

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

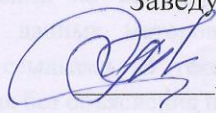
ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК
Кафедра программного обеспечения

РЕКОМЕНДОВАНО К ЗАЩИТЕ
В ГЭК И ПРОВЕРЕНО НА ОБЪЕМ
ЗАИМСТВОВАНИЯ

Заведующий кафедрой

к.т.н., доцент

М. С. Воробьева


24.06.2019 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(магистерская диссертация)

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОИСКА РЕСУРСА ПО
ХАРАКТЕРНОМУ ИЗОБРАЖЕНИЮ НА ПРИМЕРЕ САЙТА БИБЛИОТЕКИ
КОМИКСОВ

02.04.03. Математическое обеспечение и администрирование информационных систем

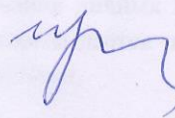
Магистерская программа «Разработка, администрирование и защита вычислительных систем»

Выполнила работу
Студентка 2 курса
очной формы обучения



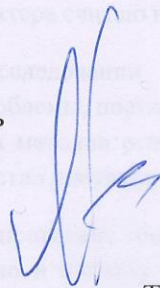
Антипова
Екатерина
Андреевна

Научный руководитель
Д.п.н., профессор



Захарова
Ирина
Гелиевна

Рецензент
Старший преподаватель
кафедры программного
обеспечения



Муравьев
Игорь
Александрович

Тюмень, 2019

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	4
ГЛАВА 1. ОПИСАНИЕ МЕТОДОВ ПОИСКА ИЗОБРАЖЕНИЙ	6
1.1 Общие вопросы поиска изображений	6
1.2 Критерии схожести изображений	10
1.3. Описание архитектуры CBIR-системы и модели поиска	13
1.4 Алгоритмы построения векторов признаков изображений	15
1.4.1 Алгоритм ORB	20
1.4.2 Алгоритм KAZE	22
1.4.3 Алгоритм AKAZE	25
1.5 Поиск и сопоставление изображений	26
1.5.1 Алгоритм кластеризации k-средних (K-Means).....	27
1.5.2 Алгоритм «Мешок визуальных слов»	28
1.5.3 Частотность терминов – обратная частотность документов.....	31
1.6 Технологии и программные средства для работы с изображениями	32
ГЛАВА 2. ТЕСТИРОВАНИЕ МЕТОДОВ ПОИСКА ИЗОБРАЖЕНИЙ ПО ОБРАЗЦУ	34
2.1 Описание используемых данных.....	34
2.2 Тестирование алгоритмов поиска особых точек изображений.....	36
2.2.1 Алгоритм ORB	37
2.2.2 Алгоритм AKAZE с дескриптором KAZE	45
2.3 Тестирование алгоритма «Мешок визуальных слов»	53
ГЛАВА 3. РЕАЛИЗАЦИЯ СИСТЕМЫ ПОИСКА ОБЛОЖКИ КОМИКСА ПО ФОТОГРАФИИ	62
3.1 Описание требуемого функционала разрабатываемой системы	62
3.2 Проектирование архитектуры разрабатываемой системы поиска.....	63
3.3 Мобильный клиент	66
3.5 Тестирование работы приложения.....	70
ЗАКЛЮЧЕНИЕ	75
СПИСОК ЛИТЕРАТУРЫ.....	77

ПРИЛОЖЕНИЕ 1	82
ПРИЛОЖЕНИЕ 2	86
ПРИЛОЖЕНИЕ 3	90

ВВЕДЕНИЕ

Когда требуется найти информацию по какому-либо ресурсу в интернете, пользователь может не знать, как этот ресурс описать, или этого описания может быть недостаточно. Но у него может быть изображение этого ресурса или похожего объекта. В таких случаях часто обращаются за помощью к сервисам для поиска изображений по загружаемому образцу.

Среди популярных поисковых систем, допускающих поиск по изображению, можно выделить Google, Яндекс и TinEye. Результаты поиска по изображению Google и Яндекс бывают очень разнообразными – от точно такого же разыскиваемого изображения, до изображений с похожей цветовой гаммой или формой. К результатам поиска может быть добавлена тематическая информация по найденному ресурсу. Поисковая система TinEye выдает список точно таких же изображений, с небольшим отличием в размере, масштабе, цветовой гамме и некоторыми деталями на изображении, а в качестве дополнительной информации предоставляет ссылки на ресурсы, где было найдено это изображение. Существуют также и узконаправленные сервисы, рассчитанные на поиск тематических ресурсов по фотографиям. Большой популярностью пользуются приложения для поиска предмета гардероба по загруженной фотографии.

Учитывая успех таких сервисов, я решила реализовать подобный уже в контексте комикс-индустрии – поиск комикса по фотографии его обложки. Данный сервис должен находить по загружаемой обложке комикс-ресурс или ресурсы, и выводить информацию о каждом из них. Изучив различные сервисы для поиска информации по комиксам, мне не удалось найти тот, который имел бы подобный функционал. Реализации нет и в формате мобильных приложений. Огромной базой данных обладает сайт Comics Vine - хорошо организованный и постоянно обновляемый ресурс информации по комиксам, данные которого поступают ежедневно от фанатов комиксов по всему миру. У сайта есть свое API, предоставляющее полный доступ к данным структурированного вики-контента.

Если рассматривать обложку комикса, то обычно обязательными атрибутами на ней являются издательство и название, но бывает и так, что этой информации недостаточно для поиска. Возможны ситуации, когда на оригинальном изображении этой информации попросту нет, когда у пользователя имеется вырезанный фрагмент обложки, или есть фотография обложки комикса, изданного в другой стране, которая может быть оформлена несколько иначе, чем ее оригинал.

Поэтому актуальной задачей остается разработка такого сервиса, который позволял бы решать перечисленные выше проблемы: одновременно искать комикс по обложке и выводить информацию, относящуюся к данному ресурсу, при этом поиск должен быть устойчив к шуму, масштабу, повороту, искажениям и возможным дополнительным артефактам, появляющимся на изображении.

Целью данной работы является разработка мобильного приложения для поиска комикса на сайте библиотеки комиксов Comics Vine по фотографии обложки. Для достижения поставленной цели были выделены следующие **задачи**:

- изучить API для загрузки данных;
- определить критерии схожести изображений;
- исследовать методы для поиска изображений по образцу;
- разработать алгоритм для поиска и сравнения изображений;
- разработать мобильное приложение, использующее алгоритм поиска комикс-ресурса по изображению.

ГЛАВА 1. ОПИСАНИЕ МЕТОДОВ ПОИСКА ИЗОБРАЖЕНИЙ

1.1 Общие вопросы поиска изображений

Изучением различных аспектов разработки методов поиска изображений занимаются достаточно давно, и многие исследователи, такие как Long F., Zhang H., Feng D, Smith J.R., Chang S.F., Haralick R.M., Shanmugam K., Dinstein I, А. П. Кирпичников, С. А. Ляшева, М. П. Шлеймович, Н. С. Васильева, Дольник А., Марков И., Новиков Б., Яремчук А, Н. Н. Венцов, В.В. Долгов, Л.А. Подколзина, В. К., Трубаков А.О., K.Velmurugan, Lt.Dr.S. Santhosh Baboo и многие другие. Поиск изображений нашел свое применение во многих сферах жизни. В их число входят электронная коммерция, медицинская диагностика, системы дистанционного зондирования Земли, обеспечение безопасности, каталогизация изображений произведений искусства, поиск изображений в Интернете и др. Данная область является актуальной областью теоретических и практических разработок и в настоящее время.

Например, в работе [10] Горев А. Ю., Шлеймович М. П., Юдинцева А. описывают классическую архитектуру системы поиска изображений по содержанию (анг. CBIR – Content-based image retrieval, CBIR-система, системы контекстного поиска изображений). Затрагиваются возможные проблемы при ее реализации. Авторы отмечают, что CBIR-системы имеют эффективное сочетание точности выдаваемого результата с временем поиска по коллекции изображений.

Термин «content-based image retrieval» был предложен Като в 1992 г. для описания проведенного им эксперимента в области автоматического поиска изображений в базе данных на основе особенностей цвета и формы [8]. С появлением первой системы QBIC (Query by Image Content) [9] был сформирован протокол работы визуального поиска в начале 90-ых годов.

Таким образом, системы поиска изображений по содержанию — это системы поиска изображений, в которых требуется найти максимально

похожие на запрос изображения в заданной коллекции изображений, в соответствии с выбранным критерием сходства [1]. В качестве поискового запроса в таких системах используется изображение или набор его характеристик.

Васильева Н. С. в работе [2] отмечает, что «создание системы поиска изображений по содержанию подразумевает решение целого ряда непростых задач, и в их число входит: анализ низкоуровневых характеристик изображений и построение векторов признаков, многомерное индексирование, проектирование пользовательского интерфейса системы и визуализация данных. Качество системы поиска зависит, в первую очередь, от используемых векторов признаков, описывающих содержание изображения».

Другим вариантом реализации системы поиска изображений является DBIR (Description Based Image Retrieval) - поиск изображений, по ключевым словам, и текстовым аннотациям [12]. К достоинствам данного алгоритма относят его применимость при поиске текстов, соответствующих семантике изображения. К недостаткам алгоритма относят сложность исполнения и субъективный характер составления аннотаций поиска [7].

Кирпичников А. П., Ляшева С. А., Шлеймович М. П. в статье [1] рассматривают методы вычисления характеристик изображений, которые применяются для эффективной организации контекстного поиска изображений. Для определения меры сходства используются характеристики изображений, которые авторы условно разделяют на четыре основные группы: характеристики цвета, характеристики текстуры, характеристики формы и характеристики объектов.

Работа [2] также посвящена обширному обзору основных алгоритмов построения векторов признаков и метрик для соответствующих им пространств. Рассматриваются признаки для таких характеристик изображения, как цвет, текстура и форма объектов. В работе приводятся результаты экспериментальных сравнений эффективности различных методов

представления и сравнения содержания изображений применительно к задачам поиска и классификации.

В статье [3] Васильева Н. С., Новиков Б. А. отмечают, что качество систем поиска изображений по содержанию, использующих низкоуровневые характеристики изображений, сложно признать удовлетворительным. Многие исследователи видят проблему в «семантическом разрыве» между низкоуровневым содержанием изображения, которым оперирует система, и семантикой изображения, необходимой пользователю.

Злотников Т. в работе [4] отмечает недостаток подхода в представлении изображений в виде набора низкоуровневых признаков: потеря большого количества нужной информации в случае, если объект изображен на одинаковых сценах, но в разных ракурсах. Автор рассказывает об использовании другого подхода в CBIR-системах - использование алгоритмов поиска ключевых точек изображений и дескрипторов для их описания на примере метода SURF, в сочетании с алгоритмом поиска ближайших соседей (Randomized KD-Tree).

Гончаренко М. О. в работе [6] представляет результаты сравнения существующих методов обнаружения ключевых точек изображения с целью детектирования изменений сцены видеоданных. Исходные видеоданные представляются в виде последовательности кадров. Для каждого изображения выполняется поиск ключевых точек и формируется дескриптор. Для соседних видеок кадров выявляются общие ключевые точки путем сравнения дескрипторов. В статье рассматриваются методы поиска ключевых точек: FAST, CenSurE, SIFT, SURF, ORB, угловой детектор Харриса, BRISK. Для построения дескрипторов на базе полученного множества точек использовались алгоритмы: SIFT, SURF, ORB, BRIEF, BRISK, FREAK. Результирующие дескрипторы сравнивались между собой следующими алгоритмами: метод ближайшего соседа, метод полного перебора.

Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary Bradski с статье [14] представляют алгоритм ORB (Oriented FAST and Rotated BRIEF) для

обнаружения и описания ключевых точек изображения. Алгоритм основан на улучшенной версии детектора ключевых особенностей FAST (Features from Accelerated Segment Test) [16], вычисляющий ориентацию каждой точки, и бинарном дескрипторе BRIEF (Binary Robust Independent Elementary Features), для которого авторы в своей работе пытаются улучшить производительность при повороте изображения. Также были проверены свойства ORB относительно алгоритмов SIFT и SURF, как на способность сопоставления изображений, так и на производительность.

Pablo F. Alcantarilla, Adrien Bartoli, Andrew J. Davison в работе [28] в 2012г. представляют алгоритм KAZE для обнаружения и описания ключевых точек изображения. Метод основан на нелинейных масштабных пространствах с использованием эффективных методов схем аддитивного разделения операторов (Additive Operator Splitting, AOS), которые стабильны при любом размере шага и распараллеливаются, и диффузии с переменной проводимостью. Несмотря на умеренное увеличение вычислительных затрат, получены хорошие результаты в производительности как в обнаружении, так и в описании по сравнению с предыдущими современными методами, такими как SURF, SIFT или CenSurE.

Pablo F. Alcantarilla, Jesús Nuevo, Adrien Bartoli в работе [15] в 2013 г. представляют алгоритм AKAZE (Accelerated-KAZE, A-KAZE). Алгоритм использует математическую структуру Fast Explicit Diffusion (FED) для поиска ключевых точек изображения, встроенную в пирамидальную структуру, для значительного ускорения нелинейных пространственных вычислений в масштабе. В качестве дескриптора используется модифицированный локально-разностный двоичный дескриптор (Modified-Local Difference Binary, M-LDB), который использует информацию градиента из пространства нелинейного масштаба. AKAZE получает сопоставимые результаты с KAZE в некоторых наборах данных, будучи на несколько порядков быстрее.

Венцов Н. Н., Долгов В. В., Подколзина Л. А. в работе [7] поднимают проблему большого количества операций чтения с диска при сравнении

искомого изображения с хранящимися изображениями в базе данных, и возникающего в последствии снижения производительности системы. В статье рассматриваются основные алгоритмы кластеризации, реализующие задачу поиска изображений по содержанию. Рассматриваемые алгоритмы осуществляют поиск изображений на основе анализа присущих им характеристик.

1.2 Критерии схожести изображений

Основной проблемой поиска по визуальным данным большинство исследователей признают так называемый «семантический разрыв»: человек, сравнивая два изображения, в первую очередь сравнивает их смысловое наполнение – семантику, в то время как оценка системы поиска по визуальным данным основывается на сравнении векторов признаков, соответствующих визуальным характеристикам изображения. Т. е. извлекая информацию из одних визуальных данных, можно получить ее разную интерпретацию со стороны пользователей и поисковых систем. Связано это с тем, что люди сравнивают изображения по разным критериям схожести, которые формируются от цели поиска изображений по заданному образцу.

Первый критерий схожести изображений - наличие или отсутствие одного и того же объекта. При этом изображения могут отличаться между собой цветом фона, наличием шумов, общей цветовой гаммой, масштабом, а также качеством освещенности и перспективным искажением объекта (рис. 1). Разница между такими изображениями может быть в повороте, объект может быть изображен под другим углом обзора, иметь разное освещение, яркость, контрастность и другой фон. Однако, интересующий объект будет присутствовать на всех изображениях, и по этой причине изображения будут считаться похожими.

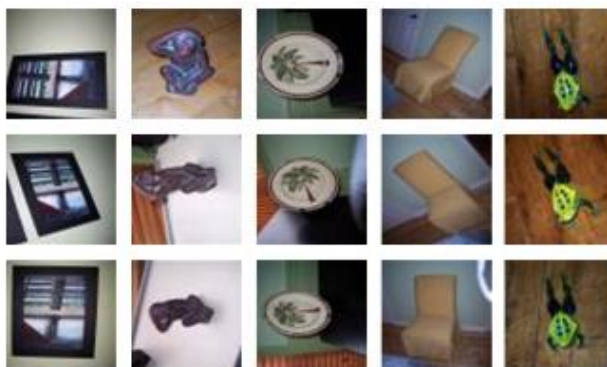


Рисунок 1. Пример изображений с одним и тем же объектом.

Вторым критерием можно считать похожие конфигурации сцен изображений, но разные по назначению (рис. 2). И как видно из примера, объекты на таких изображениях имеют схожее положение на сцене, но при этом различаются между собой полностью или в некоторой степени.



Рисунок 2. Похожие по конфигурации сцены, но разные по назначению.

Третий критерий схожести изображений — это изображения из одного класса сцен (рис.3). В примере на изображениях представлены банкетные залы, и данные изображения отнесены к одному классу по их общим атрибутам сцены, характерным для данного класса – столы, стулья, столовые приборы, меню, окна.

В качестве четвертого критерия схожести, можно выделить изображения из одного класса объектов. Классы подобных объектов определяет сам пользователь, и такое сравнение будет полезно для идентификации объектов на изображении. Пример – распознавание кошек и собак.



Рисунок 3. Изображения из одного класса сцен.

Пятый критерий - схожая цветовая гамма (рис. 4). Изображения могут иметь совершенно разные объекты, конфигурацию сцены, но будут похожи по соотношению присутствующей палитры цветов.



Рисунок 4. Изображения схожей цветовой гаммы.

Таким образом, в зависимости от выбранной цели поиска изображений по образцу, следует выбирать соответствующий критерий схожести. А уже в зависимости от выбранного критерия схожести изображений следует подбирать соответствующие алгоритмы для выделения ключевых характеристик изображений, которые будут являться основной мерой сходства изображений.

1.3. Описание архитектуры CBIR-системы и модели поиска

Поиск изображений по содержанию или контекстный поиск (CBIR, Content-Based Image Retrieval, в англоязычных источниках также известен как query by image content (QBIC) или content-based visual information retrieval (CBVIR)) является разделом дисциплины компьютерного зрения и решает задачу поиска изображений на основе анализа присущих ему характеристик. Основное направление компьютерного зрения (англ. computer vision) – это анализ и обработка изображений (в том числе и видеопотока). Алгоритмы компьютерного зрения позволяют выделять ключевые особенности на изображении (углы, границы области), производить поиск фигур и объектов в реальном времени, выполнять 3D реконструкцию по нескольким фотографиям и многое другое [11].

Поиск изображений по содержанию предполагает отсутствие какой-либо дополнительной информации об изображениях, вроде текстовых аннотаций. Для поиска изображения анализируется его содержание, вроде численных характеристик пикселей изображения [2]. Каждое изображение коллекции описывается векторами признаков (feature vector) - наборами числовых параметров, отражающих свойства низкоуровневых характеристик изображения (цвет, текстура, форма [13]). Поэтому поиск в такой системе означает поиск по векторам признаков изображений. Чем больше размерность вектора, тем подробнее он описывает характеристику изображения, но тем больше времени требуется на сравнение векторов между собой. Вычисление векторов признаков может быть очень ресурсоёмким и требовательным к памяти для их хранения.

Классическая архитектура CBIR - системы (рис.5) похожа на архитектуру классических поисковых систем. Для CBIR-систем также характерно выделение двух модулей: модуля индексирования и модуля поиска.

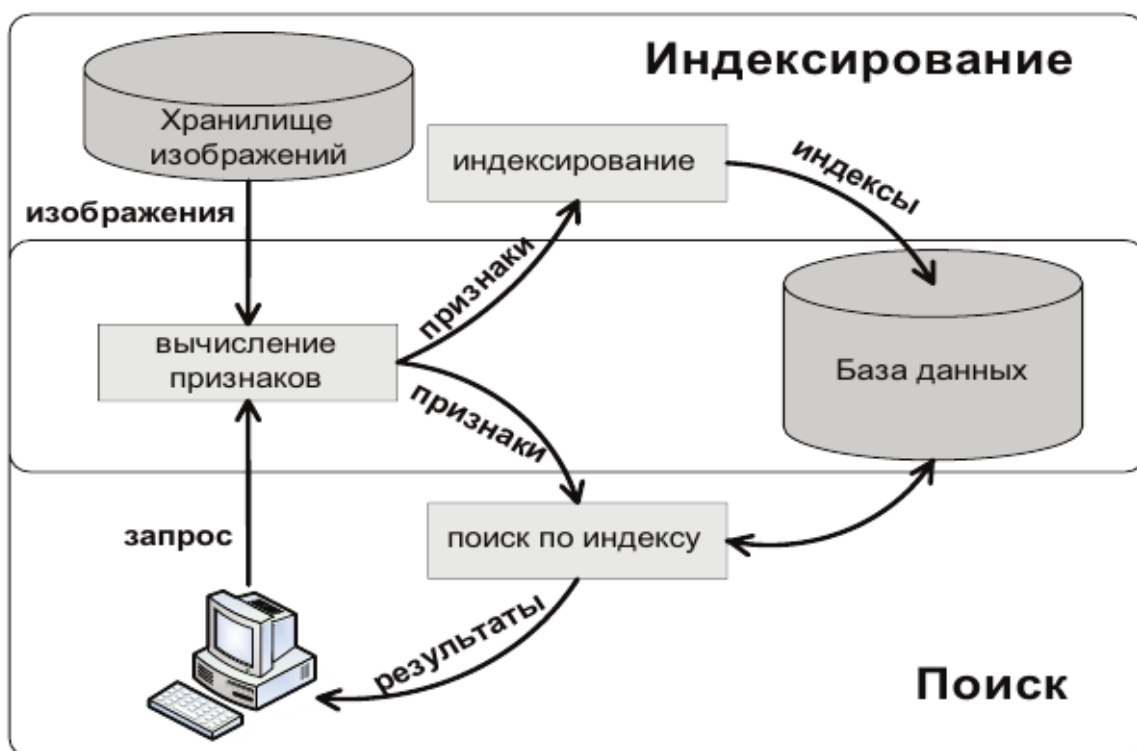


Рисунок 5. Классическая архитектура CBIR-системы [2].

Алгоритмы вычисления векторов признаков изображений являются одними из ключевых элементов CBIR-системы. На вход алгоритму подается изображение, где далее вычисляются его вектора признаков. Полученные векторы передаются в модуль индексирования, где по их значениям строится индекс. Для изображения-запроса используется тот же принцип. Полученные вектора признаков изображения-запроса используются для поиска по векторам изображений в коллекции [2].

Вектора признаков принимают значения в пространстве признаков. Задав метрику на таком пространстве, можно сравнивать изображения друг с другом, вычисляя расстояние между соответствующими им векторами. От выбора признаков, алгоритмов для выявления этих признаков и метрик для их сравнения зависит качество поиска [7].

Задачу построения CBIR-системы можно сформулировать следующим образом. Требуется создать систему, позволяющую индексировать некое множество изображений, чтобы для заданного пользователем изображения-образца запрос к системе возвращал подмножество наиболее близких по

содержанию изображений, т. е. содержащих те же объекты, что и образец, но отличающихся:

- масштабом;
- поворотом изображения;
- расположением на сцене;
- наличием шумов, заслоняющих объектов;
- значениями яркости и контраста.

Математическая модель поиска изображений по содержанию описывается следующим образом:

$$j = \arg \min\{\rho(x, i) \mid x \in I\}, \quad (1)$$

где I – множество изображений в базе данных, i – изображение-запрос, j – изображение из коллекции, наиболее похожее в смысле заданной метрики на изображение-запросе, ρ – функция расстояния между изображениями, построенная на основе заданной метрики. В реальных системах часто требуется найти не одно, а множество изображений, похожих на изображение запрос:

$$J = \{j \mid \rho(j, i) \leq \Delta\}, \quad (2)$$

где Δ – величина, определяющая допустимое отличие результата поиска от заданного изображения [10].

1.4 Алгоритмы построения векторов признаков изображений

Для определения меры сходства изображений в CBIR-системах используются характеристики изображений, которые условно разделяют на четыре основные группы: характеристики цвета, характеристики текстуры, характеристики формы и характеристики объектов [13]. Многие исследователи выделяют несколько уровней содержания изображений. Цвет и яркость относят к низкоуровневому содержанию, физические объекты (такие как человек, машина, дерево) к содержанию высокого уровня, текстуру часто

называют содержанием среднего уровня. В литературе представлено большое количество различных подходов к моделированию низкоуровневых характеристик изображения и большинство СВІR-систем построено на алгоритмах обработки низкоуровневого содержания.

Алгоритмы поиска по содержанию изображений также можно разделить на классы в зависимости от характеристики, которую использует тот или иной алгоритм. Каждый из этих классов может делиться на подклассы по типу алгоритма построения вектора признака.

Среди различных цветовых характеристик изображения наиболее часто используется гистограмма цветов, вектор цветовой связности, коррелограмма цветов, цветовые моменты, дескриптор доминантного цвета [1]. Цветовая гистограмма вычисляется в результате подсчета числа пикселей на изображении или его области, попадающих в определенные квантованные ячейки в некотором цветовом пространстве, например, RGB. Достоинством цветовых гистограмм является простота и высокая скорость получения, устойчивость к частичному перекрытию объектов, повороту, малых наклонов относительно оптической оси и масштабированию. К недостаткам относят чувствительность к масштабу и в них не фиксируется местоположение пикселей друг относительно друга, т. е. одна и та же гистограмма может быть получена для различных изображений [1]. Вектор цветовой связности позволяет объединить информацию о цветовом содержании изображения с пространственной информацией [1]. Цветовая коррелограмма – это таблица, индексированная по цветовым парам, где k -я запись для пары (i, j) определяет вероятность нахождения на изображении пикселя с цветом j на расстоянии k от пикселя с цветом i . Размер цветовой коррелограммы может быть очень большим в том случае, если учитывать всевозможные пары цветов [1]. Цветовые моменты определяются с использованием цветовых моделей $L^*u^*v^*$ и $L^*a^*b^*$. Момент третьего порядка s_3 обеспечивает увеличение эффективности поиска, но его использование повышает чувствительность к изменениям сцены [1].

Текстурные характеристики описывают пространственное распределение цветов или значений интенсивности на изображении. Текстурная характеристика является мерой таких свойств изображений или их областей, как гладкость, шероховатость и регулярность [1]. Текстурные характеристики имеют ограничения – они не несут информации о взаимном расположении элементов изображения.

В литературе описано много различных характеристик формы. Простейшими из них являются геометрические параметры областей: площадь, длина периметра, центр тяжести, округлость, прямоугольность, характеристики описывающего прямоугольника, характеристики выпуклой оболочки. Обычно эти характеристики вычисляются после сегментации изображения, заключающейся в его разделении на фон и объект [1].

Более подробно с классификацией и описанием алгоритмов формирования векторов по низкоуровневым характеристикам изображений можно ознакомиться в работах [1] и [2].

На протяжении длительного периода в большинстве существующих СВІR-системах изображения описывались набором низкоуровневых характеристик. Многие исследователи пришли к пониманию недостаточности таких алгоритмов, т. к. необходимо принимать во внимание особенности визуального восприятия человека, его ориентированность на семантику изображения. Также недостаток алгоритмов, основанных на низкоуровневых признаках изображений, заключался в возможности потери большого количества информации, когда объект изображался в разных ракурсах, разных масштабах, но на одинаковых сценах, или если на изображении присутствуют помехи [4].

За последнее десятилетие была создана большая теоретическая база в сфере обработки изображений и поиска на нём различных объектов [11]. Например, это методы контурного анализа, поиск шаблона (template matching), выделение особых точек / ключевых признаков на изображении (feature detection) и генетических алгоритмов [11]. В контексте данной работы особый

интерес представляют алгоритмы выделения ключевых признаков изображения.

Концепция feature detection в компьютерном зрении относится к методам, которые нацелены на вычисление абстракций изображения и выделения на нем ключевых особенностей. Эти особенности могут быть как в виде изолированных точек, так и в виде кривых или связанных областей. Не существует строгого определения того, что такое ключевая особенность изображения. Каждый алгоритм понимает под этим своё (углы, грани, области и т. п.) [11].

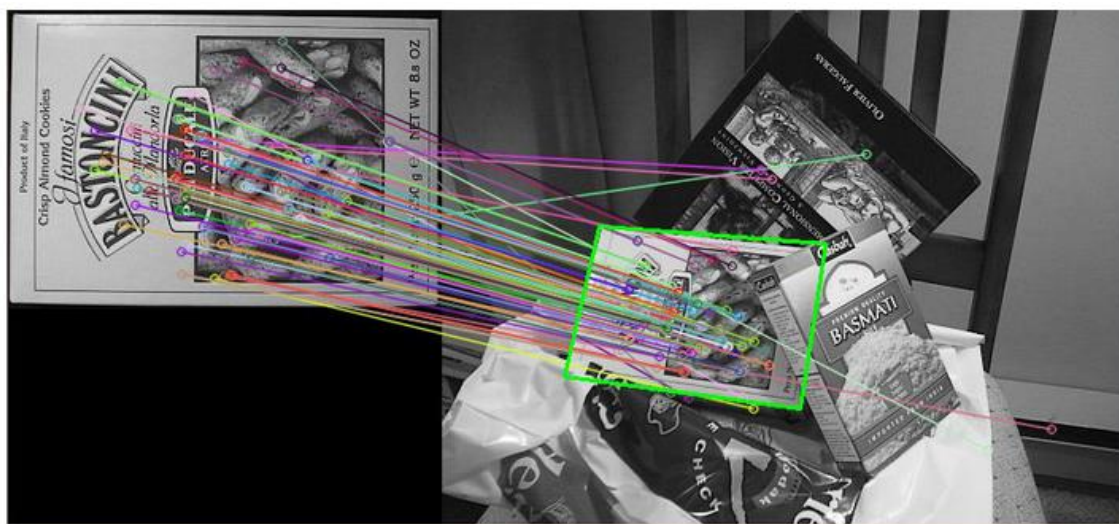


Рисунок 6. Пример соответствия между точками шаблона и тестируемого изображения.

В зависимости от используемого математического аппарата, алгоритмы поиска особых точек изображения могут быть направлены на поиск углов (особые точки, которые формируются из двух или более граней, которые, как правило, определяют границу между различными объектами и / или частями одного и того же объекта) и описание изображения в терминах регионов (blobs) [6]. К первой группе, к примеру, можно отнести детектор Харриса, метод FAST, ко второй – SIFT, SURF, BRIEF.

Процесс построения дескриптора изображения можно условно разделить на выявление особых точек изображения и формирования самого дескриптора. Особая точка – это некоторый участок картинки, который

является отличительным для заданного изображения [11]. Что именно принимается за данную точку – напрямую зависит от используемого алгоритма. Для нахождения особых точек и их последующего сравнения используются три составляющие:

- *детектор* (англ. feature detector) – осуществляет поиск особых точек на изображении;
- *дескриптор* (англ. descriptor extractor) – производит описание найденных особых точек, оценивая их позиции через описание окружающих областей; описание особой точки, определяющее особенности её окрестности, представляет собой числовой или бинарный вектор определенных параметров, где длина вектора и вид параметров определяются применяемым алгоритмом; полученные дескрипторы сохраняются в отдельный файл (или базу данных), чтобы не выполнять процесс повторно;
- *матчер* (англ. matcher) – используется для построения соответствий между двумя наборами особых точек изображений.

Требования к особым точкам:

- повторяемость (repeatability). Особенность находится в одном и том же месте сцены несмотря на изменение точки обзора и освещения (инвариантность к преобразованиям);
- значимость (saliency). Каждая особенность потенциально имеет уникальное (distinctive) описание;
- количество особенностей существенно меньше числа пикселей изображения;
- в качестве особых точек часто выступают экстремумы изображения, углы, точки границы и пятна.

Требования к дескрипторам:

- специфичность (разные точки различаются);
- локальность (зависит только от небольшой окрестности);

- инвариантность к преобразованиям;
- устойчивость к шуму;
- простота в вычислении.

При решении реальных задач дескрипторы не обладают сразу всеми указанными характеристиками, и поэтому, в зависимости от особенностей предметной области и технических характеристик анализируемых изображений, выбирается тот, который лучшим образом удовлетворяет приоритетной функциональности [6].

Самые известные алгоритмы поиска особых точек изображения - SURF [5] (Speeded Up Robust Features) и SIFT [19] (Scale-invariant Feature Transform). Алгоритм SIFT был опубликован в 1999 году David G. Lowe, и основан на гистограммах градиентов интенсивности, отчасти обладавших необходимыми типами инвариантности. Благодаря своему качеству работы и простоте SIFT - дескрипторы долгое время использовались для получения дескрипторов изображений. В 2006 году Herbert Bay, Tinne Tuytelaars и Luc Van Gool представили алгоритм SURF, являющейся улучшенной версией алгоритма SIFT. SURF - один из наиболее популярных методов выделения ключевых точек и их дескрипторов и по сей день. Алгоритм инвариантен к масштабу и повороту. SURF формирует дескрипторы, представляющие собой набор из 64 или 128 чисел для каждой ключевой точки.

Ввиду того, что алгоритмы SIFT и SURF имеют патентные ограничения, будут рассмотрены другие алгоритмы поиска особых точек изображений с открытым исходным кодом: ORB, KAZE и AKAZE.

1.4.1 Алгоритм ORB

Алгоритм ORB (Oriented FAST and Rotated BRIEF) [14] был представлен в 2011 г. авторами Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary Bradski для обнаружения и описания особых точек изображения. Алгоритм основан на улучшенной версии детектора ключевых особенностей FAST – oFAST (FAST

Keypoint Orientation), вычисляющий ориентацию каждой точки, и бинарном дескрипторе BRIEF, для которого авторы в своей работе пытаются улучшить производительность при повороте изображения.

Для поиска угловых точек в детекторе FAST поочерёдно рассматриваются окрестности по 16 пикселей вокруг каждого пикселя p . Пиксель p находится в центре этой окружности, с радиусом равным 3. Если существует N (в первоначальной версии алгоритма $N=12$, но лучшие результаты метода достигаются при $N=9$) пикселей на окружности, при условии, что все N светлее $I_p + t$, либо темнее $I_p - t$, где I_p – яркость точки p , t – пороговая величина, то точку p можно рассматривать как особую. Далее проверяются значения яркости пикселей окружности на позициях 1,5,9,13. Если для трех пикселей из четырех выполняется условие $I_i < I_p - t$ или $I_i > I_p + t$, $i = 1...4$, тогда p – особая точка (рис. 7). Для обнаруженных точек вычисляется мера Харриса [20], где точки с низким значением меры Харриса отбрасываются.

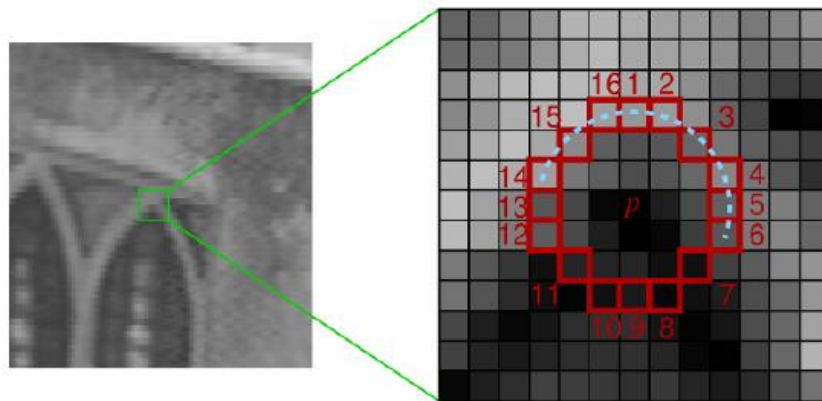


Рисунок 7. Рассматриваемая окрестность точки p FAST детектора.

Далее, вычисляется угол ориентации особой точки и моменты яркости для окрестности особой точки:

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y), \quad (3)$$

где x, y – пиксельные координаты, I – яркость. После чего, получаем угол ориентации особой точки:

$$\theta = \text{atan2}(m_{01}, m_{10}). \quad (4)$$

Дескриптор BRIEF представляет собой бинарный вектор, длиной 256 бит, созданный на основе набора бинарных тестов интенсивности вокруг особой точки p . Двоичный тест τ определяется как:

$$\tau(I; x, y) = \begin{cases} 1 : I_x < I_y \\ 0 : I_x \geq I_y \end{cases} \quad (5)$$

где x, y – области 5×5 пикселей, используемые для сравнения значений яркостей между x и y в окрестности 31×31 пиксель, I – средняя яркость выбранной области.

Для достижения инвариантности к вращению область вычисления дескриптора ориентируется по ориентации θ особой точки. Все 256 наборов x_i и y_i формируют матрицу S размерностью $2 \times n$. Затем, S с помощью матрицы поворота R_θ ориентируется в соответствии с углом θ :

$$S_\theta = R_\theta S. \quad (6)$$

А сам вектор дескриптора записывается как:

$$g_n(I, \theta) = f_n(I) \mid (x_i, y_i) \in S_\theta, \quad (7)$$

где

$$f_n(I) = \sum_{1 \leq i \leq n} 2^{i-1} \tau(I; x_i, y_i). \quad (8)$$

По результатам исследований авторов алгоритма ORB, представленный подход дает значительный выигрыш в скорости при сопоставимой или лучшей точности, чем SIFT и SURF.

1.4.2 Алгоритм KAZE

Pablo F. Alcantarilla, Adrien Bartoli, Andrew J. Davison в работе в 2012 г. представили алгоритм KAZE [28] для обнаружения и описания особых точек изображения. Метод основан на нелинейных масштабных пространствах с

использованием эффективных методов схем аддитивного разделения операторов (Additive Operator Splitting, AOS), которые стабильны при любом размере шага и распараллеливаются, и диффузии с переменной проводимостью. Несмотря на умеренное увеличение вычислительных затрат, получены хорошие результаты в производительности как в обнаружении особых точек, так и в их описании по сравнению с предыдущими современными методами, такими как SURF, SIFT.

Для определения особых точек вычисляется отклик нормализованного по шкале детерминанта Гессiana на нескольких уровнях шкалы. Для многомасштабного обнаружения признаков набор дифференциальных операторов необходимо нормализовать относительно масштаба, т. к. амплитуда пространственных производных уменьшается с масштабом:

$$L_{Hessian} = \sigma^2 (L_{xx}L_{yy} - L_{xy}^2), \quad (9)$$

где (L_{xx}, L_{yy}) - горизонтальная и вертикальная производные второго порядка соответственно, а L_{xy} - перекрестная производная второго порядка. Учитывая набор отфильтрованных изображений из линейного масштаба L^i , анализируется отклик детектора на разных уровнях масштаба σ_i . Поиск экстремумов выполняется во всех отфильтрованных изображениях, кроме $i = 0$ и $i = N$. Каждый экстремум ищется по прямоугольному окну с размерами $\sigma_i \times \sigma_i$ на текущих i , по верхнему $i + 1$ и нижнему $i - 1$ отфильтрованных изображений. Для ускорения поиска экстремумов сначала проверяются ответы по окну размером 3×3 пикселя, чтобы быстро отбрасывать не максимальные ответы. Наконец, положение ключевой точки оценивается с точностью до субпикселя с использованием метода, предложенного в [42].

Множество производных первого и второго порядка аппроксимируется с помощью 3×3 фильтров Шарра с различными размерами шагов производной σ_i . Производные второго порядка аппроксимируются с помощью последовательных фильтров Шарра в нужных координатах

производных. Эти фильтры аппроксимируют инвариантность вращения значительно лучше, чем другие популярные фильтры, вроде оператора Собеля.

Для получения инвариантных к вращению дескрипторов необходимо оценить доминирующую ориентацию в локальной окрестности, центрированной в точке расположения. Подобно алгоритму SURF, находится доминирующая ориентация в круговой области радиуса $6\sigma_i$ с шагом выборки размера σ_i . Для каждой из выборок в круговой области производные первого порядка L_x и L_y взвешиваются с Гауссовым центром в интересующей точке. Затем производные отклики представляются в виде точек в векторном пространстве, а доминирующая ориентация определяется суммированием откликов внутри скользящего круга, охватывающего угол $\pi/3$. Из самого длинного вектора получается доминирующая ориентация.

Для построения дескриптора используется M-SURF дескриптор, адаптированный к нелинейной масштабной пространственной структуре. Для обнаруженного объекта в масштабе σ_i производные первого порядка L_x и L_y размера σ_i вычисляются по прямоугольной сетке $24\sigma_i \times 24\sigma_i$. Эта сетка разделена на 4×4 подобласти, размера $9\sigma_i \times 9\sigma_i$, с перекрытием $2\sigma_i$. Производные отклики в каждом субрегионе взвешиваются с Гауссовым ($\sigma_1 = 2.5\sigma_i$) центром в центре субрегиона и суммируются в вектор дескриптора

$$d_v = \left(\sum L_x, \sum L_y, \sum |L_x|, \sum |L_y| \right). \quad (10)$$

Затем, каждый вектор подобласти взвешивается с использованием Гаусса ($\sigma_2 = 1.5\sigma_i$), определенного над маской 4×4 и центрированного на интересующей ключевой точке. При рассмотрении доминирующей ориентации ключевой точки каждый из образцов в прямоугольной сетке поворачивается в соответствии с доминирующей ориентацией. Кроме того, производные также вычисляются в соответствии с доминирующей

ориентацией. Наконец, вектор дескриптора, длиной 64, нормализуется в единичный вектор для достижения инвариантности к контрасту.

1.4.3 Алгоритм AKAZE

Pablo F. Alcantarilla, Jesús Nuevo, Adrien Bartoli в работе [15] в 2013 г. представили алгоритм AKAZE (Accelerated-KAZE, A-KAZE). Алгоритм использует математическую структуру Fast Explicit Diffusion (FED) для поиска ключевых точек изображения, встроенную в пирамидальную структуру, для значительного ускорения нелинейных пространственных вычислений в масштабе. В качестве дескриптора используется модифицированный локально-разностный двоичный дескриптор (Modified-Local Difference Binary, M-LDB), который использует информацию градиента из пространства нелинейного масштаба.

Для обнаружения особых точек вычисляется определитель Гессияна для каждого из отфильтрованных изображений L^i в нелинейном масштабном пространстве. Множество дифференциальных многомасштабных операторов нормируются относительно масштаба, используя нелинейный коэффициент масштабирования, который учитывает октаву каждого конкретного изображения в нелинейном масштабном пространстве, т. е. $\sigma_{i,norm} = \frac{\sigma_i}{2^{o_i}}$, и

$$L_{Hessian}^i = \sigma_{i,norm}^2 (L_{xx}^i L_{yy}^i - L_{xy}^i L_{xy}^i). \quad (11)$$

Для вычисления производных второго порядка используются каскадные фильтры Шарра с размером шага $\sigma_{i,norm}$. Фильтры Шарра аппроксимируют инвариантность вращения лучше, чем другие фильтры. Сначала находят максимумы отклика детектора. На каждом уровне i проверяется, что отклик детектора выше предопределенного порога и является максимумом в окне размером 3×3 . Это сделано, чтобы быстро отбросить не максимальные ответы.

Затем, для каждого из потенциальных максимумов проверяется, что ответ максимален по отношению к другим ключевым точкам от уровня $i + 1$ и $i - 1$, в окне размера $\sigma_i \times \sigma_i$ пикселей. Наконец, двумерное положение ключевой точки оценивается с точностью до субпикселя, путем подгонки двумерной квадратичной функции к определителю отклика Гессiana в окрестности 3×3 пикселей и нахождения его максимума.

Дескриптор M-LDB использует производные, вычисленные на этапе обнаружения признаков, сокращая количество операций, необходимых для создания дескриптора. Учитывая, что M-LDB вычисляет аппроксимацию среднего значения одних и тех же областей интенсивности и градиента на изображениях, булевы значения, полученные в результате сравнений, не зависят друг от друга. Ожидается, что уменьшение размера дескриптора, путем выбора случайного подмножества битов, улучшит результаты или сможет уменьшить вычислительную нагрузку без снижения производительности. Итоговый бинарный дескриптор имеет длину 486 бит.

1.5 Поиск и сопоставление изображений

Получение вектора признаков изображения (дескриптора) является только одним этапом при конструировании системы поиска по изображению-запросу. Для сопоставления дескрипторов необходимо задать метрику.

Для вещественных дескрипторов используется Евклидово расстояние. Пусть в n -мерном пространстве задано два числовых вектора $x = \{x_i\}_{i=1}^n$ и $y = \{y_i\}_{i=1}^n$. Тогда Евклидово расстояние между ними равно:

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}. \quad (12)$$

Для дескрипторов, представляющих особые точки в виде бинарной строки, используется расстояние Хемминга [24], которое вычисляется как количество неравных значений в векторах:

$$d_H(x, y) = \sum_{s=1}^n \delta_i(x, y), \quad (13)$$

где

$$\delta_i(x, y) = \begin{cases} 1, & \text{при } x_i \neq y_i \\ 0, & \text{при } x_i = y_i \end{cases}. \quad (14)$$

При запросе искомого вектора признаков поиск идет до совпадения с хранящимися векторами признаков в базе данных, что замедляет работу при наличии большого объема хранимых изображений [7]. Для повышения скорости выполнения запросов и получения оптимального результата используются кластерные алгоритмы для поиска изображений по содержанию. Определение «кластеризация данных» («data clustering») впервые было применено в публикации 1954 года, посвящённой обработке антропологических данных.

1.5.1 Алгоритм кластеризации *k*-средних (K-Means)

Метод *k*-средних (*k*-means) — наиболее популярный метод кластеризации. Алгоритм *k*-средних производит поиск заранее заданного количества кластеров в немаркированном многомерном наборе данных. Алгоритм выполняет обучение модели, состоящей из *k*-центров кластеров. Наилучшими считаются те центры, для которых расстояние от точек до соответствующих им центров минимально. По мере усложнения данных и увеличения их объема подобные алгоритмы кластеризации можно использовать для извлечения из набора данных полезной информации.

Алгоритм *k*-средних представляет собой версию EM-алгоритма. В типичном варианте метода *k*-средних применяется подход, известный под названием «максимизация математического ожидания» (expectation-maximization, EM). Подход максимизации математического ожидания состоит из следующей процедуры:

1. Выдвигают гипотезу о центрах кластеров.

2. Повторяют до достижения сходимости:

- E-шаг (шаг ожидания): приписывают точки к ближайшим центрам кластеров;
- M-шаг (шаг максимизации): задают новые центры кластеров в соответствии со средними значениями.

Таким образом, алгоритм состоит из следующих шагов:

- задается число кластеров – k .
- Шаг 1. Случайный выбор k объектов – центроидов k кластеров.
- Шаг 2. Соотнесение всех объектов по степени близости с кластерами.
- Шаг 3. Вычисление новых положений центроидов полученных кластеров – усреднение векторов-признаков объектов кластера.
- Шаг 4. Если суммарное квадратичное отклонение новых положений центроидов от старых достаточно мало (например - 0.0001), то алгоритм завершается, иначе Шаг 2.

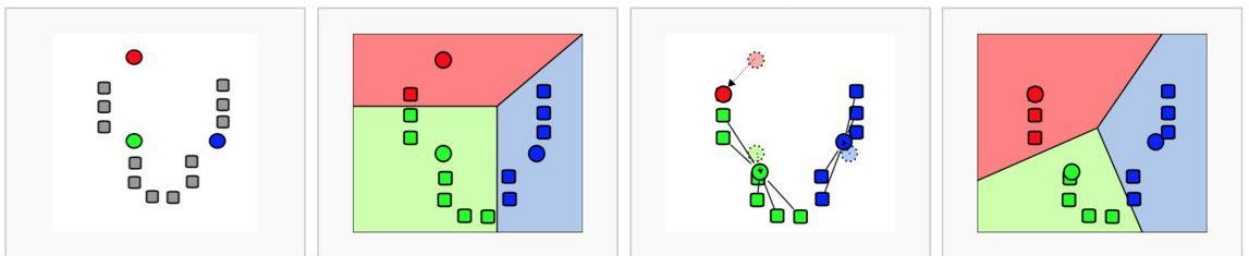


Рисунок 8. Действие алгоритма k-means в двумерном случае. Начальные точки выбраны случайно.

1.5.2 Алгоритм «Мешок визуальных слов»

Понятие «мешок визуальных слов» (Bag of Visual Words, BOVW) берется из модели «мешка слов» для анализа текста и текстовых поисковых систем. Модель BOVW нашла свое применение в классификации изображений и его содержимого, а также используется для построения высоко масштабируемых CBIR-систем. Визуальное слово («visual word») – часто повторяющийся фрагмент изображения. В изображении визуальное слово

может встречаться один раз, много раз или несколько. Словарь – набор фрагментов, часто повторяющихся в коллекции изображений. Чтобы получить словарь, нужно составить большой список всех фрагментов по всей коллекции изображений и разделить весь список на похожие группы. Все фрагменты в одной группе будут экземплярами одного и того же слова.

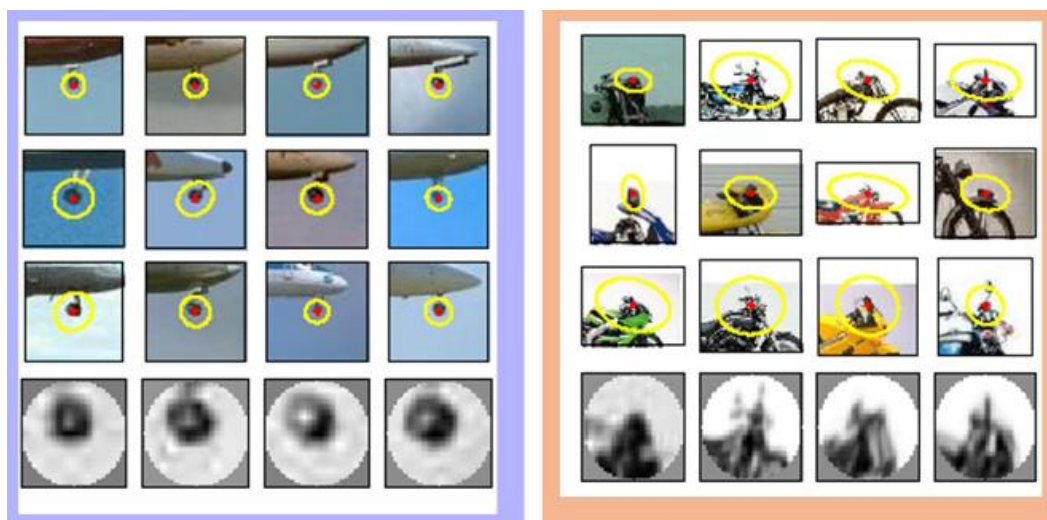


Рисунок 9. Примеры визуальных слов.

Выделяют следующие основные этапы алгоритма:

- *Извлечение особенностей.* Определение характерных признаков изображения. Обычно используют алгоритмы выделения особых точек изображения, имеющих вещественный дескриптор (например SIFT или KAZE).
- *Обучение «визуальных слов».* Построение визуального словаря путем кластеризации всех найденных особых точек изображений, с последующим частотным анализом.
- *Квантование по словарю.* Для каждого фрагмента изображения находится слово из словаря - ближайшее по дескриптору слово в словаре. Сопоставление каждого «фрагмента» изображения происходит от 1 до N, где N – размер словаря.
- *Описание изображения частотами «визуальных слов».* Построение гистограммы частот слов и запись всех «мешков» в каком-либо виде. Построение индекса для коллекции изображений.

Алгоритм имеет ряд особенностей в своей стандартной реализации, из-за чего может потребоваться значительная оптимизация. К ним относят:

- вычислительная сложность для построения словаря больших коллекций изображений: из коллекции в 5 тыс. изображений может быть получено около 20 млн. векторов признаков для дальнейшей кластеризации;
- подбор размера словаря: большой словарь – долгое обучение и возможное дальнейшее переобучение, требующее больших вычислительных мощностей, маленький словарь – грубый порог для сопоставления визуальных слов по словарю;
- из чего выходит, что сопоставление слова по словарю (квантование) при больших размерах словаря является очень затратной операцией по времени;
- большой объем памяти для хранения индекса изображения – около 1.9 Гб для хранения дескриптора коллекции в 5 тыс. изображений, около 141 Гб для хранения дескриптора коллекции из 1 млн. изображений;
- прямое сравнение дескриптора со всем словарем очень медленное;
- вектор слов в дескрипторе очень разреженный (например - 1000 ненулевых элементов из 1000 000 словаря) [29].

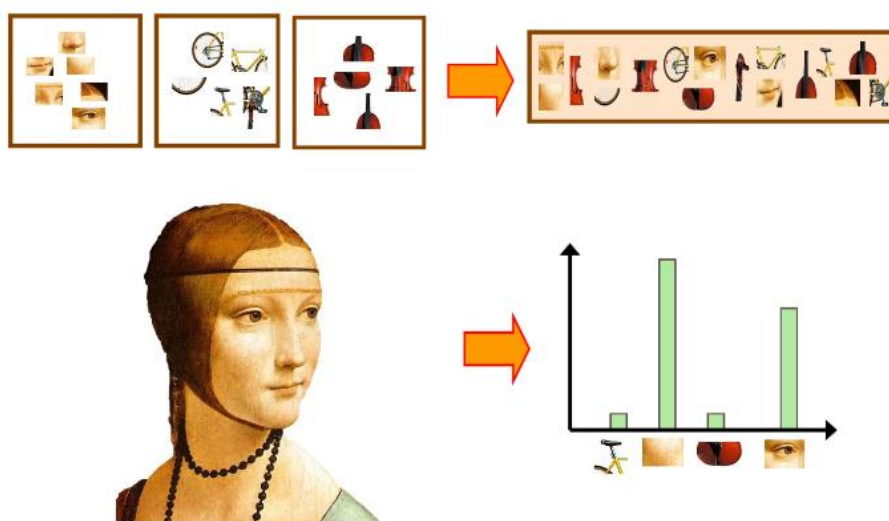


Рисунок 10. Визуальное представление основных этапов алгоритма BOVW.

1.5.3 Частотность терминов – обратная частотность документов

TF-IDF (англ. *term frequency-inverse document frequency*) [43] – метрика важности термина для какого-либо документа относительно всех остальных документов. Терм в сообщении имеет тем больше значение, чем реже он встречается в других сообщениях. Расчет TF-IDF может производиться следующим образом:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right), \quad (15)$$

где $tf_{i,j}$ – число вхождений термов i в документ j , df_i – количество документов содержащих терм i , N – общее число документов. Рассмотрим значения показателей TF и IDF подробнее.

TF (англ. *term frequency*) – частотность термина, описывающая как часто слово может встречаться в документе. Показывает важность слова в пределах отдельного документа. Рассчитывается как:

$$tf(i,j) = \frac{n_i}{\sum_k n_k}, \quad (16)$$

где n_i – число вхождений термов i в документе j , $\sum_k n_k$ – общее количество слов в документе.

IDF (англ. *inverse document frequency*) – обратная частотность документов. Показывает важность термина. Может рассчитываться как:

$$idf(i,N) = \log\frac{1+N}{1+df_i}, \quad (17)$$

где df_i – количество документов содержащих терм i , N – общее число документов. К значениям в числителе и знаменателе прибавляются единицы чтобы избежать деление на 0.

Данная метрика применяется для корпусов текстов. Однако имеет смысл рассмотреть ее и для визуальных слов, где в качестве термина i может выступать визуальное слово, в качестве документа j – изображение, как набор визуальных слов, $tf_{i,j}$ – число вхождений визуальных слов i в изображение

j, df_i – количество изображений содержащих визуальное слово i , N – общее число изображений.

1.6 Технологии и программные средства для работы с изображениями

За прошедшее десятилетие различными организациями и научными сообществами было создано несколько библиотек компьютерного зрения. Большинство из них написаны на языке C++, что делает возможным применение библиотек на различных платформах и обеспечивает хорошую скорость и производительность [11].

В последние годы, большую популярность приобрел язык программирования Python. Python – это один из наиболее популярных современных языков программирования, пригодный для решения разнообразных задач и предлагающий те же возможности, что и другие языки программирования: динамичность, поддержку ООП и кроссплатформенность.

Существуют несколько известных реализаций среды исполнения Python: CPython, Jython и Python.NET. Среда исполнения CPython обычно называется Python, состоит из интерпретатора и модулей расширения, написанных на языке C, и может использоваться на любой платформе, для которой доступен стандартный компилятор C [30].

Библиотека OpenCV (англ. Open Source Computer Vision Library) [25] является одной из самых известных и развитых библиотек компьютерного зрения. В её состав входят следующие модули:

- обработка изображений;
- построение пользовательских интерфейсов, загрузка/сохранение видео и аудиоданных;
- анализ движения и отслеживания объектов (оптический поток, шаблоны движения, устранение фона);
- поиск, анализ и сравнение ключевых точек изображений (Feature Detection and Description);

- детектирование объектов на изображении (вейвлеты Хаара, HOG и т. д.);
- методы и модели машинного обучения (SVM, деревья принятия решений и т. д.) [11].

OpenCV выпускается под лицензией BSD с открытым исходным кодом. Разработка библиотеки началась в 1999 году и продолжается по сей день. Библиотека доступна на языке C++ и Python.

За последние несколько десятилетий язык Python стал мощным инструментом для научных вычислений, включая анализ и визуализацию больших наборов данных [26]. Существует целый ряд библиотек языка Python с реализациями алгоритмов машинного обучения (Scikit-Learn, Tensorflow, Shogun, Keras, Pattern). Одна из самых известных - Scikit-Learn, пакет, предоставляющий эффективные версии множества распространенных алгоритмов [26]. Возможности библиотеки [27]:

- простые и эффективные инструменты для анализа данных и data mining;
- доступна для всех и многократно используется в различных контекстах;
- построена на библиотеках NumPy, SciPy и matplotlib;
- имеет открытый исходный код, коммерчески применима - лицензия BSD.

После изучения вышеописанных методов для поиска изображений по образцу, перспективными и представляющими особый интерес в контексте данной работы являются алгоритмы AKAZE и ORB. Оба алгоритма в своей стандартной реализации имеют бинарный дескриптор, поэтому выше был рассмотрен алгоритм KAZE. Алгоритм «мешок визуальных слов» несмотря на свою ресурсоемкость, выглядит достаточно перспективным для реализации поисковой системы по изображению. Но этот алгоритм может работать только с числовыми данными. Поэтому необходимо определиться какой из алгоритмов ORB или AKAZE с дескриптором KAZE находит и сопоставляет особые точки изображений лучше. Если преимущество окажется за вторым алгоритмом (AKAZE + дескриптор KAZE), будет рассмотрено его применение с алгоритмом «мешок визуальных слов».

ГЛАВА 2. ТЕСТИРОВАНИЕ МЕТОДОВ ПОИСКА ИЗОБРАЖЕНИЙ ПО ОБРАЗЦУ

2.1 Описание используемых данных

Comics Vine – хорошо организованный и постоянно обновляемый ресурс информации по комиксам. Инструменты для его создания были собраны несколькими разработчиками в CBS Interactive. Данные для ресурса поступают ежедневно от фанатов комиксов по всему миру. Ими же ежегодно вносятся миллионы правок на данном ресурсе. У сайта имеется свое API, предоставляющее полный доступ к данным структурированного вики-контента в формате JSON и XML. На сайте имеется подробная документация со всеми предоставляемыми данными для каждого запроса.

Для реализации мобильного клиента и поиска по обложкам комиксов понадобятся следующие запросы:

- /issues, /issue – список номеров (выпусков) и вся информация по одному номеру (под выпуском подразумевается 30-60 страничный комикс в мягком переплете, имеющий нумерацию на обложке, выходящий с определенной периодичностью);
- /volumes, /volume – список томов (сборников) и вся информация по одному тому (под томом подразумевается сборник выпусков, объединённых в одну книгу по какому-либо критерию: сюжетная арка, сборник номеров о персонаже, и т. п.);
- /characters, /character – список персонажей и более подробная информация по конкретному персонажу;
- /teams, /team – список команд персонажей и более подробная информация по каждой команде.

С подробным списком всех запросов, предоставляемых API, можно ознакомиться в документации [31]. На данный момент в базе Comics Vine насчитывается 658 215 номеров и 103 363 томов.

У каждого из приведенных выше запрашиваемых ресурсов имеется поле `image`, предоставляющее заглавное изображение персонажа/команды, или выпуска/тома. Для реализации поиска по обложке комикса будет использоваться изображение, находящиеся в этом поле. Поле хранит ссылки на изображение ресурса в 9 разрешениях (`icon_url`, `medium_url`, `screen_url`, `screen_large_url`, `small_url`, `super_url`, `thumb_url`, `tiny_url`, `original_url`).

При каждом запрашиваемом ресурсе у API имеется общая структура ответа, также доступен постраничный запрос (таб.1).

Таблица 1. Структура ответа от API ComicsVine

<code>status_code</code>	An integer indicating the result of the request. Acceptable values are: <ul style="list-style-type: none"> • 1:OK • 100:Invalid API Key • 101:Object Not Found • 102:Error in URL Format • 103:'jsonp' format requires a 'json_callback' argument • 104:Filter Error • 105:Subscriber only video is for subscribers only
<code>error</code>	A text string representing the <code>status_code</code>
<code>number_of_total_results</code>	The number of total results matching the filter conditions specified
<code>number_of_page_results</code>	The number of results on this page
<code>limit</code>	The value of the limit filter specified, or 100 if not specified
<code>offset</code>	The value of the offset filter specified, or 0 if not specified
<code>results</code>	Zero or more items that match the filters specified

2.2 Тестирование алгоритмов поиска особых точек изображений

В контексте данной работы под «поиском по характерному изображению» подразумевается поиск по фотографии обложки комикса, имеющей идентичные изображения схожего или схожих объектов, в разном масштабе, цветовой гамме, искажении, и возможным перекрывающим объектам (например надписи, с использованием различных шрифтов и на разных языках). Также характерным изображением может бы вырезанный фрагмент из этой обложки (рис. 11). Поэтому, в качестве критерия схожести двух изображений выступает наличие или отсутствие на изображениях одного и того же объекта. При этом обложки комиксов могут отличаться между собой цветом фона, наличием шумов, общей цветовой гаммой, масштабом, а также качеством освещенности и перспективным искажением объекта.



Рисунок 11. Пример похожих обложек.

Тестируемые алгоритмы для поиска особых точек изображения – алгоритм ORB и алгоритм AKAZE с дескриптором KAZE (см. выше в главе 1). Такой выбор обоснован тем, что эти алгоритмы являются самыми производительными из свободно распространяемых [14, 28, 15]. К тому же их реализации есть в библиотеке компьютерного зрения OpenCV.

Алгоритм ORB использует в своей реализации бинарный дескриптор. Алгоритм AKAZE также в своей стандартной реализации опирается на бинарный дескриптор, но для тестирования будет использоваться вещественный дескриптор KAZE. Задача тестирования алгоритмов

заключается в том, что необходимо понять, с помощью какого дескриптора лучше всего описывать найденные точки, и какой алгоритм лучше всего находит эти точки. На основе полученных результатов тестирования будет выбран алгоритм для дальнейшей реализации поиска по изображению.

В соответствии с выбранным критерием схожести изображений, тестируемые алгоритмы должны соответствовать следующим требованиям:

1. Устойчивость к наличию шумов, плохой освещенности объекта.
2. Устойчивость к искажениям и поворотам изображения-запроса.
3. Устойчивость к увеличению или уменьшению масштаба изображений.
4. Устойчивость к изменению цветовой гаммы, наличию дополнительных текстур на изображениях.
5. Устойчивость к наличию других объектов на изображении, изменению фона объекта.

Для тестирования алгоритмов была загружена небольшая коллекция изображений, состоящая из 100 обложек комиксов с помощью API сайта Comics Vine, а в качестве изображений-запросов 5 обложек, позволяющих проверить выполнение перечисленных требований для тестирования.

Тестирование выполнялось в среде PyCharm CE с использованием Python 3.7. Для реализации были использованы следующие библиотеки и модули: OpenCV 4.1.0, NumPy 1.16.3, Scikit-learn 0.21.1, модуль os для работы с операционной системой.

2.2.1 Алгоритм ORB

Тестирование алгоритма реализовано следующим образом. Для каждой обложки загруженной коллекции находятся особые точки и формируется дескриптор. Для этого, с помощью модуля os и библиотеки OpenCV загружается каждое изображение, а полученный массив пикселей передается в метод `featurePointsByORB()`, где происходит инициализация алгоритма ORB,

и с помощью метода `orb.detectAndCompute()` находятся особые точки и дескриптор изображения (Листинг 1).

Листинг 1. Инициализация ORB и получение особых точек и дескриптора изображения.

```
def featurePointsByORB(img):
    orb = cv2.ORB_create()
    kps, des = orb.detectAndCompute(img, None)
    return kps, des
```

Затем, список, состоящий из загруженного изображения, полученных особых точек, дескриптора, имени файла, пустого списка и нулевого значения (счетчик совпадений особых точек), добавляется в общий список всех дескрипторов изображений коллекции - `des_images`. После прохода по всей коллекции изображений, отдельно загружается изображение-запрос, и также с помощью метода `featurePointsByORB()` находятся его особые точки и дескриптор (Листинг 2).

Листинг 2. Получение дескрипторов коллекции изображений и изображения-запроса.

```
# получение дескрипторов всей коллекции изображений
for filename in os.listdir(folder):
    img = cv2.imread(os.path.join(folder, filename))
    if img is not None:
        kps, des = featurePointsByORB(img)
        des_images.append([img, kps, des, filename, 0, []])
        print(filename)

# получение дескриптора изображения-запроса
img_req =
cv2.imread(r"D:\Ekaterina\SourceCode\python\back_test_cs\images_request\angela.jpg")
kps_r, des_r = featurePointsByORB(img_req)
```

Далее, последовательно сравнивается каждое изображение из коллекции обложек с изображением-запросом. Для этого используется класс `BFMatcher` [37] из библиотеки `OpenCV`. Для его инициализации передаются два параметра: `cv2.NORM_HAMMING` – расстояние Хемминга, применимое для бинарных дескрипторов, и параметр `crossCheck` равный `True`, означающий, что метод `bf.match()` возвращает только те совпадения со значением (i,j) , где i -й дескриптор в наборе A имеет j -й дескриптор в наборе B как лучшее совпадение

и наоборот. Т. е. два объекта в двух наборах должны соответствовать друг другу. Результаты совпадений дескрипторов записываются в массив `matches` и сортируются по расстоянию между дескрипторами особых точек. Затем, для каждой пары изображений подсчитывается количество «хороших совпадений» дескрипторов (переменная `best_matches`) - пары дескрипторов особых точек, расстояние между которыми меньше 50 (чем меньше расстояние, тем лучше, а само контрольное значение было подобрано эмпирически). Похожими считаются те сравниваемые изображения, которые имеют наибольшее значение в переменной `best_matches`. Значение `best_matches` добавляется в список `res_matches`, который позже понадобится для отбора найденных похожих обложек. Также значения переменных `best_matches` и `matches` записываются в `item` (Листинг 3).

Листинг 3. Сравнение пары изображений.

```
# последовательное сравнение каждого изображения с изображением-запросом
res_matches = []
for item in des_images:
    print("-----")
    print(item[3])
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des_r, item[2])
    matches = sorted(matches, key=lambda x: x.distance)
    best_matches = 0
    for m in matches:
        if m.distance < 50:
            best_matches += 1
            #print(m.distance)
    res_matches.append(best_matches)
    item[4] = best_matches
    item[5] = matches
```

Для получения контрольного значения наибольшего количества совпадений список `res_matches` сортируется по убыванию. Чтобы вывести хотя бы 5 найденных похожих обложек, берется значение 5-го элемента списка `res_matches` в качестве контрольного (Листинг 4).

Листинг 4. Вывод результатов.

```
res_matches = sorted(res_matches, reverse = True)
print(res_matches)
```

```
control_val = res_matches[5]
for item in des_images:
if item[4] >= control_val:
imgg = item[0]
kps_g = item[1]
img3 = cv2.drawMatches(img_req, kps_r, imgg, kps_g, item[5][:item[4]],
None,
flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
image = resizeImage(img3, 700)
cv2.imshow(str(item[4]), image)
cv2.waitKey(0)
```

Вывод похожих обложек реализован следующим образом: в окне выводится общее изображение, состоящее из изображения-запроса (в левой части) и найденного изображения из коллекции (справа). В заголовке окна результата указаны количество сравнений дескрипторов особых точек (matches) двух изображений и количество лучших совпадений из этих сравнений (good). Линии, проведенные между изображениями, показывают лучшие совпадения особых точек. В приведенном тестировании похожие обложки имеют самое большое количество хороших значений.

Ниже представлены самые показательные результаты работы алгоритма ORB, полученные при тестировании. С полным списком всех результатов можно ознакомиться в Приложении 1.

Первая тестируемая обложка является фотографией обложки переведённого комикса на русский язык. Разрешение фотографии - 400×711px. Изображение-запрос имеет небольшое перспективное искажение и размытость. Найденное изображение с разрешением 400×621px, имеет более четкие контуры, увеличенный масштаб и дополнительные объекты в левом верхнем и нижнем углу. Текст аналогичного шрифта, но на другом языке. Найденное изображение в коллекции было единственной обложкой, соответствующей изображению-запросу, имело 178 сопоставлений (показатель matches), из которых хороших – 68 (показатель good). Это единственное изображение в коллекции, набравшее такой большой результат в показателе good (рис.12) – у всех остальных изображений в коллекции значение меняется в диапазоне от 0 до 9.

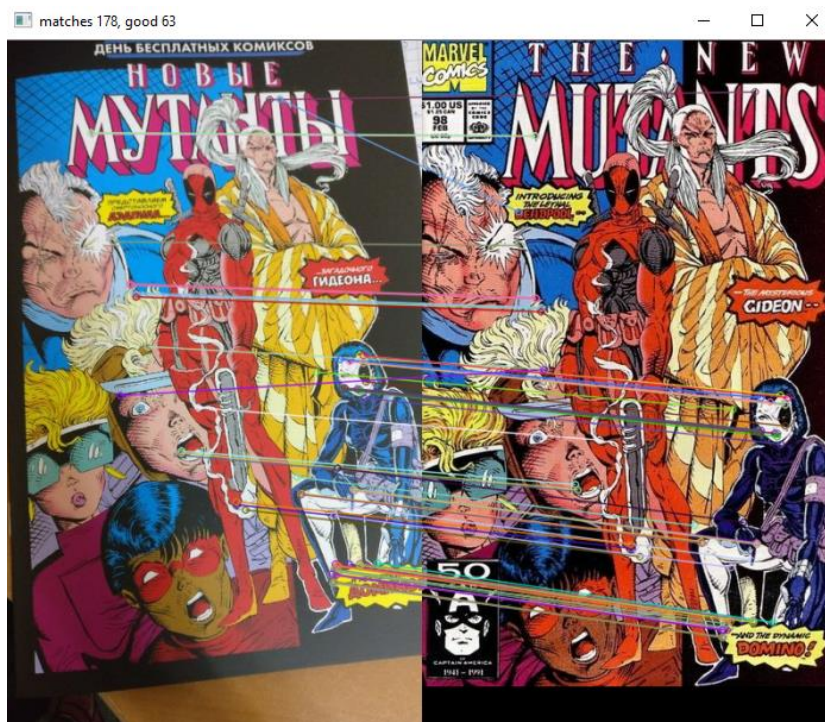


Рисунок 12. Результат ORB для первого изображения-запроса.

Второе изображение-запрос — оригинальная обложка комикса, с четкими контурами и яркими контрастными цветами. Обложка ранее имела очень высокое разрешение — 1867×2800 px, которое было уменьшено до 400×600 px. В коллекции изображений есть две идентичные обложки. Первая практически полностью дублирует запрос, с небольшим отличием в яркости цветов и наличии небольшой текстуры бумаги, разрешение - 300×463 px. Вторая обложка с более явными различиями: другая цветовая гамма, значительно больше теней и текстур, особенно на фоне, изменен текст в верхней части изображения. Разрешение - 339×499 px, изображение перевернуто (рис. 13). Алгоритму удалось выявить эти обложки - они имеют наибольшее количество хороших сравнений, хотя у второй обложки показатель good меньше практически в 3 раза. Показатели первой обложки - matches 224, good 153, второй - matches 151, good 46. У все остальных обложек в коллекции показатель good меняется от 0 до 14.

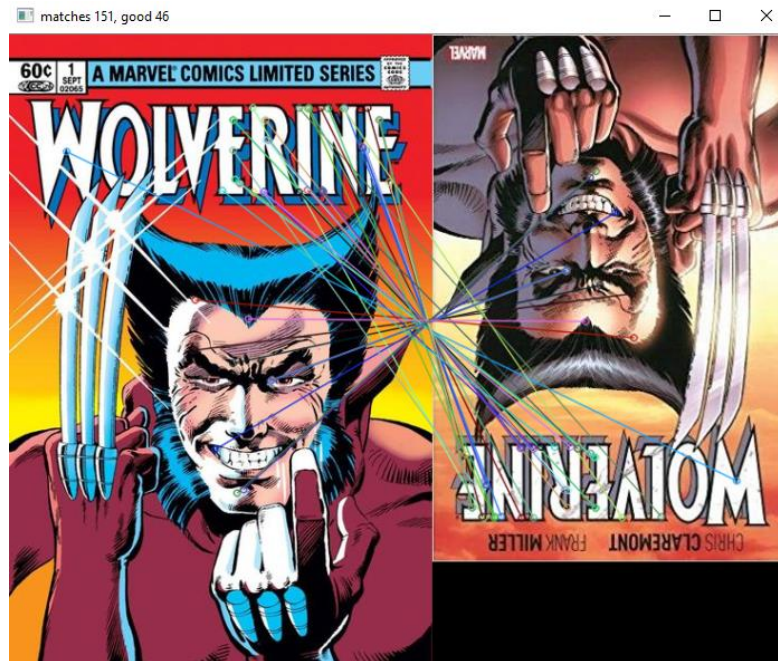


Рисунок 13. Результат ORB для второго изображения-запроса со второй выявленной обложкой.

Далее упор в тестировании алгоритма ORB делался на сравнении обложек, имеющих более заметные различия, чем у предыдущих. Третью обложку-запрос с оригинальной объединяет только наличие одинакового изображения главного персонажа (рис. 14). Разрешение – 319×442px. В коллекции есть три похожих изображения - дубликат запроса и две оригинальных обложки комикса, имеющих разный текст и шрифт: англоязычная версия (225×350px) и переведенная на русский язык (400×601 px). С дубликатом запроса полное совпадение - matches 500, good 500. У второй найденной обложки показатель matches равен 154, показатель good 36. У третьей - показатели matches 138, good 16, при этом на полученном результате можно заметить, что одна из пар точек, соединённая зеленой линией, расположены на разных частях изображений (рис. 14). Все три выявленных изображения имеют наибольшее значения в показателе good по сравнению с другими обложками в коллекции - их значения от 0 до 10.

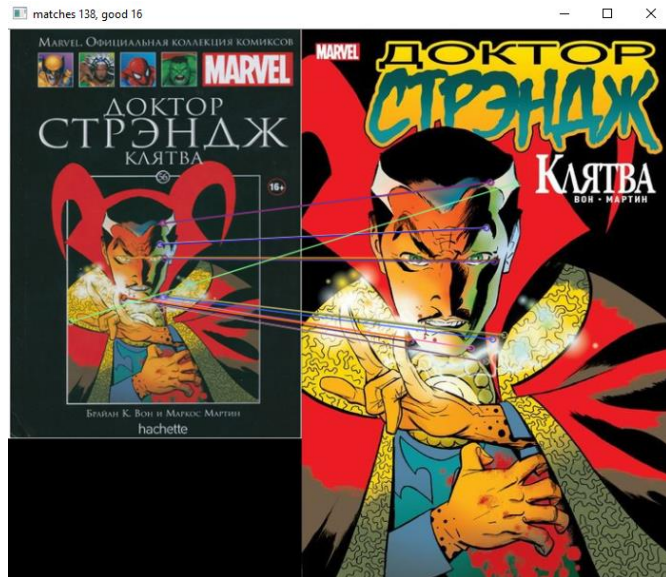


Рисунок 14. Результат ORB для третьего изображения-запроса с третьей выявленной обложкой.

Четвертое изображение-запрос является вырезанным фрагментом из оригинальной обложки с разрешением 400×670px, и преобразовано в чёрно-белое в режиме «мозаика». В коллекции присутствует три изображения, здесь подробно рассмотрено третье (рис.15). В отличие от предыдущих тестов алгоритм смог выявить только две обложки из трех – первую (400×608px) и вторую (400×434px).



Рисунок 15. Результат ORB для четвертого изображения-запроса с не выявленной третьей обложкой.

Первое выявленное изображение – оригинальная обложка комикса, matches 168, good 61. Вторая обложка имеет наибольшее значение в показателе good – 164, matches – 255, т. к. именно из этого изображения было получено изображение-запрос. А вот третья обложка (493×640px) выявлена алгоритмом не была, т. к. показатель good имеет очень низкое значение – 2, при этом matches равно 152 (рис.15). Значение у других обложек менялось от 0 до 13.



Рисунок 16. Результат ORB для последнего изображения-запроса со второй выявленной обложкой.

Последнее тестируемое изображение-запрос - фотография обложки переведённого комикса на русский язык. Изображение слегка размыто, разрешение - 400×611px. В коллекции обложек есть два схожих изображения, и при тестировании оба были выявлены. Первая обложка (500× 756px) — изображение обложки с фотографии-запроса, только без размытия и с четкими контурами. Результаты показателей – matches 242, good 171. Второе изображение - оригинальная обложка комикса, с разрешением 400×623px. Изображение по сравнению с обложкой-запросом имеет другую цветовую гамму, присутствуют дополнительные объекты. Обложка-запроса оказалась частью обложки оригинала. Результаты показателей – matches 63, good 46 (рис.

16). По количеству хороших совпадений это вторая обложка в коллекции. Значение `good` для других изображений – 0 до 24.

По результатам тестирования алгоритма ORB для поиска особых точек на изображении были сделаны следующие выводы:

1. Алгоритм устойчив к шуму и плохой освещенности объекта.
2. Алгоритм устойчив к искажениям и поворотам.
3. Алгоритм устойчив к увеличению или уменьшению масштаба общей сцены изображения.
4. Алгоритм устойчив к изменению цветовой гаммы, наличию дополнительных текстур на изображении.
5. Алгоритм плохо обнаруживает обложки, у которых сильно изменен фон, но при этом присутствует изображение одного и того же объекта, результат может иметь небольшое количество совпадений или практически никаких.

Качество работы алгоритма зависит от разрешения каждого из сравниваемых изображений. Если одно из них превышает разрешение другого в несколько раз, могут быть так называемые «ложные сопоставления».

2.2.2 Алгоритм AKAZE с дескриптором KAZE

Тестирование алгоритма AKAZE с дескриптором KAZE реализовано схожим образом, как и для ORB. Для каждой обложки загруженной коллекции находятся особые точки и формируется дескриптор. С помощью модуля `os` и библиотеки `OpenCV` загружается изображение обложки, а полученный массив пикселей передается в метод `featurePointsByAKAZE()`, где происходит инициализация алгоритма AKAZE, и с помощью метода `detector.detectAndCompute()` находятся особые точки и дескриптор изображения (Листинг 5).

В библиотеке `OpenCV` в реализации алгоритма AKAZE доступно несколько дескрипторов [38]:

- DESCRIPTOR_KAZE_UPRIGHT=2 (Upright descriptors, not invariant to rotation),
- DESCRIPTOR_KAZE=3,
- DESCRIPTOR_MLDB_UPRIGHT=4 (Upright descriptors, not invariant to rotation),
- DESCRIPTOR_MLDB = 5.

Т. к. я использую дескриптор KAZE и для корректной реализации сравнения изображений важна инвариантность к повороту, в конструктор класса AKAZE передается descriptor_type=3.

Листинг 5. Инициализация AKAZE с дескриптором KAZE.

```
def featurePointsByAKAZE(image):
    detector = cv2.AKAZE_create(descriptor_type=3)
    kps, des = detector.detectAndCompute(image, None)
    return kps, des
```

Затем, список, состоящий из загруженного изображения, полученных особых точек, дескриптора, имени файла, пустого списка и нулевого значения (будущий счетчик совпадений), добавляется в общий список всех дескрипторов изображений коллекции - des_images. Затем отдельно загружается изображение-запрос, и также с помощью метода featurePointsByAKAZE() находят его особые точки и дескриптор изображения (Листинг 6).

Листинг 6. Получение дескрипторов всей коллекции изображений и изображения-запроса.

```
des_images = []
for filename in os.listdir(folder):
    # print(filename)
    img = cv2.imread(os.path.join(folder, filename))
    if img is not None:
        kps, des = featurePointsByAKAZE(img)
        # print("des: {}".format(des))
        des_images.append([img, kps, des, filename, []])

query_image =
cv2.imread(r"D:\Ekaterina\SourceCode\python\back_test_cs\images_request\surfer_n.jpg")
kps, des = featurePointsByAKAZE(query_image)
```

Далее, последовательно сравнивается каждое изображение из коллекции обложек с изображением-запросом. Этот процесс был вынесен в отдельный метод `useBFMatcher()`, принимающий в качестве аргументов два сравниваемых дескриптора изображений (Листинг 7). Как и при тестировании алгоритма ORB, используется класс `BFMatcher` из библиотеки `OpenCV`. Для того, чтобы эмпирически не подбирать контрольное значение для проверки расстояния между дескрипторами, использовался метод `bf.knnMatch(des1, des2, k=2)` – находящий k лучших совпадений для каждого дескриптора из набора запросов. Параметр k означает количество лучших совпадений, найденных для каждого дескриптора запроса или меньше, если дескриптор запроса имеет меньше k возможных совпадений [39]. Далее применяется тест соотношения D.Lowe, описаний в его статье [40].

Листинг 7. Сравнение изображений по двум дескрипторам KAZE.

```
def useBFMatcher(des1, des2):
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1, des2, k=2)
    #для вывода «хороших особых точек»
    matchesMask = [[0, 0] for i in range(len(matches))]
    # Apply ratio test
    k_good = 0
    i=0
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            matchesMask[i] = [1, 0]
            k_good +=1
        i +=1
    #print("matches: {}".format(len(matches)))
    #print("good: {}".format(k_good))
    d = dict()
    d['matchesMask'] = matchesMask
    d['k_good'] = k_good
    d['matches'] = matches
    return d
```

Как и при тестировании алгоритма ORB, в список `res_k_good` добавляется количество «хороших совпадений» дескрипторов, которые позже понадобятся для отбора найденных похожих обложек.

Для получения контрольного значения наибольшего количества совпадений список `res_k_good` сортируется по убыванию. Чтобы вывести хотя бы 5 найденных похожих обложек, берется значение 5-го элемента списка `res_k_good` в качестве контрольного (Листинг 8).

Листинг 8. Получение особых точек, дескрипторов, количества хороших сравнений и подсчет результатов.

```
res_k_good = []
for item in des_images:
    print("-----")
    print(item[3])
    dr = useBFMatcher(des, item[2])
    item[4] = dr
    # print(dr['k_good'])
    res_k_good.append(dr['k_good'])

res_k_good = sorted(res_k_good, reverse=True)
#print(res_k_good)
control_val = res_k_good[5]
for item in des_images:
    print("-----")
    print(item[3])
    if item[4]['k_good'] > control_val:
        print(item[4]['matches'])
        print(item[4]['k_good'])
        printBestMatchedImageWithDict(700, query_image, kps,
item[0], item[1], item[4])
```

Вывод похожих обложек реализован следующим образом: в окне выводится общее изображение, состоящее из изображения-запроса (в левой части) и найденного изображения из коллекции (справа). В заголовке окна результата указаны количество сравнений дескрипторов особых точек (`matches`) двух изображений и количество лучших совпадений из этих сравнений (`good`). Линии, проведенные между изображениями, показывают лучшие совпадения особых точек. В приведенном тестировании похожие обложки имеют самое большое количество хороших сравнений. Для тестирования алгоритма АКАZE вывод результата был вынесен в отдельный метод (Листинг 9).

Листинг 9. Вывод результата для алгоритма AKAZE.

```
def printBestMatchedImageWithDict(size, image, kps, image2, kps2, dr):
    draw_params = dict(matchColor=(0, 255, 0),
                       singlePointColor=(255, 0, 0),
                       matchesMask=dr['matchesMask'],
                       flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
    img3 = cv2.drawMatchesKnn(image, kps, image2, kps2, dr['matches'],
                              None, **draw_params)
    image3 = resizeImage(img3, size)
    # plt.imshow(img3),plt.show()
    cv2.imshow("matches " + str(len(dr['matches'])) + ", good " +
              str(dr['k_good']), image3)
    cv2.waitKey(0)
```

Для тестирования использовались те же изображения-запросы и коллекция обложек, как и при тестировании алгоритма ORB. Ниже представлены самые показательные результаты работы алгоритма AKAZE с дескриптором KAZE, полученные при тестировании. Используются те же обозначения для показателей matches и good. С полным списком всех результатов можно ознакомиться в Приложении 2.

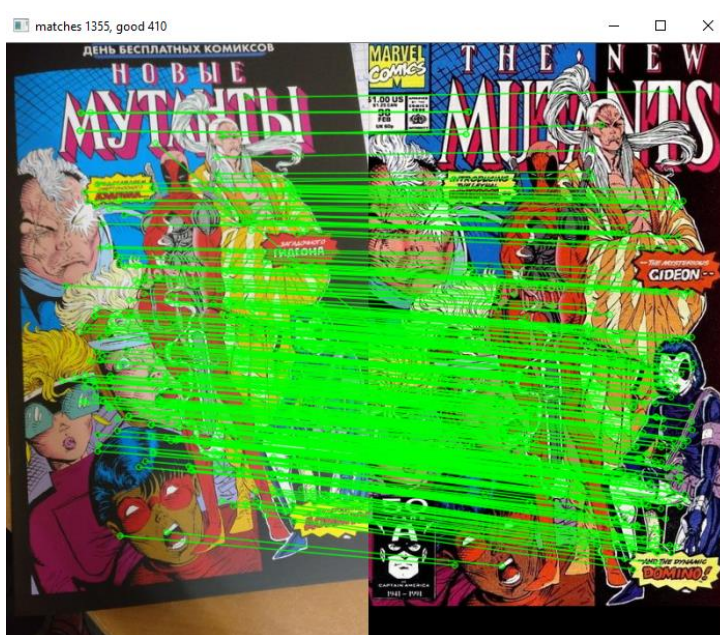


Рисунок 17. Результат AKAZE для первого изображения-запроса.

Для первого изображения-запроса алгоритм AKAZE сопоставил 1355 дескрипторов, из которых хороших сравнений 410 (рис.17). При этом разброс значений показателя good для других изображений составил от 0 до 7.

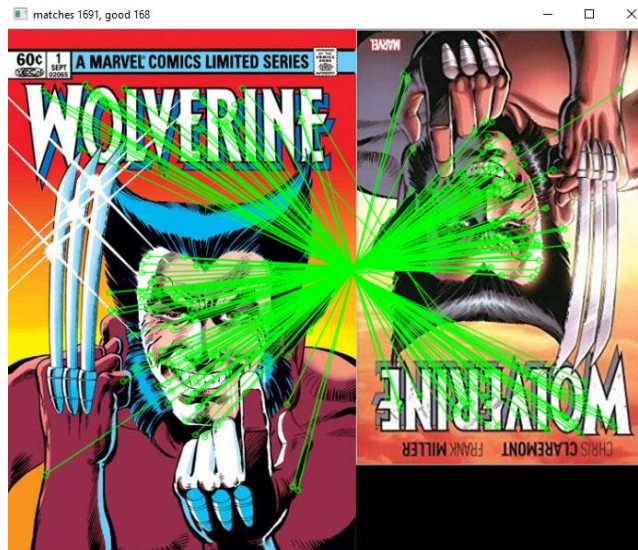


Рисунок 18. Результат AKAZE для второго изображения-запроса со второй обложкой.

Второе изображение-запрос при сравнении с коллекцией обложек выявило два схожих изображения, аналогично алгоритму ORB. Показатели для первой обложки - matches 1691, good 585. Для второй обложки - matches 1691, good 168 (рис.18). Разброс значений good для других изображений составлял от 0 до 20.

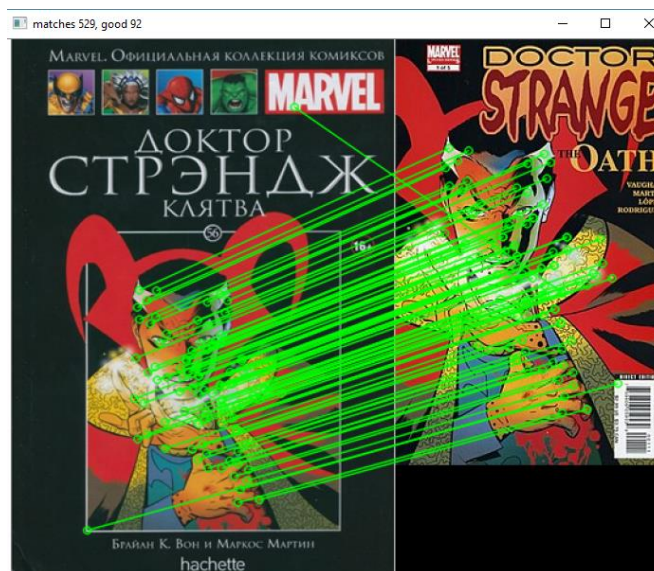


Рисунок 19. Результат AKAZE для третьего изображения-запроса со второй выявленной обложкой.

Сравнение по третьему изображению с алгоритмом AKAZE дало следующие результаты. Дубликат запроса имеет полное совпадение - matches 529, good 529. У второй обложки показатель matches 529, good 92. Замечено

одно ложное сопоставление (рис.19). Третья обложка имеет показатели matches 529, good 95. На этой паре изображений очевидных «ложных» сопоставлений точек нет. Все три выявленных обложки имеют наибольшие значения в показателе good по сравнению с другими обложками в коллекции - от 0 до 5.

Результаты по четвертому изображению-запросу в тестировании оказались гораздо лучше, чем у алгоритма ORB. Алгоритм смог выявить все три похожие обложки в коллекции. Значения показателей первого изображения – matches 1219, good 402, у второй обложки чуть больше – good 414. Третья обложка имеет 140 хороших совпадений, при этом на изображении нет очевидных ложных сравнений (рис.20). Разброс значений good для других изображений в коллекции - от 0 до 7.



Рисунок 20. Результат АКАZE для четвертого изображения-запроса с третьей обложкой.

Сравнение по пятому изображению-запросу с коллекцией изображений также дало два схожих результата, аналогичных алгоритму ORB. Показатели для первой обложки - matches 670, good 366. Показатели второй обложки - matches 670, good 77 (рис.21). Разброс значений good для других изображений в коллекции - от 0 до 7.



Рисунок 21. Результат АКАZE для пятого изображения-запроса со второй выявленной обложкой.

В результате тестирования алгоритма АКАZE с дескриптором KAZE были получены следующие результаты:

1. Алгоритм устойчив к шуму и плохой освещенности объекта.
2. Алгоритм устойчив к искажениям и поворотам.
3. Алгоритм устойчив к увеличению или уменьшению масштаба общей сцены изображения.
4. Алгоритм устойчив к изменению цветовой гаммы, наличию дополнительных текстур на изображениях.
5. Алгоритм устойчив к наличию других объектов на изображении, изменению фона объекта, при этом количество других объектов заметно не влияет на результат как в случае с алгоритмом ORB.

Сравнивая два протестированных алгоритма, можно сделать следующие выводы. У алгоритма АКАZE имеется схожая особенность, как и у алгоритма ORB – зависимость результата сопоставления особых точек от разрешения сравниваемых изображений. Если одно из них превышает разрешение другого в несколько раз, могут быть так называемые «ложные сопоставления». У

алгоритма ORB такие сопоставления наблюдались чаще, чем у алгоритма AKAZE, притом, что детектор AKAZE находил значительно больше особых точек на изображении. Также прослеживается закономерность таких ложных срабатываний – алгоритм AKAZE реагирует на углы в окрестностях пикселей, имеющих белый цвет.

Алгоритм AKAZE находит гораздо больше особых точек, чем алгоритм ORB, а также благодаря вещественному дескриптору KAZE гораздо лучше сопоставляются найденные дескрипторы особых точек. Также, в случае с алгоритмом ORB приходилось самостоятельно подбирать пороговое значение расстояния между дескрипторами, когда для вещественного дескриптора KAZE оказался применимым тест соотношения D.Lowe [40].

Хороших сопоставлений дескрипторов особых точек у AKAZE значительно больше, чем у ORB. Это можно заметить по показателю good алгоритма ORB у выявленных изображений в коллекции, со значением показателя good у изображений с совершенно другим содержанием: например, для обложки на рисунке 14 было найдено всего 16 хороших совпадений, при максимальном значении 10 хороших сопоставлений для непохожих изображений.

Алгоритм AKAZE заметно лучше распознает отдельные объекты на изображении, чем алгоритм ORB. Что видно по результатам тестирования на четвертом изображении-запросе - рисунок 15 и рисунок 20.

Подводя итог, очевидное преимущество за алгоритмом AKAZE в сочетании с дескриптором KAZE перед алгоритмом ORB. Также алгоритм AKAZE с дескриптором KAZE применим для работы с алгоритмом «Мешок визуальных слов».

2.3 Тестирование алгоритма «Мешок визуальных слов»

Тестирование выполнялось в среде PyCharm CE, Python 3.7. Использовались следующие модули и библиотеки: модуль os для работы с

операционной системой, OpenCV 4.1.0, NumPy 1.16.3, Scikit-learn 0.21.1, модули preprocessing и joblib, SciPy 1.2.1, модуль - scipy.cluster.vq, pylab, модуль из библиотеки PIL – Image.

Реализация алгоритма разделена на два этапа – построение словаря «визуальных слов» и поиск по коллекции изображений с применением полученного словаря.

Для формирования словаря визуальных слов использовалась ранее загруженная коллекция обложек, как и при тестировании алгоритма ORB и AKAZE. Размер тестовой коллекции 100 изображений. Прежде чем построить словарь, необходимо извлечь «визуальные слова» из всей коллекции загруженных изображений. Для этого использовался алгоритм AKAZE для поиска особых точек изображения с вещественным дескриптором KAZE, где дескрипторы каждой особой точки выступают в роли фрагментов изображений, и далее записываются в общий массив descriptors. Таким образом, формируется список всех возможных значений «визуальных слов», которые встречаются в данной коллекции изображений (Листинг 10).

Листинг 10. Получение списка всех дескрипторов со всей коллекции изображений.

```
train_path = r"D:\Ekaterina\SourceCode\python\back_test_cs\images2"
training_names = os.listdir(train_path)
print(training_names)
# получение списка файлов
image_paths = []
for training_name in training_names:
    image_path = os.path.join(train_path, training_name)
    image_paths += [image_path]
print(image_paths)

detector = cv2.AKAZE_create(3)
# извлечение дескрипторов из каждого изображения
des_list = []
for i, image_path in enumerate(image_paths):
    img = cv2.imread(image_path)
    print("Extract AKAZE of %s image, %d of %d images" %
          (training_names[i], i, len(image_paths)))
    kps, des = detector.detectAndCompute(img, None)
    des_list.append((image_path, des))
print(len(des_list))
```

```
# формирование списка всех дескрипторов для кластеризации
descriptors = des_list[0][1]
for image_path, descriptor in des_list[1:]:
    descriptors = np.vstack((descriptors, descriptor))
print(descriptors.shape)
```

Выбранный размер словаря - 1000 слов. Для кластеризации использовался алгоритм k-means. Количество заданных кластеров равно количеству заданных слов для словаря (Листинг 11).

Листинг 11. Формирование словаря визуальных слов.

```
# кластеризация k-средних
numWords = 1000
print("Start k-means: %d words, %d key points" %(numWords,
descriptors.shape[0]))
voc, variance = kmeans(descriptors, numWords, 1)
```

Затем производится «квантование» по словарю: для каждого дескриптора особой точки находится ближайшее по дескриптору визуальное слово в словаре, и строится вектор, с количеством повторений каждого слова на изображении (Листинг 12).

Листинг 12. Квантование по словарю.

```
# расчет гистограммы признаков
img_features = np.zeros((len(image_paths), numWords), "float32")
for i in range(len(image_paths)):
    words, distance = vq(des_list[i][1], voc)
    for w in words:
        img_features[i][w] += 1
```

Для всех полученных векторов производится нормализация с помощью метрики TF-IDF и L2-нормализации [41]. Список векторов коллекции изображений, список адресов изображений на диске, метрика idf, размер словаря и сам словарь записываются в файл для дальнейшего использования в модуле поиска (Листинг 13).

Листинг 13. Нормализация векторов изображений и сохранение словаря в файл.

```
#векторизация Tf-Idf
nbr_occurences = np.sum( (img_features > 0) * 1, axis = 0)
idf = np.array(np.log((1.0*len(image_paths)+1) / (1.0*nbr_occurences
+ 1)), 'float32')
```

```
# L2 normalization
img_features = img_features*idf
img_features = preprocessing.normalize(img_features, norm='l2')
joblib.dump((img_features, image_paths, idf, numWords, voc),
"bof2.pkl", compress=3)
```

Модуль поиска организован следующим образом. Загружается изображение-запрос, и с помощью алгоритма AKAZE с дескриптором KAZE извлекаются особые точки и дескрипторы изображения (Листинг 14).

Листинг 14. Загрузка изображения-запроса и получение дескрипторов его особых точек.

```
# загрузка изображения
img_q_path =
r"D:\Ekaterina\SourceCode\python\bovw_test\images_request\angela.jpg"
# загрузка коллекции изображений и их векторов, словаря
img_features, img_paths, idf, numWords, voc = joblib.load("bof2.pkl")

# инициализация алгоритма AKAZE
detector = cv2.AKAZE_create(3)
# список, где будут храниться все дескрипторы загруженного
изображения
des_list = []
img = cv2.imread(img_q_path)
kps, des = detector.detectAndCompute(img, None)
des_list.append((img_q_path, des))
```

Затем производится квантование изображения-запроса по словарю (Листинг 15). Здесь `test_features` – полученный вектор, описывающий изображение-запрос, количеством повторений каждого визуального слова на изображении.

Листинг 15. Загрузка изображения-запроса и получение дескрипторов его особых точек.

```
descriptors = des_list[0][1]
# квантование по словарю
test_features = np.zeros((1, numWords), "float32")
words, distance = vq(descriptors,voc)
for w in words:
    test_features[0][w] += 1
```

Для нормализации вектора изображения-запроса используется ранее загруженная метрика IDF и L2-нормализация. Затем, вектор изображения-запроса перемножается с матрицей векторов изображений коллекции. На

основе переменной score вычисляются индексы сортировки (rank_ID) списка img_paths (Листинг 16).

Листинг 16. Нормализация полученного вектора и подсчет результатов схожести изображений.

```
# Perform Tf-Idf vectorization and L2 normalization
# нормализация вектора изображения из 1000 слов
test_features = test_features*idf
test_features = preprocessing.normalize(test_features, norm='l2')

# подсчет результатов
score = np.dot(test_features, img_features.T)
#print(score)
rank_ID = np.argsort(-score)
#print(rank_ID)
```

Далее, выводятся результаты с похожими изображениями к загруженному изображению-запросу (Листинг 17).

Листинг 17. Отображение результатов.

```
figure(num=None, figsize=(10, 10), dpi=120, facecolor='w',
edgecolor='k')
gray()
subplot(5,4,1)
imshow(img[:, :, :-1])
axis('off')
for i, ID in enumerate(rank_ID[0][0:16]):
    img = Image.open(img_paths[ID])
    print(img_paths[ID])
    gray()
    subplot(5,4,i+5)
    imshow(img)
    axis('off')
show()
```

Ниже представлено несколько результатов работы алгоритма «мешок визуальных слов» (BOVW), в сочетании с алгоритмом поиска особых точек AKAZE и вещественным дескриптором KAZE. Исходное изображение-запрос выделено синим цветом, а далее перечислены 10 пронумерованных обложек из коллекции изображений в порядке схожести с изображением-запросом.

Результат, представленный на рисунке 23 примечателен тем, что алгоритм в выдаче смог найти не только обложку оригинал под номером 1),



Рисунок 23. Результат алгоритма BOVW для первого изображения-запроса.

так и обложку под номером 2), где присутствуют некоторые персонажи из обложки-запроса, но изображенные немного иначе. На обложках 6), 8) и 9) центральные персонажи имеют схожую форму головы, как и центральный персонаж на обложке-запросе.

На рисунке 24, изображение-запрос имеет схожие объекты практически в каждой найденной обложке. Алгоритм смог найти все ранее рассмотренные обложки как ORB и AKAZE – 1), 5) и 7).

Интересны результаты 2) и 9). У изображения-запроса сверху есть 4 изображения других персонажей. Первые две миниатюры есть в обложке под номером 6), третья присутствует на 2) и 9), также на этого персонажа похожи центральные персонажи на 3) и 4) обложке. На рисунке 25 можно рассмотреть эти сходства поближе, миниатюра из персонажей выделена желтым цветом, найденные фрагменты в других обложках – белым.



Рисунок 24. Результат алгоритма BOVW для второго изображения-запроса.

Результат, представленный на рисунке 26 для обложки запроса, показывает, что алгоритм смог найти не только обложки под номером 1) и 2), но и также опознал обложки под номером 3) и 8), с теми же персонажами и объектами на изображении.



Рисунок 25. Найденные схожие объекты в других обложках.

С остальными результатами работы алгоритма можно ознакомиться в Приложении 3.

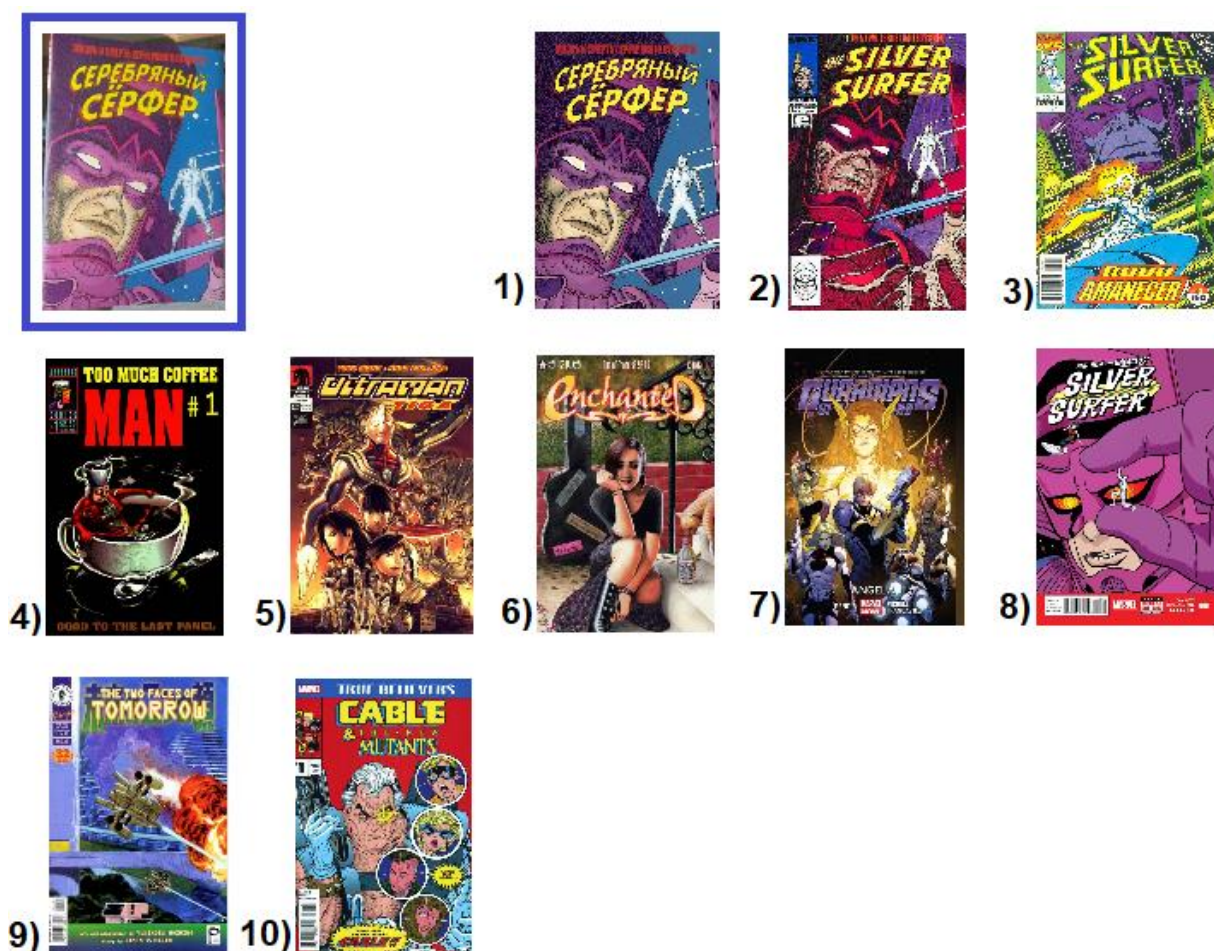


Рисунок 26. Результат BOVW для последнего изображения-запроса.

Таким образом, сочетание алгоритма мешка визуальных слов и алгоритмов поиска особых точек на изображение дает неплохие результаты при сравнении изображений. Притом как для обложек дубликатов, так и для изображений, имеющих схожие объекты: в некоторых случаях алгоритм смог найти, как и идентичные изображения объектов, так и отличающиеся некоторыми деталями или изображенные в другом ракурсе.

Алгоритм тестировался на достаточно маленькой коллекции изображений, и на относительно маленьком словаре, сам словарь был построен на 100 изображениях, поэтому по некоторым обложкам сложно сказать, чем они могут быть схожи с обложкой-запросом. Также алгоритм не смог найти обложку, ранее не найденную при рассмотрении алгоритма ORB (рис. 16).

Полученные результаты пригодны для применения в дальнейшей разработке поиска по обложке для мобильного приложения, с использованием алгоритма «мешок визуальных слов», в сочетании с алгоритмом для поиска особых точек на изображении AKAZE и вещественным дескриптором KAZE. Ключевыми вопросами остаются выбор размера словаря и размер коллекции изображений, на которой нужно построить словарь визуальных слов.

ГЛАВА 3. РЕАЛИЗАЦИЯ СИСТЕМЫ ПОИСКА ОБЛОЖКИ КОМИКСА ПО ФОТОГРАФИИ

3.1 Описание требуемого функционала разрабатываемой системы

Основная идея разрабатываемого мобильного приложения – поиск информации по комикс-ресурсам (выпуски, тома, персонажи и команды), где информацию можно получить либо по названию запрашиваемого ресурса, либо если этот ресурс является комиксом (выпуск или том), загрузив изображение его обложки. Существует достаточно веб-сайтов, предоставляющих информацию по данной тематике. Однако, мне не удалось найти подобные сервисы в формате мобильного приложения, охватывающие все издательства комиксов и имеющий подобный функционал поиска по изображению. Поэтому разрабатываемое мобильное приложение должно обладать следующими возможностями:

- визуализация информации по комикс-ресурсам;
- поиск по названию ресурса с фильтрацией категории - по отдельности или все сразу;
- реализация поиска комикса по фотографии обложки или характерного изображения;
- сохранение комиксов в «избранное» (локально на устройстве).

Данные для приложения получены с использованием API сайта Comics Vine. Визуализация информации по комикс-ресурсам подразумевает следующее:

- отображение данных по категориям – выпуски, тома, персонажи и команды;
- на главном экране – просмотр новых выпусков и томов, загруженных на сайт Comics Vine за последние 2-3 дня;
- подробный просмотр каждой категории (кнопка «More»);
- подробный просмотр информации по каждому ресурсу – выпуск, том, персонаж и команда.

Сохранение комиксов в «избранное» подразумевает, что в приложении есть раздел, куда при желании пользователь сможет сохранять информацию по заинтересовавшим его комиксам, для обеспечения быстрого доступа к подробной информации.

Реализация поиска комикса по обложке также должна быть вынесена в отдельный раздел приложения, т. к. это отдельный сервис, не связанный с поиском по названию комикс-ресурса. Для поиска по изображению в этом разделе должна быть предоставлена возможность загрузки изображения с внутренней памяти мобильного устройства.

3.2 Проектирование архитектуры разрабатываемой системы поиска

Беря во внимание классическую архитектуру CBIR-системы (см. выше в главе 1, пункт 1.3), кроме мобильного клиента необходима реализация веб-сервиса, который возьмет на себя всю вычислительную работу по загрузке и преобразованию изображений в векторный вид, и поиск по хранилищу изображений.

На данный момент в базе сайта Comics Vine насчитывается 658 215 номеров и 103 363 томов. Т. к. для реализации поиска по коллекции изображений будет применяться достаточно ресурсоемкий алгоритм «мешок визуальных слов», было решено сузить охват поиска, и организовать его только для категории обложек томов, при этом для реализации взять часть коллекции обложек томов.

Серверная часть разрабатывалась в среде PyCharm CE, Python 3.7. В качестве базы данных - PostgreSQL 11. Использовались следующие модули и библиотеки: OpenCV 4.1.0, NumPy 1.16.3, Scikit-learn 0.21.1, Flask 1.0.2, psycopg 2 2.8.2, requests 2.22.0, модули preprocessing и joblib, SciPy 1.2.1, модуль - scipy.cluster.vq, pylab, модуль из библиотеки PIL – Image.

Первый этап — это формирование словаря визуальных слов. С помощью API сайта Comics Vine выборочно загружаются данные по комиксам и

записываются в базу данных. В эти данные входит минимально необходимая информация для идентификации комикса (id) и загрузки его обложки (image), а также ссылка на получение более подробной информации по ресурсу (api_detail_url) [31]. В зависимости от предполагаемого размера словаря и размера коллекции изображений, по которой будет осуществляться поиск, можно загрузить данные либо сразу всей коллекции изображений, либо загрузить выборочно только для построения словаря, а потом дополнить остальными.

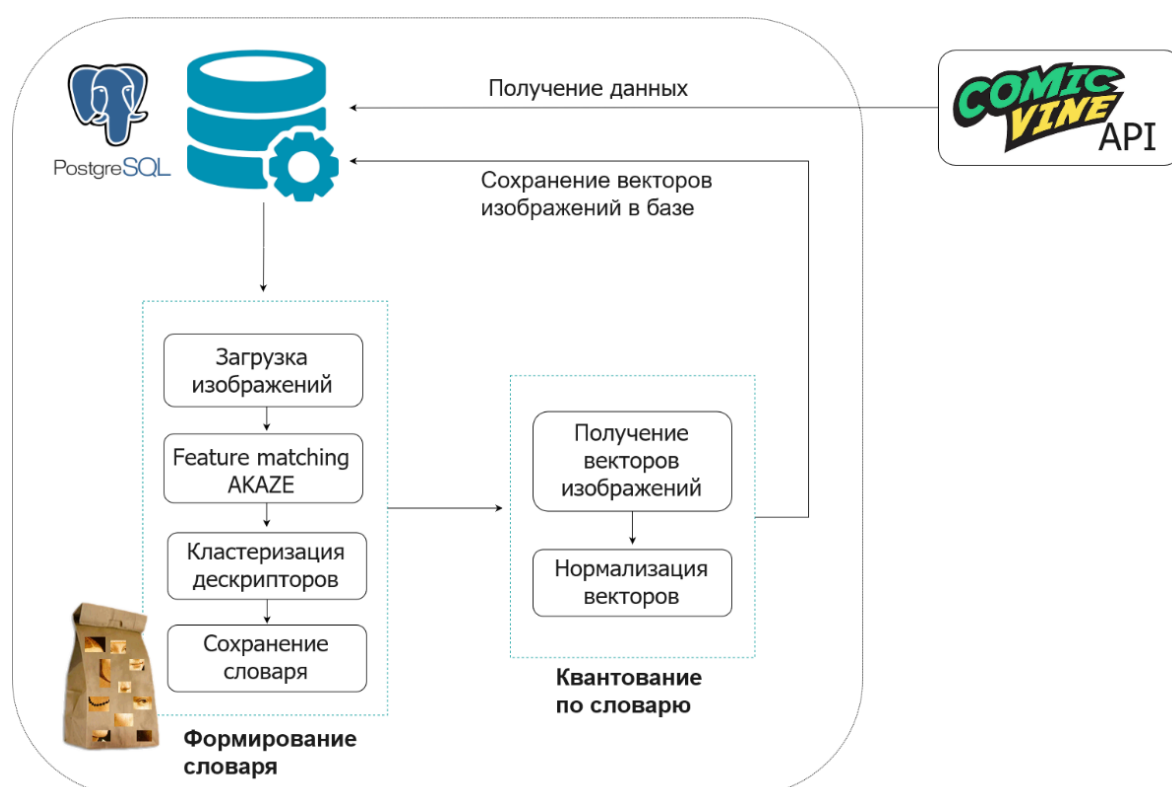


Рисунок 27. Построение словаря «визуальных слов» для разрабатываемой системы поиска.

Загруженные данные считываются из базы, производится скачивание файлов изображений. С помощью алгоритма для поиска особых точек AKAZE и дескриптора KAZE формируются дескрипторы особых точек, все дескрипторы собираются в один массив и проводится их кластеризация. Полученный словарь хранится на серверной части в файле формата rkl. Загруженная коллекция изображений сопоставляется с полученным словарем (квантование по словарю, рисунок 27) и формируются таким образом новые

вектора изображений. Вектора нормализуются и добавляются к записям в базе данных, дополняя уже ранее загруженные данные.

Поиск по коллекции изображений организован следующим образом. Пользователь загружает с мобильного устройства изображение, изображение отправляется с помощью развернутого API веб-сервису. Полученное изображение загружается и обрабатывается, и с помощью алгоритма AKAZE с дескриптором KAZE выделяются особые точки изображения и формируются дескрипторы. Далее, полученные дескрипторы сопоставляются по ранее созданному словарю визуальных слов и формируется вектор загруженного изображения-запроса.



Рисунок 28. Архитектура разрабатываемой системы поиска

Сформированный вектор изображения-запроса приводится к нормализованному виду, и производится поиск похожих векторов в хранилище изображений. Полученный результат из базы данных отправляется в виде JSON структуры мобильному приложению.

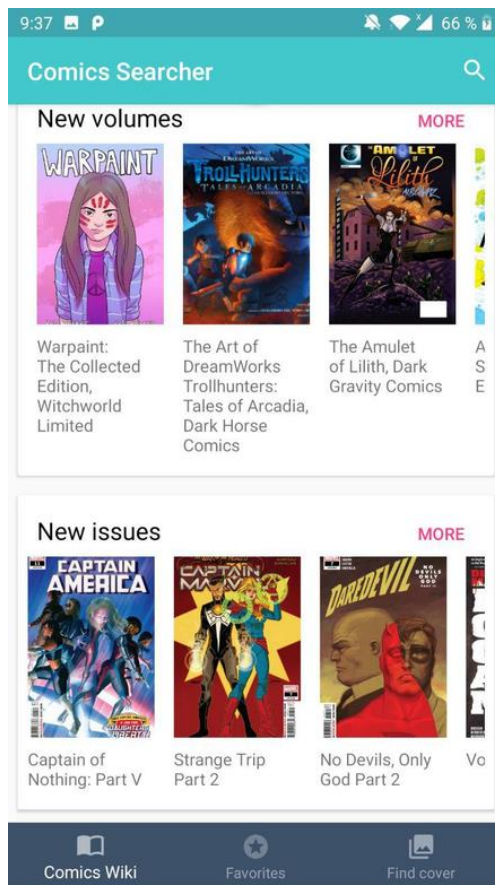
3.3 Мобильный клиент

Реализация мобильного клиента производилась для ОС Android. Минимально поддерживаемая версия операционной системы для приложения, с учетом текущего распределения версий Android [32] - 5.0 Lollipop. Приложение разработано на языке Java 8, в интегрированной среде разработки для работы с платформой Android - Android Studio 3.3. Отладка и тестирование приложения проводилась на собственном устройстве - OnePlus 3t с версией Android 9 Pie.

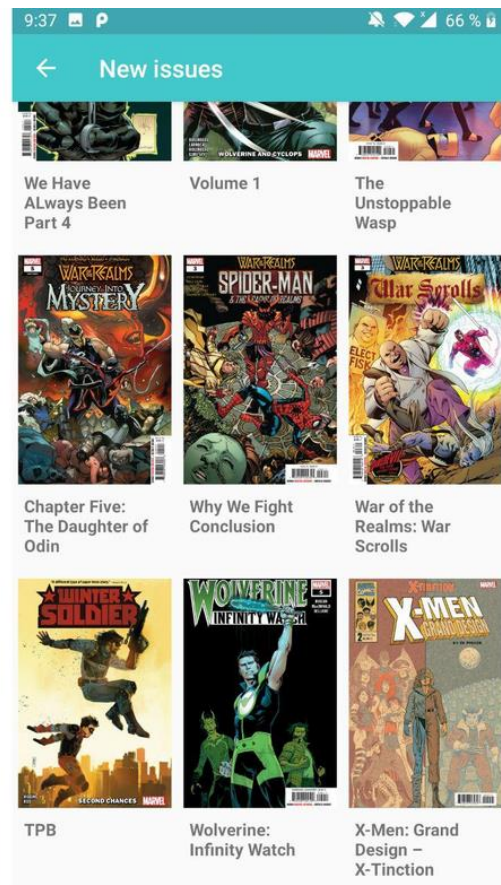
Для реализации мобильного клиента использовались следующие библиотеки:

- Retrofit [33] – типобезопасный (англ. type safety) HTTP-клиент для Android и Java, используемый для работы с API в клиент-серверных приложениях.
- Picasso [34] - библиотека для асинхронной загрузки, кэширования и отображения изображений для Android.
- SQLite [35] - компактная встраиваемая СУБД, часто используемая в качестве локальной базы данных на устройствах с ОС Android. Используется в приложении для кэширования ранее загруженных данных по каждому ресурсу и для хранения избранных комиксов пользователя.
- Sugar ORM [36] - ORM для упрощения работы с SQLite в Android.

Далее представлен разработанный функционал мобильного приложения. На главном экране – просмотр новых выпусков и томов (New volumes, New issues), загруженных на сайт ComicsVine (рис. 29). На этом же окне доступен просмотр других категорий комикс-ресурсов (All volumes, All issues, All characters, All teams). На главном экране у каждой категории отображается прокручиваемый горизонтальный список из 15 элементов. Просмотр полного списка по каждой категории доступен при нажатии на кнопку «More» (рис. 29).



(a)



(b)

Рисунок 29. (a) Стартовый экран приложения. (b) Просмотр полного списка по каждой категории.

В интерфейсе приложения реализован просмотр подробной информации по каждому отдельному ресурсу (выпуск, том, персонаж и команда) при нажатии на изображение обложки или его название (рис. 30). Информации по каждому ресурсу может быть достаточно много, поэтому здесь используется сворачивающаяся панель инструментов (Collapsing Toolbar) и прокрутка окна для просмотра длинного текстового содержимого (Scrolling View) [44].

Реализован поиск по названию комикс-ресурса в отдельном окне приложения. У используемого API сайта Comics Vine есть эндпоинт /search, с подробным описанием которого можно ознакомиться в документации [31]. Для поиска доступна фильтрация – каждая категория комикс-ресурса по отдельности или все сразу (рис.32).

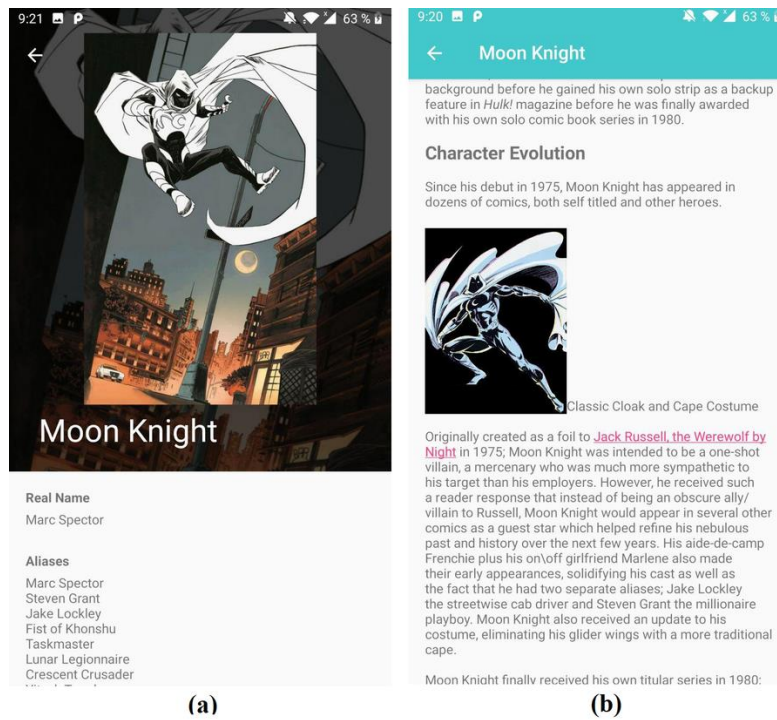


Рисунок 30. Просмотр информации по персонажу. (а) Сворачивающаяся панель с изображением персонажа. (б) Просмотр длинного текстового содержимого.

Сохранение в избранное доступно в подробной информации по комикс-ресурсу только для томов и выпусков. Сохраненные комиксы добавляются в локальную базу данных SQLite и отображаются во вкладке Favorites (рис. 31).

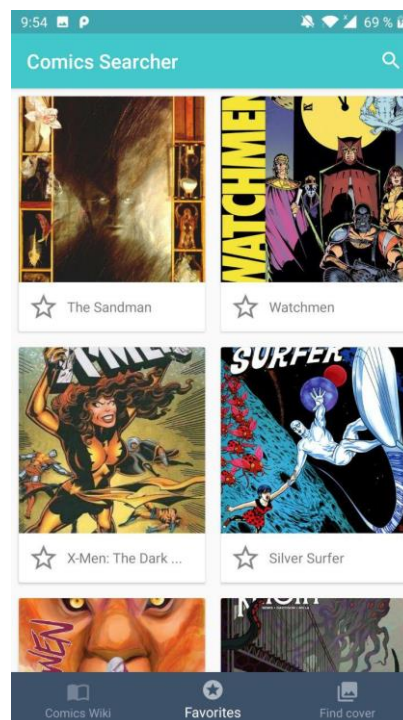


Рисунок 31. Раздел «Избранное» (Favorites).

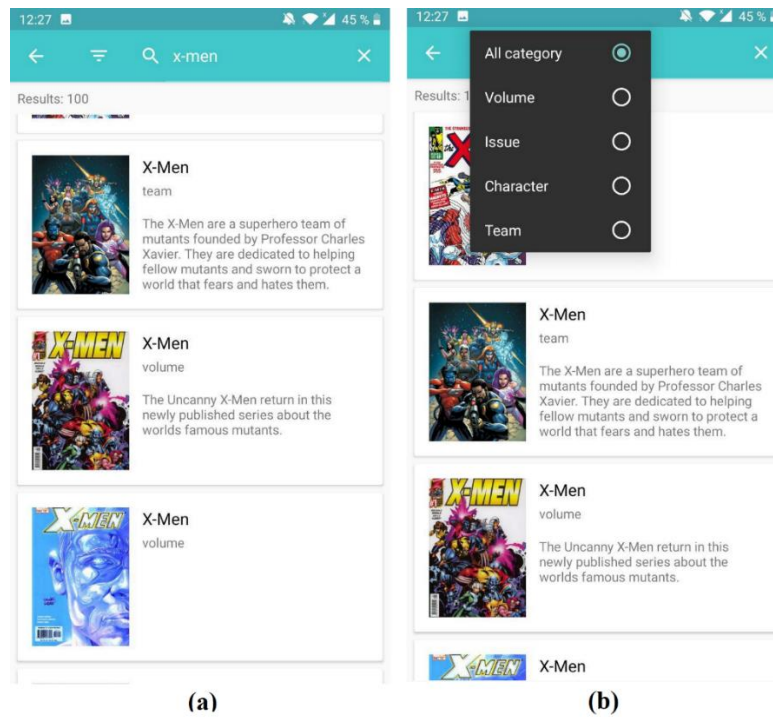


Рисунок 32. (а) Результат поисковой выдачи по названию комикс-ресурса. (б) Меню фильтрации по категориям.

Интерфейс для поиска комикса по обложке помещен в раздел Find cover в нижнем меню приложения. Данная функция доступна только для обложек томов комиксов (рис. 33).

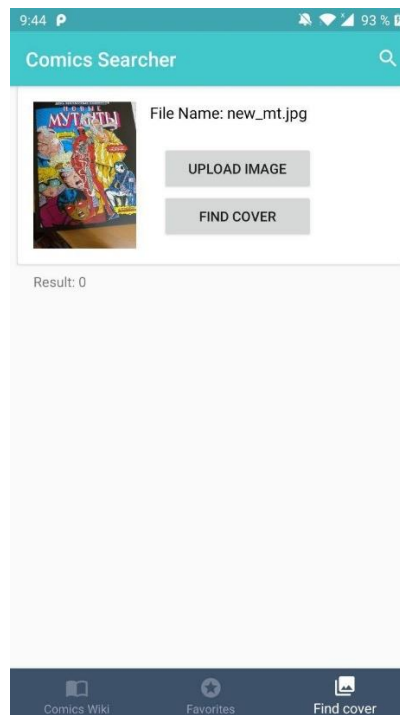


Рисунок 33. Интерфейс для поиска комикса по обложке.

3.5 Тестирование работы приложения

Для тестирования поиска, реализованного с помощью алгоритма «мешок визуальных слов», было создано 2 словаря. Оба состояли из 1100 слов. Первый словарь был сформирован на 200 изображениях, второй на - 1000. На создание первого ушло 2 минуты, на создание второго около 40 минут.

Изначально планировалось построить словарь на коллекции изображений размером в 2000 обложек (а после построить на 5000). Количество слов было тем же - 1100. Скрипт для формирования этого словаря запускался на машине со следующими характеристиками: оперативная память DDR4 объемом 8 ГБ, процессор Intel (R) Core (TM) i7-7700HQ CPU 2.80GHz, 4 ядра 8 потоков, HDD диск на 1ТБ. Формирование такого словаря выполнялось больше часа, при кластеризации дескрипторов оперативная память была забита на 6.5 ГБ, диск загружен на 100%, и компьютер оставался в таком состоянии все последующее время. По этим причинам было решено не ждать завершения программы, и построить словарь на коллекции меньшего объема. Дальнейшая попытка строить словарь на 5000 изображениях не производилась.

Самая ресурсоемкая операция на этапе формирования словаря - кластеризация, и зависит она от количества подаваемых на вход дескрипторов. Для первого словаря, построенного на 200 изображениях, было получено 295 749 дескрипторов, для 1000 изображений - 1 332 293 дескриптора. На 2000 изображениях получено 2 832 220 дескрипторов. Очевидно, что кластеризация такого объема данных требует огромных вычислительных мощностей.

Размер коллекции изображений, на которой тестировался данный алгоритм составляла 7000 изображений, т. к. процесс квантования по словарю занимает достаточно времени. Квантование по первому словарю заняло около 12 минут, по второму – 13. Сравнение изображения-запроса с коллекцией проводилось по тому же принципу как при тестировании алгоритма в главе 2

пункте 2.3. Для данного размера коллекции изображений проблем с быстродействием поиска обнаружено не было.

Ниже представлены результаты поисковой выдачи реализованного поиска по загруженной обложке в интерфейсе мобильного приложения. Представлены результаты поиска для новых изображений-запросов, т. к. в новой коллекции обложек не присутствовали ранее используемые обложки-запросы. Размер поисковой выдачи имеет фиксированное значение – 16 найденных обложек.

Первая обложка - оригинал, переведенный на русский язык. Загружалась для того, чтобы проверить может ли алгоритм в таких условиях найти фактический дубликат. На рисунке 34 (a) представлен результат для первого словаря, построенного на 200 изображениях, 34 (b) – для второго словаря, построенного на 1000 изображений.

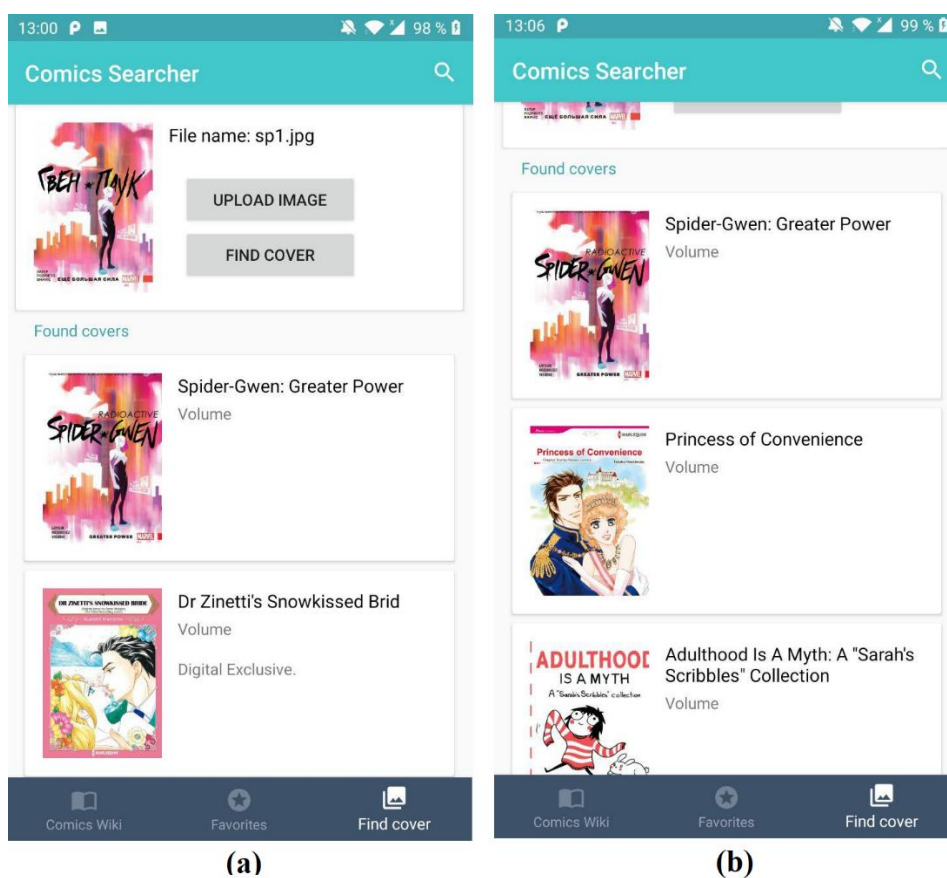


Рисунок 34. Первое изображение-запрос. (a) Результат поиска для словаря, построенного на 200 обложках. (b) Результат поиска для словаря, построенного на 1000 обложках.

Как видно из полученных результатов, ожидаемая обложка была найдена и была на первой позиции в списке. Менялись только последующие обложки или их порядок. Для полученной поисковой выдачи была характерная схожая цветовая гамма.

Второе изображение - фотография обложки без надписей, сделанная в плохом качестве и с искажением (рис.35). Поиск, построенный по первому словарю, смог найти оригинальную обложку, но поместил ее только на 10 место из 16 – рисунок 35 (а). Поиск, построенный на втором словаре, вывел найденную обложку на первое место – рисунок 35 (b). Для остальных найденных обложек как в первом, так и во втором случае, был характерен белый фон и контрастные объекты с четкими линиями.

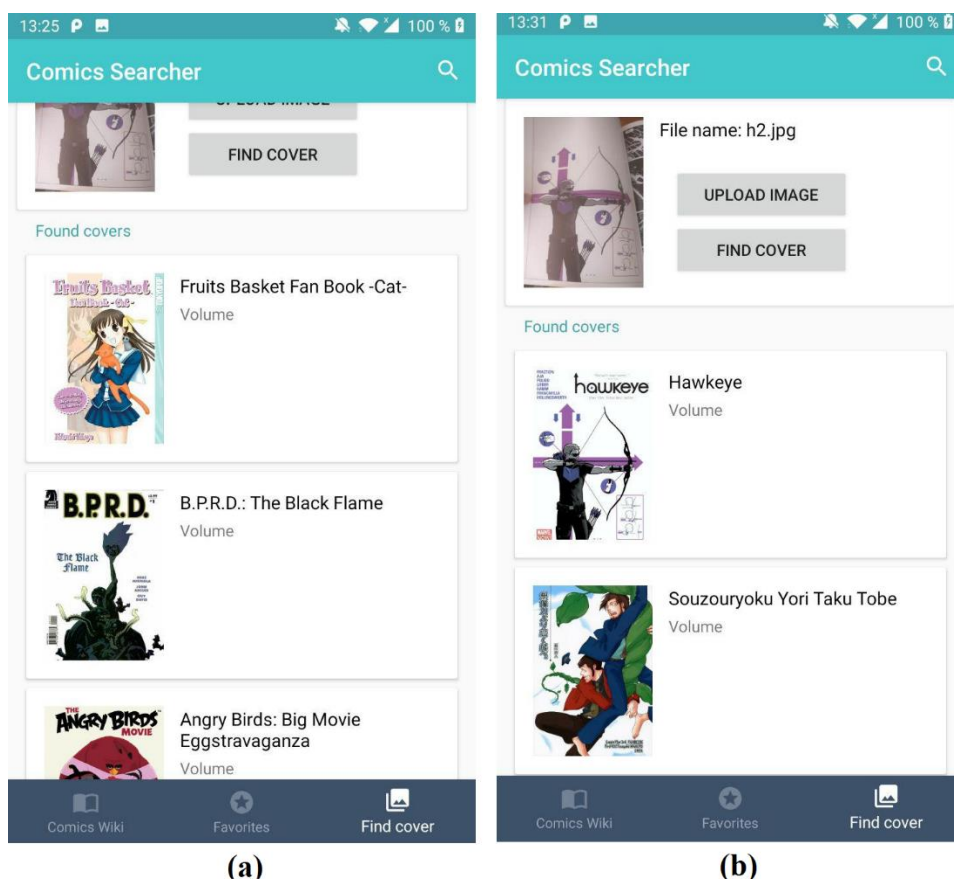


Рисунок 35. Вторая обложка-запрос. (a) Результат поиска для первого словаря.
(b) Результат поиска для второго словаря.

Для последующих двух загруженных изображений (рис.36) результаты поиска для первого и второго словаря оказались идентичными. Загруженные изображения - фотографии обложек одной серии комиксов, первый и второй

том соответственно (рис.36). В коллекции изображений была обложка только второго тома - рисунок 36 (b), но и для первого изображения - рисунок 36 (a), алгоритм опознал ее как идентичную. Стоит отметить, что второй найденный комикс для этой обложки похож цветовой гаммой и расположением этих цветов на изображении.

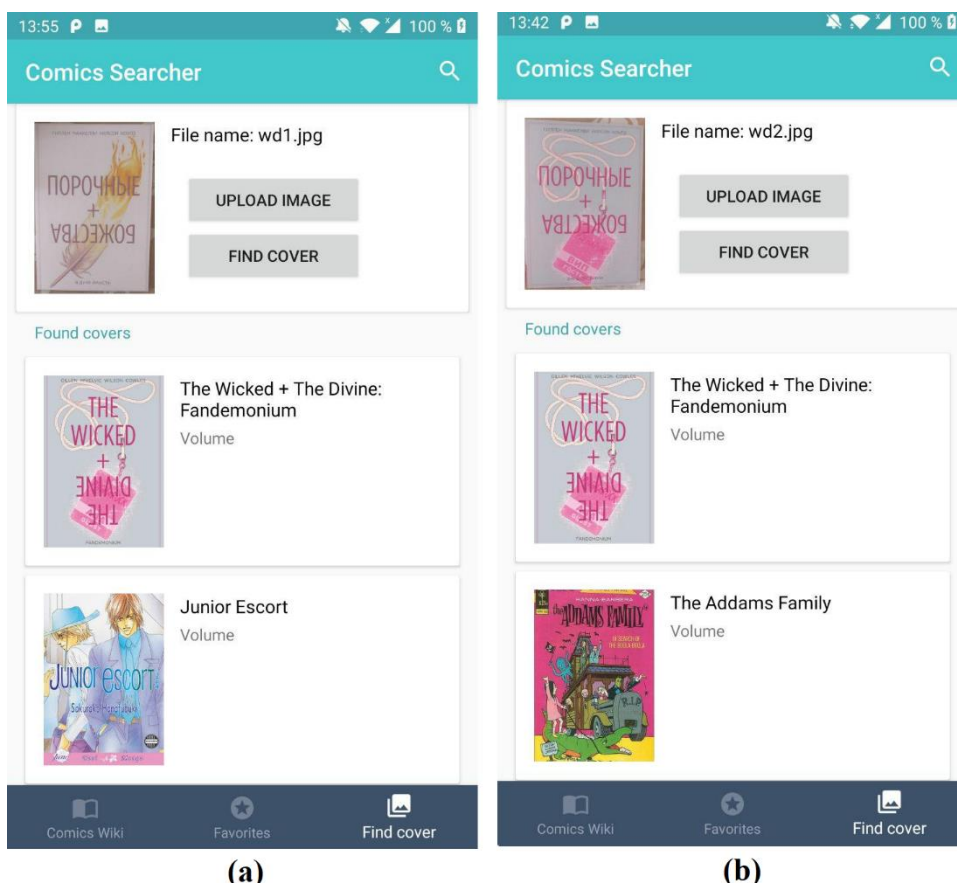


Рисунок 36. Результаты выдачи для (a) первой и (b) второй обложки-запроса.

Последнее изображение-запрос дало неоднозначные результаты поисковой выдачи. Поиск, построенный на первом словаре (200 изображений), отработал лучше, чем поиск на втором словаре. На рисунке 37 представлены найденные результаты для первого словаря. Ожидаемое изображение обложки появилось только на 10 позиции выдачи. Перед ней – обложка комикса, с персонажем дважды присутствующем на изображении запросе. На первой и второй позиции списка результатов - рисунок 37(a), обложки схожей цветовой гаммы. Также можно заметить, что у персонажей на этих обложках есть

схожий жест руки. Поиск с использованием второго словаря смог выдать только обложку на 9 позиции (рис.37). Ожидаемая обложка найдена не была.

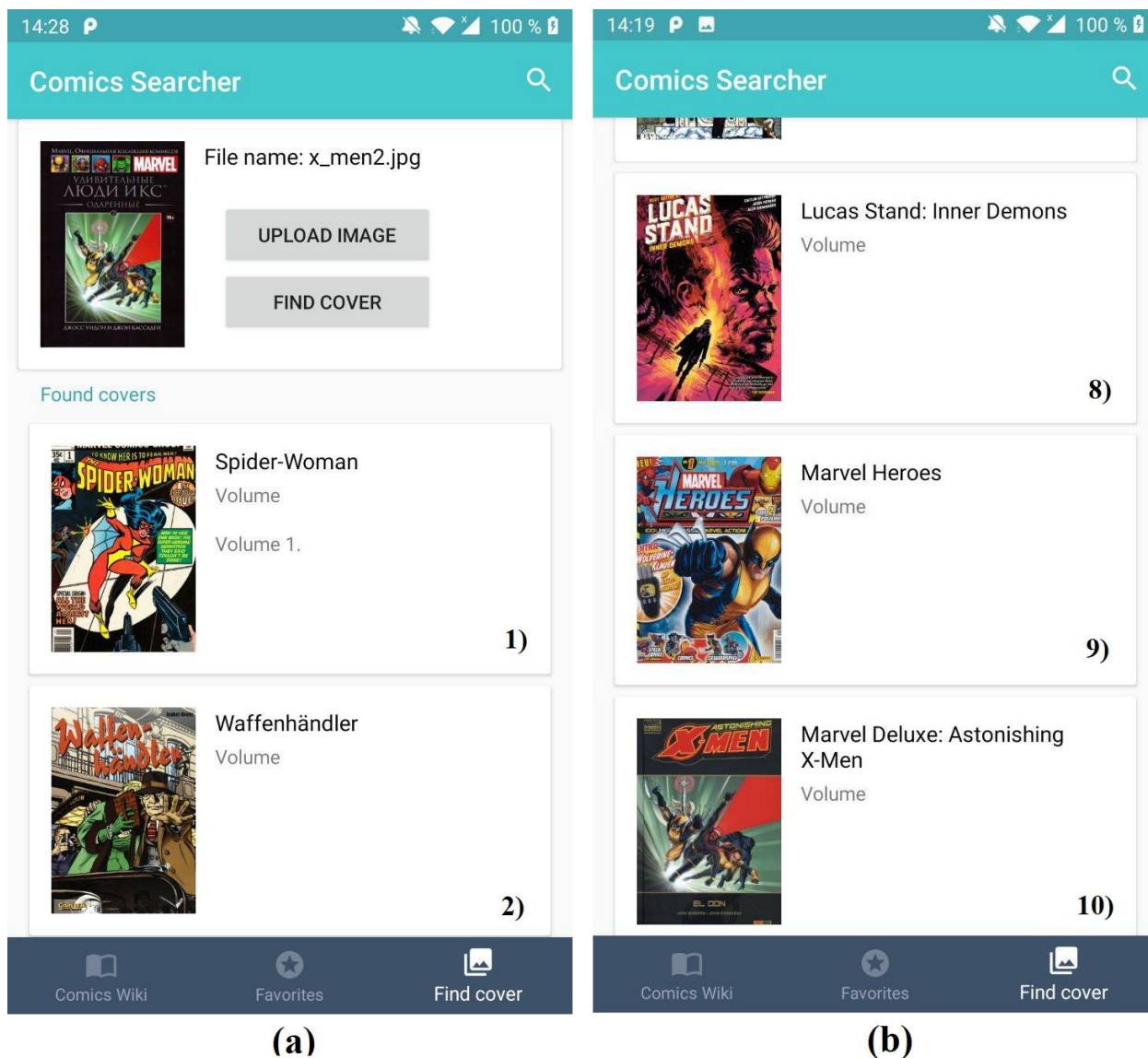


Рисунок 37. Результаты выдачи для последнего изображения-запроса. (а) Обложки, находящиеся на позиции 1) и 2) со схожим жестом руки и общими цветами. (б) Ожидаемая обложка на позиции 10), на комиксе 9) изображен персонаж как на обложке-запросе.

ЗАКЛЮЧЕНИЕ

В процессе выполнения данной работы была исследована широкая теоретическая база в области поиска изображений по образцу в коллекции других изображений. Были изучены и протестированы несколько алгоритмов компьютерного зрения, а также реализован и протестирован алгоритм «мешок визуальных слов». Был разработан мобильный клиент для поиска информации по комикс-ресурсам, для которого также была предпринята попытка реализовать свой поиск ресурса по изображению.

Для достижения поставленной цели было изучено предоставляемое API сайта Comics Vine. Была проведена классификация основных критериев схожести изображений и описан каждый из них. Была подробно рассмотрена архитектура классической CBIR-системы.

Был проведен анализ и сравнение алгоритмов из области feature matching – поиска особых точек на изображении. Тестирование и сравнение проводилось между алгоритмами ORB и AKAZE с вещественным дескриптором KAZE. Алгоритм ORB оказался устойчив практически ко всем поставленным требованиям, однако плохо справлялся с поиском обложек, где был сильно видоизменен фон. Алгоритм AKAZE с вещественным дескриптором KAZE показал хорошие результаты при выявлении похожих изображений и был устойчив ко всем предъявленным требованиям. В ходе тестирования алгоритмов выявилась следующая особенность – зависимость результата сопоставления особых точек от разрешения сравниваемых изображений. Если одно из них превышает разрешение другого в несколько раз, могут быть «ложные сопоставления».

Был реализован алгоритм «мешок визуальных слов», где для построения визуального словаря использовались: алгоритм для поиска особых точек AKAZE с вещественным дескриптором KAZE, кластеризация k-means, метрика TF-IDF для нормализации векторов изображений. В результате были получены неплохие результаты по сопоставлению изображений на небольшой

коллекции обложек. На основе полученных результатов алгоритма мешка визуальных слов было решено использовать его для реализации поиска в мобильном приложении.

Была спроектирована архитектура разрабатываемой системы поиска, описаны процессы построения словаря «визуальных слов» и поиска изображения по образцу. Разработано мобильное приложение для визуализации и отображения загружаемого контента с помощью API сайта Comics Vine, реализующего весь требуемый функционал. Для реализации поиска по изображению был поднят веб-сервис с развёрнутым API и базой данных. Был разработан модуль для формирования словаря визуальных слов по коллекции изображений, модуль для загрузки и преобразования изображений к векторному виду с помощью алгоритма AKAZE с дескриптором KAZE и словаря визуальных слов. Был разработан модуль для поиска и сравнения изображений.

Для тестирования разработанного поиска по изображению, было сформировано два словаря, размером 1100 слов, построенного на коллекции изображений размером в 200 и 1000 обложек. Увеличить размер коллекций для словаря и количество слов не удалось, ввиду отсутствия вычислительной мощностей. По этой причине пришлось сократить и размер загруженной коллекции изображений, по которой осуществляется поиск. Представленные результаты поиска демонстрируют, что алгоритм хорошо справляется с обложками дубликатами, отличающихся надписями, шрифтом, искажением объектов и плохой освещенностью. Однако обилие других объектов на обложке-запросе, может сильно сказаться на полученном результате. По этой причине необходимо строить словарь с большим количеством слов в словаре, чем изображений в коллекции для формирования словаря.

Рассмотренная реализация поиска по изображению остается актуальной задачей и по сей день и нуждается в проведении отдельного исследования для поиска методов оптимизации алгоритма «мешок визуальных слов».

СПИСОК ЛИТЕРАТУРЫ

1. Кирпичников А. П., Ляшева С. А., Шлеймович М. П. Контекстный поиск изображений // Вестник Казанского технологического университета. 2014. №18.
2. Васильева Н. С. Методы поиска изображений по содержанию // Программирование. – 2009. – Т. 35. – №. 3. – С. 51-80.
3. Васильева Н. С., Новиков Б. А. Построение соответствий между низкоуровневыми характеристиками и семантикой статических изображений // Труды седьмой всероссийской конференции RCDL. – 2005. – С. 236-240.
4. Злотников Т. Система поиска изображений по содержанию // Компоненты и технологии. – 2012. – №. 1. – С. 58-59.
5. Bay H., Tuytelaars T., Van Gool L. Surf: Speeded up robust features // European conference on computer vision. – Springer, Berlin, Heidelberg, 2006. – С. 404-417.
6. Гончаренко М. О. Сравнительный анализ методов формирования дескрипторов изображений в контексте задачи сегментации видеопотока. – 2015.
7. Венцов Н. Н., Долгов В. В., Подколзина Л. А. Обзор алгоритмов кластеризации, используемых в задачах поиска изображений по содержанию // Инженерный вестник Дона. – 2016. – Т. 42. – №. 3 (42).
8. A Sketch Retrieval Method for Full Color Image Database. Kato, T., Kurita, T., Otsu, N., Hirata, K. Query by Visual Example, Proc. of Int. Conf. on Pattern Recognition, 1992. pp. 530-533.
9. The QBIC Project: Querying Images by Content, Using Color, Texture, and Shape / Wayne Niblack, Ron Barber, William Equitz et al. // Storage and Retrieval for Image and Video Databases, San Jose, CA, USA, January 31 - February 5, 1993. — 1993. — Pp. 173–187.

10. Горев А. Ю., Шлеймович М. П., Юдинцева А. О. Контекстный поиск изображений в Web-системах //Вестник Казанского технологического университета. – 2014. – Т. 17. – №. 19.
11. Благовещенский И. А., Демьянков Н. А. Технологии и алгоритмы для создания дополненной реальности //Моделирование и анализ информационных систем. – 2015. – Т. 20. – №. 2. – С. 129-138.
12. Anna Saro Vijendran, S. Vinod Kumar. A New Content Based Image Retrieval System by HOG of Wavelet Sub Bands. International Journal of Signal Processing, Image Processing and Pattern Recognition Vol. 8, No. 4 (2015), pp. 297-306
13. Шапиро Л., Стокман Д. Компьютерное зрение. – БИНОМ. Лаб. знаний, 2006.
14. Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary Bradski: "ORB: an efficient alternative to SIFT or SURF", Computer Vision (ICCV), IEEE International Conference on. IEEE, pp. 2564 – 2571, 2011.
15. X. Yang, K. T. Cheng: "LDB: An ultra-fast feature for scalable augmented reality". In IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR), pp. 49 – 57, 2012.
16. Rosten, Edward, Tom Drummond: "Machine learning for high-speed corner detection", 9th European Conference on Computer Vision (ECCV), pp. 430 – 443, 2006.
17. Michael Calonder, Vincent Lepetit, Christoph Strecha, Pascal Fua, "BRIEF: Binary Robust Independent Elementary Features", 11th European Conference on Computer Vision (ECCV), pp. 778 – 792, 2010.
18. S. Grewenig, J. Weickert, C. Schroers, A. Bruhn: "Cyclic Schemes for PDEBased Image Analysis", In International Journal of Computer Vision, 2013
19. Lowe David G. Object Recognition from Local Scale-Invariant Features // IC-CV. — 1999. — Pp. 1150–1157.
20. Harris, C., Stephens, M.: "A Combined Corner and Edge Detector". Proceedings of the 4th Alvey Vision Conference, pp. 147 – 151, 1988.

21. J. Weickert, H. Scharr.: “A scheme for coherence-enhancing diffusion filtering with optimized rotation invariance”, *Journal of Visual Communication and Image Representation*, pp. 103–118, 2002.
22. Herbert Bay, Tinne Tuytelaars, Luc Van Gool, "SURF: Speeded Up Robust Features". *Proceedings of the ninth European Conference on Computer Vision*, pp. 404 – 417, 2006.
23. Unsupervised Nearest Neighbors [Электронный ресурс] URL: <https://scikit-learn.org/stable/modules/neighbors.html#unsupervised-neighbors> (дата обращения: 11.04.2019).
24. Hamming distance // Wikipedia URL: https://en.wikipedia.org/wiki/Hamming_distance (дата обращения: 11.04.2019).
25. AI Courses by OpenCV // OpenCV URL: <https://opencv.org/> (дата обращения: 03.04.2019).
26. Виндер П. Python для сложных задач: наука о данных и машинное обучение // Спб.: Питер. – 2018. – 576 с.
27. Scikit-learn // Scikit-learn URL: <https://scikit-learn.org/> (дата обращения: 11.04.2019).
28. Alcantarilla P. F., Bartoli A., Davison A. J. KAZE features // *European Conference on Computer Vision*. – Springer, Berlin, Heidelberg, 2012. – С. 214 - 227.
29. R. Datta, D. Joshi, J. Li, and J. Z.Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 2008.
30. Возможности языка и основы синтаксиса // IBM Developer URL: https://www.ibm.com/developerworks/ru/library/l-python_part_1/ (дата обращения: 2.05.2019).
31. Free Comic Database and API // Comics Vne URL: <https://comicvine.gamespot.com/api/documentation> (дата обращения: 15.06.2019).

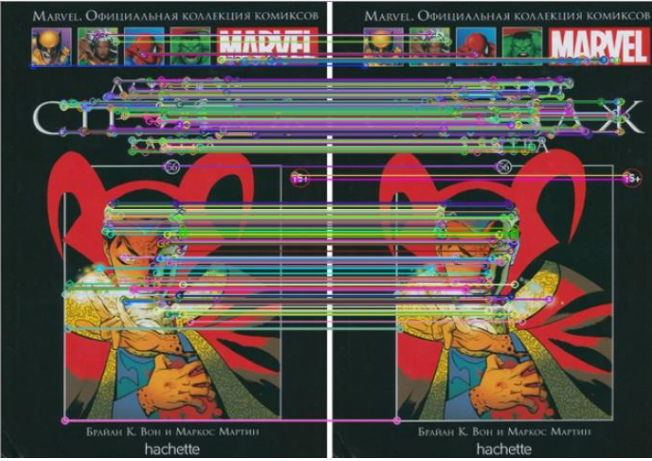
32. Distribution dashboard // Android Developer URL: <https://developer.android.com/about/dashboards> (дата обращения: 4.05.2019).
33. Retrofit // Retrofit URL: <https://square.github.io/retrofit/> (дата обращения: 16.05.2019).
34. Picasso // Picasso URL: <https://square.github.io/picasso/> (дата обращения: 16.05.2019).
35. SQLite // SQLite URL: <https://www.sqlite.org/index.html> (дата обращения: 18.04.2019).
36. Sugar ORM // Sugar ORM URL: <http://satyan.github.io/sugar/> (дата обращения: 18.04.2019).
37. BFMatcher Class Reference // OpenCV URL: https://docs.opencv.org/3.4/d3/da1/classcv_1_1BFMatcher.html (дата обращения: 30.05.2019).
38. AKAZE Class Reference // OpenCV URL: https://docs.opencv.org/4.1.0/d8/d30/classcv_1_1AKAZE.html (дата обращения: 30.05.2019).
39. DescriptorMatcher Class Reference // OpenCV URL: https://docs.opencv.org/4.1.0/db/d39/classcv_1_1DescriptorMatcher.html (дата обращения: 30.05.2019).
40. Lowe D. G. Distinctive image features from scale-invariant keypoints // International journal of computer vision. – 2004. – Т. 60. – №. 2. – С. 91-110.
41. sklearn.preprocessing.normalize // Scikit-learn v0.21.2 URL: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.normalize.html> (дата обращения: 30.05.2019).
42. Brown M., Lowe D. G. Invariant features from interest point groups //BMVC. – 2002. – Т. 4.
43. Salton G., Buckley C. Term-weighting approaches in automatic text retrieval //Information processing & management. – 1988. – Т. 24. – №. 5. – С. 513-523.

44. Add code from a template // Android Developers URL:
<https://developer.android.com/studio/projects/templates#ScrollingActivity>
(дата обращения: 15.06.2019).



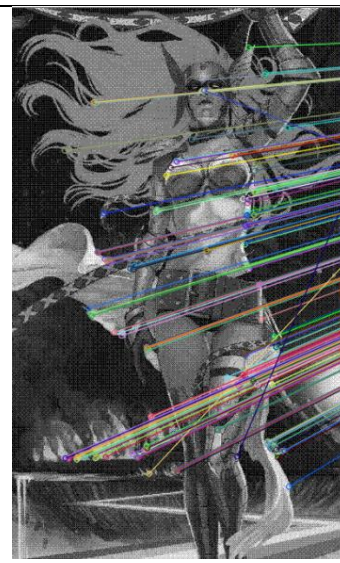
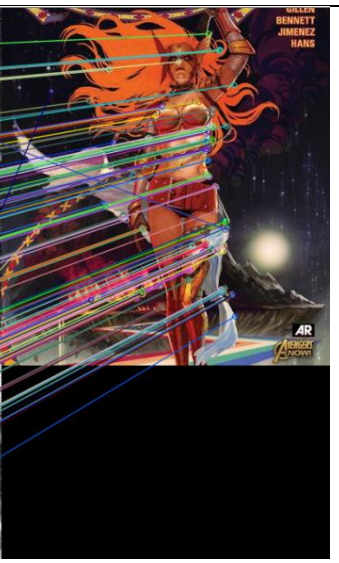


Результаты тестирования алгоритма ORB

Результат сопоставления		Описание
1) Изображение–запрос	2) Ответ	
		<p>1) 400×711px Перспективное искажение и размытость, надписи на другом языке.</p> <p>2) 400×621px Четкие контуры, увеличенный масштаб, дополнительные объекты, текст аналогичного шрифта, но на другом языке. Показатели: matches 178 good 68.</p> <p>Другие изображения: good от 0 до 9.</p>
		<p>1) 400×600px Оригинальная обложка, четкие контуры, яркие контрастные цвета, изменен размер (ранее 1867×2800px).</p> <p>2) 300×463px Дублирует запрос, небольшое отличие в яркости цветов и текстуры, наличие дополнительных объектов. Показатели: matches 224, good 153.</p> <p>Другие изображения: good от 0 до 14.</p>
		<p>1) 400×600px Оригинальная обложка, четкие контуры, яркими контрастные цвета, изменен размер (ранее 1867×2800px).</p> <p>2) 339×499px Дублирует запрос, другая цветовая гамма, больше теней и текстур, изменен текст, изображение перевернуто. Показатели: matches 151, good 46.</p> <p>Другие изображения: good от 0 до 14.</p>


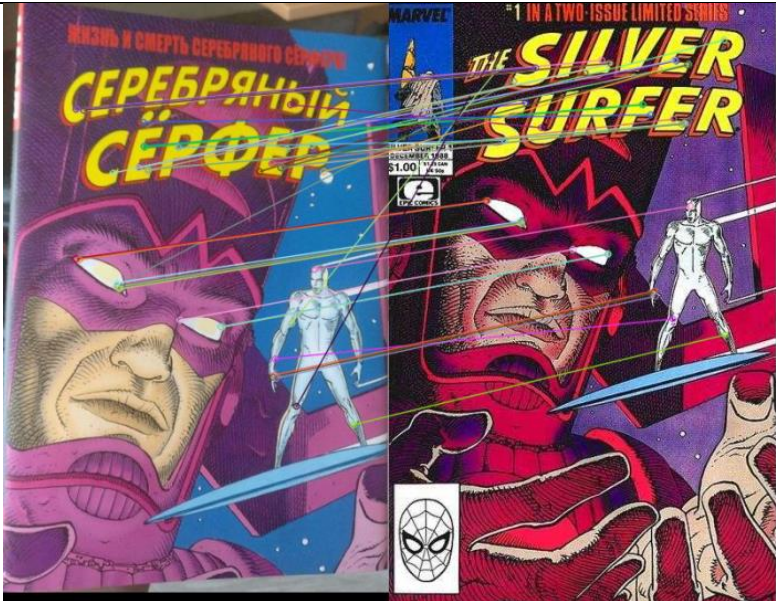
Результаты тестирования алгоритма ORB

Результат сопоставления		Описание
1) Изображение–запрос	2) Ответ	
		<p>1) 319×442px Наличие одинакового изображения главного персонажа.</p> <p>2) 319×442px Дубликат запроса. Показатели: matches 500, good 500.</p> <p>Другие изображения: good от 0 до 10.</p>
		<p>1) 319×442px Другое оформление обложки.</p> <p>2) 225×350px Англоязычная версия оригинальной обложки комикса, другой шрифт, изображение ярче. Показатели: matches 154, good 36.</p> <p>Другие изображения: good от 0 до 10.</p>
		<p>1) 319×442px Другое оформление обложки.</p> <p>2) 400×601px Русскоязычная версия оригинальной обложки комикса, другой шрифт, изображение ярче, но размыто. Показатели: matches 138, good 16.</p> <p>Другие изображения: good от 0 до 10.</p>

Результаты тестирования алгоритма ORB





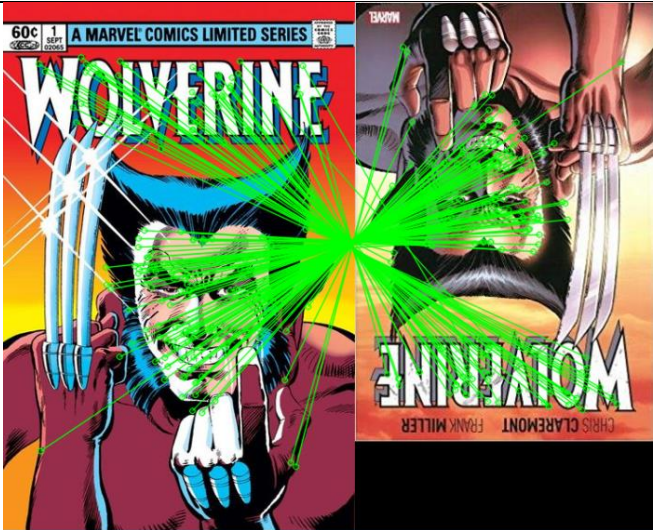
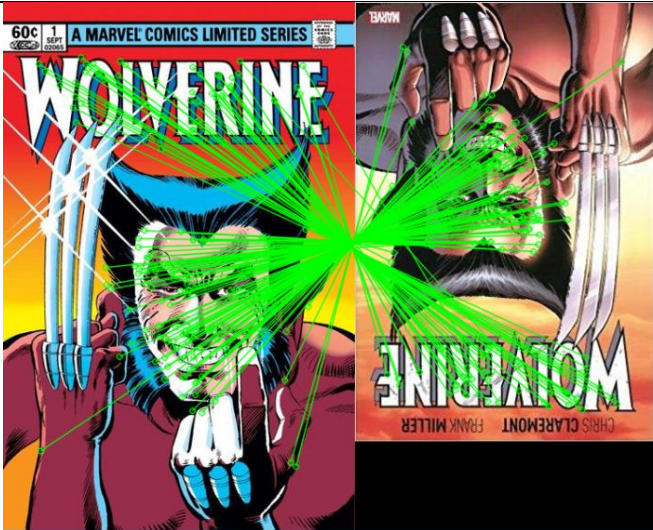
Результат сопоставления		Описание
1) Изображение-запрос	2) Ответ	
		<p>1) 400×670px Вырезанный фрагмент из оригинальной обложки, изображение преобразовано в чёрно-белое в режиме «мозаика».</p> <p>2) 400×608px Оригинальная обложка комикса, цветное изображение. Показатели: matches 168, good 61.</p> <p>Другие изображения: good от 0 до 13.</p>
		<p>1) 400×670px Вырезанный фрагмент из оригинальной обложки, изображение преобразовано в чёрно-белое в режиме «мозаика».</p> <p>2) 400×434px Исходное неотредактированное изображение. Показатели: matches 255, good 164.</p> <p>Другие изображения: good от 0 до 13.</p>
		<p>1) 400×670px Вырезанный фрагмент из оригинальной обложки, изображение преобразовано в чёрно-белое в режиме «мозаика».</p> <p>2) 493×640px Цветное изображение, изменённый фон, много других объектов на изображении. Показатели: matches 152, good 2.</p> <p>Другие изображения: good от 0 до 13.</p>

Результаты тестирования алгоритма ORB

Результат сопоставления		Описание
1) Изображение-запрос	2) Ответ	
		<p>1) 400×611px Фотография обложки переведённого комикса на русский язык. Есть размытость и небольшое искажение обложки.</p> <p>2) 500× 756px Изображение обложки с фотографии-запроса, только без искажения и размытия, с четкими контурами. Показатели: matches 242, good 171.</p> <p>Другие изображения: good от 0 до 24.</p>
		<p>1) 400×611px Фотография обложки переведённого комикса на русский язык. Есть размытость и небольшое искажение обложки.</p> <p>2) 400×623px Оригинальная обложка комикса, имеет другую цветовую гамму, расширенная сцена, имеются дополнительные объекты. Показатели: matches 63, good 46.</p> <p>Другие изображения: good от 0 до 24.</p>

ПРИЛОЖЕНИЕ 2

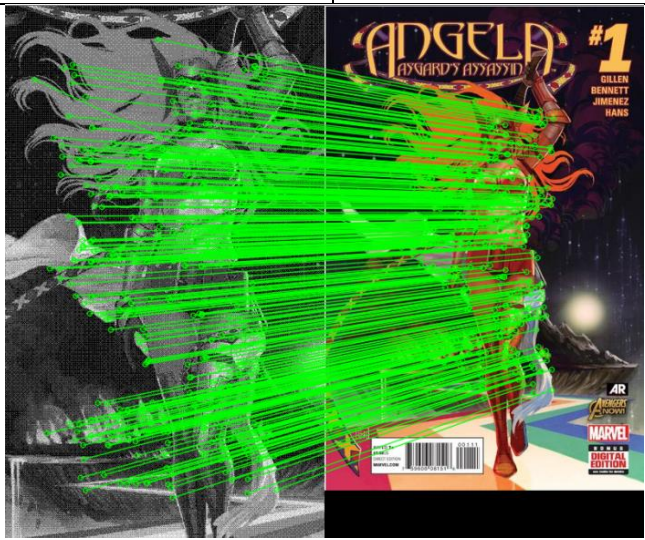
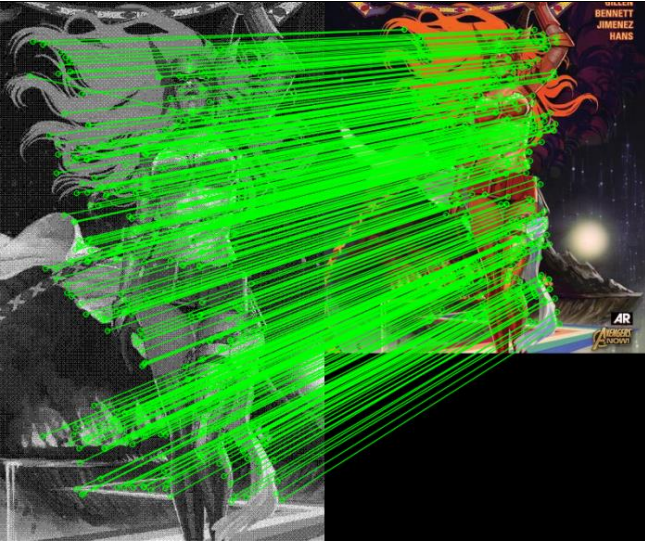

Результаты тестирования алгоритма АКАZE с дескриптором KAZE

Результат сопоставления		Описание
1) Изображение–запрос	2) Ответ	
		<p>1) 400×711px Перспективное искажение и размытость, надписи на другом языке.</p> <p>2) 400×621px Четкие контуры, увеличенный масштаб, дополнительные объекты, текст аналогичного шрифта, но на другом языке. Показатели: matches 1355, good 410.</p> <p>Другие изображения: good от 0 до 7.</p>
		<p>1) 400×600px Оригинальная обложка, четкие контуры, яркие контрастные цвета, изменен размер (ранее 1867×2800px).</p> <p>2) 300×463px Дублирует запрос, небольшое отличие в яркости цветов и текстуры, наличие дополнительных объектов. Показатели: matches 1691, good 585.</p> <p>Другие изображения: good от 0 до 20.</p>
		<p>1) 400×600px Оригинальная обложка, четкие контуры, яркими контрастные цвета, изменен размер (ранее 1867×2800px).</p> <p>2) 339×499px Дублирует запрос, другая цветовая гамма, больше теней и текстур, изменен текст, изображение перевернуто. Показатели: matches 1691, good 168.</p> <p>Другие изображения: good от 0 до 20</p>

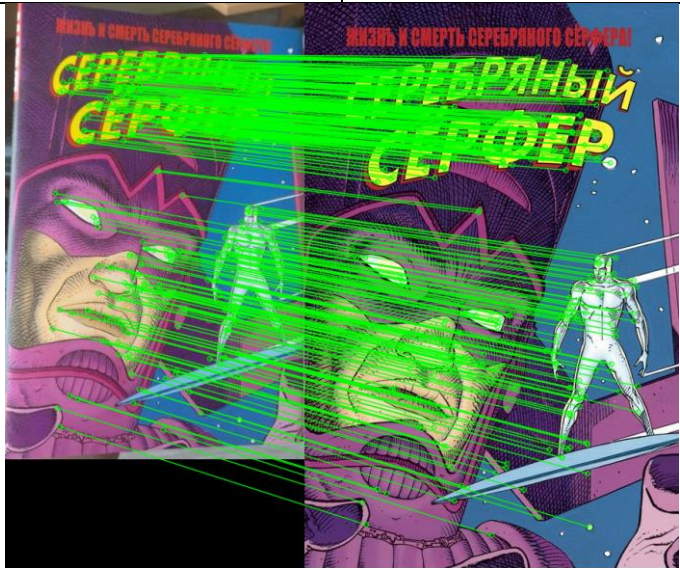

Результаты тестирования алгоритма АКАZE с дескриптором KAZE

Результат сопоставления		Описание
1) Изображение-запрос	2) Ответ	
		<p>1) 319×442px Наличие одинакового изображения главного персонажа.</p> <p>2) 319×442px Дубликат запроса. Показатели: matches 529, good 529.</p> <p>Другие изображения: good от 0 до 5.</p>
		<p>1) 319×442px Другое оформление обложки.</p> <p>2) 225×350px Англоязычная версия оригинальной обложки комикса, другой шрифт, изображение ярче. Показатели: matches 529, good 92.</p> <p>Другие изображения: good от 0 до 5.</p>
		<p>1) 319×442px Другое оформление обложки.</p> <p>2) 400×601px Русскоязычная версия оригинальной обложки комикса, другой шрифт, изображение ярче, но размыто. Показатели: matches 529, good 95.</p> <p>Другие изображения: good от 0 до 5.</p>

Результаты тестирования алгоритма АКАZE с дескриптором KAZE

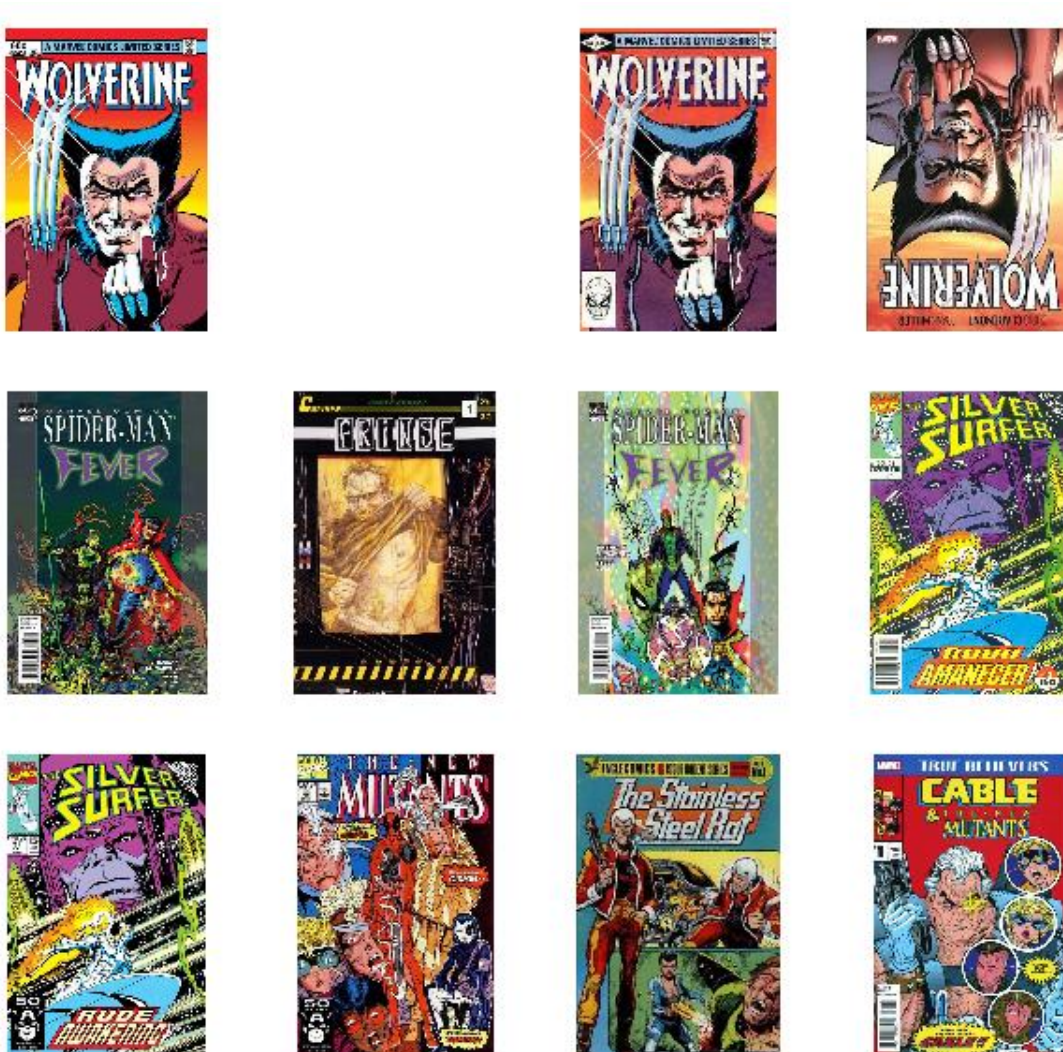
Результат сопоставления		Описание
1) Изображение–запрос	2) Ответ	
		<p>1) 400×670px Вырезанный фрагмент из оригинальной обложки, изображение преобразовано в чёрно-белое в режиме «мозаика».</p> <p>2) 400×608px Оригинальная обложка комикса, цветное изображение. Показатели: matches 1219, good 402.</p> <p>Другие изображения: good от 0 до 7.</p>
		<p>1) 400×670px Вырезанный фрагмент из оригинальной обложки, изображение преобразовано в чёрно-белое в режиме «мозаика».</p> <p>2) 400×434px Исходное неотредактированное изображение. Показатели: matches 1219, good 414.</p> <p>Другие изображения: good от 0 до 7.</p>
		<p>1) 400×670px Вырезанный фрагмент из оригинальной обложки, изображение преобразовано в чёрно-белое в режиме «мозаика».</p> <p>2) 493×640px Цветное изображение, изменённый фон, много других объектов на изображении. Показатели: matches 1219, good 140.</p> <p>Другие изображения: good от 0 до 7.</p>

Результаты тестирования алгоритма АКАZE с дескриптором KAZE

Результат сопоставления		Описание
1) Изображение–запрос	2) Ответ	
		<p>1) 400×611px Фотография обложки переведённого комикса на русский язык. Есть размытость и небольшое искажение обложки.</p> <p>2) 500× 756px Изображение обложки с фотографии-запроса, только без искажения и размытия, с четкими контурами. Показатели: matches 670, good 366.</p> <p>Другие изображения: good от 0 до 7.</p>
		<p>1) 400×611px Фотография обложки переведённого комикса на русский язык. Есть размытость и небольшое искажение обложки.</p> <p>2) 400×623px Оригинальная обложка комикса, имеет другую цветовую гамму, расширенная сцена, имеются дополнительные объекты. Показатели: matches 670, good 77.</p> <p>Другие изображения: good от 0 до 7.</p>

ПРИЛОЖЕНИЕ 3

Результаты тестирования алгоритма «Мешок визуальных слов»



Результат алгоритма BOVW для второго изображения-запроса.

Результаты тестирования алгоритма «Мешок визуальных слов»



Результат алгоритма BOVW для четвертого изображения-запроса.