НУКУССКИЙ ФИЛИАЛ ТАШКЕНТСКОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ИМЕНИ МУХАММЕДА АЛЬ-ХОРЕЗМИЙ МИНИСТЕРСТВО ПО РАЗВИТИЮ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙ РЕСПУБЛИКИ УЗБЕКИСТАН

ФАКУЛЬТЕТ «КОМПЬЮТЕРНЫЙ ИНЖИНИРИНГ»
КАФЕДРА ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

У Т В Е Р Ж Д А Ю

Зав. кафедрой
_____Туреniязова А.И.

«____» _____ 2019 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему

«Метод распознавания изображений на мобильном телефоне Android»

Выпускника: _____ У. Кенгесбаев
Научный руководитель: _____ Куандыков К.
Научный консультант _____ Бекбосынов А.

НУКУС - 2019 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Эта выпускная квалификационная работа является частью исследования, проведенного исследовательской ячейкой Embedded Vision группы разработчиков встраиваемых систем EmSD в De Nayer Instituut, возглавляемой Toon Goedemé.

В 2008 году Google выпустила свою первую версию Android SDK, которая представляет собой инструментарий для создания приложения для мобильного телефона с открытым исходным кодом OS Android. Целью нашей работы является использование платформы Android для разработки путеводителя по виртуальному музею для мобильных телефонов. По сути, пользователь должен иметь возможность сфотографировать картину, и приложение должно отреагировать, показав информацию, относящуюся к этой картине.

Для создания этого приложения нам нужно найти или создать алгоритм распознавания изображений, идеальный для нашего приложения. Наше приложение требует алгоритм, который не займет слишком много времени для выполнения. Пользователь, конечно же, не хочет ждать пять минут перед каждой интересующей его картиной. Для нашей дипломной работы нас попросили выполнить алгоритм на телефоне, и поскольку мобильные телефоны имеют ограниченные технические характеристики оборудования, мы должны найти легкий алгоритм.

Другая часть выпускной квалификационной работы - исследование платформы Android. Android - это платформа с открытым исходным кодом, поэтому у нее много возможностей. Он поощряет повторное использование существующих компонентов, поэтому нам придется оценить все его варианты, чтобы выбрать те вещи, которые нам нужны. Объединив эти существующие строительные блоки с нашим собственным кодом, мы сможем создать свободное и простое в использовании приложение.

Первое проливает свет на исследования, которые мы провели в Matlab для определения идеального алгоритма для нашего приложения. Последний

подробно описывает, как мы реализовали это в Android. Мы решили, что музеи, которые хотят использовать наше приложение, должны иметь возможность легко создавать информацию о своих картинах для нашего приложения. Вот почему мы пришли к идее интерфейса управления, написанного на Java.

ГЛАВА-I ОБЗОР СИСТЕМНОЙ АРХИТЕКТУРЫ ANDROID

1.1 Обзор системной архитектуры Android

Android - это первая открытая и бесплатная платформа для мобильных устройств, разработанная членами Open Handset Alliance. Open Handset Alliance - это группа из более чем 40 компаний, включая Google, ASUS, Garmin, HTC, Эти компании собрались вместе, чтобы ускорить и улучшить разработку мобильных устройств. Тот факт, что это открытая платформа, имеет много преимуществ:

• Потребители могут покупать более инновационные мобильные устройства по более низким ценам.

• Операторы мобильной связи могут легко настраивать свои продуктовые линейки и быстрее предлагать новые услуги.

• Производители мобильных телефонов выиграют от более низких цен на программное обеспечение.

• Разработчики смогут легче создавать новое программное обеспечение, поскольку они имеют полный доступ к исходному коду системы и обширной документации API.

В Ryan Paul дает объяснение, касающееся лицензии на open source проекта Android. Как основной участник Android, Google решил выпустить этот проект с открытым исходным кодом под версией 2 Apache Software License (ASL). ASL является разрешающей лицензией, что означает, что код, выпущенный по этой лицензии, может использоваться в продуктах с закрытым исходным кодом, в отличие от лицензий с авторским левом, таких как GPL. Это побуждает все больше компаний вкладывать средства в разработку новых приложений и сервисов для Android, поскольку они могут получить прибыль от этих улучшений для платформы. Обратите внимание, что ASL применяется только к компонентам пользовательского пространства платформы Android. Базовое ядро Linux лицензируется по GPLv2, а сторонние приложения могут быть выпущены под любой лицензией.

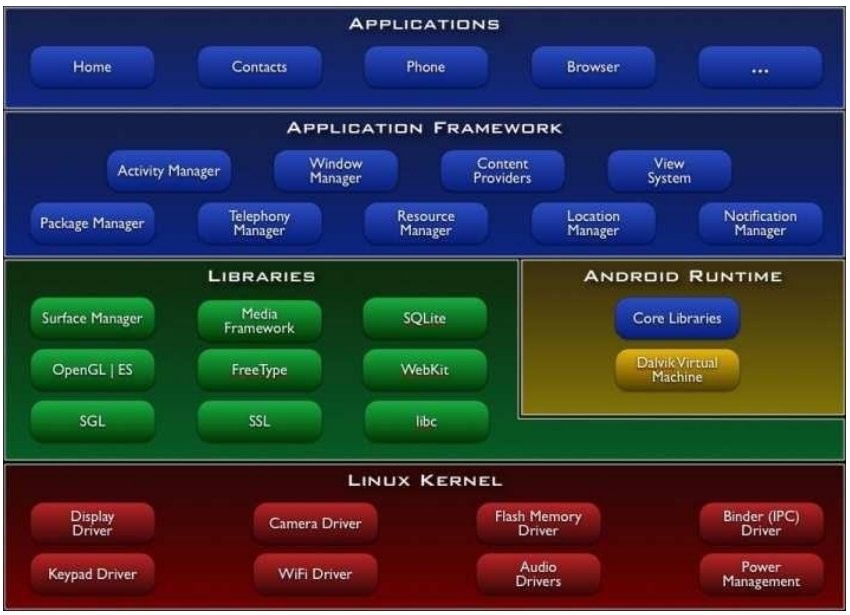

Рисунок-1: Архитектура системы.

В следующих параграфах мы объясним некоторые подробности о платформе Android, основываясь на информации, найденной на официальном сайте Android.

1.1.1 Обзор архитектуры

Android разработан, чтобы быть полным стеком ОС, Middleware и приложений. На рисунке-1 представлен обзор его компонентов.

Android использует ядро Linux 2.6. Google решил использовать Linux в качестве уровня аппаратной абстракции, потому что у него есть рабочая модель драйвера и часто существующие драйверы. Он также обеспечивает управление памятью, управление процессами, безопасность, сети, Следующий уровень содержит библиотеки. Они написаны на C / C ++ и обеспечивают основную мощь платформы Android. Этот уровень состоит из нескольких различных компонентов. Сначала есть Surface Manager. Этот менеджер используется для отображения разных окон в нужное время. OpenGL/ES и SGL - графические библиотеки системы. Первая - это 3D-библиотека, вторая - 2D-библиотека. Их можно комбинировать для использования 2D и 3D в одном приложении. Следующий компонент - Media Framework. Эта структура предоставляется участником Open Handset Alliance, Packet Video. Он содержит большинство кодов, необходимых для обработки наиболее популярных

форматов мультимедиа. Например: MP3, MPEG4, AAC, FreeType отображает все растровые и векторные шрифты, представленные на платформе Android. Для хранения данных используется система управления базами данных SQLite. Webkit - это движок браузера с открытым исходным кодом, который используется как ядро браузера, предоставляемого Android. Они оптимизировали его, чтобы он хорошо отображался на небольших дисплеях.

На этом же уровне находится Android Runtime. Android Runtime содержит одну из самых важных частей Anroid: виртуальную машину Dalvik. Эта виртуальная машина использует Dex-файлы. Эти файлы являются байт-кодами, полученными в результате преобразования .jar-файлов во время сборки. Dex-файлы созданы для работы в ограниченных средах. Они идеально подходят для небольших процессоров, они эффективно используют память, их структуры данных могут быть разделены между различными процессами, и они имеют высокую оптимизированный для процессора интерпретатор байт-кода. Кроме того, можно запустить несколько экземпляров виртуальной машины Dalvik. В среде выполнения Android мы также можем найти библиотеки ядра, содержащие все классы коллекций, утилиты, операции ввода-вывода, Прикладная платформа написана на Java и является инструментарием, который используют все приложения. Это приложения, поставляемые с телефоном, приложения, написанные Google, или приложения, написанные домашними пользователями. Диспетчер операций управляет жизненным циклом всех приложений . Диспетчер пакетов отслеживает приложения, установленные на устройстве. Window Manager - это абстракция JPL поверх сервисов более низкого уровня, предоставляемых Surface Manager. Диспетчер телефонии содержит API-интерфейсы для телефонных приложений. Поставщики контента образуют структуру, позволяющую приложениям обмениваться данными с другими приложениями. Например, приложение «Контакты» может делиться всей своей информацией с другими приложениями. Диспетчер ресурсов используется для хранения локализованных строк, растровых изображений и

других частей приложения, которые не являются кодом. Система просмотра содержит кнопки и списки. Он также обрабатывает отправку событий, макет, рисование, Диспетчер местоположения, Диспетчер уведомлений и Диспетчер XMPP также являются интересными API .

Уровень приложений содержит все приложения: Главная страница, Контакты, Браузер, Пользовательские приложения и так далее. Все они используют Application Framework.

1.1.2 API

В этом разделе перечислены некоторые API, содержащиеся в Application Framework. Теперь мы подробнее рассмотрим более важные из них:

- Менеджер местоположения
- XMPP Сервис
- Менеджер уведомлений
- Просмотры

Менеджер местоположений позволяет получать географическую информацию. Он может дать вам знать ваше текущее местоположение и с помощью намерений может вызывать уведомления, когда вы приближаетесь к определенным местам. Этими местами могут быть заранее определенные местоположения, местоположение друга и так далее. Менеджер местоположений определяет ваше местоположение на основе доступных ресурсов. Если у вашего мобильного телефона есть GPS, Менеджер местоположений может определить ваше местоположение очень подробно в любой момент. В противном случае ваше местоположение будет определяться с помощью идентификаторов вышек сотовой связи или возможной географической информации, содержащейся в сетях WiFi.

Служба XMPP используется для отправки сообщений с данными любому пользователю под управлением Android на своем мобильном телефоне. Эти сообщения данных являются Интентами с парами имя значение. Эти данные могут быть отправлены любым приложением, поэтому они могут содержать информацию любого типа. Его можно использовать для разработки

многопользовательских игр, программ чата .Но также его можно использовать для отправки географической информации другим пользователям. Например, вы можете отправить свое текущее местоположение своим друзьям. Таким образом, их менеджер местоположения может уведомить ваших друзей, когда вы находитесь в их районе. Служба XMPP работает с любой учетной записью Gmail, поэтому приложения могут использовать ее без создания какой-либо серверной инфраструктуры.

Диспетчер уведомлений позволяет добавлять уведомления в строку состояния. Уведомления добавляются из-за связанных событий (Intents), таких как новые SMS-сообщения, новые сообщения голосовой почты и в ближайшее время. Сила уведомлений в строке состояния может быть распространена на любое приложение. Это позволяет разработчикам уведомлять пользователей о важных событиях, которые происходят в их приложении. Это много типов уведомлений, в то время как каждый тип представлен определенной иконкой в строке состояния. Нажав на значок, можно отобразить соответствующий текст. Если вы щелкните по нему еще раз, вы попадете в приложение, которое создало уведомление.

Представления используются для создания приложений из стандартных компонентов. Android предоставляет списки, сетку, галерею, кнопки, флажки Система просмотра также поддерживает несколько методов ввода, размеры экрана и раскладки клавиатуры. Кроме того, есть еще два инновационных взгляда. Первый - это MapView, представляющий собой реализацию механизма рендеринга карт, используемого в приложении «Map». Это представление позволяет разработчикам встраивать карты в свои приложения, чтобы легко предоставлять географическую информацию. Представление может полностью контролироваться приложением: настройка местоположения, масштабирование.

Второе инновационное представление - это веб-представление. Это позволяет встраивать html-контент в ваше приложение.

1.1.3 Анатомия приложения

При написании приложения для Android необходимо учитывать, что приложение можно разделить на четыре компонента:

- Activity
- IntentReceiver
- Service
- ContentProvider

Activity - это пользовательский интерфейс. Например, приложение Почта имеет 3 действия: список почты, окно для чтения писем и окно для составления писем. IntentReceiver - это зарегистрированный код, который запускается при запуске внешним событием. С помощью XML можно определить, какой код должен выполняться при каких условиях (доступ к сети, в определенное время,...). Service - это задача, которая не имеет пользовательского интерфейса. Это долгоживущая задача, которая работает в фоновом режиме. ContentProvider - это часть, которая позволяет обмениваться данными с другими процессами и приложениями. Приложения могут хранить данные любым удобным для них способом, но чтобы сделать их доступными для других приложений, они используют ContentProvider. На рисунке-2 показано это использование ContentProvider.

Основным файлом в приложении Android является файл AndroidManifest.xml. Он содержит информацию, которая должна быть у системы, прежде чем она сможет запустить код приложения. Он описывает компоненты, упомянутые выше, например, он определяет, какое действие будет запущено при запуске приложения. Он также объявляет различные разрешения, такие как доступ в Интернет. Имя приложения и значок, используемые в приложении Home, также определены в этом файле. Когда понадобятся определенные библиотеки, это будет упомянуто здесь. Это лишь некоторые из многих вещей, которые делают файлы AndroidManifest.xml.

Android разработан для поощрения повторного использования и замены компонентов. Предположим, мы хотим написать приложение, которое хочет выбрать фотографию. Вместо того, чтобы писать собственный код для выбора

этой фотографии, мы можем использовать встроенный сервис. Для этого нам нужно сделать запрос или намерение. Затем система с помощью диспетчера пакетов определит, какое установленное приложение лучше всего соответствует нашим потребностям. В этом случае это будет встроенная фотогалерея. Если в дальнейшем мы решим заменить встроенную фотогалерею, наше приложение продолжит работать, потому что система автоматически выберет это новое приложение для сбора фотографий.

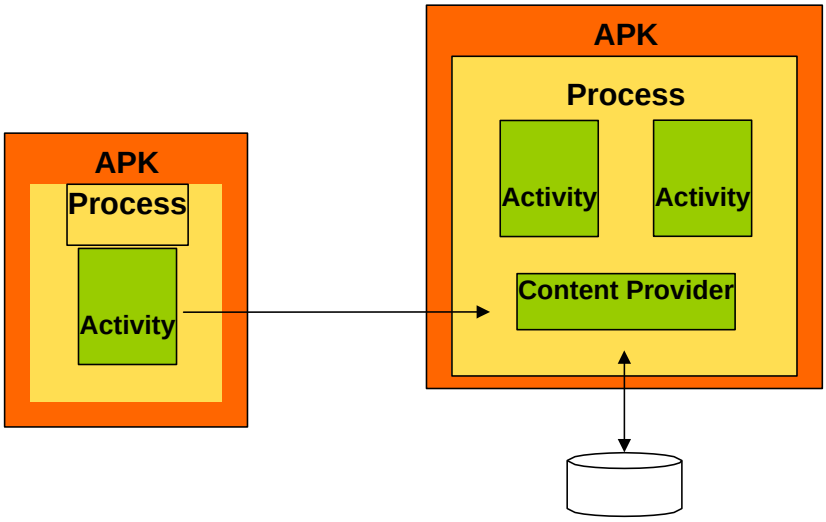

Рисунок -2: Обзор анатомии базового приложения Android

1.1.4 Модель приложения: приложения, задачи, потоки и процессы

В приложении для Android есть три важных условия:

- Пакет Android
- Задача
- Процесс

В то время как в других операционных системах существует сильная корреляция между исполняемым изображением, процессом, значком и приложением, в Android более плавная согласованность.

Пакет Android представляет собой файл, содержащий исходный код и ресурсы приложения. Так что это в основном форма, в которой приложения распространяются среди пользователей. После установки приложения на

экране появится новый значок для доступа к нему. На рисунке-3 показано, как создаются пакеты Android.

При нажатии на значок, созданный при установке пакета, появится новое окно. Пользователь увидит это как приложение, но на самом деле это задача, с которой он имеет дело. Пакет Android содержит несколько действий, одним из которых является точка входа верхнего уровня. При нажатии на значок, задача будет создана для этого действия. Каждое новое начатое действие будет выполняться как часть этой задачи. Таким образом, задача может рассматриваться как стек действий. Новая задача может быть создана с намерением действия, которое определяет, какая это задача. Это намерение можно запустить с помощью флага Intent.FLAG_ACTIVITY_NEW_TASK. Действия, запущенные без этого флага, будут добавлены к той же задаче, что и операция, запускающая ее. Если используется флаг, сходство задач, уникальное статическое имя для задачи (по умолчанию это имя пакета Android), будет использоваться для определения, существует ли уже задача с таким же сходством. Если нет, новая задача будет создана. В противном случае существующее задание будет перенесено, и действие будет добавлено в его стек действий.

Процесс - это то, о чем пользователь не знает. Обычно весь исходный код в пакете Android выполняется в одном низкоуровневом процессе ядра, как также показано на рисунке- 3 Это поведение можно изменить, используя тег процесса. Основное использование процессов:

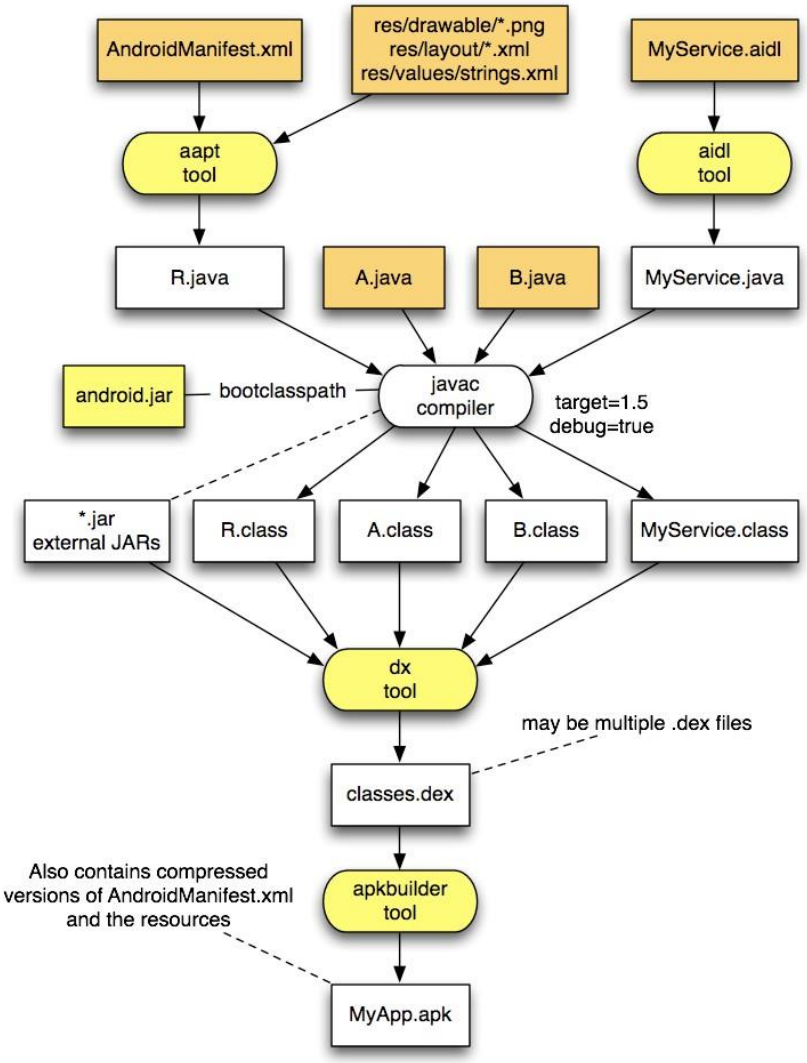

Рисунок-3: Процесс разработки пакета Android

• Повышение стабильности и безопасности за счет помещения ненадежного или нестабильного кода в отдельные процессы.

• Сокращение накладных расходов за счет запуска кода нескольких пакетов Android в одном процессе.

• Разделение тяжелого кода в процессах, которые могут быть уничтожены в любое время, без необходимости уничтожения всего приложения.

Обратите внимание, что два пакета Android, работающих с разными идентификаторами пользователей, не могут быть запущены в одном и том же процессе во избежание нарушения безопасности системы. Каждый из этих пакетов получит свой отдельный процесс. Android пытается сохранить все свои запущенные процессы однопоточными, если приложение не создает свои собственные дополнительные потоки. Поэтому при разработке приложений

важно помнить, что для каждого экземпляра Activity, BroadcastReceiver, Service или ContentProvider не создаются новые потоки. Таким образом, они не должны содержать длинные или блокирующие операции, поскольку они будут блокировать все другие операции в процессе.

1.1.5 Жизненный цикл приложения

В этих параграфах мы собираемся объяснить, как Android работает с разными приложениями, работающими одновременно. Сложность заключается в том, что на Android у нас ограниченные ресурсы. Поэтому управление памятью является одной из важнейших функций Android-системы. Конечно, фреймворк заботится об этом, и разработчик приложений или конечный пользователь даже не должен знать об этих событиях, происходящих в фоновом режиме. Чтобы обеспечить цельный и удобный интерфейс, Android всегда предоставляет пользователю кнопку «назад», которая работает во всех приложениях. Что бы ни делал пользователь, когда он нажимает кнопку «назад», должен появиться предыдущий экран.

Типичным примером для демонстрации и объяснения этой системы является пользователь, желающий прочитать свою электронную почту, щелкнув ссылку на веб-сайт, который открывается в его браузере. Пользователь запускается с домашнего экрана, запускаемого «домашним» системным процессом, где он щелкает, чтобы открыть почтовое приложение. Почтовое приложение запускается путем создания системного процесса для почтового приложения и по умолчанию запускает действие «список рассылки». Деятельность можно рассматривать как одно окно графического интерфейса. Там пользователь нажимает на электронную почту. Перед запуском этого нового окна для просмотра сообщения «системная посылка» сохраняется в системном процессе для почтовой рассылки, и эта посылка содержит текущее состояние активности почтовой рассылки. Это необходимо, потому что в будущем системный процесс может быть удален из памяти (поскольку другому приложению нужна память), но когда пользователь возвращается к действиям со списком рассылки, он ожидает увидеть то же

окно, которое видел раньше. Значение: прокрутка до той же страницы в списке, выбор правильного элемента, Каждый раз, когда начинается новое действие, либо из того же приложения, либо из другого, пакет состояний сохраняется в системном процессе старого действия. Возвращаясь к примеру, активность сообщения открывается, и пользователь может прочитать его сообщение электронной почты. Пользователь нажимает на URL веб-сайта внутри сообщения, поэтому нам нужно запустить процесс браузера. Кроме того, состояние активности сообщений сохраняется в процессе почтовой системы, и пользователь видит браузер, открывающий URL-адрес. Теперь предположим, что на странице есть ссылка на карту. Еще раз пользователь хочет посмотреть на это, он щелкает, и посылка с информацией о состоянии для браузера сохраняется в его системном процессе. Однако теперь у нас есть домашний экран, почтовое приложение и браузер, и нам не хватает памяти. Итак, мы должны убить приложение, чтобы освободить место. Главный экран не может быть убит: он нужен нам, чтобы система реагировала. Браузер также был бы плохим выбором для уничтожения, но почтовое приложение на пару позиций назад в стеке, и убивать безопасно. Приложение карт запускается, создает системный процесс и действие, и пользователь может перемещаться.

Затем он хочет вернуться. Так что теперь активность браузеров восстановлена на первый план. Пакет состояний, который он сохранил при запуске приложения карт, можно отбросить, так как информация о состоянии браузера все еще загружается. Возвращаясь к бэк-стеку, есть активность сообщений из почтового приложения. Этот процесс больше не выполняется, поэтому мы должны запустить его снова. Чтобы сделать это, мы должны предоставить доступную память, и поскольку приложение карт больше не используется, оно уничтожается. Так что теперь почта запускается снова и открывает новую активность сообщений. Это пустое действие без какого-либо полезного контента для пользователя. Однако в системном процессе мы сохранили информационную посылку, и теперь она используется для заполнения активности сообщения. Обратите внимание, что посылка не

содержит содержимого самого действия (почтовое сообщение), а содержит только информацию, необходимую для перестроения исходного действия. В этом случае это может быть описание того, как получить почтовое сообщение из базы данных. Таким образом, почтовое приложение восстанавливает содержание активности сообщения. Пользователь видит сообщение электронной почты точно так же, как оно появилось в прошлый раз, не зная, что действие было прервано загрузкой приложения карт!

1.2 Распознавание изображений

Обработка изображения может быть описана как любое возможное действие, выполняемое над изображением. Это может быть просто обрезка изображения, увеличение контрастности или масштабирование. С тех пор, как оцифровка изображений вошла в компьютерный мир, возникла потребность в распознавании изображений. Распознавание изображений является классической проблемой при обработке изображений. В то время как люди могут легко извлекать объекты из изображения, компьютеры не могут. Для компьютера изображение - это не более чем матрица пикселей. Искусственный интеллект, необходимый для распознавания объектов в этой матрице, должен быть создан программистом. Поэтому большинство существующих методов решения этой проблемы применимы только к конкретным объектам или задачам.

Примеры:

- Оптическое распознавание символов или OCR преобразует изображения в текст.
- Распознавание человеческих лиц, часто используемых в цифровых камерах.
- Оценка позы для оценки ориентации и положения объекта на камеру.
- Контентный поиск изображений для поиска изображений в большем наборе изображений.

Поскольку нам приходилось распознавать картины на изображении, методика, которую мы будем использовать, является примером последней упомянутой задачи распознавания изображений.

Широко используемый метод распознавания контента в изображении показан на рисунке-4 и состоит из следующих двух частей:

- Определение функций, характерных для определенного изображения.
- Сопоставление различных изображений на основе этих функций

В этом разделе мы рассмотрим несколько часто используемых методов, где приведено краткое описание некоторых методов определения характеристик с последующим более подробным описанием метода, который мы будем использовать (т. е. инварианты цветовых моментов). В этом разделе мы объясним некоторые методы, которые можно использовать для сопоставления изображений на основе их функций. Мы сконцентрируемся на методах сопоставления изображений, представленных инвариантами цветовых моментов.

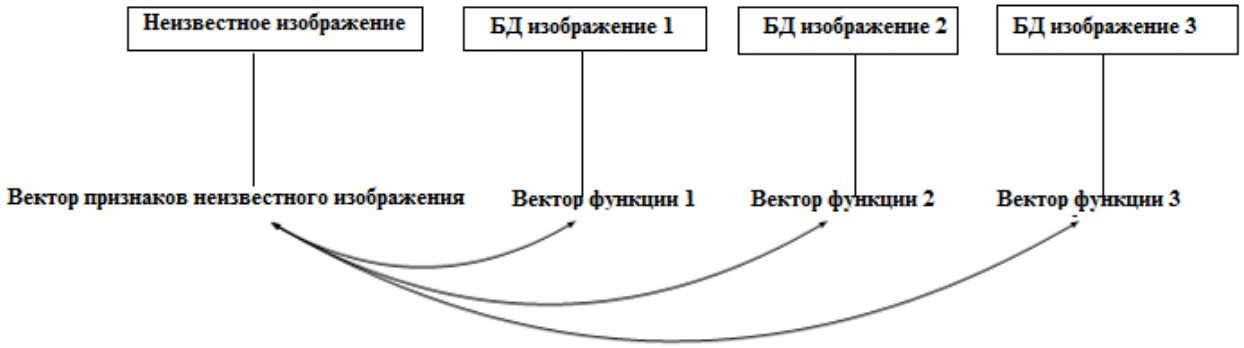

Рисунок-4: Принцип распознавания изображений

1.2.1 Определение функций

Чтобы распознать изображение, нам нужно определить некоторые функции, которые можно объединить, чтобы создать уникальную идентичность для этого изображения. Два совпадающих изображения, например два изображения одной и той же картины, должны иметь почти одинаковые идентичности. Поэтому важно, чтобы мы использовали надежную технику для определения функций, которые не зависят от условий окружающей среды, таких как точка обзора и освещение.

1.2.1.1 Доступные методы

На сегодняшний день самый простой способ распознавания изображений - попиксельное соответствие. Каждый пиксель из исходного изображения сравнивается с соответствующим пикселем в картинке библиотеки. Изображение должно быть точно таким же, чтобы соответствовать. Этот метод совсем не интересен, так как он не учитывает изменения угла обзора, освещения или шума. Поэтому, чтобы сравнить снимки, сделанные на Android-телефоне, с изображениями в базе данных музейных экспонатов, нам нужны более продвинутые методы.

Один метод использует локальные особенности изображения; Есть два обычно используемых алгоритма:

SIFT или масштабно-инвариантное преобразование объектов - это алгоритм, который использует функции, основанные на появлении объекта в определенных точках интереса. Эти особенности являются инвариантами изменений шума, освещенности и обзора. Их также относительно легко извлечь, что делает их очень подходящими для сопоставления с большими базами данных. Эти точки интереса или ключевые точки обнаруживаются путем свертки изображения с гауссовыми фильтрами в разных масштабах, что приводит к размытым по Гауссу изображениям. Ключевыми точками являются максимумы и минимумы разности последовательных размытых изображений. Однако это приводит к слишком большому количеству ключевых точек, поэтому некоторые из них будут отклонены на более поздних этапах алгоритма, поскольку они имеют низкую контрастность или расположены вдоль края.

SURF или ускоренные надежные функции основаны на SIFT, но алгоритм работает быстрее, чем SIFT, и обеспечивает лучшую инвариантность преобразований изображения. Он обнаруживает ключевые точки на изображениях, используя гессенскую матрицу. Этот метод, например, используется в «Интерактивном музейном руководстве: быстрое и надежное распознавание музейных предметов».

Метод, который мы будем использовать, генерирует инвариантные функции на основе цветовых моментов. Это метод, использующий глобальные функции, и не основанный на локальных функциях, таких как SIFT и SURF. При использовании локальных функций необходимо сравнивать каждую функцию, а время обработки значительно выше, чем при использовании глобальных функций. Тем не менее, можно использовать локальные функции, и они будут работать лучше на трехмерных объектах. Однако наша главная цель - 2D-распознавание, и мы думаем, что цветовые моменты могут справиться с этим с хорошим балансом скорости распознавания и скорости. Цветовые моменты объединяют силу координат пикселей и интенсивности пикселей. Эти цветовые моменты можно комбинировать различными способами для создания инвариантов цветовых моментов. В следующем разделе мы дадим более подробное объяснение этого метода.

1.2.1.2 Наш метод: цветовые инварианты момента

Инварианты цветовых моментов - это особенности изображения, основанные на обобщенных цветовых моментах, которые разработаны, чтобы быть инвариантными относительно точки обзора и освещения. Инварианты, использованные в, сгруппированы по определенному типу геометрического и фотометрического преобразования. Мы охватим все группы (GPD, GPSO, PSO, PSO * и PAFF) и решим, какая группа лучше всего подходит для нашего приложения.

Поскольку инварианты основаны на обобщенных цветовых моментах, мы начнем с объяснения того, что такое обобщенные цветовые моменты. Рассмотрим изображение с площадью Ω. Комбинируя координаты x и y каждого пикселя с их соответствующими интенсивностями цвета I_R, I_{tt} и I_B, мы можем вычислить обобщенный момент цвета:

1.2.1 Определение функций

Чтобы распознать изображение, нам нужно определить некоторые функции, которые можно объединить, чтобы создать уникальную идентичность для этого изображения. Два совпадающих изображения,

например два изображения одной и той же картины, должны иметь почти одинаковые идентичности. Поэтому важно, чтобы мы использовали надежную технику для определения функций, которые не зависят от условий окружающей среды, таких как точка обзора и освещение.

1.2.1.1 Доступные методы

На сегодняшний день самый простой способ распознавания изображений - попиксельное соответствие. Каждый пиксель из исходного изображения сравнивается с соответствующим пикселем в картинке библиотеки. Изображение должно быть точно таким же, чтобы соответствовать. Этот метод совсем не интересен, так как он не учитывает изменения угла обзора, освещения или шума. Поэтому, чтобы сравнить снимки, сделанные на Android-телефоне, с изображениями в базе данных музейных экспонатов, нам нужны более продвинутые методы.

Один метод использует локальные особенности изображения; Есть два обычно используемых алгоритма:

SIFT или масштабно-инвариантное преобразование объектов [8] - это алгоритм, который использует функции, основанные на появлении объекта в определенных точках интереса. Эти особенности являются инвариантами изменений шума, освещенности и обзора. Их также относительно легко извлечь, что делает их очень подходящими для сопоставления с большими базами данных. Эти точки интереса или ключевые точки обнаруживаются путем свертки изображения с гауссовыми фильтрами в разных масштабах, что приводит к размытым по Гауссу изображениям. Ключевыми точками являются максимумы и минимумы разности последовательных размытых изображений. Однако это приводит к слишком большому количеству ключевых точек, поэтому некоторые из них будут отклонены на более поздних этапах алгоритма, поскольку они имеют низкую контрастность или расположены вдоль края.

SURF или ускоренные надежные функции основаны на SIFT, но алгоритм работает быстрее, чем SIFT, и обеспечивает лучшую инвариантность

преобразований изображения. Он обнаруживает ключевые точки на изображениях, используя гессенскую матрицу. Этот метод, например, используется в «Интерактивном музейном руководстве: быстрое и надежное распознавание музейных предметов».

Метод, который мы будем использовать, генерирует инвариантные функции на основе цветовых моментов. Это метод, использующий глобальные функции, и не основанный на локальных функциях, таких как SIFT и SURF. При использовании локальных функций необходимо сравнивать каждую функцию, а время обработки значительно выше, чем при использовании глобальных функций. Тем не менее, можно использовать локальные функции, и они будут работать лучше на трехмерных объектах. Однако наша главная цель - 2D-распознавание, и мы думаем, что цветовые моменты могут справиться с этим с хорошим балансом скорости распознавания и скорости. Цветовые моменты объединяют силу координат пикселей и интенсивности пикселей. Эти цветовые моменты можно комбинировать различными способами для создания инвариантов цветовых моментов. В следующем разделе дается более подробное объяснение этого метода.

1.2.1.2 Наш метод: цветовые инварианты момента

Метод, который мы реализовали, основан на исследованиях, проведенных Mindruetal. В этом разделе мы объясняем нашу интерпретацию этого исследования.

Для нашего тезиса мы хотим иметь возможность распознавать картины на фотографиях, сделанных в разных ситуациях. Это означает, что картины претерпят геометрические и фотометрические преобразования. Наш способ справиться с этими преобразованиями - получить набор инвариантных функций. Эти особенности не меняются, когда фотография сделана под разными углами и при разных условиях освещения. Инварианты, используемые в нашей диссертации, основаны на обобщенных цветовых моментах. В то время как цветовые гистограммы характеризуют изображение на основе его цветового распределения, обобщенные цветовые моменты

объединяют координаты пикселя и интенсивности цвета. Таким образом, они дают информацию о пространственном расположении цветов.

Инварианты цветовых моментов - это особенности изображения, основанные на обобщенных цветовых моментах, которые разработаны, чтобы быть инвариантными относительно точки обзора и освещения. Инварианты, использованные в, сгруппированы по определенному типу геометрического и фотометрического преобразования. Мы охватим все группы (GPD, GPSO, PSO, PSO * и PAFF) и решим, какая группа лучше всего подходит для нашего приложения.

Поскольку инварианты основаны на обобщенных цветовых моментах, мы начнем с объяснения того, что такое обобщенные цветовые моменты. Рассмотрим изображение с площадью Ω. Комбинируя координаты x и y каждого пикселя с их соответствующими интенсивностями цвета I_R, I_{tt} и I_B, мы можем вычислить обобщенный момент цвета:

$$M_{pq}^{abc} = \iint_{\Omega} x^p y^q I_R^a I_G^b I_B^c \, dx \, dy \qquad (1)$$

В формуле (2.1) сумма p + q определяет порядок, а a + b + c определяет степень цветового момента. Если мы изменим эти параметры, мы можем создать много разных моментов с их собственным значением. Далее мы будем использовать только моменты максимального порядка 1 и максимальной степени 2.

Мы приведем небольшой пример, чтобы объяснить, как эти обобщенные цветовые моменты можно использовать для создания инвариантов. Как уже упоминалось выше, различные цветовые моменты могут быть созданы путем изменения параметров a, b, c, p и q:

$$M_{10}^{100} = \iint_{\Omega} x I_R \, dx \, dy \qquad (2)$$

$$M_{00}^{100} = \iint_{\Omega} I_R \, dx \, dy \qquad (3)$$

Уравнение (2) приводит к сумме интенсивностей красного каждого пикселя, оцениваемых по его координате x. Уравнение (3) дает общую сумму интенсивностей красного каждого пикселя. Рассмотрим картину, освещенную

красной лампой, поэтому интенсивность красного в каждом пикселе масштабируется с коэффициентом R.

Результат уравнений (2) и (3) не может быть инвариантным к этому эффекту, поскольку они оба масштабируются на R. Чтобы устранить этот эффект, мы можем использовать (2) и (3) следующим образом:

$$\frac{M_{10}^{100}}{M_{00}^{100}} \qquad (4)$$

Разделив (2.2) на (2.3), получим среднюю координату x для красного цвета. Однако это число является инвариантом эффекта масштабирования, поскольку коэффициент масштабирования R присутствует как в знаменателе, так и в знаменателе. Это только очень простой пример, поэтому мы не можем использовать его для сравнения двух изображений в реальных ситуациях. Инварианты, которые мы будем использовать для нашего алгоритма распознавания изображений, имеют более высокую степень сложности.

(а) оригинал→(б) евклидово→ (в) аффинное→ (г) проективное

Рисунок-5: Геометрические преобразования

Как мы уже говорили, различные инварианты, перечисленные в, сгруппированы по типу геометрических и фотометрических преобразований. На рисунке-5 показаны некоторые примеры геометрического преобразования. Однако большинство геометрических преобразований можно упростить до аффинных преобразований:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a11 & a12 \\ a21 & a22 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \qquad (5)$$

Если в любом случае проективное преобразование нельзя упростить, нам придется нормализовать его, прежде чем мы сможем вычислить инварианты, потому что проективные инвариантные моментные инварианты не существуют.

Если мы рассмотрим фотометрические преобразования, существует три типа:

• Тип D: диагональ

$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} = \begin{pmatrix} S_R & 0 & 0 \\ 0 & S_G & 0 \\ 0 & 0 & S_B \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \qquad (6)$$

Тип SO: масштабирование и смещение

$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} = \begin{pmatrix} S_R & 0 & 0 \\ 0 & S_G & 0 \\ 0 & 0 & S_B \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} O_R \\ O_G \\ O_B \end{pmatrix} \qquad (7)$$

• Тип AFF: аффинный

$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} = \begin{pmatrix} a_{RR} & a_{RG} & a_{RB} \\ a_{GR} & a_{GG} & a_{GB} \\ a_{BR} & a_{BG} & a_{BB} \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} O_R \\ O_G \\ O_B \end{pmatrix} \qquad (8)$$

Первые две группы инвариантов - это те, которые используются в случае аффинного геометрического преобразования. Они называются GPD и GPSO. G обозначает их инвариантность для аффинных геометрических преобразований. Первый инвариантен для фотометрических преобразований типа диагональ (уравнение 6), второй для фотометрических преобразований типа масштабирование-смещение (уравнение 7). Диагональ означает только масштабирование в каждой цветовой полосе.

Вторые две группы можно использовать как в случае аффинных, так и проективных геометрических преобразований. Обратите внимание, что нормализация геометрического преобразования необходима, потому что инвариантные функции для проективных преобразований не могут быть построены. Эти группы называются PSO и PAFF. Первый используется для масштабных фотометрических преобразований (уравнение 7), второй для аффинных преобразований (уравнение 8). В они заметили, что в случае PSO-инвариантов может иметь место численная нестабильность. Вот почему они перепроектировали эти инварианты и назвали эти новые инварианты PSO *.

1.2.2 Сопоставление изображений

В этом разделе поясняется, как получить характеристики, инвариант точек обзора и изменения освещения. Много разных инвариантов возможно, комбинируя обобщенные цветовые моменты по-разному. Представьте, что мы нашли тип инварианта (то есть GPD, GPSO, PSO * или PAFF), который удовлетворяет нашим потребностям. Мы можем сгруппировать некоторые инварианты этого типа в так называемый вектор признаков. Этот вектор признаков можно рассматривать как уникальную идентификацию изображения. Когда два изображения представляют один и тот же контент, вектор их признаков будет почти одинаковым. Математически это означает, что расстояние между этими векторами очень мало. На основании этих расстояний мы можем попытаться найти совпадение изображений в большой базе данных изображений.

Рассмотрим два вектора в n-мерном пространстве: x_1 x_2 ... x_n и y_1 y_2 ... y_n. Расстояние между этими векторами существует в разных формах. Первое расстояние - это евклидово расстояние (уравнение 9). Это расстояние является прямым расстоянием между двумя векторами, как показано на рисунке 6, и основано на теореме Пифагора

$$d_E(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \qquad (9)$$

Второе расстояние - это расстояние до Манхэттена (уравнение 10). Это расстояние является суммой расстояний в каждом измерении, как показано на рисунке 7.

$$d_M(x,y) = \sqrt{\sum_{i=1}^{n}|x_i - y_i|} \qquad (10)$$

Оба этих расстояния могут использоваться для сопоставления изображений, но может возникнуть проблема. Предположим, мы используем два инварианта для построения нашего векторного признака. Первый инвариант x_1 приводит к значениям в диапазоне [x_1, min, x_1, max]. Второй инвариант x_2 приводит к гораздо более широкому диапазону значений [x_2, min, x_2, max]. Если мы вычислим евклидово расстояние или манхэттенское

расстояние по одной из приведенных выше формул, второй инвариант будет оказывать гораздо большее влияние на результат. Таким образом, процесс сопоставления изображений основан только на втором инварианте. Эта проблема может быть решена путем нормализации значений каждого измерения.

Рисунок 8 показывает пару векторов признаков, состоящих из x_1 и x_2. Значения x_1 и x_1 имеют стандартное отклонение σ_1 соответственно σ_2. Каждый вектор может быть нормализован путем деления значения

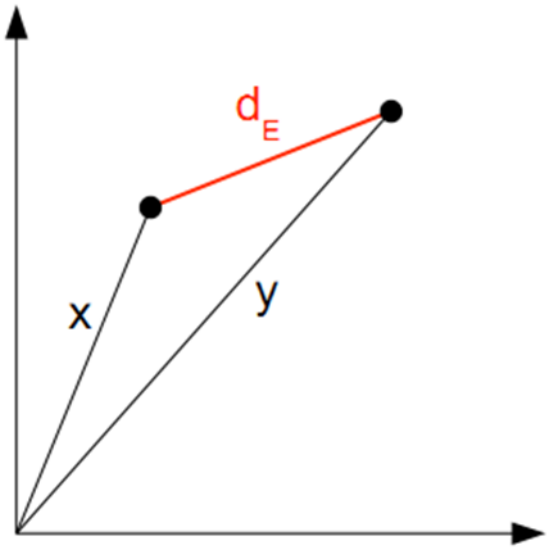

Рисунок 6: Пример евклидова расстояния

каждое измерение x_i их стандартным отклонением σ_i. Мы можем адаптировать евклидово расстояние и манхэттенское расстояние таким же образом, что приведет к нормированному евклидову расстоянию (уравнение 11) и нормализованному манхэттенскому расстоянию (уравнение 12).

$$d_{E,n}(x,y) = \sqrt{\sum_{i=1}^{n}\left(\frac{x_i - y_i}{\sigma_i}\right)^2} \qquad (11)$$

$$d_{M,n}(x,y) = \sqrt{\sum_{i=1}^{n}\left|\frac{x_i - y_i}{\sigma_i}\right|} \qquad (12)$$

Эти расстояния также известны как расстояния Махаланобиса.

Если мы хотим сопоставить изображение с большой базой данных, нам нужно рассчитать расстояние до каждого изображения в этой базе данных. Это приводит к списку расстояний, из которых наименьшее расстояние

представляет наиболее подходящее изображение. Это, однако, не гарантирует, что это изображение имеет тот же контент, что и исходное изображение. Возможно, что база данных не содержит такого изображения, поэтому алгоритм распознавания изображений выберет наиболее похожее, но неисправное изображение. Нам нужно отклонить этот результат. Это можно сделать, задав определенные ограничения для лучшего соответствия, найденного в базе данных. Есть две возможности:

$d_{min} < d_{th}$: определено определенное пороговое расстояние d_{th}, и наименьшее расстояние d_{min} должно быть меньше этого порогового расстояния. Определить это пороговое расстояние непросто, и поэтому это не очень хорошее решение.

$\frac{d_{min}}{d_{min+1}} < r_{th}$: расстояние во втором лучшем матче должно быть в r_{th} раз больше, чем в лучшем. Это лучшее решение, поскольку оно дает больше гибкости.

Для сопоставления изображений нам нужно решить, какое расстояние мы будем использовать, и решить, нужна ли нам нормализация. На основании рассчитанных расстояний мы можем выбрать лучшее соответствие из базы данных,

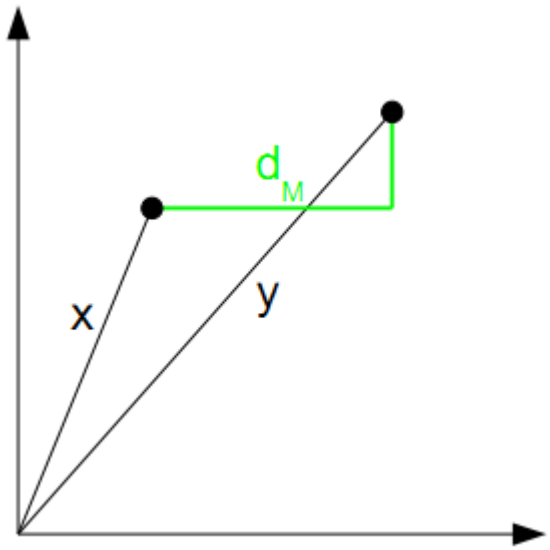

Рисунок 7: Пример манхэттенского расстояния.

но в некоторых случаях нам нужно отклонить этот результат. В главе 2 объясняются наши исследования и тесты по этой технике. Тем не менее, другие уже провели исследования по той же теме (распознавание изображений на мобильном телефоне), используя другие методы. Следующий раздел проливает свет на эту работу.

ГЛАВА 2 РАСПОЗНАВАНИЕ ИЗОБРАЖЕНИЙ В MATLAB

2.1 Реализация инвариантов

Основано на работе Mindruetal. мы упомянули выше, где мы решили реализовать некоторые из предложенных инвариантов в Matlab.

Мы выбрали пару инвариантов с различной вычислительной сложностью и разными целями: инвариант для геометрических преобразований или нет, и т. д. Для простоты использования мы дали этим инвариантам «дружественные имена»: GPDS1, GPDD1, GPSOS12, GPSOD1, GPSOD2, PSO1, PSO2 и PSO3. Все эти инварианты задаются с помощью «обобщенных цветовых моментов», как определено формулой (1) .

Вот список протестированных нами инвариантов:

Тип 1 (GPD)

GPD: инвариант для геометрических и фотометрических преобразований типа Diagonal GPDS1:

$$S_{02} = \frac{M_{00}^{2} M_{00}^{0}}{(M_{00}^{1})^{2}} \qquad (1)$$

GPDD1:

$$D_{02} = \frac{M_{00}^{11} M_{00}^{00}}{M_{00}^{10} M_{00}^{01}} \qquad (2)$$

Тип 2 (GPSO)

GPSO: геометрические и фотометрические преобразования масштабирования и набор GPSOS12:

$$S_{12} = \frac{\left\{ \begin{matrix} M_{10}^{2} M_{01}^{1} M_{00}^{0} - M_{10}^{2} M_{00}^{1} M_{01}^{0} - M_{01}^{2} M_{10}^{1} M_{00}^{0} \\ + M_{01}^{2} M_{00}^{1} M_{10}^{0} - M_{00}^{2} M_{10}^{1} M_{01}^{0} - M_{00}^{2} M_{01}^{1} M_{00}^{0} \end{matrix} \right\}^{2}}{(M_{00}^{0})^{2} \left[M_{00}^{2} M_{00}^{0} - (M_{00}^{1})^{2} \right]^{3}} \qquad (3)$$

GPSOD1:

$$D_{02} = \frac{\left[M_{00}^{11} M_{00}^{00} - M_{00}^{10} M_{00}^{01} \right]^{2}}{\left[M_{00}^{20} M_{00}^{00} - (M_{00}^{01})^{2} \right] \left[M_{00}^{02} M_{00}^{00} - (M_{00}^{01})^{2} \right]} \qquad (4)$$

GPSOD2:

$$D_{12}^{1} = \frac{\left\{ \begin{matrix} M_{10}^{10}M_{01}^{01}M_{00}^{00} - M_{10}^{10}M_{00}^{01}M_{01}^{00} - M_{01}^{10}M_{10}^{01}M_{00}^{00} \\ + M_{01}^{10}M_{00}^{01}M_{10}^{00} - M_{00}^{10}M_{10}^{01}M_{01}^{00} - M_{00}^{10}M_{01}^{01}M_{10}^{0} \end{matrix} \right\}^{2}}{(M_{00}^{00})^{4}\left[M_{00}^{20}M_{00}^{00} - (M_{00}^{10})^{2}\right]\left[M_{00}^{02}M_{00}^{00} - (M_{00}^{01})^{2}\right]^{1}} \quad (5)$$

Тип 3 (PSO)

PSO: инвариант для фотометрических преобразований типа Scaling и Offset PSO1:

$$S_{11} = \frac{M_{00}^{0}M_{10}^{1} - M_{10}^{1}M_{00}^{1}}{M_{00}^{0}M_{01}^{1} - M_{01}^{0}M_{00}^{1}} \quad (6)$$

PSO2:

$$S_{12}^{1} = \frac{M_{pq}^{0}M_{pq}^{2} - (M_{pq}^{1})^{2}}{(M_{00}^{0}M_{pq}^{q} - M_{pq}^{0}M_{00}^{1})^{2}},\ pq \in \{01, 10\} \quad (7)$$

PSO3:

$$S_{12}^{2} = \frac{M_{00}^{0}M_{00}^{2} - M_{00}^{1}M_{00}^{1}}{(M_{00}^{0}M_{00}^{1} - M_{10}^{0}M_{00}^{1})(M_{00}^{0}M_{01}^{1} - M_{01}^{0}M_{00}^{1})} \quad (8)$$

Реализация всех алгоритмов в Matlab была упрощена благодаря использованию самописной GCM-функции («обобщенные цветовые моменты»), которая выглядела так:

```
1 function [m]=gcm(I,p,q,a,b,c)
2 m=0;
3 [height,width,k]=size(I);
4
5 for x=1:width
6 for y=1:height
7 m=m+(x^p*y^q*I(yx,1*)^a*I(y,*x,2)^b*I(y,x,3)^c);
8 end
9 end
```

Listing 2.1. Оригинальная реализация функции Generalized Color Moments

Этот подход сработал, однако производительность не была отличной. Мы также заметили, что выполнение происходит только на одном из ядер наших компьютеров, в то время как наша версия Matlab должна поддерживать несколько ядер. В особенности потому, что мы хотели протестировать производительность алгоритмов на множестве изображений, повышение

производительности стало очень важным для сокращения времени вычислений, поэтому мы решили попробовать и оптимизировать код. Причина нашей низкой производительности заключается в том, что Matlab действительно не оптимизирован для циклов. Мы использовали два вложенных цикла For, и их устранение оказалось бы полезным. Поэтому мы адаптировали наш код для использования матричных операций, которые Matlab может выполнять более эффективно. После оптимизации наш код был закончен:

```
1 function [m]=gcm(I,p,q,a,b,c)
2 [height,width,k]=size(I);
3
4 Iscaled = I(:,:,1).^a .*I(:,:,2).^b .*I(:,:,3).^c;
5 LinX = 1:width;
6 LinY = 1:height;
7 LinX = LinX .^p;
8 LinY = LinY .^q;
9 LinY = transpose(LinY);
10 Mat = LinY     LinX;
11 Ipos = Iscaled .   double(Mat);
12
13 m = sum(sum(Ipos));
```

Listing 2.2. Оптимизированная реализация функции обобщенных цветовых моментов

2.2 Пригодность инвариантов

Предыдущий раздел (2.1) дал некоторые возможные алгоритмы, которые будут использоваться для нашего приложения. Тем не менее, мы сначала должны были выяснить, какой из них был наиболее подходящим. Чтобы протестировать алгоритмы, были использованы три набора изображений, специально разработанные для тестирования алгоритмов распознавания изображений. Три набора были основаны на одном объекте, будь то книга, другая книга или бутылка. Цель состояла только в том, чтобы найти алгоритм, который хорошо работает на 2D-объектах, поэтому бутылка была наименее актуальной. Однако, поскольку бутылку всегда рассматривали

с одной и той же стороны, мы решили включить ее в наши тесты. Наборы изображений, которые мы использовали для этого, можно найти в приложении.

Затем был рассчитан вектор с инвариантами цветовых моментов для каждого из изображений, по 11 изображений на объект, т.е. всего 33 вектора. Конечно, мы написали небольшой Matlab-скрипт для автоматизации этого и вывода результирующих векторов в CSV-файл.

Этот CSV-файл был входом для другого Matlab-скрипта, который вычисляет евклидово расстояние всех возможных комбинаций. Исходя из N= 33 изображений, мы получили результирующую матрицу размером N * N, которая затем была масштабирована. Каждая ячейка матрицы содержала число от 0 до 1, где 0 означает «нет разницы между изображениями», а 1 - «максимальная разница между изображениями во всем наборе». Вскоре мы поняли, что очень сложно судить об алгоритме, просто взглянув на цифры. Вот почему мы преобразовали эту матрицу в изображение в оттенках серого размером 33 * 33 пикселей. Когда два вектора абсолютно одинаковы, их расстояние равно 0 или, если смотреть на изображение, черный пиксель. Соответствующее значение 1 привело к значению серого 255, белый. Это изображение будет называться «картой» в остальной части нашего выпускная квалификационная работа.

После реализации инвариантов, перечисленных выше карты были созданы для каждого инварианта. Результаты можно найти ниже. В идеальной ситуации каждый из 3 объектов соответствует самому себе, и ни один из других объектов. Карта для идеального инварианта - рисунок 9. Следовательно, о пригодности инварианта можно судить, сравнивая полученную карту с идеальной картой.

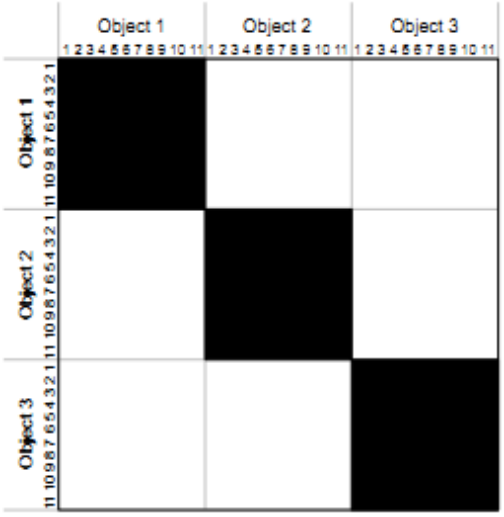

Рисунок 9: Идеальный результат: карта для идеально работающей инвариантной формулы

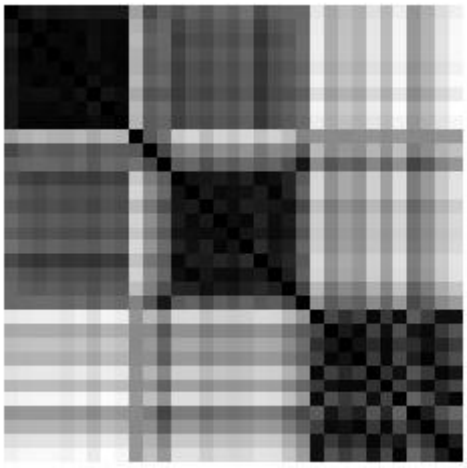

Рисунок 10: Сгенерированный результат для GPDS1-инварианта

2.3 Соответствие векторов объектов

Растровые изображения, описанные в предыдущем разделе, дают хороший общий обзор возможностей каждого инварианта. Следующим шагом в нашем исследовании Matlab было тестирование, можем ли мы на самом деле найти соответствие для изображения в базе данных. Такая база данных заполнена расчетами векторов признаков набора тестовых изображений.

Как объяснено выше, алгоритм сопоставления пытается найти наилучшее совпадение в базе данных на основе расстояния между векторами признаков. Объект базы данных, вектор характеристик которого имеет наименьшее расстояние до вектора «неизвестного» объекта, является наилучшим совпадением. Однако, как мы уже объяснили, даже когда в базе данных нет реальных совпадений, мы всегда найдем наилучшее совпадение. Поэтому мы используем пороговое значение, чтобы отклонить эти ложные совпадения. Число, полученное в результате деления наименьшего расстояния на второе наименьшее, должно быть меньше порогового значения.

В файле Matlab findmatch.m мы реализовали то, что объяснялось выше. Функция findmatch ожидает имя набора тестов, тип инварианта и пороговое значение. Затем он загружает файл CSV банка данных и файл CSV,

содержащий векторы всего набора. Затем он начинает поиск наилучшего соответствия для каждого вектора в тестовом наборе. Каждое совпадение сохраняется в матрице CSV, что в конечном итоге будет результатом функции findmatch.

```
1 function [CSV]=findmatch(setname,invariant,thr)
2
3 DB=csvread(['./testresultaten/databank/db_',
upper(invariant),'.csv']);
4 objectnr=0;
5 testfile=['./testresultaten/',setname,'
/testres_',upper(invariant),'.csv'];
6 testset=csvread(testfile);
7 CSV=zeros(size(testset,1),5);
8
9  for l = 1:size(testset,1);
10        imgnr=testset(l,1);
11        if imgnr==1
12          objectnr=objectnr+1;
13        end
14
15        res=testset(l,2:4);
16    %инициализировать расстояния в 2000. Это гарантирует, что
первое вычисленное расстояние будет меньше.
17 smallest=[2000 0];
18 sec_smallest=[2000 0];
19
20 for nr=1:size(DB,1)
21          dist=sqrt((res(1)-DB(nr,2))^2+ ...
22(res(2)-DB(nr,3))^2+ ...      %вычисление евклидова
расстояния
23             (res(3)-DB(nr,4))^2);
24
25          if dist<smallest(1)
26            sec_smallest=smallest;
27            smallest=[dist nr];
28          end
29        end
```

```
30
31 if smallest(1)/sec_smallest(1)<thr  %проверить,
соответствует ли лучшее соответствие требованию
32 res=smallest(2);
33 else
34 res=0;
35    end
36
37
     CSV(l,:)=[objectnr,imgnr,smallest(2),sec_smallest(2),r
es];
38 end
```

Listing 2.3. Findmatch: тестовая реализация алгоритма сопоставления

Обратите внимание, что мы решили проверить евклидово расстояние, а не манхэттенское. Мы создали растровые изображения для обоих расстояний, но евклидово привело к более четкой разнице между положительными и отрицательными совпадениями. Мы также решили не объединять различные инвариантные типы, потому что мы заметили, что вычисление одного инвариантного типа уже заняло некоторое время в Matlab. Вычисление двух из этих инвариантов заняло бы слишком много времени, чтобы быть полезным в нашем возможном приложении для Android. Вот почему мы также не использовали расстояния Махаланобиса, как описано выше.

Функция findmatch генерирует длинный список чисел в качестве вывода. Первый и последний номер каждой строки списка являются наиболее интересными для нас. Если эти числа равны, мы нашли положительное совпадение. Если эти цифры различаются, мы нашли ложноположительное совпадение. Это означает, что наш алгоритм решил, что изображения совпадают, но на самом деле это другой объект. Последнее число также может быть 0, что означает, что мы нашли совпадение, а затем отклонили его, поскольку оно не соответствовало спецификации порогового значения.

2.4 Нахождение идеального порогового значения

Результаты функции findmatch, описанные в предыдущем разделе , полезны для определения, можем ли мы найти достаточно совпадений, а также

для нахождения идеального порогового значения для нашего алгоритма. Однако ручное изучение выходных списков findmatch было бы слишком трудоемким, и мы не получили бы хорошего обзора результатов. Поэтому мы создали другую функцию с именем findmatchbatch, которая многократно выполняет функцию findmatch, увеличивая пороговое значение с шагом 0,01. Каждый результат findmatch проверяется, и findmatchbatch отслеживает количество совпадений, ложных срабатываний и отклоненных совпадений для каждого порогового значения. Когда все пороговые значения были изучены, он генерирует график, содержащий проценты каждого из вышеупомянутых чисел.

```
1 function findmatchbatch(setname,invariant)
2
3 for i=1:100;
4findmatchres = findmatch(setname,invariant,i/100);
5        res_orig_obj = findmatchres(:,1);
6res_found_match = findmatchres(:,5);
7total = size(findmatchres(:,5));
8matches(i) = sum((res_found_match-res_orig_obj)==0)/total(1);
9 iszero(i) = sum(res_found_match==0)/total(1);
10 end
11
12 misses = 1 - matches - iszero;
13 YMatrix1=[matches' misses' iszero']. 100; % матрица,
используемая для построения графа
14
15 %Создание графа
16 ...
```

Listing 2.4: findmatchbatch: Создание графиков, показывающих влияние порогового значения на скорость распознавания

На основе сгенерированного графика мы смогли определить пороговое значение, которое нам понадобится в нашем приложении для Android. Пороговое значение должно обеспечивать высокий процент положительных совпадений, и, по возможности, большинство ложных срабатываний должно

быть отклонено. Результаты этой функции и наша интерпретация графиков приведены в следующем разделе.

2.5 Результаты

В предыдущих двух разделах описаны методы определения пригодности различных инвариантов и способы получения идеального порогового значения для сопоставления. Тестирование алгоритмов, как описано выше, позволило создать карты для измерения пригодности алгоритмов, а метод, описанный в предыдущем разделе , рассчитал идеальное пороговое значение.

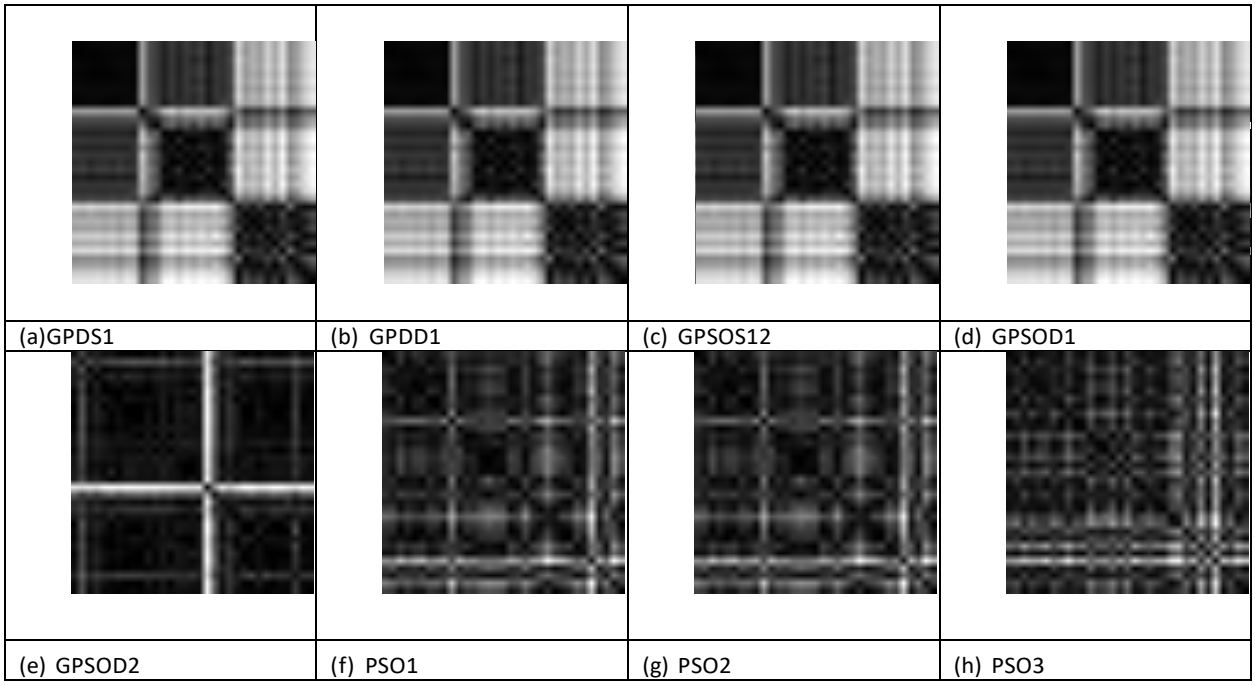

Рисунок 11: Карты для алгоритмов, созданы с использованием метода,

Карты на рисунке 11 были рассчитаны для всех инвариантов, перечисленных выше, для идентичного набора из 33 изображений с 11 изображениями каждого из 3 объектов. Из этих карт мы выбрали 3 наиболее полезных для продолжения тестирования: GPDS1, GPDD1 и GPSOD1, потому что они лучше всего напоминают идеальную карту (рисунок 9).

Следующим шагом в нашем исследовании было создание графиков с помощью функции findmatchbatch. Мы протестировали GPDS1, GPDD1 и GPSOD1 на наборе из 475 фотографий реальных картин, выставленных в музее Помпиду в Париже, Франция. Эти фотографии можно найти в приложении.

Результаты показаны на рисунок 11. Каждая фигура содержит зеленый, синий и красный график. Зеленые графики представляют процент правильных совпадений для каждого порогового значения. Как сказано выше, правильные совпадения - это строки выходных данных findmatch, в которых первое и последнее числа равны. Красные графики - это количество строк, в которых первое и последнее число различаются. Это неправильные совпадения, которые не были обнаружены как неправильные. Это число должно быть как можно ниже. Сумма оставшихся строк представлена синим графиком. Эти результаты - все правильные и неправильные совпадения, которые были отклонены из-за пороговых требований.

На рисунке 12 показан график результатов сопоставления инварианта GPDS1. Первое, на что нужно обратить внимание, это то, что порог в 1 или 100% допускает 90% положительных совпадений (зеленый график). Остальные 10% являются ложными срабатываниями (красный график), поскольку порог 1 никогда не может отклонить результат (синий график). При уменьшении порогового значения сначала количество ошибочных совпадений начинает уменьшаться, а количество положительных совпадений остается практически равным. Это означает, что пониженное пороговое значение влияет только на ложные срабатывания. Однако при превышении лимита в 65% происходит значительное уменьшение количества положительных совпадений.

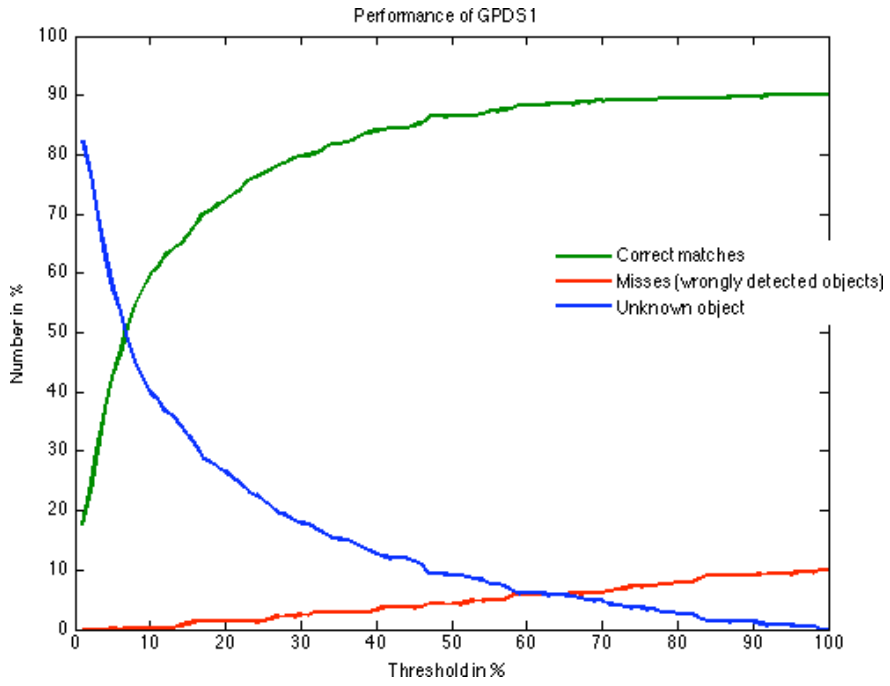

Рисунок 12: Результаты сопоставления GPDS1 в зависимости от порога

Причину этого эффекта можно найти в том, что обнаружено только 5% ложных срабатываний. По статистике сейчас более вероятно, что положительный результат будет отклонен. Поэтому мы решили, что 65% будет хорошим пороговым значением для этого инварианта. Это позволяет распознавать около 88% и только 5% необнаруженных плохих совпадений. Следующий график, который мы создали, показан на рисунке 13 Он визуализирует результаты сопоставления инварианта GPDD1. При пороге 100% уровень распознавания составляет около 88%, что немного ниже, чем на предыдущем графике. При снижении порогового значения количество положительных совпадений остается стабильным до достижения порогового значения 75%. При уменьшении этого значения можно обнаружить тот же эффект, что и на рисунке 12. Таким образом, в этом случае пороговое значение 75% обеспечивает наилучшие результаты. С этим значением может быть достигнут уровень распознавания около 87% и около 7% необнаруженных плохих совпадений.

Третий инвариант, который мы выбрали, был GPSOD1. Результаты findmatchbatch с этим инвариантом показаны на рисунке 14 Этот график ясно показывает, что, хотя этот инвариант допускает более высокую скорость

распознавания, на него также больше влияет пороговое значение. При пороге 100% он имеет около 92% положительных совпадений и только 8% ложных совпадений. При снижении порога в основном положительные совпадения отклоняются. Можно сделать вывод, что этот инвариант менее стабилен. Тем не менее, при пороге 75%, только 4% ложных срабатываний не обнаруживаются при распознавании 90% объектов.

Таблица 1 суммирует результаты, которые мы собрали. На основании этих результатов нам пришлось выбрать один инвариант для нашей реализации Android. Математически инвариант GPSOD1 обеспечивает наилучшие результаты. Но, имея в виду, что нам придется реализовать его на мобильном телефоне, вычислительно он в два-три раза тяжелее, чем два других варианта. Небольшое преимущество немного лучшей скорости распознавания не компенсирует его время вычислений. Поэтому мы решили не использовать инвариант GPSOD1. Из оставшихся двух инвариантов инварианты GPDS1 имеют немного лучшую скорость распознавания.

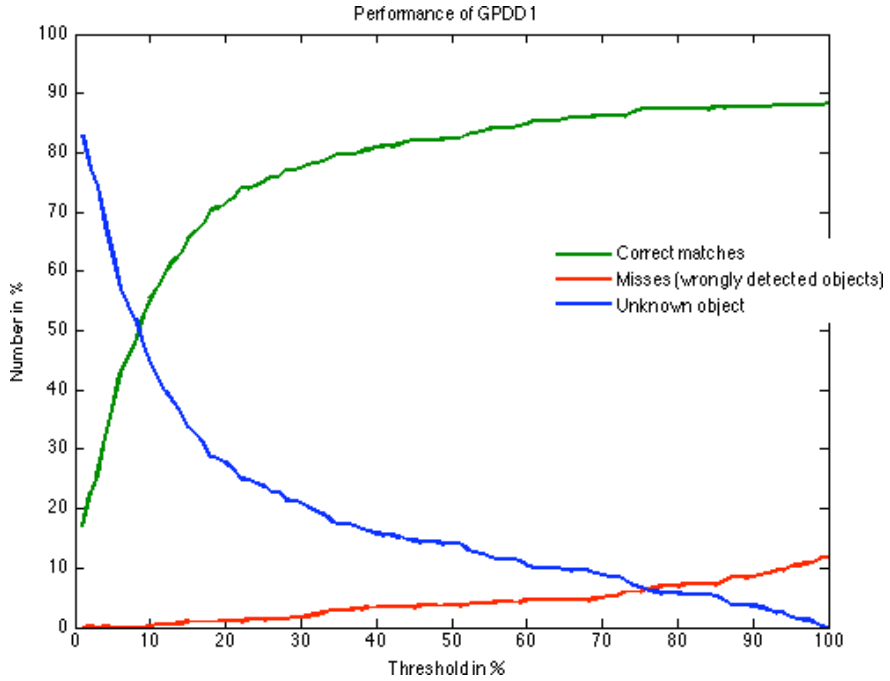

Рисунок 13 Результаты сопоставления GPDD1 в зависимости от порога

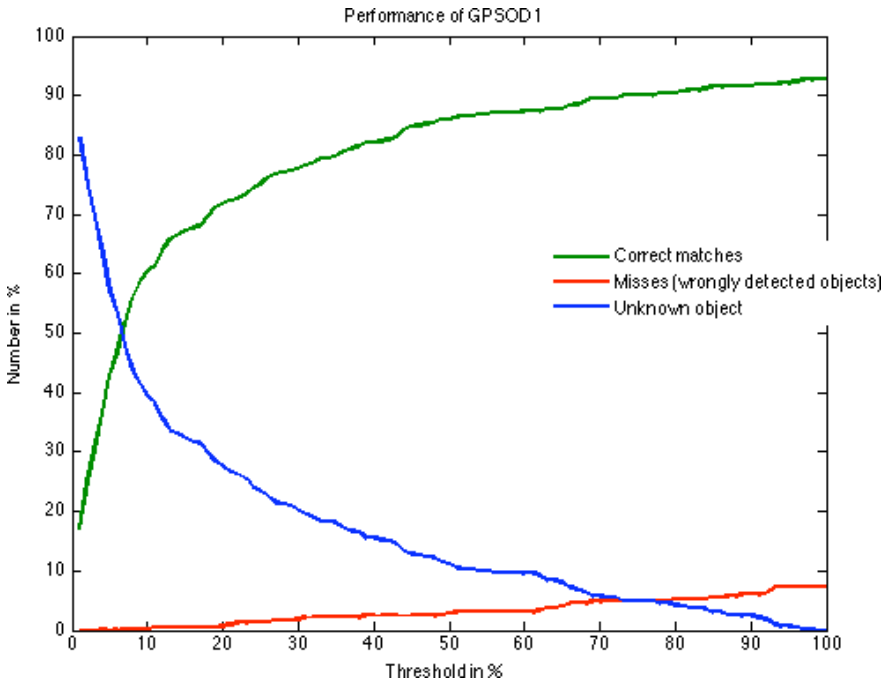

Рисунок 14: Результаты сопоставления GPSOD1 в зависимости от порога

У него нет недостатков по сравнению с инвариантом GPDD1, поэтому мы решили, что это будет лучший инвариант для нашего приложения с пороговым значением 65%.

Invariant	**Threshold**	**Pos. matches**	**Bad matches**
GPDS1	65%	88%	5%
GPDD1	75%	87%	7%
GPSOD1	75%	90%	4%

Таблица 1: Обзор соответствия результатов

Графики дали хороший обзор скорости распознавания каждого инварианта. На их основе мы выбрали инвариант GPDS1 для нашего приложения. Однако графики не содержали никакой информации о том, почему не удалось сопоставить данные, что может быть интересно учесть при разработке приложения для Android. Неудачные совпадения состоят из неверных совпадений и отклоненных результатов совпадений. Изучив один из выходных файлов CSV функции findmatch, мы заметили, что в основном снимки, сделанные с экстремальных углов обзора или с другой ориентацией камеры, обычно плохо совпадают. Мы использовали эти знания, чтобы добавить решение этой проблемы в возможное приложение. Как объяснено в ниже, мы позволили одной картине иметь несколько векторов объектов,

каждый из которых имеет свой угол обзора или ориентацию. Например: один векторный элемент для портретной ориентации и один для альбомной ориентации.

ГЛАВА 3. РАСПОЗНАВАНИЕ ИЗОБРАЖЕНИЙ В ANDROID

3.1 Идея распознавание изображений в android

Общая блок-схема нашего приложения Android Museum Guide (AMuGu) приведена в 15:

Рисунок 15: Поток программ для приложения Android

Короче говоря, когда пользователи запускают приложение, им предоставляется главное меню, которое имеет 2 варианта: либо прочитать XML-файл в базу данных SQLite, как описано ниже, либо запустить экскурсию по музею, показав изображение предварительного просмотра с камеры. Когда пользователь начинает экскурсию, он может направить мобильный телефон к картине и сделать снимок с помощью сенсорного экрана или физической

кнопки на устройстве. Затем начинается расчет, и в это время пользователь видит индикатор выполнения. Когда расчет завершен, в базе данных выполняется поиск соответствующей известной картины. Если не найдено ни одного совпадения или если оно не совпадает достаточно точно, пользователь получает сообщение на своем экране. В противном случае информационное окно отображается с «красивой» фотографией картины. Это резюме картины также содержит название, художника и т.д. Если музей добавил ссылку на информационное окно, появляется дополнительная вкладка «Дополнительная информация». В основном это веб-браузер, предназначенный для отображения дополнительной информационной страницы о картине. Преимущество такого подхода заключается в гибкости, поскольку музей может изменять макет и стиль по своему усмотрению. Они также могут выбрать, хотят ли они сохранить веб-страницу на внутреннем или внешнем веб-сервере (интрасети или в Интернете), или же они хотят сохранить страницу на телефоне Android. Когда пользователь заканчивает чтение информации или закрывает сообщение, если совпадение не найдено, пользователь возвращается к операции предварительного просмотра камеры.

3.2 Базы данных

Чтобы найти соответствие определенной картине картины и предоставить информацию об этой картине, нам нужна была база данных. Несколько вещей должны были быть рассмотрены при работе с базой данных. Первым и самым важным шагом была разработка хорошей структуры базы данных. После создания базы данных нам нужно было найти интуитивно понятный интерфейс, обеспечивающий простой способ загрузки данных в базу данных и их изменения. Когда данные были созданы, нам нужно было только чтобы наше приложение запрашивало их при необходимости. Все эти шаги будут объяснены в этом разделе.

3.2.1 Проектирование базы данных

Наша база данных должна быть в состоянии предоставить все данные, необходимые для работы нашего приложения. Поэтому нам нужно создать достаточные таблицы с их полями данных, и таблицы должны быть связаны таким образом, чтобы они могли легко предоставлять связанные данные. База данных должна содержать библиотеку картин. В основном эта информация состоит из:

- имя
- Исполнитель
- изображение картины
- другая дополнительная информация
- вектор инвариантов фотографии, снятой с картины

Мы могли бы просто бросить все это в один стол для рисования, но мы решили не делать этого. Вместо этого мы создали отдельные векторные, графические и информационные таблицы. Мы создали отдельную векторную таблицу, чтобы иметь возможность предоставить более одного описывающего вектора для определенной картины. Таким образом, мы можем добавить векторы, рассчитанные по фотографиям, сделанным с более экстремальных углов, которые, как мы объяснили выше, не всегда распознаются. Разделять таблицы живописи и информации на самом деле не было необходимости, но мы решили сделать это, чтобы обеспечить большую гибкость в отношении возможных изменений в будущем. Кроме того, мы создали музейный стол, чтобы обеспечить большую гибкость. Например, это необходимо, если мы хотим расширить наше приложение, чтобы оно могло содержать библиотеки более чем одного музея. Теперь каждая таблица должна быть заполнена соответствующими полями данных и дополнительным полем идентификатора, так называемым первичным ключом, который уникально описывает каждую запись таблицы. Это привело к следующим структурам таблицы:

- роспись стола: пид
- векторная таблица: inv1, inv2, inv3, vid

• информационная таблица: имя, исполнитель, img, необязательно, iid

• музейный стол: имя, середина

Обратите внимание, что для необязательных полей и полей img мы решили, что они должны быть путями соответственно для HTML и файла изображения. Размещение необязательной информации в html-файлах позволяет музеям придать этой информации индивидуальность и разметку. Это также позволяет изменять информацию (а также изображение) без необходимости изменять записи в базе данных. В нашем приложении мы будем использовать WebView для визуализации этой информации.

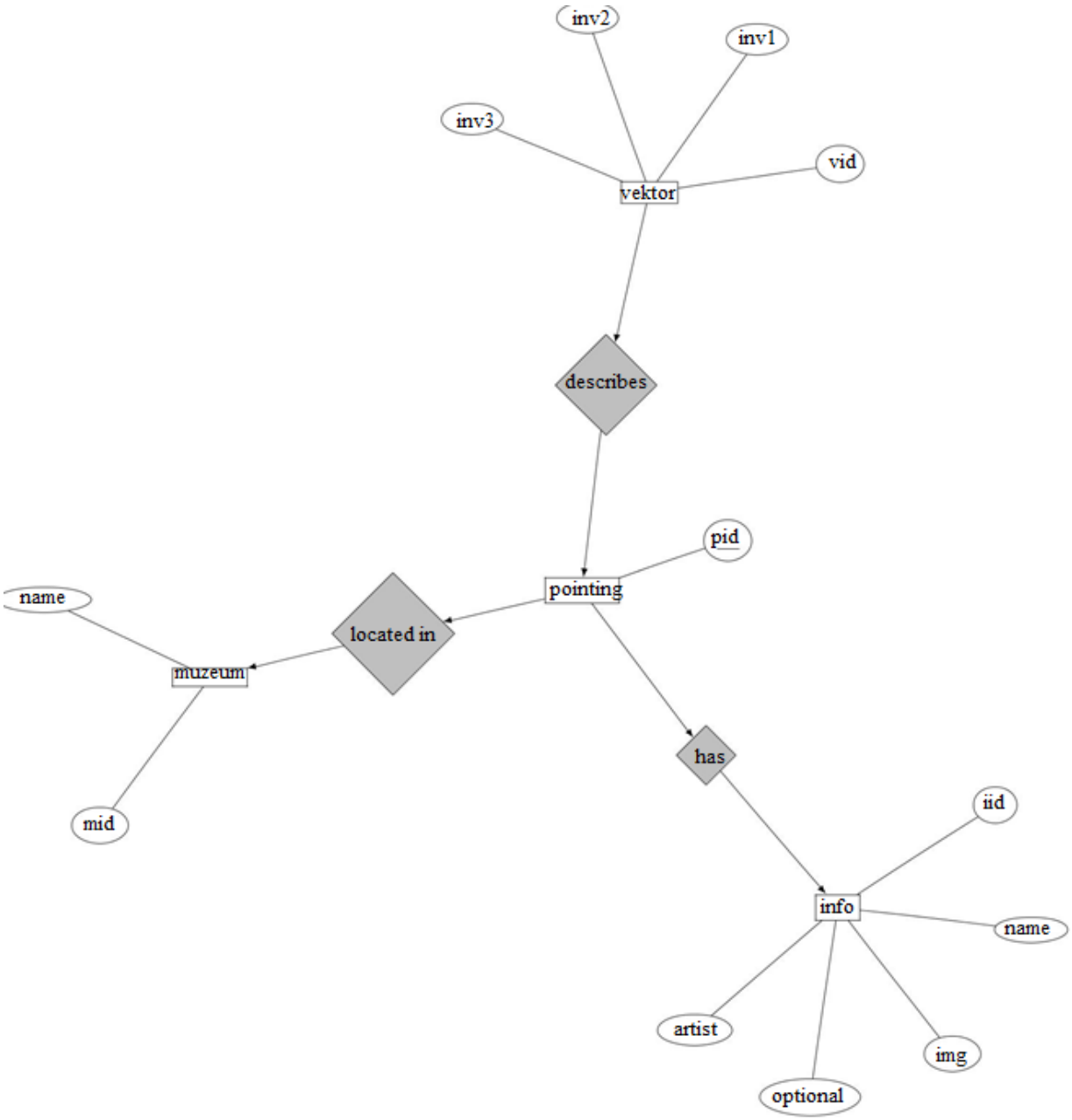

Рисунок 16 Модель данных Entity-Relation базы данных AMuGu

На рисунке 16 показана модель данных Entity-Relation нашей базы данных. Он показывает, какие таблицы присутствуют и какие поля содержит каждая таблица. Но он также описывает, как связаны таблицы: одна или несколько векторных таблиц описывают одну запись картины, запись картины имеет одну соответствующую запись информации, и каждая картина находится в одном музее. Программно эти ссылки достигаются путем добавления одного дополнительного поля ссылки к связанным таблицам (это не показано на рисунке). Это поле соответствует первичному ключу таблицы, с которой оно связано. Таким образом, таблица векторов имеет дополнительное поле pid, а таблица раскраски - дополнительные поля mid и iid.

3.2.2 Создание и использование базы данных в Android

В предыдущем разделе мы разработали модель базы данных для хранения данных о картинах. Прежде чем мы смогли использовать его в нашем приложении, нам сначала пришлось его создать. Для этого мы использовали систему управления базами данных SQLite и файлы XML, содержащие структурированные данные рисования. В этом разделе мы объясним, почему мы это сделали.

Как мы уже говорили выше, Android использует систему управления базами данных SQLite, доступ к которой можно получить через небольшую библиотеку C-программирования. Там, где другие системы управления базами данных являются в основном автономными процессами, SQLite запускается как часть программы, из которой он вызывается. Программа может использовать SQLite, выполняя простые вызовы функций, что уменьшает задержку, поскольку не требуется межпроцессное взаимодействие. Поэтому SQLite идеально подходит для приложений на мобильных телефонах. В нашем приложении нам не нужно было использовать эти вызовы функций C, потому что Google предоставил классы интерфейса базы данных под названием

SQLiteDatabase и SQLiteOpenHelper. В нашем приложении мы создали общий класс интерфейса базы данных под названием DbAdapter. Этот класс содержит объект класса SQLiteDatabase и расширение класса SQLiteOpenHelper, называемое DatabaseHelper. Кроме того, он имеет несколько методов для открытия и закрытия связи с базой данных, создания записей, удаления записей, извлечения данных из базы данных и выполнения необработанных запросов.

Чтобы использовать базу данных, нам нужно сначала открыть ее, чтобы разрешить связь. Поэтому необходимо создать новый экземпляр класса DbAdapter и выполнить метод open ().

```
1 public DbAdapter open() throws SQLException {
2    Log.v("DbAdapter", "Start Opening...");
3    mDbHelper = new DatabaseHelper(mCtx);
4    mDb = mDbHelper.getWritableDatabase();
5    Log.v("DbAdapter","Opened succesful! Version: " +
mDb.getVersion());
6    return this ;
7 }
```

Listing 3.1: open (): Инициализация связи с базой данных

По сути, этот метод создает новый экземпляр mDbHelper класса DatabaseHelper и запрашивает объект SQLiteDatabase mDb. Первый будет использоваться для создания, открытия, обновления и закрытия базы данных. Последний будет использоваться для выполнения запросов. Класс DatabaseHelper расширяет класс SQLiteOpenHelper, поэтому он также содержит его методы. Одним из этих методов является onCreate (), показанный в Listing e 3.2. Этот метод вызывается автоматически при создании объекта этого класса, поэтому мы решили переопределить этот метод для автоматического создания базы данных при необходимости.

```
1 @Override
2 publicvoid  onCreate(SQLiteDatabase db) {
3    db.execSQL(DATABASE_CREATE_MUSEUM);
4    Log.v("DbAdapter","Museum table created.");
```

```
5     db.execSQL(DATABASE_CREATE_VECTOR);
6     Log.v("DbAdapter","Vector table created.");
7     db.execSQL(DATABASE_CREATE_PAINTING);
8     Log.v("DbAdapter","Painting table created.");
9     db.execSQL(DATABASE_CREATE_INFO);
10    Log.v("DbAdapter","Info table created.");
11 }
```

Listing 3.2. Создание новых таблиц базы данных

Как видите, мы создаем каждую таблицу базы данных, вызывая execSQL. Аргументом этих методов являются строки, которые мы также создали в классе DbAdapter. Напр

```
1     privatestaticfinal String DATABASE_CREATE_VECTOR =
2     "create table " + TABLE_VECTOR
3     + " ("
4     + KEY_P_VID + " integer primary key autoincrement,        "
5     + KEY_F_PID + " integer, "
6     + KEY_INV1 + " double, "
7     + KEY_INV2 + " double, "
8     + KEY_INV3 + " double "
9     + ");";имер:
```

Listing 3.3. Создание таблицы векторов

Итак, подведем итог: когда метод open () вызывается для объекта DbAdapter, создается новый экземпляр класса DatabaseHelper, в результате чего вызывается метод onCreate (), который создает базу данных, если ее еще не было. Затем, вызвав метод getWritableDatabase () для этого экземпляра, будет возвращен объект SQLiteDatabase, что сделает базу данных доступной для выполнения запросов.

При разработке нашего приложения мы решили, что нам и возможным пользователям нашего приложения будет легко создавать множество новых записей без необходимости выполнения реальных запросов SQL. Еще один отличный способ объединения структур, состоящих из примитивных данных, - это использование файлов XML. Вот почему мы решили создать нашу базу данных путем анализа файла XML. XML-файлы имеют довольно простую структуру (показанную в Listing e 3.4), что позволяет музеям легко создавать

новые базы данных или редактировать их. На более позднем этапе нашей диссертации мы даже начали разработку небольшого Java-приложения, которое выступает в качестве графического интерфейса для создания и редактирования файла XML.

```
1 <museum name="Pompidou">
2 <painting>
3 <vector>
4 <inv1>1.2157</inv1>
5 <inv2>1.2329</inv2>
6 <inv3>1.2708</inv3>
7 </vector>
8 <info>
9 <name>L'adoration du veau</name>
10 <artist>Francis  Picabia</artist>
11 <img>veau.jpeg</img>
12 <opt>veau.html</opt>
13 </info>
14 </painting>
15 <painting>
16 <vector>
17 <inv1>1.3891</inv1>
18 <inv2>1.7651</inv2>
19 <inv3>1.0372</inv3>
20 </vector>
21 <info>
22 <name>La vache spectrale</name>
23 <artist>Salvador Dali</artist>
24 <img>vachespec.jpeg</img>
25 <opt>vachespec.html</opt>
26 </info>
27 </painting>
28 </museum>
```

Listing 3.4. Пример файла базы данных XML

На рисунке 11 показан программный поток для создания базы данных из файла XML. Мы заметили, что чтение большой базы данных занимало много времени, поэтому мы решили запустить ее в фоновом режиме. Мы запускаем этот сервис в методе onCreate () основного действия нашего приложения. Мы

также предоставили возможность перечитать базу данных на случай, если что-то пойдет не так при запуске приложения.

Класс синтаксического анализатора XML, который мы использовали в нашем приложении, имеет три метода только для анализа XML-файла. Первый метод startElement () вызывается при обнаружении открывающего тега XML. Как видно из Listing a 3.5, этот метод устанавливает некоторые логические значения, чтобы определить, прошел ли он определенный открывающий тег, и извлекает ли атрибуты из этих тегов, если они есть. Подробности этой функции

```
1 / **
2 * Вызывается при обнаружении открывающего тега XML.
3* Может также обрабатывать атрибуты.
4 */
5 @Override
6 publicvoid  startElement(String namespaceURI, String
localName,
7 String qName, Attributes atts) throws SAXException {
8 if (localName.equals("museum")) {
9 long lm = new
File("/sdcard/files/AMuGu/db.xml").lastModified();
10 museum.setMDate(lm);
11 //Атрибут Extractan
12 String attrValue = atts.getValue("name");
13 museum.setMName(attrValue);
14 AMuGu.dbHelper.createEntry(museum);
15 this.in_museum = true;
16 } elseif (localName.equals("painting")) {
17 this.in_painting = true;
18 Log.v("XMLHANDLER",- "Adding new Painting...");
19 previous_p=p;
20 p++;
21 } elseif (localName.equals("vector")) {
22 this.in_vector = true;
23 previous_p=p;
24 } elseif (localName.equals("inv1")) {
25 this.in_inv1 = true;
26 } elseif (localName.equals("inv2")) {
27 this.in_inv2 = true;
```

```
28 } elseif (localName.equals("inv3")) {
29 this.in_inv3 = true;
30 } elseif (localName.equals("info")) {
31 this.in_info = true;
32 } elseif (localName.equals("name")) {
33 this.in_name = true;
34 } elseif (localName.equals("artist")) {
35 this.in_artist = true;
36 } elseif (localName.equals("img")) {
37 this.in_img = true;
38 } elseif (localName.equals("opt")) {
39 this.in_opt = true;
40 }
41 }
```

Listing 3.5: startElement (): определить, передан ли открывающий тег

Второй метод (перечисление 3.6) вызывается автоматически, когда читается содержимое между двумя тегами. Основываясь на вышеупомянутых логических значениях, мы определяем, где этот контент должен храниться. Мы решили временно сохранить этот контент в Java-объектах, представляющих записи таблицы. Это облегчило бы его сохранение в базе данных впоследствии.

```
1 /**
2 Iscalledon<tag>content</tag>structure.
3@paramch-arrayofcharacterscontainingcontentbetweenXMLtags
4@paramstart-startingoffsetinch
5@paramlength-numbersofcharacterstouse
6/
7 @Override
8 publicvoid   characters(char ch[], int start, int length) {
9data = String.copyValueOf(ch, start, length);
10
11 **/
12* SettingupanInvVectorobject.
13*/
14 if (this.in_vector) {
15 vectors.add(new InvVector());
16 this.in_vector=false;
17 }
18 if (this.in_inv1) {
```

```
19 vector=(InvVector) vectors.get(vectors.size()-1);
20 vector.setInv1(Float.parseFloat(data));
21 }
22 if (this.in_inv2) {
23 vector=(InvVector) vectors.get(vectors.size()-1);
24 vector.setInv2(Float.parseFloat(data));
25 }
26 if (this.in_inv3) {
27 vector=(InvVector) vectors.get(vectors.size()-1);
28 vector.setInv3(Float.parseFloat(data));
29 }
30
31 **/
32* SettingupanInfoobject
33*/
34 if (this.in_artist) {
35 info.setArtist(data);
36 }
37 if (this.in_name) {
38 info.setName(data);
39 }
40 if (this.in_img) {
41 info.setImg("/sdcard/files/AMuGu/info/" + data );
42 }
43 if (this.in_opt) {
44 info.setOptInfo("/sdcard/files/AMuGu/info/" + data);
45 }
46 }
```

Listing 3.6. Извлечение данных между тегами XML

При передаче закрывающего тега мы должны снова установить логическое значение в false, и, как показано на рисунке 3.3, при передаче закрывающего тега рисования мы должны сохранить рисунок в базе данных. Эта последняя часть выполняется путем вызова метода commitPainting ().

```
1 /**
2*Iscalledtocreatethedatabaseentrywiththedataextractedfro
mtheXMLfile.
3 /*
```

```
4 privatevoid    commitPainting() {
5 AMuGu.dbHelper.createEntry(info);
6 Cursor c = AMuGu.dbHelper.fetchInfo();
7 c.moveToLast();
8 int iid = c.getInt(c.getColumnIndex(DbAdapter.KEY_P_IID));
9 painting.setIid(iid);
10 c.close();
11 c = AMuGu.dbHelper.fetchMuseum();
12 c.moveToLast();
13 int mid = c.getInt(c.getColumnIndex(DbAdapter.KEY_P_MID));
14 painting.setMid(mid);
15 c.close();
16 AMuGu.dbHelper.createEntry(painting);
17
18 c = AMuGu.dbHelper.fetchPaintings();
19 c.moveToLast();
20 int pid = c.getInt(c.getColumnIndex(DbAdapter.KEY_P_PID));
21 Log.v("XMLHANDLER",info.toString());
22 Log.v("XMLHANDLER",painting.toString());
23 Iterator<InvVector> i = vectors.listIterator();
24 while(i.hasNext()) {
25 InvVector v = i.next();
26 v.setPid(pid);
27 AMuGu.dbHelper.createEntry(v);
28 Log.v("XMLHANDLER",v.toString());
29 }
30 c.close();
31 vectors.clear();
32 Log.v("XMLHANDLER", "Painting added!");
33 }
```

Listing 3.7. CommitPainting (): сохранение рисунка, извлеченного из файла XML, в базе данных

Каждый шаг метода commitPainting () указан в Listing е 3.7. Метод сначала создает запись в информационной таблице в базе данных. Эта запись автоматически получит новое значение iid из системы управления базами

данных SQLite. Вызывая метод fetchInfo (), мы получаем курсор над всеми записями информационной таблицы. Последняя созданная запись может быть достигнута путем вызова метода moveToLast () для объекта курсора. Затем мы можем получить поле iid из этой записи для использования в записи рисования, которая будет создана далее. То же самое сделано для получения середины последней записи музея. Когда новая запись рисования создана, мы выполняем те же шаги для создания векторных записей.

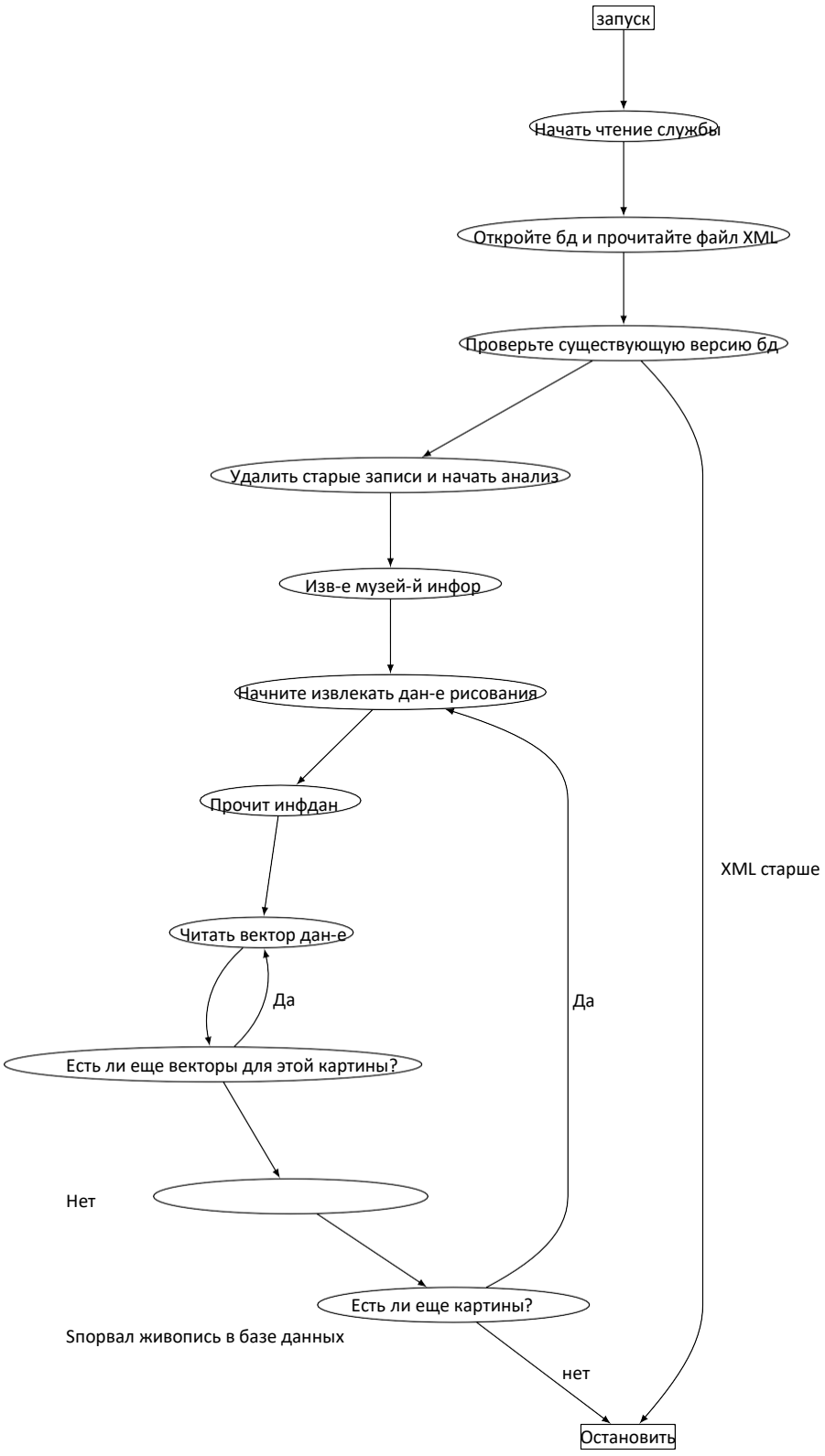

Рисунок 17 Чтение базы данных во время запуска приложения

3.3 Захват изображений

После того, как пользователь выбрал опцию «Начать тур» в главном меню, он запускает тур, который представляет предварительный просмотр изображения с камеры. Мы основали наш код на примере кода Google

CameraPre-view.java. Конечно, нам пришлось расширить код, потому что нам нужен не только предварительный просмотр, мы хотим сделать снимок, нажав кнопку «Сделать снимок». Чтобы сделать это, мы должны были создать OnClickListener, который вызывает процедуру takePicture нашей камеры при нажатии кнопки. В качестве аргументов для вызова takePicture, мы передаем недавно созданный jpegCallback. Android имеет 2 возможных обратных вызовов для изображений: JPEG и RAW. Однако RAW пока не поддерживается, поэтому мы должны использовать JPEG-опцию. Эта функция обратного вызова сначала декодирует JPG-данные в объект Bitmap и передает растровое изображение объекту GPDS1 для расчета. Расчет еще не начался. Создается обработчик, чтобы объект Tour мог получать уведомления о завершении вычислений, а затем пора начинать выполнение нашего кода распознавания изображений с помощью объекта GPDS1.

```
1 publicclass Tour extends Activity {
2
    //CreateourPreviewviewandsetitasthecontentofouractivity
.
3 mPreview = new Preview(this);
4 mLL = new LinearLayout(this);
5 mLL.setOrientation(LinearLayout.VERTICAL);
6
7 bGPDS1 = new Button(this);
8 bGPDS1.setText("Take picture");
9 bGPDS1.setVisibility(View.VISIBLE);
10 bGPDS1.setClickable(true);
11 bGPDS1.setOnClickListener(mGPDS1Listener);
12
13 mLL.addView(bGPDS1);
14 mLL.addView(mPreview);
15
16 setContentView(mLL);
17 }
18
19 @Override
20 protectedvoid   onResume() {
```

```
21 super.onResume();
22 bGPDS1.setClickable(true);
23 }
24
25 OnClickListener mGPDS1Listener = new OnClickListener() {
26 publicvoid onClick(View v) {
27 bGPDS1.setClickable(false);
28 mPreview.mCamera.takePicture(shutterCallback, null,
jpegCallback);
29 }
30 };
31
32 Camera.PictureCallback jpegCallback = new
Camera.PictureCallback() {
33 publicvoid onPictureTaken(byte[] _data, Camera _camera) {
34 Bitmap bmpTmp = BitmapFactory.decodeByteArray(_data, 0,
_data.length);
35 oGPDS1 = new GPDS1(TourActivity,bmpTmp);
36 Handler wait4calc = new Handler() {
37 publicvoid    handleMessage(Message msg) {
38 Result = oGPDS1.getResult();
39AMuGu.getPhoto().setInv(Result[0],Result[1],Result[2]);
40 Intent i = new Intent(TourActivity,ShowInfo.class);
41 i.putExtra("KEY_PID", AMuGu.getPhoto().findMatch());
42 startActivity(i);
43 }
44 };
45 oGPDS1.execute(wait4calc);
46 }
47 };
```

Listing 3.8. Захват изображений

GPDS1 -объект начинается с сохранения обработчика для объекта Tour в переменной класса. Затем он создает ProgressDialog, настраивает и запускает его. Для выполнения расчета создается отдельная нить. Если бы мы не запустили новый поток, было бы невозможно одновременно взаимодействовать с пользователем и проводить расчеты. Это будет означать, что вычисление займет 100% процессора, и пользователь не увидит никакого

прогресса на своем дисплее, устройство будет выглядеть замороженным. Отделяя расчет от пользовательского интерфейса с помощью потоков, пользователь видит индикатор выполнения обновления. Новый поток вычислений фактически является самим объектом GPDS1. Это возможно, потому что мы сделали это реализующим Runnable. Таким образом, когда поток запускается, вызывается функция запуска GPDS1, которая выполняет 2 действия: сначала она сама начинает вычисления (более подробно это описано в разделе 3.4) и вызывает внутренний обработчик, который отклоняет ProgressDialog. Когда вычисления заняты, ProgressDialog обновляется потоком GPDS1. Когда функция convertbatch возвращается, вектор результатов известен, а ProgressDialog отклоняется.

```
1 publicclass  GPDS1 extends Object implements Runnable{
2 *     / TheGPDS1-
classwillgeneratetheinvariantcolormomentofagivenimageby
3 *usingtheGPDS1-algorithm.
4 */
5 private Bitmap bmpImage = null;
6 private Float[] Result = new Float[3];
7 privateint     pdupdatenrs = 25 ;
//updatesofprogressbarperimage
8 private ProgressDialog pd;
9 private Context cntParent;
10 private Handler ParentHandler;
11 private Thread GPDS1thread;
12
13 public GPDS1(Context _cnt, Bitmap _bmpImage) {
14 super();
15 bmpImage = _bmpImage;
16 cntParent = _cnt;
17 }
18
19 public Float[] getResult() {
20 return Result;
21 }
22
23 publicvoid    execute(Handler _refHandler) {
```

```
24 ParentHandler = _refHandler;
25 pd = new ProgressDialog(cntParent);
26 pd.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
27 pd.setTitle("Working...");
28 pd.setMessage("Processing image");
29 pd.setMax(pdupdatenrs);
30 pd.show();
31 GPDS1thread = Thread.currentThread();
32 Thread thread = new Thread(this);
33 thread.start();
34 }
35
36 private Float[] calculatebatch() {
37 ...
38 return Result;
39   }
40
41 publicvoid run() {
42 Result = calculatebatch();
43 handler.sendEmptyMessage(0);
44 }
45
46 private Handler handler = new Handler() {
47 publicvoid  handleMessage(Message msg) {
48 pd.dismiss();
49 ParentHandler.sendEmptyMessage(0);
50 }
51 };
52 }
```

Listing 3.9. Реализация объекта GPDS1, иллюстрирующая использование потоков

Теперь мы знаем вектор изображения, который мы можем сообщить Хендлеру о нашей туристической деятельности. Оттуда начинается сопоставление, как более подробно описано ниже.

Полный код для захвата изображений можно найти в приложении в файлах tour.java и gpds1.java.

3.4 Оптимизация кода

После наших Matlab-экспериментов мы продолжили преобразовывать алгоритмы в Java-код. Чтобы сгенерировать результаты тестов из 475 фотографий в Matlab с разумной скоростью, нам пришлось максимально оптимизировать Matlab-код. В основном это было достигнуто путем попытки избежать циклов for и замены их матричными операциями. Благодаря этому наш код стал более эффективным, но не так легко переносимым на Java. Другими словами, мы начали с оптимизированного кода Matlab и в основном должны были отменить большинство созданных нами оптимизаций. Этот дополнительный шаг не был полностью бесполезным, благодаря этому у нас было лучшее представление о том, что именно происходит в коде.

Оригинальная версия

Наша первая Java-реализация алгоритма, который мы назвали «GPDS1», выглядела примерно так:

```
1 private Double[] calculate() {
2 Double[] dblResult = new Double[]{0.0d, 0.0d, 0.0d};
3 Double dblTmp;
4 int i;
5 for (i=0; i<3; i++) {
6 dblTmp = gcms(0,0,1,i);
7 dblResult[i] = gcms(0,0,2,i) gcms(0,0,0,i)/(dblTmp dblTmp);
8}
9 return dblResult;
10 }
11 private Double gcms(int p, int q, int scale, int rgb) {
12 Double m = new Double(0.0d);
13 int iPixel;
14 double i,j;
15 Integer[] iRGB = new Integer[]{0,0,0};
16 double[] dRGB = newdouble []{0,0,0};
17 int height = bmpImage.getHeight();
18 int width = bmpImage.getWidth();
```

```
19
20 for (i=0; i<height; i++) {
21 for (j=0; j<width; j++) {
22 iPixel = bmpImage.getPixel((int)j,(int)i);
23 iRGB[0] = Color.red(iPixel);
24 iRGB[1] = Color.green(iPixel);
25 iRGB[2] = Color.blue(iPixel);
26 dRGB[0] = iRGB[0].doubleValue()/255;
27 dRGB[1] = iRGB[1].doubleValue()/255;
28 dRGB[2] = iRGB[2].doubleValue()/255;
29 m += ( Math.pow(j,p)
30 Math.pow(i,q)
31 Math.pow(dRGB[rgb],scale)
32 );
33 }
34 }
35 return
36 }
```

Listing 3.10. Оригинальная реализация GPDS1 для Android

Выполнение этого кода в Android-эмуляторе заняло около 1 м 43 с на одну фотографию на компьютере, который мы использовали для тестирования. Поскольку у нас не было доступного Android-оборудования, эти смоделированные результаты синхронизации дают реальную оценку. Тем не менее, мы понимали, что мобильный телефон будет медленнее по сравнению с ПК, потому что у него меньше MIPS, но с другой стороны, Android-телефоны могут ускорять байт-код Java в своих процессорах. Общий подход, который мы использовали, был следующим: если мы сможем ускорить это, сделайте это.

Первая оптимизация

присмотреться к коду в (Listing 3.10), можно заметить, что gcms вызывается 9 раз. Каждый gcms сканирует все изображение. Чтобы оптимизировать это, мы сократили количество полных изображений до 1. Это было сделано путем создания функции gcmsbatch, которая принимает массив опций GCMS в качестве параметра. При этом несколько операций могут

выполняться на пикселе, пока только сканирование изображения один раз. Конечно, функция вычисления должна создать массив команд GCMS. Мы также отменили функцию Math.pow, потому что мы использовали только экспоненты 0, 1 и 2. Экспонента 0 означает, что результат всегда равен 1, экспонента 1 означает, что результат является исходным значением, а экспонента 2 означает значение * значение. Вычисление квадрата значения намного быстрее при использовании простого умножения, чем при использовании функции Math.pow, как описано в Функция calcPow была нашим ответом на эту проблему. Кроме того, тип данных для наших расчетов был выполнен с двойной точностью, что, вероятно, не требуется. Поплавки быстрее вычисляются. После настройки наш код выглядел так:

```
private Float[] calculatebatch() {
2   Float[] floResult = new Float[]{0.0f, 0.0f, 0.0f};
3   Float floTmp;
4   int i,offset;
5
6   GPDS1.GCMScommand[] gcmsc = new GPDS1.GCMScommand[9];
7   //insertcommandsforbatchprocessing
8   for (i=0; i<3; i++) {
9     offset = i 3; //offset
10    gcmsc[offset+0] = new GPDS1.GCMScommand(0,i);
11    gcmsc[offset+1] = new GPDS1.GCMScommand(1,i);
12    gcmsc[offset+2] = new GPDS1.GCMScommand(2,i);
13  }
14
15  //executecommands
16  gcmsc = gcmsbatch(gcmsc);
17
18  //post-processing:combiningindividualgcms-
resultsintoRGB-results
19  for (i=0; i<3; i++) {
20    offset = i 3; //offset
21    floTmp = gcmsc[offset+1].getResult();
22    floResult[i] = gcmsc[offset+2].getResult()        *
gcmsc[offset+0].getResult()/(floTmp floTmp);
```

```
23  }
24 return floResult;
25 }
26
27   private GPDS1.GCMScommand[]gcmsbatch(GPD
S1.GCMScommand[] gcmsc){
28   int iPixel;
29   int i,j,k;
30   Integer[] iRGB=new  Integer[] {0,0,0};
 31 Float[] fRGB = new
Float[]{0.0f,0.0f,0.0f};
32      int height = bmpImage.getHeight();
33      int width = bmpImage.getWidth();
34      int numpixels = height width;
35      int pdupdate = numpixels/pdupdatenrs;
36
37      for (i=0; i<height; i++) {
38        for (j=0; j<width; j++) {
39          iPixel = bmpImage.getPixel(i,j);
40          iRGB[0] = Color.red(iPixel);
41          iRGB[1] = Color.green(iPixel);
42          iRGB[2] = Color.blue(iPixel);
43          fRGB[0] = iRGB[0]/255;
44          fRGB[1] = iRGB[1]/255;
45          fRGB[2] = iRGB[2]/255;
46
47          for (k=0; k<gcmsc.length; k++) {
48gcmsc[k].addToResult(calcPow(fRGB[gcmsc[k].rgb],
gcmsc[k].scale));
49          }
50      }
51      return gcmsc;
52 }
53
54 privateclass GCMScommand extends Object {
55      //commands
```

```
56      int p = 0;
57      int q = 0;
58      int scale  = 0;
59      int rgb    = 0;
60      //result
61      float m    = 0.0f;
62
63      GCMScommand(int _scale, int _rgb) {
64        scale = _scale;
65        rgb = _rgb;
66      }
67
68      GCMScommand(int _p, int _q, int _scale, int _rgb) {
69        p = _p;
70        q = _q;
71        scale = _scale;
72        rgb = _rgb;
73      }
74
75      publicvoid addToResult(float d) {
76        m += d;
77      }
78
79      publicfloat    getResult() {
80        return m;
81      }
82 }
83
84 privatefloat calcPow(float a, int b) {
85      if (b==1) {
86        return a;
87   } elseif  (b==2) {
88    return a a;
89   } else {
90     return 1.0f;
91   }
92 }
```

Listing 3.11. Первая оптимизация: использование gcmsbatch

Выше кода удалось улучшить время с 1m43s до 0m31s. Хотя это означает возможное ускорение с коэффициентом 3,3, мы все еще не были полностью удовлетворены. С одной стороны, мы хотели сделать это быстрее, а с другой стороны добавить дополнительную функциональность: индикатор выполнения, чтобы пользователь знал, что что-то происходит. Android-платформа уже содержит объект Progressbar, так что это не должно быть слишком сложно.

Вторая оптимизация

Чтобы повысить скорость, мы рассмотрели случай, когда calcPow возвращает a * a. Это не невероятно медленно, потому что мы не используем функцию Math.pow, но значение «a» может встречаться на изображении более одного раза. Мы думали, что сможем улучшить, создав своего рода примитивный кеш. Дополнительным преимуществом этого подхода было устранение разногласий. В (Listing 3.11) целочисленные значения для R, G и B были разделены на 255 для каждого пикселя. Теперь целочисленное значение передается кеш-функции и используется как индекс массива хранения. Также «GPDS1.GCMScommand (0, i)» запускался 3 раза для разных значений i, но результат всегда был одинаковым. Поэтому рекомендуется запрашивать только 7 вместо 9 команд GCMS.

```
1 GPDS1.GCMScommand[] gcmsc = new GPDS1.GCMScommand[7];
2 //insertcommandsforbatchprocessing
3 for (i=0; i<3; i++) {
4    ♂ffset = i 2; //offset
5    gcmsc[offset+0] = new GPDS1.GCMScommand(1,i);
6    gcmsc[offset+1] = new GPDS1.GCMScommand(2,i);
7 }
8 //performance:GCMScommand(0,0,0,i)isequalforeveryi,
9 //soexecuteitonlyonceandputitasthelastelementofthearray
10
11 gcmsc[gcmsc.length-1] = new GPDS1.GCMScommand(0,0);
12 //executecommands
```

```
13 gcmsc = gcmsbatch(gcmsc);
14
15 //post-processing:combiningindividualgcms-resultsintoRGB-
results
16 for (i=0; i<3; i++) {
17   offset = i 2; //offset
18   dblTmp = gcmsc[offset].getResult();
19   dblResult[i] = gcmsc[offset+1].getResult()
* gcmsc[gcmsc.length-].getResult()/(dblTmp dblTmp);
20 }
```

Listing 3.12. Вторая оптимизация: 7 команд GCMS вместо 9

Что касается прогресса, мы столкнулись с некоторыми проблемами. Несмотря на то, что объекты панели прогресса были предоставлены в библиотеке Android, их использование было немного сложным. Идея индикатора выполнения состоит в том, что вы видите прогресс во время выполнения длительной операции. Вычисление наших инвариантных моментов пришлось отделить от графического интерфейса пользователя, используя несколько потоков. Мы имеем некоторые трудности с запуском потоков при сохранении обновленного экрана. Сам индикатор прогресса также вводит дополнительное время вычислений. Сначала мы пытались обновлять полосу один раз каждую горизонтальную линию на изображении, но это оказалось довольно неэффективным. Затем мы ввели переменную, чтобы указать общее количество обновлений для изображения. Значение 25 оказалось хорошим компромиссом между скоростью и отзывчивостью.

Благодаря этой оптимизации время обработки было сокращено до 19 секунд.

Третья оптимизация

Кэш, введенный в предыдущей оптимизации, был переписан. Каждый раз, когда значение было необходимо, структура if проверяла, знает ли кэш это значение. Это была пустая трата времени, и справочная таблица (LUT) была лучшим решением. Более того, мы решили заполнить LUT в начале расчета, используя OpenGL | ES-инструкции. В библиотеке Android есть пара функций

для вычисления произведения двух матриц 4x4 с плавающей точкой. Используя это, мы могли бы использовать диагональные матрицы и вычислить 4 значения в одной инструкции. Поскольку значение находилось в диапазоне 0-255, нам пришлось выполнить это 64 раза. Вместо того, чтобы реально кэшировать результаты, мы теперь вычисляем все возможные квадратные значения при инициализации. Мы рискуем рассчитать некоторые значения, которые нам не нужны, но это приемлемый риск. Нам не нужно каждый раз проверять, существует ли уже вычисленное значение или его нужно вычислять.

Вместо выполнения функции getPixel для каждого пикселя в gcmsbatch создается массив целых чисел, который получает все значения для растрового изображения.

```
1 int[] iPixels = new int [width height];
2 bmpImage.getPixels(iPixels, 0, width, 0, 0, width, height);
```

Listing 4.13. Третья оптимизация: getPixels

Некоторые for -loops также могут быть записаны в оптимизированной форме. Использование "for (int item: iPixels) {" более эффективно, чем "for (i = 0; i <высота; i ++) {for (j = 0; j <ширина; j ++) {". Эта конструкция называется «расширенной для цикла» [12]. Код теперь выглядит так:

```
1 for (int item : iPixels) {
2   iRGB[0] = Color.red(item);
3   iRGB[1] = Color.green(item);
4   iRGB[2] = Color.blue(item);
5
6   for (k=0; k<gcmsc.length; k++) {
7   gcmsc[k].addToResult(calcPow(iRGB[gcmsc[k].rgb],gc
    msc[k].scale));

8   }
9 }
```

Listing 3.14. Третья оптимизация: расширенный цикл for

В итоге одно изображение теперь обрабатывается за 8 секунд, более чем в 12 раз быстрее, чем наша первоначальная реализация. Графическое сравнение времени выполнения приведено в таблице 3.1. Полностью оптимизированный код можно найти в приложении.

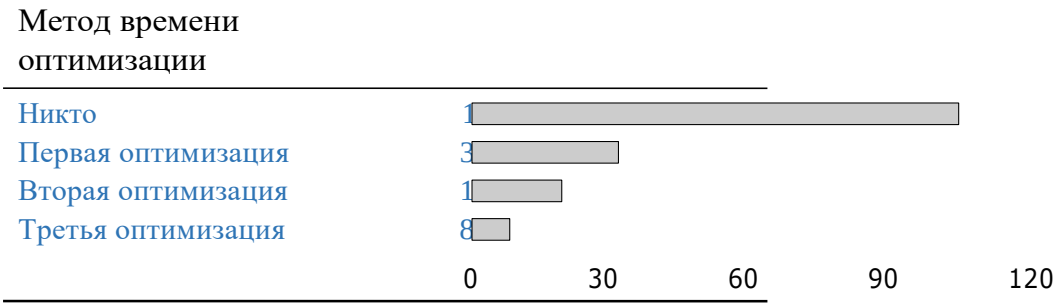

Таблица 3.1: Сравнение времени выполнения

Согласование

В предыдущем разделе мы объяснили, как вычисляются инварианты новой картинки. Следующим шагом является сопоставление этих инвариантов с базой данных изображений. На рисунке 18 показаны различные этапы процесса сопоставления. Как мы уже говорили в разделе 3.3, когда снимок сделан, его результаты сохраняются в объекте MuseumPhoto. На основе этих значений мы пытаемся найти соответствие для этого объекта MuseumPhoto. Это делается методом findMatch (), который выполняет большинство шагов рисунка 18

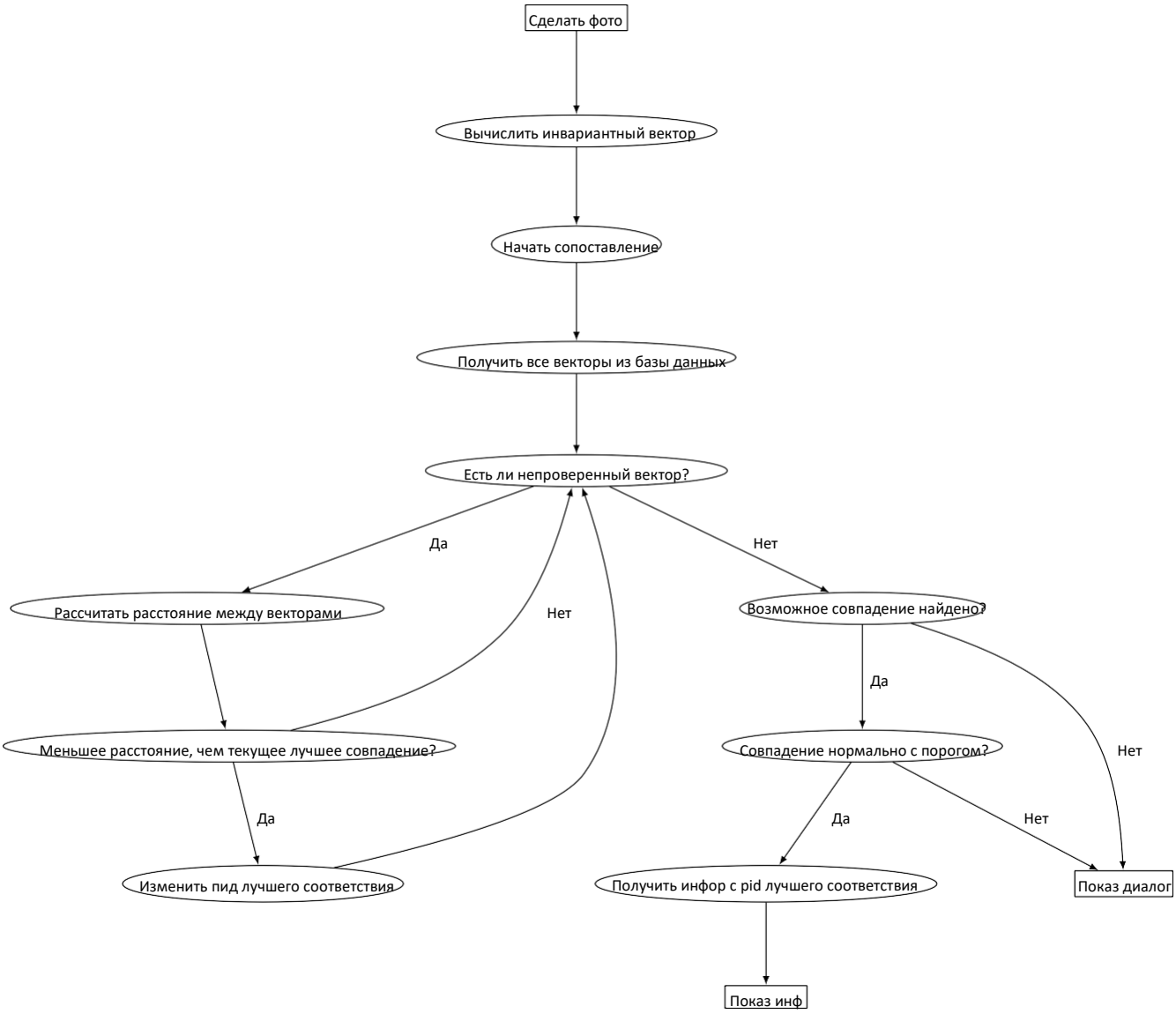

Рисунок 18 Процесс сопоставления

Сначала загружаются векторы в базе данных. Это делается так же, как загрузка записей информационной таблицы в методе commitPainting (), использованном в разделе 3.2. Расстояние каждого вектора до вектора снятого изображения вычисляется. Если рассчитанное расстояние меньше расстояния предыдущего лучшего соответствия, мы нашли лучшее и, следовательно, обновим значение pid для лучшего соответствия. Все это делается с помощью кода в Listing e 3.15.

```
1 do {
2    //calculation
3    m_inv1  = c.getFloat(c.getColumnIndex(DbAdapter.KEY_INV1));
4    m_inv2  = c.getFloat(c.getColumnIndex(DbAdapter.KEY_INV2));
5    m_inv3  = c.getFloat(c.getColumnIndex(DbAdapter.KEY_INV3));
6    m_pid = c.getInt(c.getColumnIndex(DbAdapter.KEY_F_PID));
7
8    dist = (float) Math.sqrt((inv1-m_inv1) (inv1-m_inv1)
9       + (inv2-m_inv2) (inv2-m_inv2) + (inv3-m_inv3) (inv3-
m_inv3));
10   if (dist<smallest_dist) {
11     Log.v("MUSEUMPHOTO", "Best match changed");
12     smallest = m_pid;
13     s_smallest_dist=smallest_dist;
14     smallest_dist=dist;
15   }
16 } while (c.moveToNext());
```

Listing 3.15. Расчет расстояния до всех векторов базы данных

Обратите внимание, что мы также отслеживаем наименьшее и второе наименьшее расстояние, которое мы нашли. Для этого есть веская причина. В разделе 2.2.2 мы объяснили, что если в базе данных есть изображение, мы всегда найдем наименьшее расстояние и, следовательно, лучшее совпадение. Это, однако, не означает, что это правильное соответствие. Поэтому мы решили использовать пороговое значение, чтобы уменьшить количество несоответствий. В разделе 3.5 мы объяснили, что 0,65 является хорошим пороговым значением для инварианта GPDS1. Поэтому, прежде чем вернуть pid с лучшим соответствием, мы сначала проверяем, соответствует ли он пороговым значениям. Это показано в Listing 3.16.

```
1 if (smallest==0) { //Novectorswereloadedfromthedatabase
2    Log.v("MUSEUMPHOTO","No Match Found");
3    return -1;
4  } elseif     (smallest_dist/s_smallest_dist
< THR) { //ifbestmatchisok(qualifies
withthethreshold)
```

```
5    Log.v("MUSEUMPHOTO","Match Found");
6    return smallest;
7 } else {
8    Log.v("MUSEUMPHOTO","Bad Match");
9    return 0;
10 }
```

Listing 3.16. Проверка пороговых требований

Для отображения информации, относящейся к соответствующей картине, мы создали новый Java-класс ShowInfo, который расширяет класс Android Activity. Мы запускаем это действие, когда процесс сопоставления завершен. В ShowInfo мы загружаем связанную информацию из базы данных в объект Info, используя его метод setInfoFromPID ().

```
1 publicvoid  setInfoFromPID(int _pid) {
2Cursor c;
3AMuGu.dbHelper.open();
4c = AMuGu.dbHelper.executeSql("select from info " +
5"inner join painting on info._iid=painting.iid " +
6"where painting._pid="+_pid+";");
7if (!c.moveToFirst()) return;
8iid= c.getInt(c.getColumnIndex(DbAdapter.KEY_P_IID));
 //retrieveindexoftheiid
9name =
c.getString(c.getColumnIndex(DbAdapter.KEY_NAME));
//retrieveindexofthe name-column
10 artist =
c.getString(c.getColumnIndex(DbAdapter.KEY_ARTIST));
//retrieveindexof theartist-column
11 img                                                =
c.getString(c.getColumnIndex(DbAdapter.KEY_IMG));
 //retrieveindexofthe img-column
12 optinfo =
c.getString(c.getColumnIndex(DbAdapter.KEY_OPT));
 //retrieveindexofthe opt-column
13 AMuGu.dbHelper.close();
14 }
```

Listing 4.17. SetInfoFromPID (): инициализировать объект Info для отображения в действии ShowInfo

Как вы можете видеть в Listing e 3.17, мы не используем один из предопределенных запросов SQLite. Мы присоединяемся к информационному столу и столу для рисования. Поскольку релевантны только строки, в которых iid информационной таблицы равен iid таблицы рисования, мы объединяем их на основе этого равенства. Из полученных строк мы выбираем ту, в которой pid совпадает с той, которую мы нашли в процессе сопоставления. Теперь мы заполнили все поля данных объекта Info, мы можем использовать его для создания нашего GUI. Мы опишем, как это делается в разделе 3.6.

ЗАКЛЮЧЕНИЕ

Наша выпускная квалификационная работа возникла с «Распознавания изображений на мобильном телефоне Android». Цель состояла в том, чтобы создать приложение-путеводитель по музею для Android, операционной системы мобильного телефона с открытым исходным кодом.

В этой работе представлена и объяснена операционная система Android. Были прочитаны статьи о распознавании изображений, и мы решили использовать метод, основанный на глобальных особенностях, называемых инвариантами цветовых моментов. Выбор этого метода состоял главным образом в том, чтобы уменьшить вычислительную сложность, поскольку мобильный телефон имеет строго ограниченную вычислительную мощность.

Следующим шагом было наше исследование с использованием Matlab (глава 2). Мы реализовали некоторые из методов, описанных в нашем исследовании литературы, и продолжили сравнивать результаты для различных инвариантов и обнаружили, что один из простейших инвариантов, который мы назвали GPDS1, подходит для нашего приложения. Это был не лучший инвариант в тесте, но он был довольно близок к лучшему, без необходимости сложных вычислений. Сопоставление векторов инвариантов было описано ниже, где также был рассчитан порог.

Узнав, как сделать распознавание, мы сделали реализацию в Android (глава 3), где сделали наше AMuGu-приложение. Мы начали с описания того, что должно делать приложение, и перешли к разработке базы данных. Алгоритм был переведен в код Java, и мы оптимизировали нашу реализацию, чтобы сократить время вычислений. Первоначальная реализация в Android требовала 103 секунд для вычисления инвариантов для одного изображения, после оптимизации она была уменьшена до 8 секунд. Соответствие было также переписано для Android, и наш графический интерфейс пользователя был объяснен. Во время нашей реализации мы обнаружили, что Android - гибкая и удобная для разработчиков платформа. У нас не было доступного

аппаратного обеспечения Android, однако мы создали приложение, которое работает в эмуляторе Android и возвращает те же результаты, что и наш Matlab-код.

В целом, если оглянуться на главу 1 о цели выпускной квалификационной работы мы можем заключить, что достигли наших первоначальных целей. Мы провели исследование о распознавании изображений и возможностях Android, нашли легкий алгоритм и реализовали его как приложение для Android, которое действует как музейный гид.

СПИСОК ЛИТЕРАТУРЫ