");
        messageBox.setWindowTitle("Error.");
        messageBox.exec();
        return;
    }


    //существует ли папка с БД
    QDir directory(dir);
    if(!directory.exists())
    {
        QMessageBox messageBox(this);
        messageBox.setText("No such directory!");
        messageBox.setWindowTitle("Error.");
        messageBox.exec();
        return;
    }

    //открывает файл БД для записи
    QFile databaseFile(dir + "/Database.txt");
    //если файл не отрылся выдает ошибку
    if(!databaseFile.open(QIODevice::WriteOnly))
    {
        QMessageBox messageBox(this);
        messageBox.setText("Can't open file!");
        messageBox.setWindowTitle("Error.");
        messageBox.exec();
        return;
    }
    //открывает текстовый поток
    QTextStream out(&databaseFile);
    //получает список файлов в паке с раширением jpg
    QStringList imagesFiles = directory.entryList(QStringList() << "*.jpg" <<
"*.JPG", QDir::Files);

    ui->progressBar->setValue(0); //установить значение прогресс-бара в 0
    //перебор фото в папке
```

```
    for(int i = 0; i < imagesFiles.count(); ++i)
    {
        QPoint eye1;
        QPoint eye2;

        QString imageName = imagesFiles[i]; //сохранение имени файла
        imageName.truncate(imageName.length() - 4); //отрезается от имени
расширение
        QStringList eyesString = imageName.split('_'); //строка делится на
части с разделителем _
        eye1.setX(eyesString[0].toInt()); //установка глаз исходя из имени
        eye1.setY(eyesString[1].toInt());
        eye2.setX(eyesString[2].toInt());
        eye2.setY(eyesString[3].toInt());


        QString imagePath = dir + "/" + imagesFiles[i]; //сохранение пути до
файла

        QImage realImage(imagePath); //открытие файла
        unsigned int picSize = this->ui->comboBox->currentText().toInt();
        QImage transformatedImage = ImageVector::prepareImage(realImage,
eye1, eye2, picSize); //трансформация фото

        ImageAttributes* attributes =
ImageVector::calculateAttributes(transformatedImage);//вычисление признаков
трансформированного фото
        out << imagesFiles[i] << "\n"; //запись в файл имени фото
        out << QString::number(attributes->getBlocksCount()*16) + "\n";
//запись в файл колличества блоков
        //перебор всех блоков
        for(int i = 0; i < attributes->getBlocksCount(); ++i)
        {
            //перебор всех признаков
            for(int j = 0; j < 16; ++j)
            {
                out << QString::number(attributes->getAttribute(i, j)) +
"\n"; //запись в файл признаков
            }
        }

        delete attributes; //удаление признаков текуей фото
        ui->progressBar->setValue((int)(100./imagesFiles.count()*(i+1)));
//установка прогресса в бар
    }
}
```

---

main.cpp

```
#include "mainwindow.h"
#include "makedatabasewindow.h"
#include <QApplication>
#include <QString>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    QMainWindow* w = NULL;
    // открытие второго окна по ключу s
```

```
    if ((argc == 2) && (QString(argv[1]).compare("-s", Qt::CaseInsensitive)
== 0))
        w = new MakeDatabaseWindow();
    else
        w = new MainWindow();

    w->show();

    int result = a.exec();
    delete w;

    return result;
}
```