МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

02.03.03 МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ И АДМИНИСТРИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

БАКАЛАВРСКАЯ РАБОТА

на тему
«Сегментация изображений с помощью алгоритмов кластеризации»

Студент А.О. Авдеев _____

Руководитель В.С. Климов _____

Консультант по аннотации А.В. Кириллова _____

Допустить к защите
Заведующий кафедрой к.т.н. А.В. Очеповский _____

«_____» _____ 2017 г.

Тольятти 2017

АННОТАЦИЯ

Тема данной выпускной квалификационной работы: Сегментация изображений с помощью алгоритмов кластеризации.

Работа посвящена исследованию проблемы автоматической сегментации изображений с помощью алгоритмов кластеризации.

Объектом исследования является автоматизация процессов сегментации объектов на изображениях. Предметом исследования является фильтрация изображений методом свертки и последующая их сегментация с помощью алгоритма кластеризации.

Целью работы является – повышение автоматизации и точности кадрирования объектов на изображениях за счет оптимизации алгоритма сегментации путем фильтрации изображений методом свертки.

Во введение дается краткая характеристика исследования.

В первой главе проводится анализ состояния вопроса. Рассматриваются существующие программные решения для автоматизации процесса кадрирования изображений, проводится анализ недостатков существующих программных решений и предлагается методы по их решению.

Во второй главе проводится анализ и проектирование программного обеспечения для сегментации изображений с применение алгоритма свертки.

В третьей главе проводится программная реализация предложенных решений. Приводится алгоритм работы с изготовленным, в рамках бакалаврской работы, программным продуктом.

В заключение приводятся выводы по работе.

Затем приведен список используемой литературы, в котором присутствуют отечественные и зарубежные книги.

Выпускная квалификационная работа содержит пояснительную записку объемом 45 страниц, включая 24 рисунка, одну таблицу и 7 формул, а также список литературы из 22 источников.

ABSTRACT

The title of graduation project is Segmentation of images using clustering algorithms. This graduation project is about the program which selects and analyzes segments in a digital image. The program should solve problems with finding background segments and cropping the image.

The author dwells on edge detection including a variety of mathematical methods that aim at identifying points in a digital image. The key issue of the graduation project is the development of solutions for the clustering images of different algorithms for example, the k-means algorithm. The work touches upon analysis of edge detection algorithms and clustering methods, description of the clustering method k-means and analysis of existing clustering methods and their runtime.

The graduation project may be divided into several logically connected parts which are: description of the subject area and formalization of requirements for the software product, development of software for automated detection, selection and cropping edges of image objects and software testing.

We give full coverage to the methods of detecting and selecting edges in a image that are represented in the graphical user interface. The GUI can be presented as a portable program version on a PC using the Java language.

The work is of interest for a wide circle of readers. Overall, the results suggest that the program of detecting and cropping objects is useful to automate process of image editing and gives users several methods to choose.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

С ростом количества информации, которая окружает человека, становится все сложнее обрабатывать окружающую нас информацию. Множество изображений, которые делает человек ежесекундно, содержит постороннюю информацию в виде фона или посторонних объектов.

Сегментация или по-другому обработка изображений призвана упростить процесс обработки изображений:

- уменьшить объем изображений;
- упрощение изображения для того, чтобы его было проще и легче анализировать;
- избавить изображения от посторонней информации, путем обозначения главных объектов исходных изображений (рисунок 1).

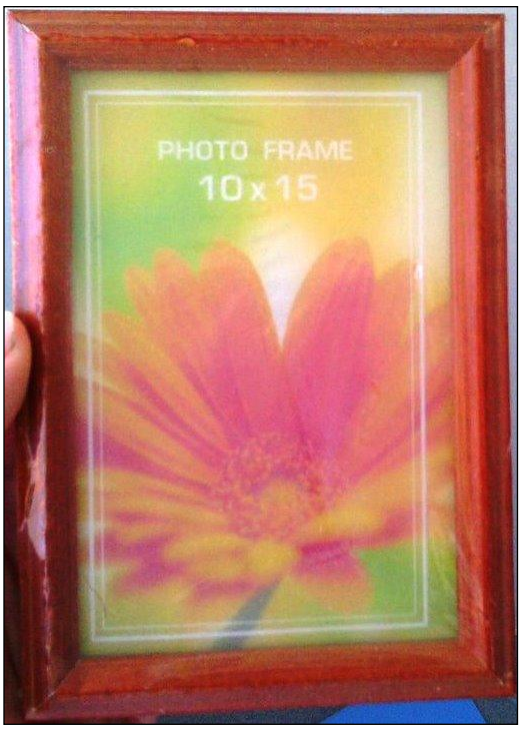


Рисунок 1 – Пример обозначения главного объекта изображения

Сегментация изображений обычно используется для выделения объектов и границ этих объектов на изображениях. Для человека процесс разделение объектов на изображениях является несложной задачей, однако в компьютерном зрении данная задача является непростой. Для обработки

изображений алгоритмы кластеризации могут применять сравнение пикселей изображения, яркости или текстуру фона.

Процесс сегментации изображения можно разделить на анализ низкого уровня и анализ высокого уровня. Отделение объектов находящихся на изображении от фона является процессом анализа высокого уровня сегментации (рисунок 1).

Рисунок 2 – Пример кадрирования объекта изображения

На рисунке 1, фото справа, видно, что сегментация прошла успешно только в верхней области, а в правой области никаких изменений не произошло. Основная сложность обработки изображений заключается в том, что изображение не равномерно освещено, поэтому алгоритм практически не отличает фон от объекта.

Новизна работы заключатся в оптимизации работы алгоритма кластеризации при кадрировании изображений путем применения предварительной фильтрации изображения методом свертки.

Цель работы повышение автоматизации и точности кадрирования изображений, за счет разработки алгоритма основанного на свертки.

Объектом исследования является автоматизация сегментации изображений.

Предметом исследования является увеличение точности сегментации изображений с помощью алгоритмов кластеризации .

1 АНАЛИЗ СОСТОЯНИЯ АЛГОРИТМОВ КЛАСТЕРИЗАЦИИ ИЗОБРАЖЕНИЙ

1.1 Обзор предметной области

На сегодняшний день в технике нет почти ни одной области, которую так или иначе не затрагивала бы цифровая обработка изображений. С ростом производительности процессоров и увеличением оперативной памяти обработка изображений становится, все более, доступна конечным пользователям. С ростом количества текстовой и графической информации приходится тратиться все больше и больше времени на ее обработку и хранение. Например, при обработке изображений, часто возникает потребность в отделении объекта изображения от посторонних объектов, пусть то будет фон или другие объекты.

Задачу по кластеризации изображений можно отнести к машинному зрению, обработке изображений или анализу изображений. Все эти направления тесно пересекаются и зачастую имеют общие решения.

Машинное зрение в основном направлено на решение задач промышленности и крупного сектора. Примерами может служить системы по контролю качества изделий на предприятиях или предприятия по изготовлению робототехники.

Анализ изображений и обработка изображений сосредоточены на обработке цифровых изображений, преобразование одного изображения в другое. В качестве примера операций можно привести:

- операции по кадрированию изображений;
- выделению краев объектов на изображениях;
- операцию по поиску объектов на изображениях;
- операции по изменению яркости, контрастности, насыщенности и интенсивности изображений;
- операции по повороту и отражению изображений.

Машинное зрение применяется для решения различного рода задач, которые происходят с полным или частичным участием человека.

С ростом вычислительных характеристик процессоров и видеокарт области применения машинного зрения стали только увеличиваться. Из основных областей можно выделить:

- медицинская область (автоматический анализ медицинских снимков, томография, УЗИ);
- системы распознавания рукописного текста;
- применение контроля качества, промышленных роботов в таких сферах как: автомобилестроение, электроника, машиностроение;
- охранные системы (идентификация личности, распознавание и отслеживание движущихся объектов).

1.2 Обзор программных аналогов

Рассмотрим, какое программное обеспечение существует для сегментации изображений. Под сегментацией будет пониматься набор нескольких функций таких как: поиск объекта на изображение, выделение его границ и кадрирование изображения до нужного объекта.

Существуют несколько разновидностей программ для сегментации изображений. Программы, основной функционал которых заключается в кадрирование изображений. А также программы, для которых кадрирование изображений является лишь частью функционала, которые данные программы предоставляют. В программных продуктах подобное действие может называться «auto crop» или «auto image cropping», если программы имеют язык интерфейса отличный от русского. В ходе обзора программных аналогов будут рассматриваться две разновидности программ описанных ранее.

Пользуясь поиском на русском языке, удалось найти первый программный продукт – reaConverter. Данный продукт имеет три версии, из которых только одна бесплатная. Продукт позиционирует себя как

многофункциональный инструмент для работы с изображениями, основной особенностью которого является поддержка более 500 форматов изображений и их конвертирование в самые популярные форматы: «JPEG», «PNG» и так далее. Данный продукт поддерживает кадрирование изображений, поэтому выступает в роли аналога. Пользовательский интерфейс приложения представлен на рисунке 3.

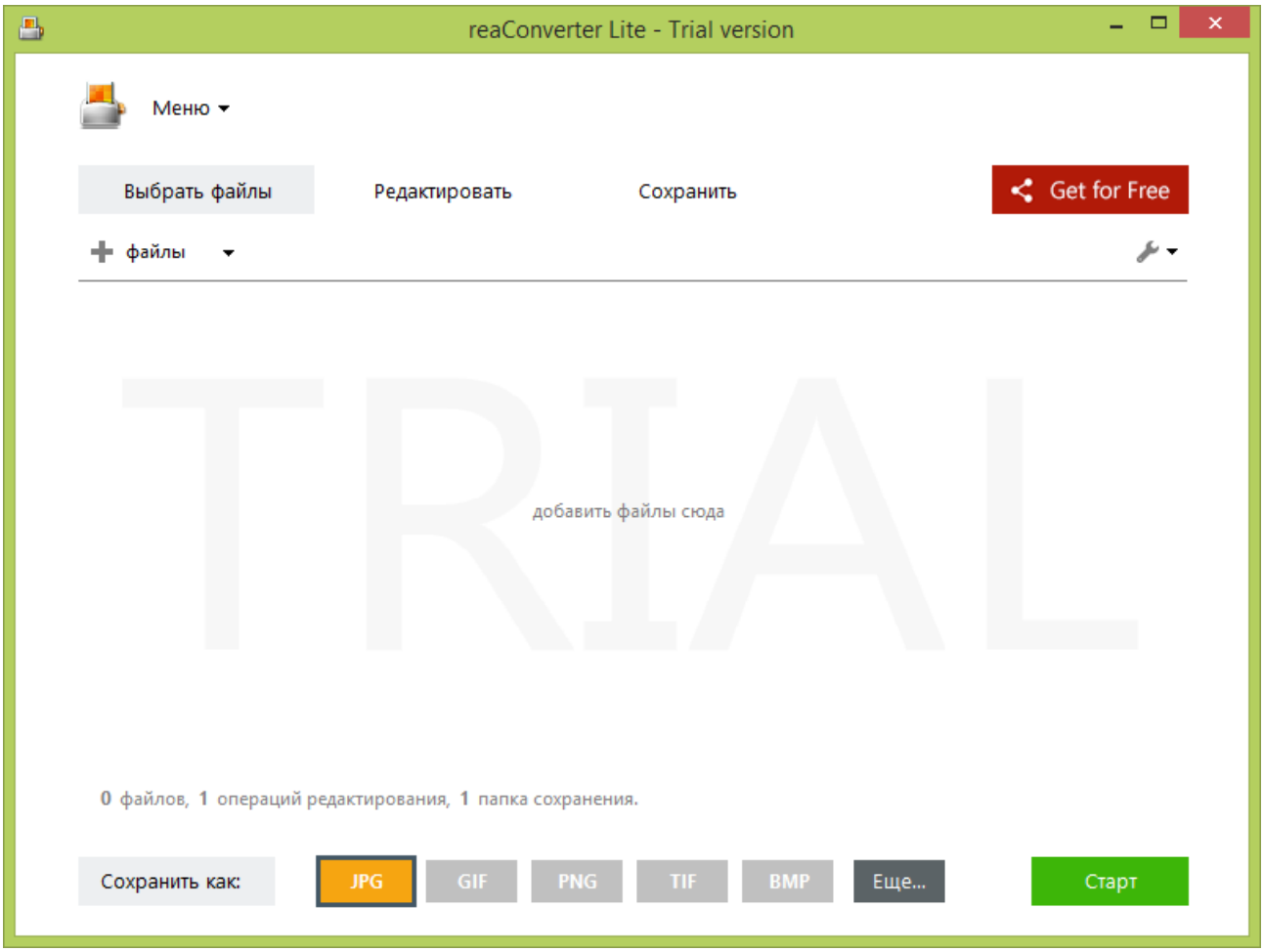


Рисунок 3 – Главная страница приложения «reaConverter»

Интерфейс пользователя достаточно прост и информативен, требуется выбрать исходное изображение, после этого выбрать вкладку «Редактировать» и добавить действие «Автоматическое кадрирование». Стоит отметить, что приложение поддерживает русский язык интерфейса. Окно вкладки «Редактировать» содержит окно для предпросмотра изображения после применения кадрирования и окно с заданием начальной цвета фона пикселя, то есть можно выбрать цвет пикселя или указать его

расположение на исходном изображении. Так же можно задать процент отклонения цвета пикселя фона от изначально выбранного пикселя в соответствии с рисунком 4.

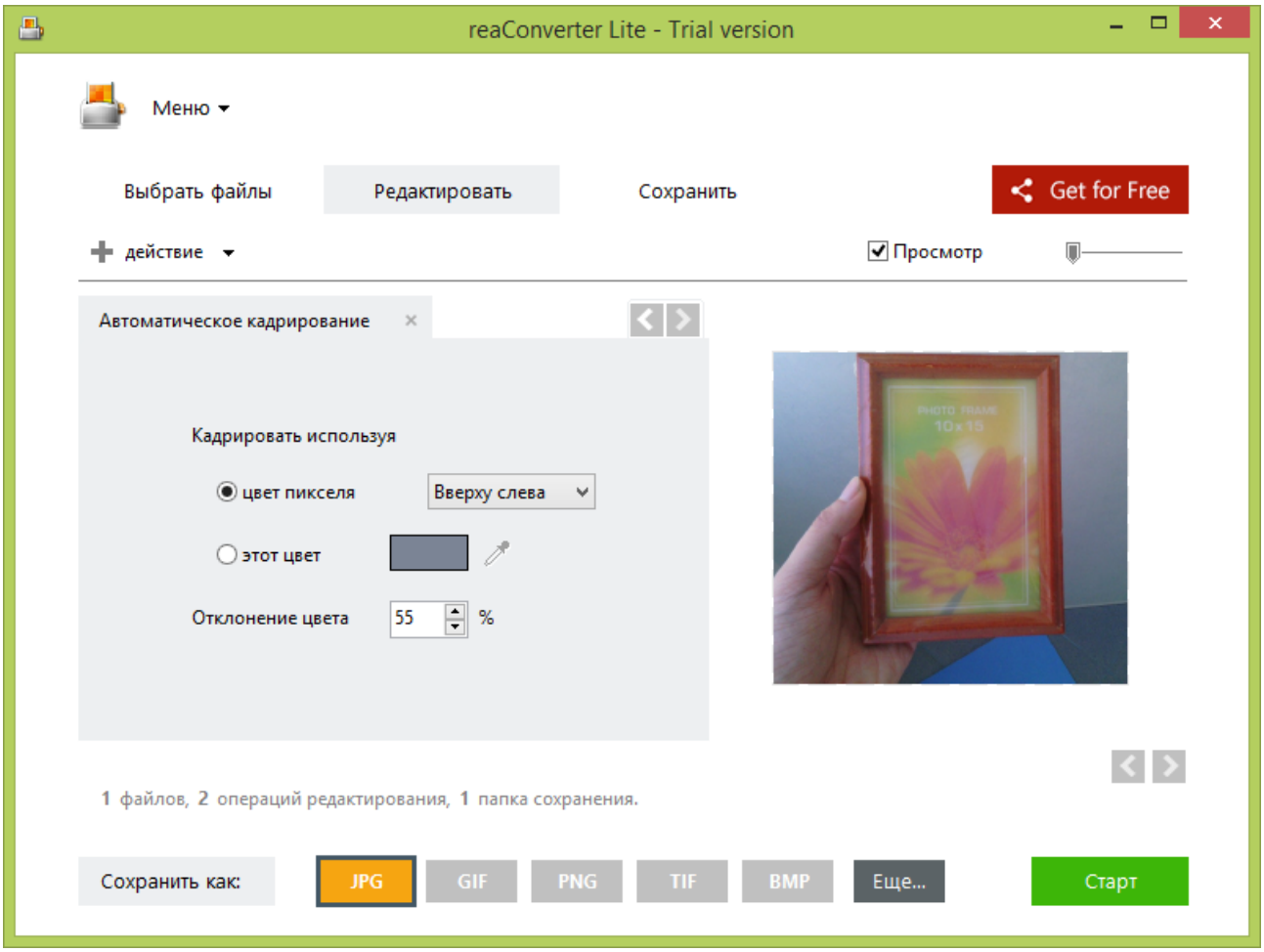


Рисунок 4 – Страница «Редактировать» приложения «reaConverter»

Данный программный продукт является «облегченной» версией стандартного продукта «reaConverter» и для снятия ограничений предлагается зарегистрироваться или купить стандартную версию программы за 50$. Из недостатков можно отметить, что программный продукт поддерживает только операционную систему Windows и Windows Server, что не дает возможности использовать его под операционные системы семейств Linux и MacOS.

Другим программным инструментом для кадрирования изображений является BatchCrop, призванным упростить кадрирование изображений, как говориться на главной странице сайта программы. Язык пользовательского

интерфейса по умолчанию английский и поддержки русского в стандартном пакете нет. Главная страница приложения выглядит, как показано на рисунке 5.

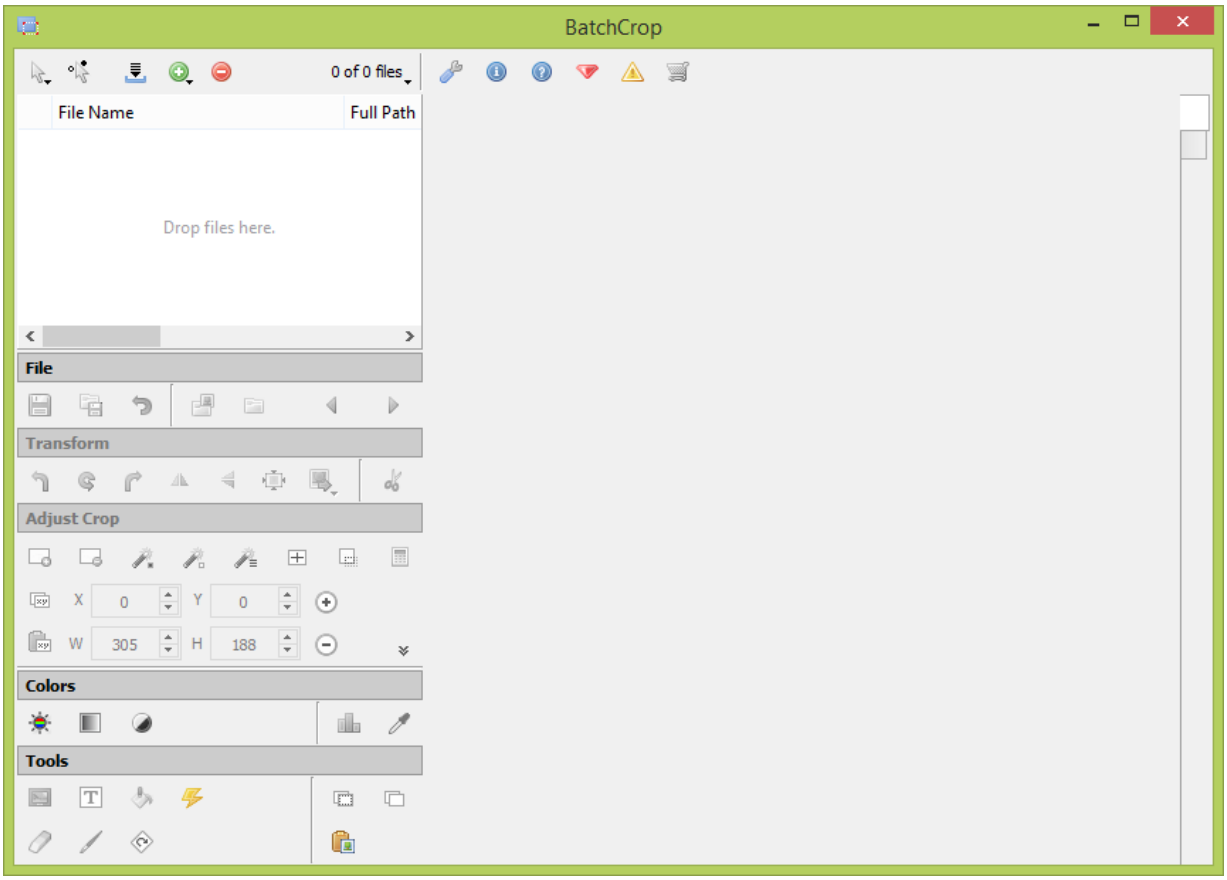


Рисунок 5 – Главная страница приложения «BatchCrop»

Интерфейс пользователя разделен на две области: правая область, которая включает окно для загрузки изображений и кнопки для редактирования изображений и левую область, отвечающую за предпросмотр изображений.

Из особенностей приложения можно отметить что, в шапке меню, перейдя на вкладку с настройками и выбрав пункт «Auto Crop» можно задать отступ от границы объекта, например 30px, при операции кадрирования в соответствие с рисунком 6.

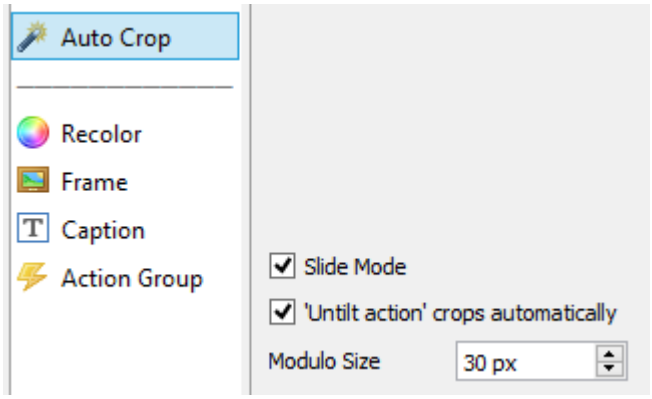


Рисунок 6 – Пример задания отступа от границ объекта

К достоинствам приложения можно отнести поддержку горячих клавиш для популярных действий, также и оформление интерфейса функций в виде иконок, что упрощает работу с приложением.

Из недостатков приложения можно отметить:

- отсутствие поддержки операционных систем Linux;
- зависимость от сторонней библиотеки «Microsoft Visual C++»;
- отсутствие поддержки русского языка интерфейса приложения;

Можно использовать пробную версию приложения без лицензии, однако имеются некоторые ограничения на число одновременное загружаемых файлов, а так же напоминание о том, что используется пробная версия продукта и рекомендуется приобрести лицензию за 49$.

Приложение предлагает три варианта кадрирования: кадрирование по светлым границам, кадрирование по темным границам и кадрирование по тексту или документу. Приложение не смогло автоматически кадрировать объект изображения, если его границы явно не выделяются, что является существенным недостатком в соответствии с рисунком 7.

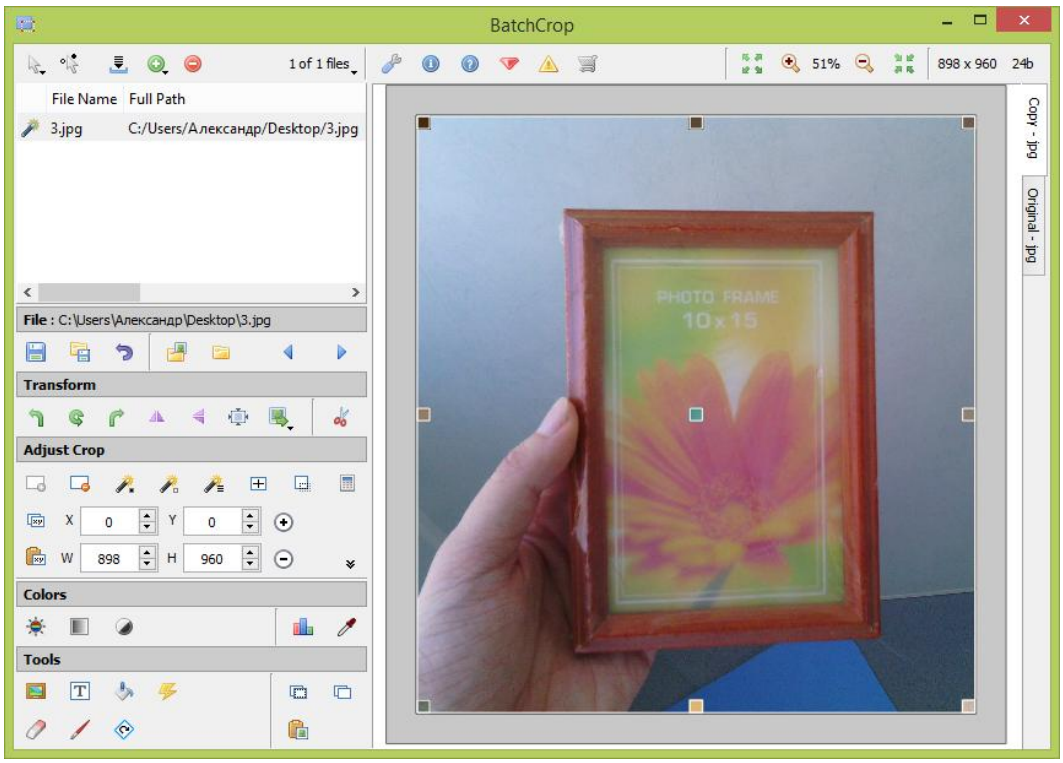


Рисунок 7 – Пример кадрирования изображения приложения «BatchCrop»

Еще одним инструментом для редактирования изображений с функцией автоматического кадрирования является онлайн-редактор Lunapic. Если сравнивать с предыдущими аналогами, то веб-приложение Lunapic работающее в режиме «онлайн», является зависимым от браузера и использования сети интернет. Также приложение позиционирует себя как средство для редактирования фотографий прямо в браузере и поддерживает загрузку изображений по прямому адресу в сети интернет. Главная страница редактора Lunapic представлена на рисунке 8.

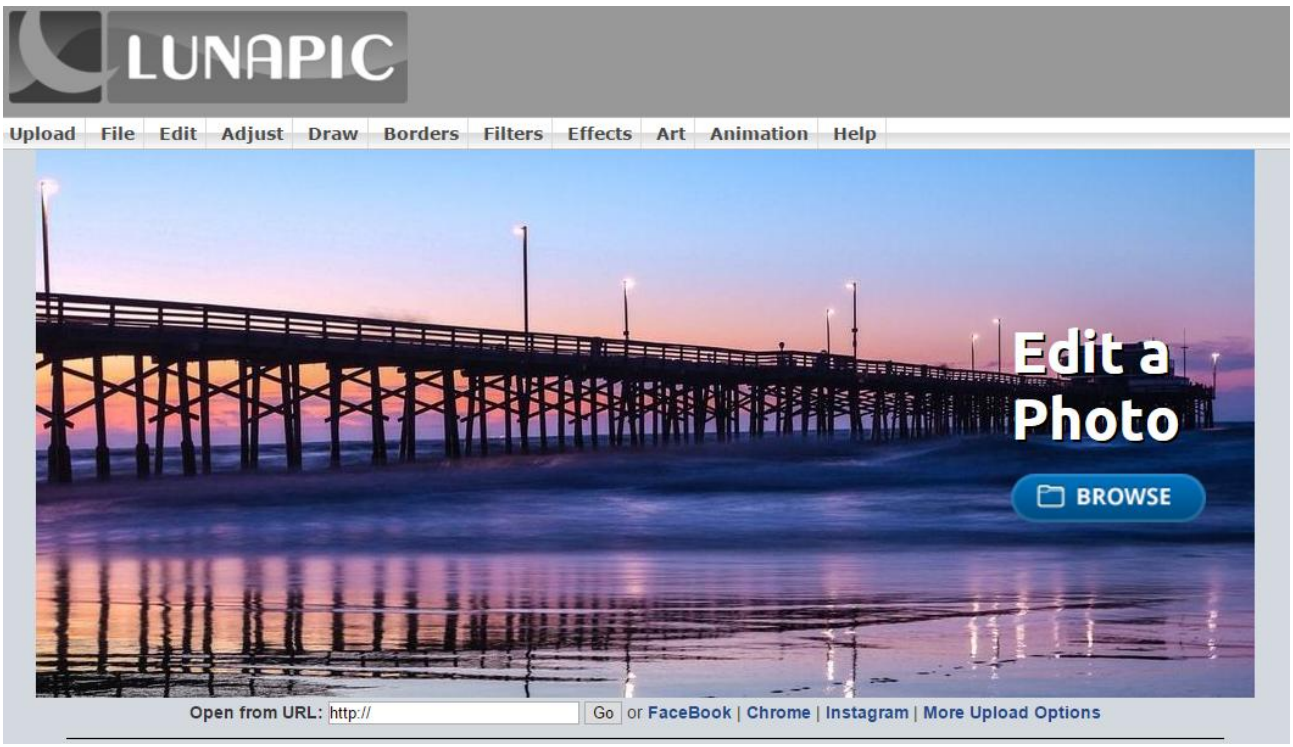


Рисунок 8 – Главная страница онлайн-редактора «Lunapic»

После загрузки исходного изображения (рисунок 9) с компьютера или сети интернет, следует выбрать пункт меню «Edit» и выбрать действие «AutoCrop Image».

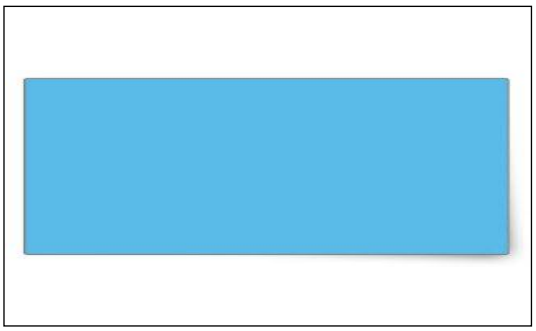


Рисунок 9 – Исходное изображение перед кадрированием

Затем приложение попытается найти и кадрировать объект на изображение, не предлагая никаких дополнительных опций. Результат кадрирования представлен на рисунке 10.

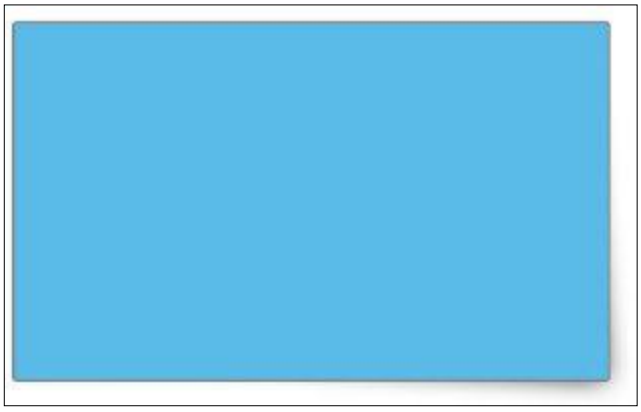


Рисунок 10 – Результат кадрирования изображения в онлайн-редакторе «Lunapic»

В отличие от других аналогов кадрирование в веб-приложение «Lunapic» происходит в автоматическом режиме. Также поддерживается большинство популярных типов изображений, и осуществляется платформонезависимая поддержка приложения, так как работа приложения осуществляется в браузере. Однако существенным недостатком является то, что для работы требуется подключения к сети интернет и невозможность работать оффлайн, используя приложение, установленное на компьютере.

По результатам анализа состояния вопроса была составлена сводная таблица сравнения программных реализаций по сегментации изображений и их кадрированию (таблица 1).

Таблица 1 – Сравнение программных реализаций по кадрированию исходных изображений

Способ кадрирования изображения	**Разработчик**	**Сложность использования**	**Стоимость использования**	**Поддержка ОС**	**Автоматическое кадрирование**
Приложение reaConverter (рисунок 2).	reaConverter LLC (программа reaConverter Lite)	Средняя	49$ - стандартная версия 99$ - профессиональная версия	MS Windows, MS Windows Server	Частично (выбор цвета пикселя)
Приложение BatchCrop (рисунок 4).	Atarca Software	Средняя	Лицензия 49$/приложение	Mac OS X, MS Windows	Частично (выбор способа кадрирования)
Веб-приложение Lunapic (рисунок 7).	-	Низкая	Бесплатно	ОС с поддержкой браузера	Автоматическое

Собственная разработка	ФГБОУ ВО ТГУ, кафедра прикладная математика и информатика	Низкая	Бесплатно	ОС с поддержкой JRE	Автоматическое

Из проведенного анализа состояния вопроса можно сделать следующие выводы:

1) Программы для автоматического кадрирования распространены в основном за границей и практически не поддерживают русского языка интерфейса;

2) Реализация программ нацелена на одну или несколько конкретных платформ и не охватывает все платформы одновременно.

3) Приложения, нацеленные на профессионально использование, имеющие множество функций по редактированию изображений, являются лицензионными и стоят около 50 долларов.

4) Интерфейсы, предоставляемые пользователю, зачастую отличается друг от друга функциональностью и оформлением, что требует дополнительных затрат времени на их анализ и изучение.

5) Кадрирование изображений и определение объектов на изображениях выполняется с большой погрешностью и требует дополнительных ручных средств, в виде обрезки изображения, для окончательного выделения объектов на изображениях

Таким образом, актуальной проблемой, требующей решения можно признать – необходимость разработки алгоритма для более точной сегментации изображений. Алгоритм должен оптимизировать процесс нахождения объектов на изображениях, уменьшить значимость посторонних цифровых шумов на и других незначительных объектов, которые могут попасть в кадр.

Таким образом, целью работы является – повышение автоматизации и точности сегментации изображений за счет разработки алгоритма свертки.

2 АНАЛИЗ И ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ СЕГМЕНТАЦИИ ИЗОБРАЖЕНИЙ С ПРИМЕНЕНИЕМ АЛГОРИТМОВ СВЕРТКИ

2.1 Описание алгоритма свертки

Метода обработки изображений в большинстве случаев являются проблемно ориентированными. Для решений конкретных проблем при обработке изображений применяются разные подходы. Одним из подходов можно считать алгоритм свертки.

По определению свертка – это математическая операция, применяемая к двум функциям, которые порождают третью функцию, которая может рассматриваться как модификация одной из первоначальных функций. Общая формула свертки выглядит следующим образом (1).

$$(f * g)(x) = \int_{R^d} f(y)g(x-y)dy = \int_{R^d} f(x-y)g(y)dy \quad (1)$$

где R^d – пространство, на котором рассматривается свертка.

В общем виде применить формулу пока нельзя, требуется обозначить входные функции как дискретные. Ввиду того, что мы рассматриваем алгоритм свертки применимо к изображениям, то следует рассматривать двумерное пространство значений R^2. Тогда дискретная свертка двух входных функций размерами $M * N$ определятся формулой (2).

$$f(x,y) * h(x,y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n)h(x-m,y-n) \quad (2)$$

Знаки минус означают, что функция зеркально отражается относительно начала координат. Если обозначить $F(u,v)$ и $H(u,v)$ за фурье-образы функций $f(x,y)$ и $h(x,y)$ соответственно, то одна половина теоремы о свертки утверждает, что функция $f(x,y) * h(x,y)$ и $F(u,v)H(u,v)$ образуют фурье-пару. Можно заметить, что свертка в частной области приводит к умножению в пространственной области и наоборот (формулы 3-4).

$$f(x,y) * h(x,y) \leftrightarrow F(u,v) * H(u,v) \quad (3)$$

$$f(x,y)h(x,y) \leftrightarrow F(u,v) * H(u,v) \quad (4)$$

Если представить изображений в виде матрицы, а элементы матрицы принять за пиксели изображения, то можно воспользоваться формулой свертки для получения новых значений пикселей. Вторую функцию тогда можно принять за матрицу свертки.

Подходы для улучшения изображений можно разделить на две категории:

- методы обработки в пространственной области (пространственные методы);
- методы обработки в частной области (частные методы).

В пространственных методах фильтрации изображений, матрицу свертки принято называть фильтром, а значения элементов матрицы коэффициентами. Схема фильтрации представлена на рисунке 11.

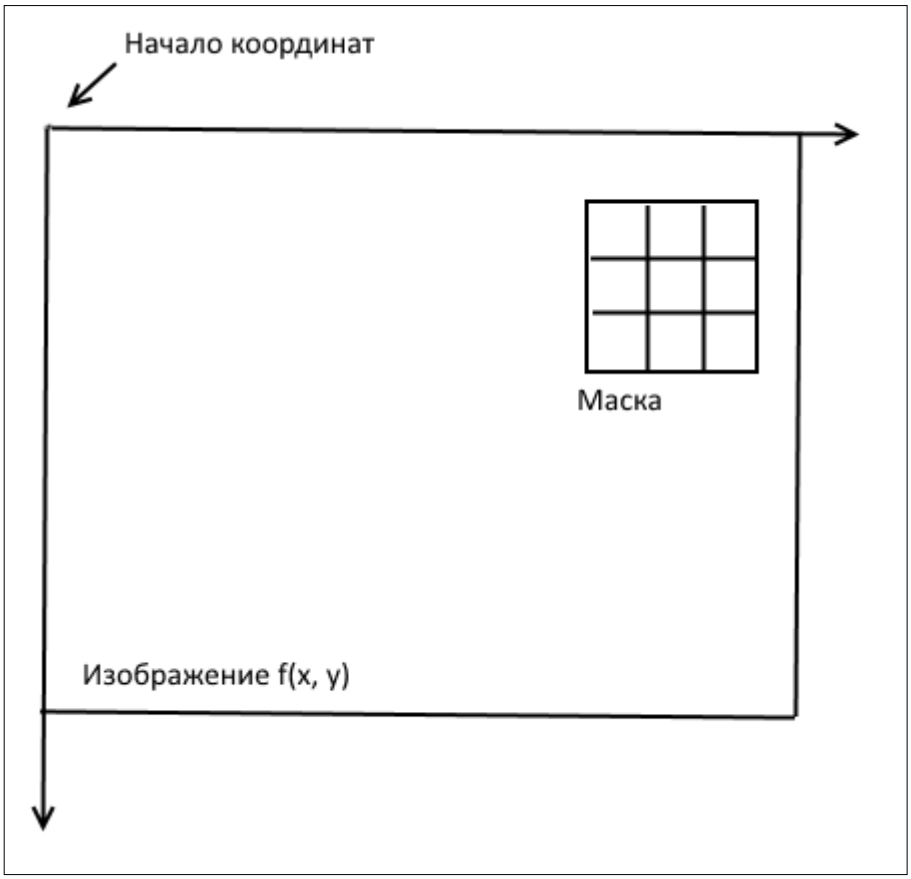


Рисунок 11 – Схема фильтрации изображения

При фильтрации изображения методом «Размытия» изображения, матрицу свертки считают по закону Гауссовского распределения (формулы 5-6):

$$h_g(r,c) = e^{-(r^2 + c^2)/(2\delta^2)} \quad (5)$$

$$h(r,c) = \frac{h_g(r,c)}{\sum_{r=1}^{M} \sum_{c=1}^{N} h_g(r,c)} \quad (6)$$

где M и N – размеры масок; δ – среднеквадратическое отклонение распределения Гаусса.

Фильтр Гаусса относится к низкочастотным фильтрам, поэтому в отличие от усредняющих фильтров он в меньшей степени размывает изображение.

Если воспользоваться формулой свертки и представить произведение матрицы пикселей изображения с матрицей свертки, то получится новое изображение с эффектом «Размытия» (рисунок 12).

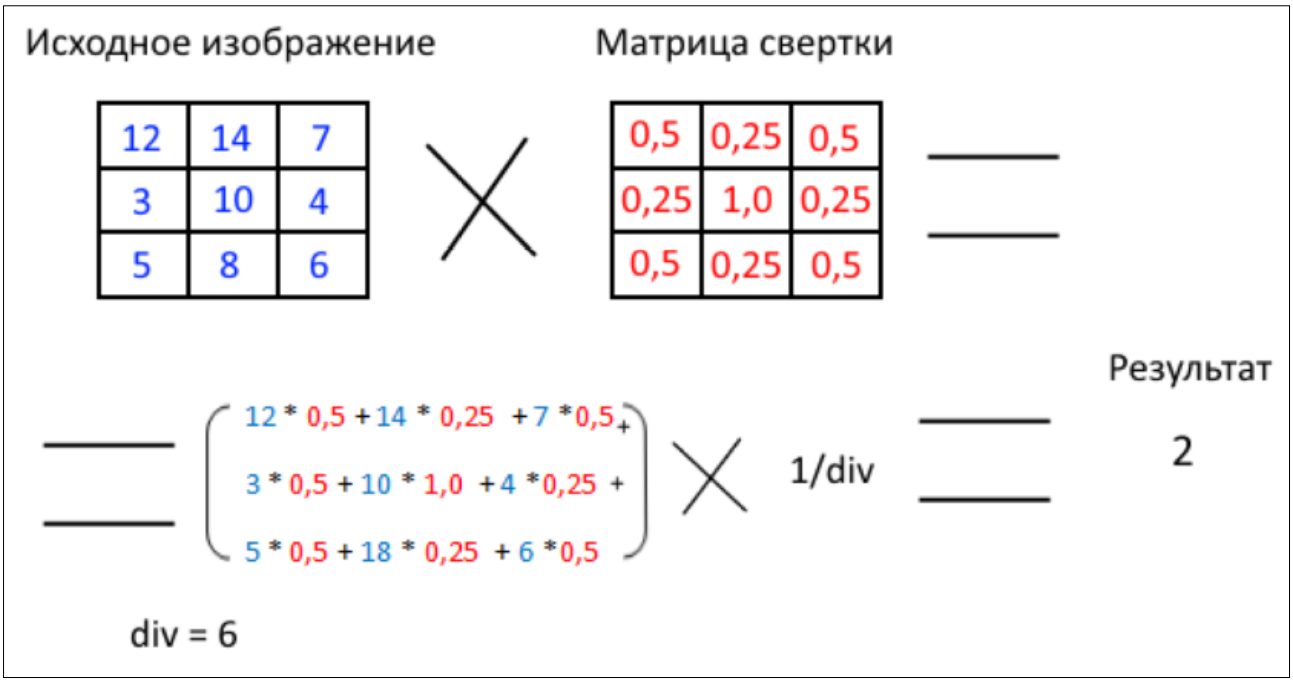


Рисунок 12 – Пример перемножения матрицы свертки и части изображения

Размер матрицы принято брать нечетный. Так же размер свертки напрямую влияет на степень размытости изображения, чем он выше, тем выше степень размытости (рисунок 13).

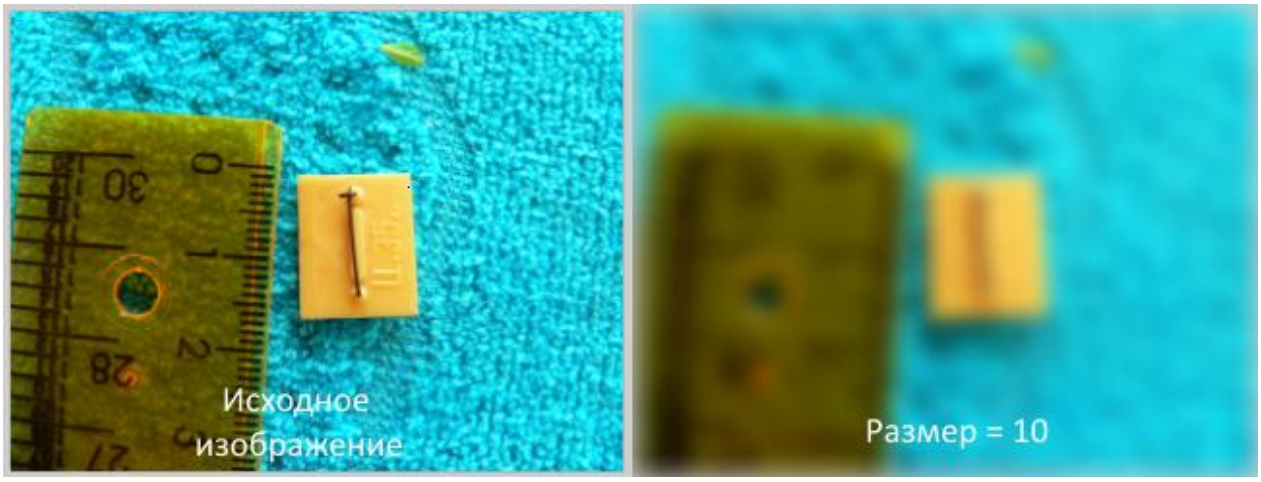


Рисунок 13 – степень размытости изображения от размера матрицы

Стоит упомянуть и о граничных условиях. Если размер матрицы больше чем 1x1, то возникает граничное условие для крайних пикселей. Метод решения достаточно прост, требуется создать временное изображение, увеличив все края изображения на половину длину матрицы свертки. При этом новое место достаточно заполнить повторяющимися пикселями краев старого изображения.

2.2 Описание алгоритма сегментации к-средних

Сегментация – процесс разбиения некоторых объектов или областей на кластеры или классы по схожим характеристикам. Основная цель сегментации – изолировать интересующие области друг от друга. Конечный успех сегментации изображений представляется точностью сегментации, по этой причине значительное внимание должно быть уделено повышению ее надежности.

Как правило, алгоритмы сегментации изображений основываются на одном из двух базовых свойств яркости: однородности и разрывности. В первом случае используется разбиение изображения на области по заранее заданным критериям. Во втором случае разбиение состоит на основание резкого изменения сигнала, например перепады яркости на изображениях.

Пример кластеризации точек представлен на рисунке 14.

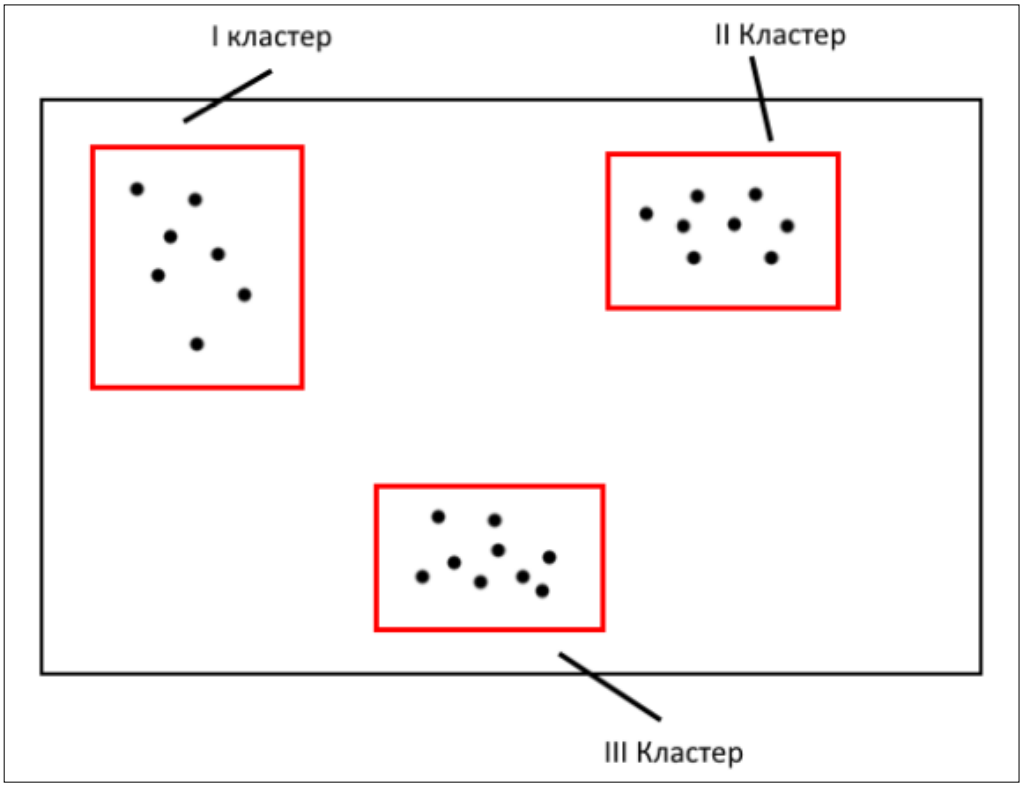


Рисунок 14 – Пример кластеризации точек

Одним из самых популярных алгоритмов кластеризации является «k-means». Метод к-средних обладает простотой реализации и быстрой сходимостью. Сам алгоритм можно записать в виде формулы 7:

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2 \quad (7)$$

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

Точки, являющиеся центрами кластеров, называются также центрами масс этих кластеров.

Алгоритм кластеризации к-средних, на примере изображений, можно описать последовательностью нескольких шагов:

1) выбор первоначального количества классов;

2) выбор из множества пикселей исходного изображения случайно или с помощью определенного алгоритма первоначальных центров кластеров;

3) вход в цикл, который выполняется до тех пор, пока центры масс кластеров не перестанут изменять свои значения;

4) обход каждого пикселя изображения и определение его к конкретному кластеру, используя некий алгоритм на основе цвета пикселя;

5) после окончания выполнения цикла высчитываются новые центры кластеров по определенному алгоритму;

6) проверка окончания итерации за счет сравнения значений предыдущих кластеров и новых: если проверка удачная, то заканчиваем выполнение цикла, иначе продолжаем с пункта 3.

Для выбора первоначального количества кластеров чаще всего применяется логический анализ той области, в которой алгоритм кластеризации будет применяться. В данном случае предполагается, что объект на изображение только один, а остальное является фоном. Исходя из такого предположения, можно сделать вывод, что количество кластеров будет ровняться двум.

При сегментации, а в частности, при выборе первоначальных кластеров изображений оптимальных подходов не существует. Выбор центра масс при первой итерации является существенным фактором, который влияет на скорость работы алгоритма, поэтому требуется определить как можно более точный способ их задания.

Исходя из предметной области обработки изображений и того, что количество кластеров не превышает двух, можно сделать предположение, что центр масс одного из кластеров следует расположить в самом центре изображения, а последующий на любом из краев изображения. Основываясь на том, что чаще производится фотосъемка сверху в низ, можно сделать уточнение, что центр второго кластера лучше выбирать в верхней части изображения, например, рядом с началом координат. Пример оптимального выбора центра масс показан на рисунке 15.

Рисунок 15 – Пример оптимального выбора центра масс

2.3 Анализ сегментации изображений с применением алгоритма свертки

Сегментация изображений методом к-средних чувствительна к шуму и посторонним объектам незначительного размера, вследствие чего кластеризация проходит с большой долей погрешности.

Для уменьшения значения данной погрешности к изображению применяется фильтр «Размытия», основанный на алгоритме свертки. Степень размытия зависит от размерности матрицы свертки, поэтому требуется вычислить оптимальное значение размера матрицы, при котором кластеризация будет проходить наиболее эффективно. Сделать это можно практическим путем, наблюдая за результатом кластеризации.

Для начала было выбрано изображение, которое содержит посторонний объект, который является малозаметным на изображение (рисунок 16).

Затем экспериментальным путем выбран размер матрицы равный 20. Результат сегментации представлен на рисунке 17.

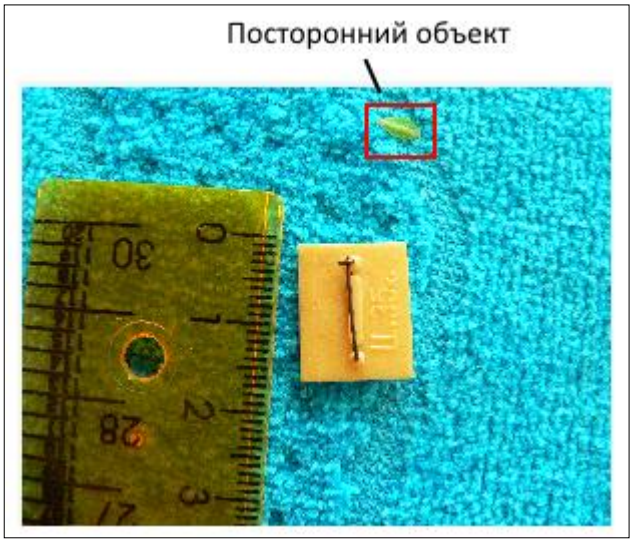


Рисунок 16 – Изображение содержащее посторонний объект

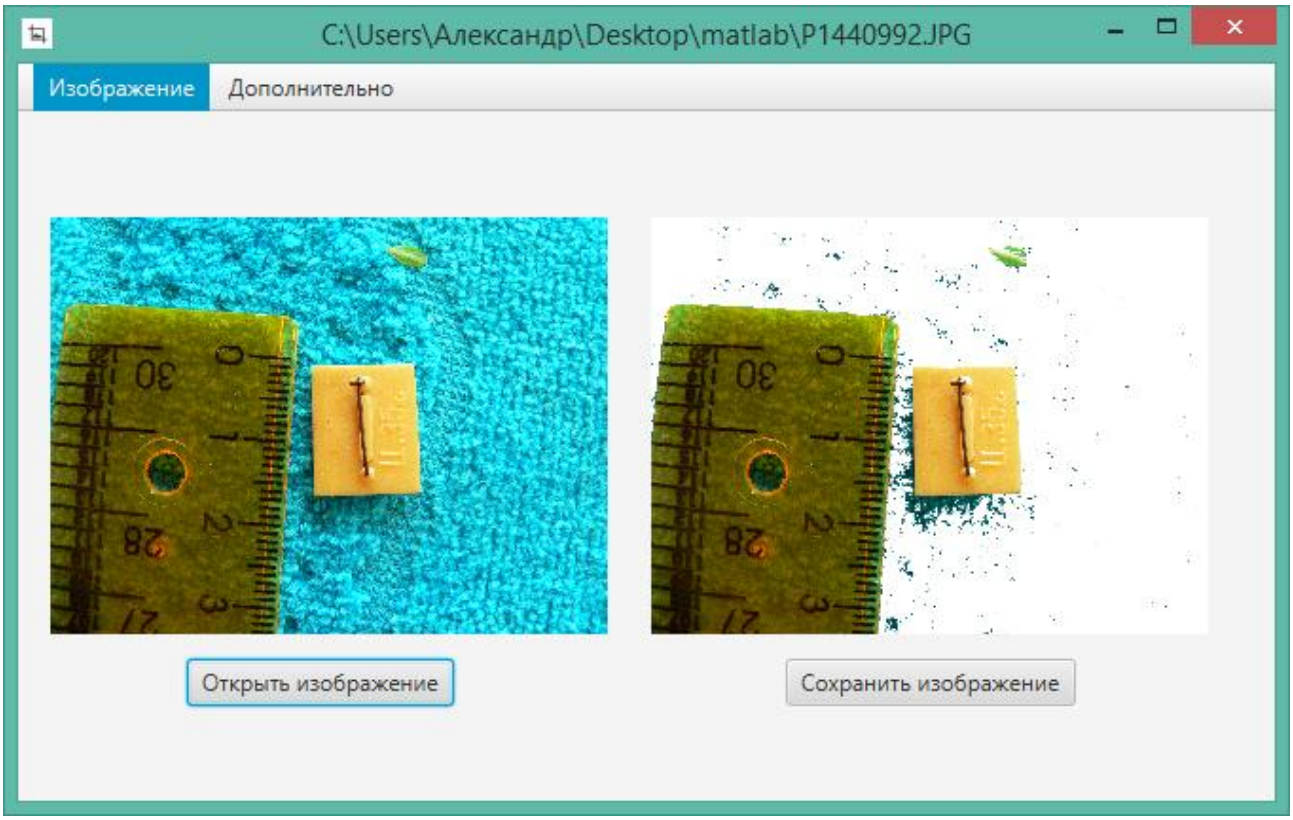


Рисунок 17 – Результат сегментации изображения

На изображение справа (рисунок 17) видно, что после применения фильтра и сегментации посторонний объект все еще виден и принадлежит к кластеру основного объекта изображения. Поэтому было решено увеличить размер матрицы свертки до 80 (рисунок 18).

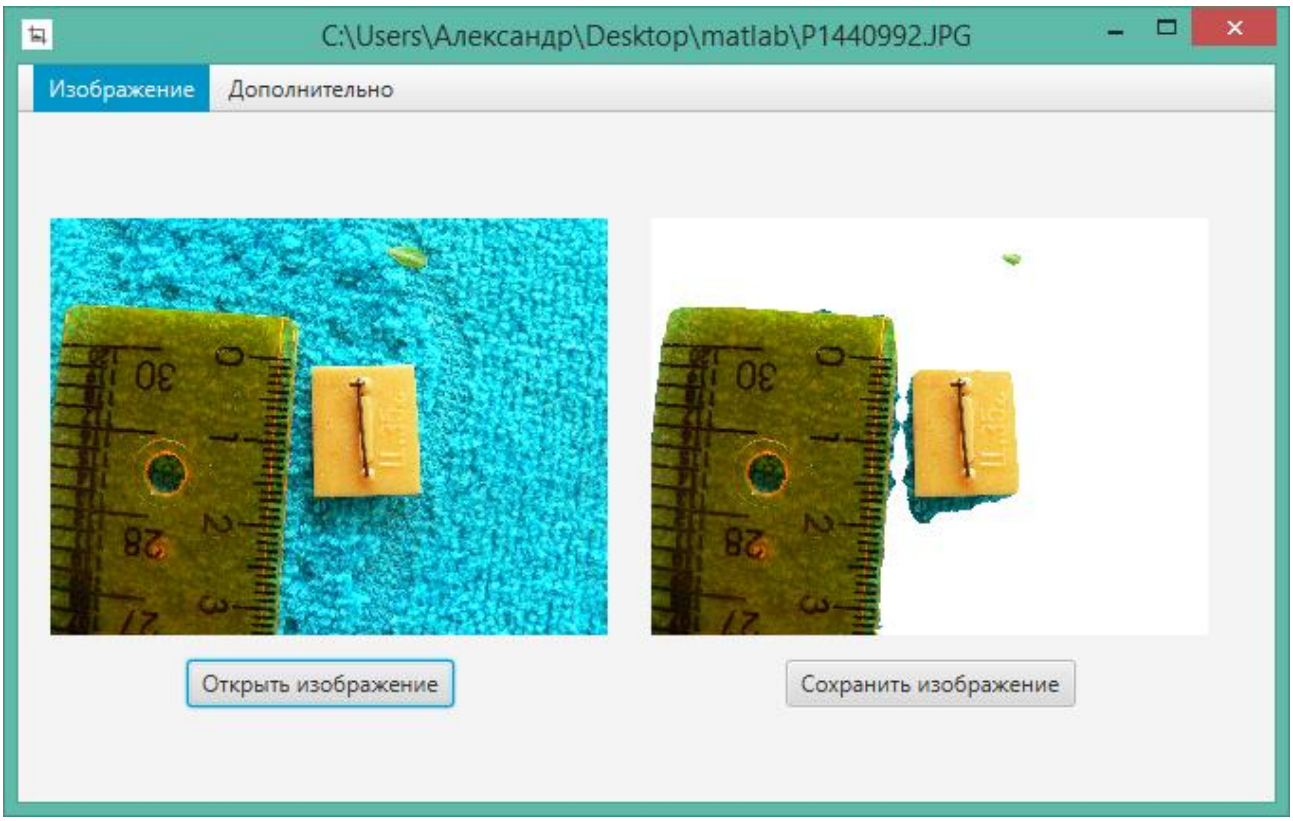


Рисунок 18 – Результат сегментации изображения

На изображение справа (рисунок 18) видно, что посторонний объект пропал с изображения, так же исчезли все посторонние шумы. Исходя из этого, можно сделать вывод, что размерность матрицы определена, и сегментация изображения происходит успешна. Сравнивая основной объект «значок» на рисунках 16 и 17, можно утверждать, что значимость объекта визуально уменьшилась незначительно, что является окончанием эксперимента подбора размера свертки матрицы для алгоритма кластеризации k-means.

3 ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ СЕГМЕНТАЦИЙ ИЗОБРАЖЕНИЙ С ПРИМЕНЕНИЕМ АЛГОРИТМОВ СВЕРТКИ

Для написания программной реализации алгоритмов и их практической визуализации использовался язык программирования Java и среда разработки IntelliJ IDEA (рисунок 19).

Для разработки приложения использовались следующие средства разработки и проектирования:

- платформа – Java Platform, Standard Edition, сокращенно Java SE и ее реализация от компании Oracle – Java Development Kit, сокращенно JDK;
- платформа – JavaFX, которая используется для создания графического интерфейса пользователя;
- средство для проектирования графического интерфейса – Scene Builder;
- схема разделения данных model-view-controller, сокращенно MVC;
- интегрированная среда разработки приложений – IntelliJ IDEA.

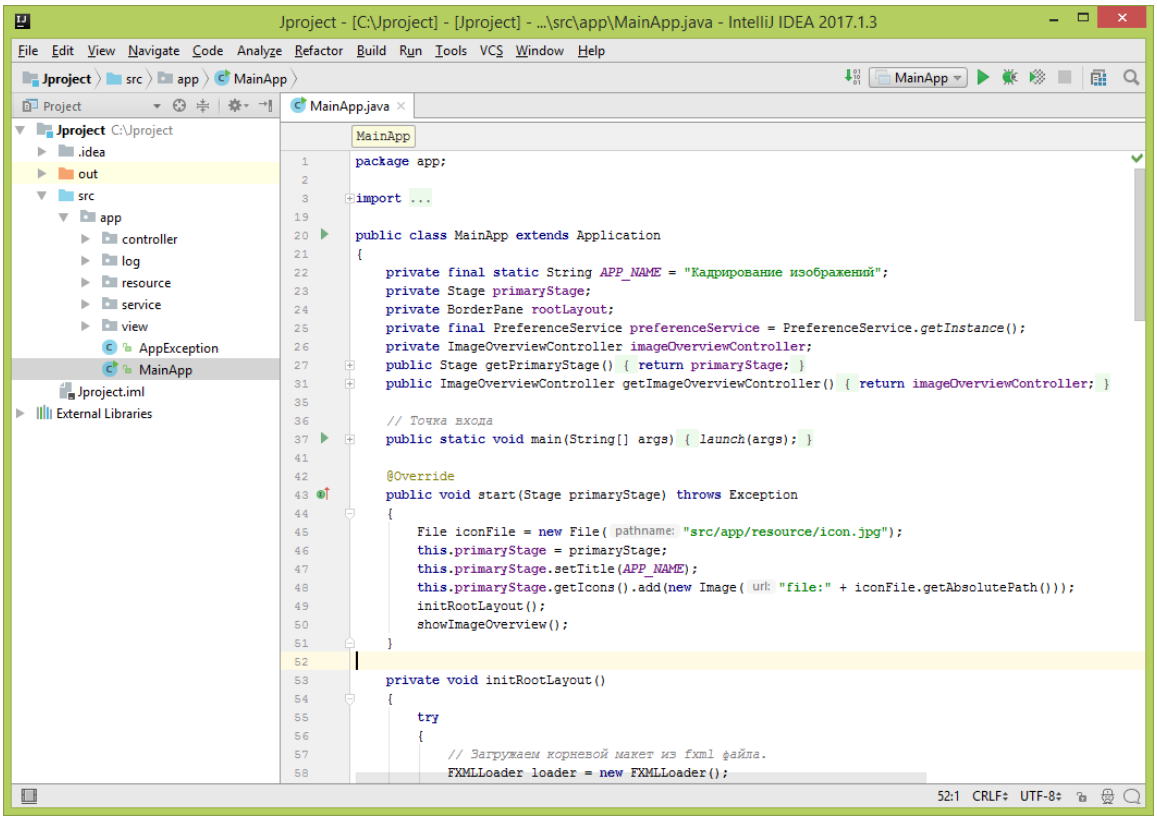


Рисунок 19 – Проект приложения в среде IntelliJ IDEA

Реализация структуры каталогов программы организованна следующим образом (рисунок 19).

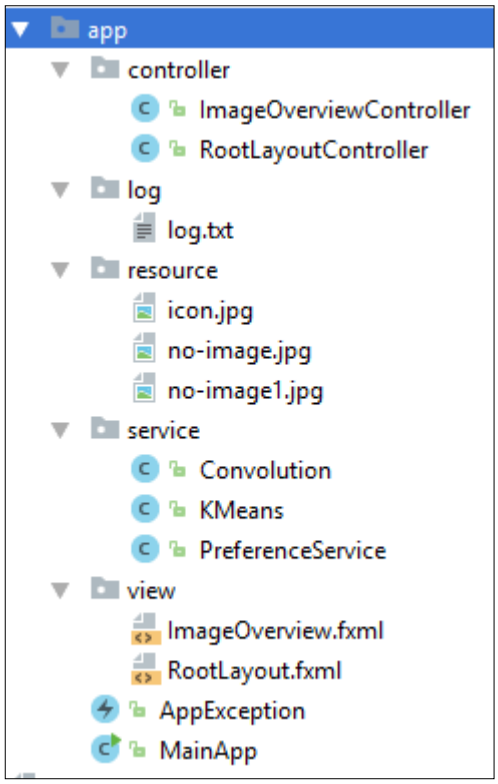


Рисунок 20 – Структура каталог приложения

Корневая папка проекта «app» содержит:

- файл «MainApp.java», который является контрольной точкой при запуске приложения, содержит методы для вызова контроллеров приложения;
- файл «AppException.java», который отвечает за обработку исключительных ситуаций и их логирование в файл «log.txt»;
- папку «resource», которая содержит подгружаемые в ходе открытия приложения изображения;
- папка «view», которая содержит файлы формата «fxml», основанные на языке разметки xml. Шаблоны, отвечающие за отображение программы, хранят пользовательский интерфейс, написанный на языке fxml;
- папка «controller», которая содержит файлы контроллеров. В данных файлах описывается основной функционал приложения, а так же методы напрямую связанные с шаблонами отображения, описанными в предыдущем пункте.

Так же каталог проекта содержит папку «service», которая содержит вспомогательные файлы, направленные на решение специфических задач пользователя. Например, файл «KMeans.java» содержит алгоритм кластеризации к-средних и методы работы с изображениями. Аналогично файл «Convolution.java» содержит алгоритм свертки и методы по работе с изображениями. Файл «PreferenseService.java» содержит методы по записи последней открытой директории в реестр системы, реализованные, для того, чтобы при частом открытии изображений не искать последнюю открытую директорию.

3.1 Реализация пользовательского интерфейса

Интерфейс пользователя представляет собой единое окно (рисунок 21), которое содержит:

- пункт меню «Изображение»:
  - подпункт «Открыть изображение»;
  - подпункт «Сохранить изображение»;
- исходное и кадрированное изображение;
- кнопки в визуальной части:
  - «Открыть изображение»;
  - «Сохранить изображение»;
- название приложения или прямой путь до изображения;
- иконку приложения;

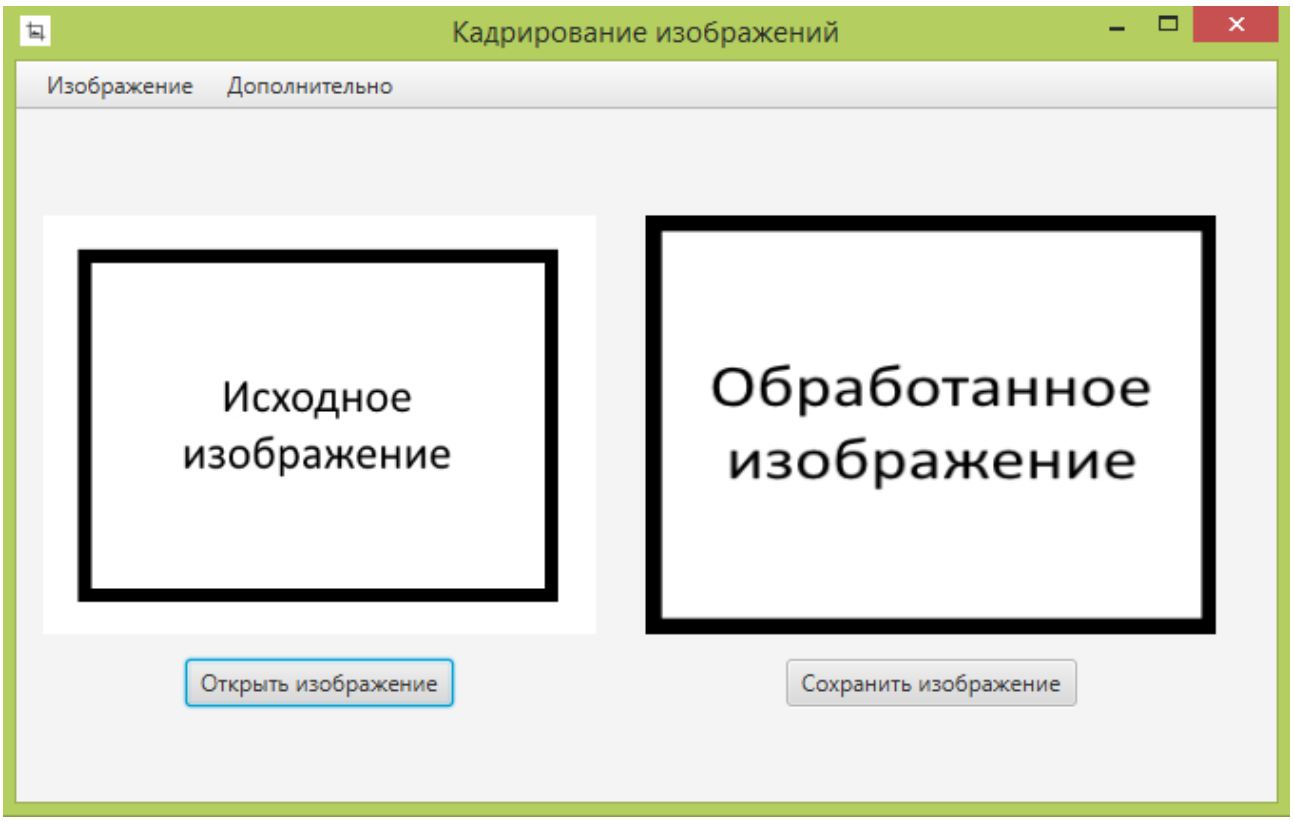


Рисунок 21 – интерфейс пользователя при открытии приложения

Также приложение поддерживает горячие клавиши для действий открытия и сохранения приложения и имеет во вкладке «Дополнительно» пункт «О программе», где указан автор и его контактная информация в соответствии с рисунком 22.

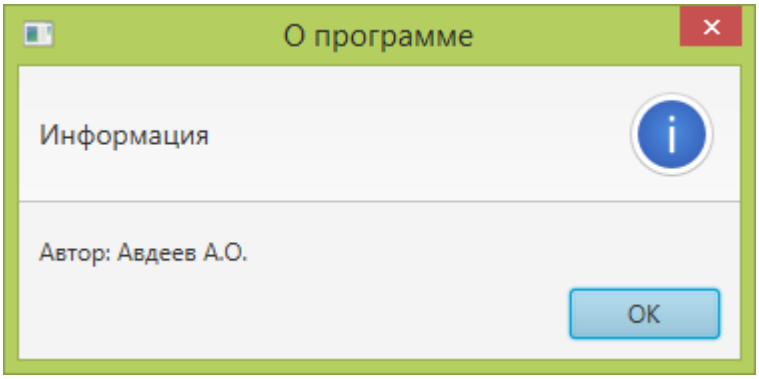


Рисунок 22 – вкладка меню «О программе»

При проектировании было учтено требование по минимизации и упрощению интерфейса пользователя, поэтому приложение состоит только из главной сцены. Главная сцена размещает два объекта, которые содержат исходное изображение и кадрированное. Также после загрузки программы на фон объектов, содержащих изображения, были подгружены исходные

изображения, которые, по мнению автора, наглядно демонстрируют функционал приложения (рисунок 21). Пример загрузки изображений с помощью операции «Открытия изображения» продемонстрированы на рисунках 22-23.

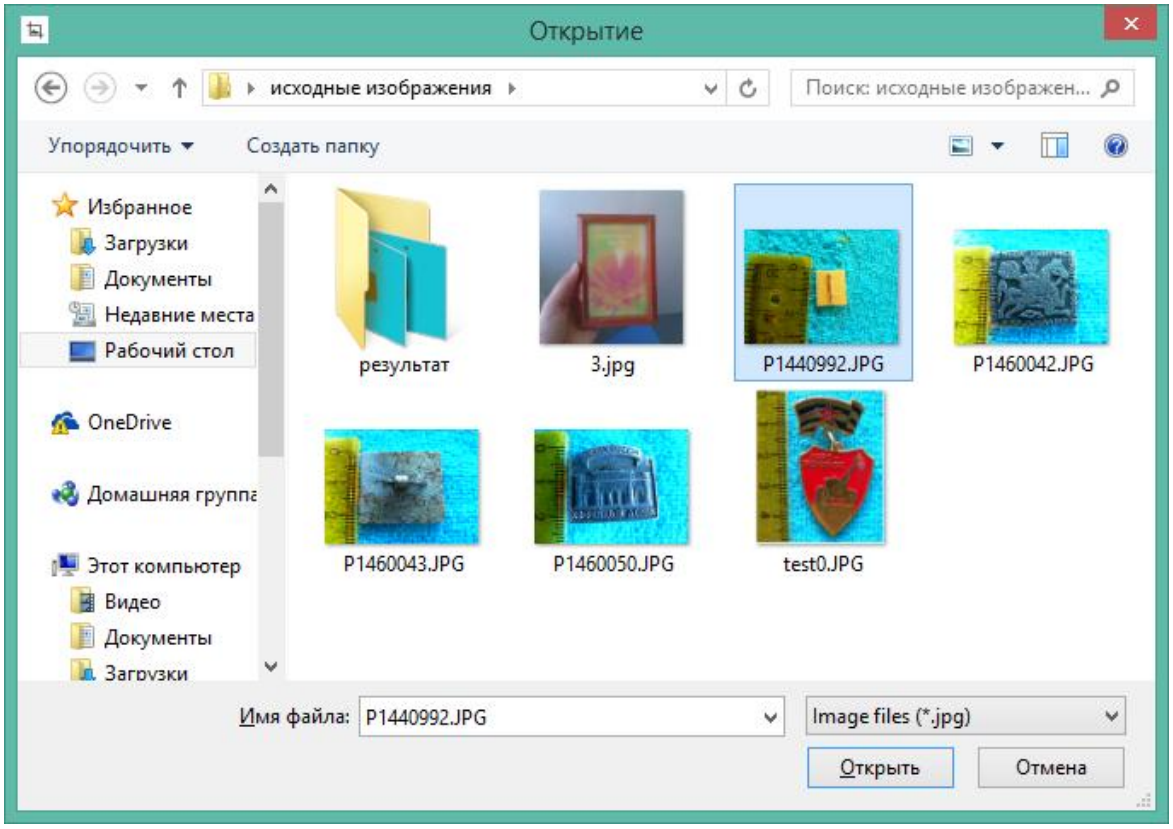


Рисунок 22 – Диалоговое окно выбора изображения

Рисунок 23 – Пример открытия изображения «Прямоугольник»

3.2 Тестирование пользовательского интерфейса

Часто в процессе проектирования приложений программист не учитывает «поведение пользователя» программы, а ведь учитывать все возможные последствия и действия пользователя и программы является одной из главных задач хорошо спроектированного приложения. Для этих целей применяется тестирование.

Для выявления случаев возникновения исключительных и непредвиденных ситуаций в Java существуют исключения и их обработчики. Для фиксирования всех исключительных ситуаций в проект был добавлен класс «AppException», который принимает объект исключения и фиксирует его в лог файл, который находится в корневой директории проекта, в папке log. Лог файл имеет расширение «txt» и работает со стандартными потоками ввода/вывода.

Для выявления исключительных ситуаций, сначала требуется дать определение таких ситуаций. Под исключительной ситуацией понимают ситуации, которые нарушают алгоритм выполнения программы, во время возникновение которой приложение прекращает выполнение определенной операции, выполняемой в данный момент или в редких случаях самого приложения, и начинает работать произвольным образом или прекращает свою работу. Ситуации, предусмотренные разработчик заранее, в большинстве случаев фиксируются и обрабатываются, на начальном этапе проектирования приложения. Например, главный класс приложения «MainApp» содержит два метода: «initRootLayout» и «ShowImageOverview» в которых инициализируются соответствующие контроллеры «RootLayoutController» и «ImageOverviewControoller». Каждый из контроллеров может выбрасывать исключительные ситуации, которые в самом классе контроллера могут никак не обрабатываться. Поэтому инициализация классов контроллеров в каждом методе главного класса приложения обернуты в блоки try-catch, для того, чтобы фиксировать исключительные ситуации в соответствии с рисунком 24.

```
private void showImageOverview()
{
    try
    {
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(MainApp.class.getResource( name: "view/ImageOverview.fxml"));
        AnchorPane lotOverviewPage = loader.load();

        rootLayout.setCenter(lotOverviewPage);

        imageOverviewController = loader.getController();
        imageOverviewController.setMainApp(this);
    }
    catch(IOException e)
    {
        AppException.Throw(e);
    }
}
```

Рисунок 24- Пример обертки кода класса контроллера в блок try/catch

Рекомендуется обрабатывать исключения локально, однако, если исключение во время проектирование локально не фиксируется, то фиксирование исключения на уровне выше является «защитной мерой» для отлова исключительных ситуаций. Пример локальной обработки исключения демонстрируется на рисунке 25.

```
if(!file.exists())
    try
    {
        //noinspection ResultOfMethodCallIgnored
        file.createNewFile();
    }
    catch(IOException e)
    {
        AppException.Throw(e);
    }
```

Рисунок 25 - Пример локальной обработки исключительной ситуации

Так как интерфейс приложения исключительно прост, то есть состоит из двух кнопок и двух диалоговых окон, то пользователь оказывает минимальное влияние на работу приложения, что является хорошей практикой в процессе разработки пользовательского интерфейса программного продукта.

1.3 Пример работы программы

Предположим, что требуется кадрировать несколько примитивных изображений, а затем сохранить их.

Для этого запустим приложение. После появления главной сцены приложения (рисунок 21), нажмет сочетание горячих клавиш «CTRL + O». После появления диалогового окна (рисунок 22), нужно открыть каталог с изображениями, если он не открыт, и выбрать исходное изображение для кластеризации и кадрирования.

После выбора изображения, программа начнет автоматически кластеризовать изображение, затем кадрирует его по найденному сегменту объекта изображения (рисунок 23). Если процесс кадрирования вас удовлетворил, то можно сохранить полученное изображения, нажав сочетания горячих клавиш «CTRL + S» и выбрать директорию для сохранения (рисунок 25). По умолчанию директория для сохранения изображения совпадает с последней открытой директорией.

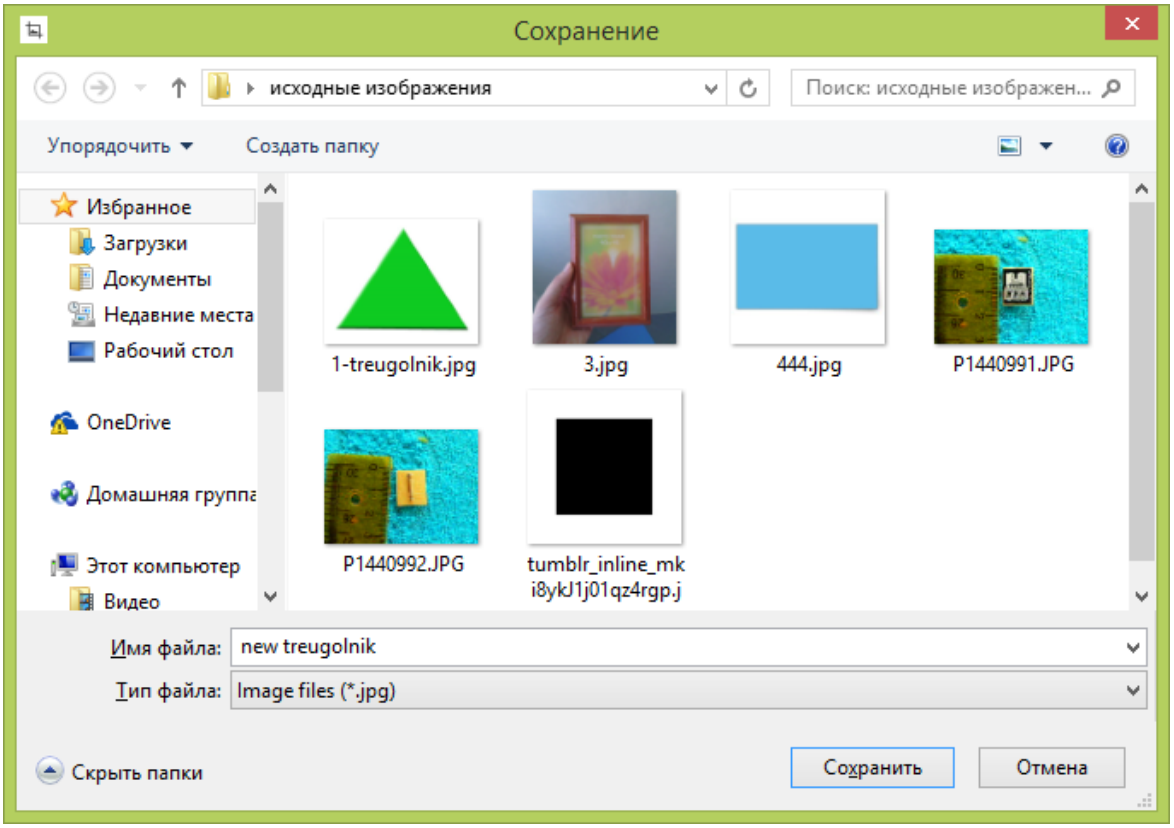


Рисунок 25 – Диалоговое окно сохранения полученного изображения

ЗАКЛЮЧЕНИЕ

В ходе выпускной квалификационной работы была описана актуальность рассматриваемой темы, определен объект и предмет выпускной работы, поставлена цель и выявлены задачи. Также был проведен анализ состояния вопроса, а именно проведен обзор существующих программных аналогов. Кроме этого, были формализованы требования к разрабатываемому программному продукту, описан язык программирования, с помощью которого разработаны алгоритмы программы, а также была описана работа данного программного обеспечения.

В результате было показано, что задачу сегментации изображений можно рассматривать как задачу по кластеризации объектов на изображениях. Было установлено, что автоматизация и точность сегментации изображений возможна за счет применения алгоритмов свертки.

Разработанное в ходе выпускной квалификационной работы программное обеспечение является автоматическим, однако имеет низкую точность сегментации, при слишком большом значении размера матрицы свертки, что может компенсировать высокой скоростью обработки изображений, поэтому программный продукт имеет перспективу улучшения и развития. Для успешного решения задачи сегментации изображения, опытным путем была подобрана величина матрицы свертки.

Данное программное обеспечение может быть дополнено новыми функциональными алгоритмами, работа существующих алгоритмов может быть распределена между несколькими потоками для увеличения производительности программы, а размер матрицы свертки может быть задана в окне пользовательского интерфейса. Так же разработанный программный продукт является бесплатным и имеет простой интерфейс пользователя.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Нормативно-правовые акты

1. ГОСТ 2.105 – 95. Общие требования к текстовым документам [Текст]. – М.: Изд-во стандартов, 1996. – 29 с. – (Единая система конструкторской документации).

2. ГОСТ 7.1-2003. Библиографическая запись. Библиографическое описание документа.

3. ГОСТ 7.32-2001. Отчет о научно-исследовательской работе. Структура и правила оформления.

4. ГОСТ 7.82-2001. Библиографическая запись. Библиографическое описание электронных ресурсов.

5. ГОСТ 19.701 – 90. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения (ИСО 5807–85) [Текст]. Введен 1992–01–01. – М.: Изд-во стандартов, 1992. – 14 с. – (Единая система программной документации).

Научная и методическая литература

6. Егоров, А.Г. Правила оформления выпускных квалификационных работ по программам подготовки бакалавра и специалиста: учебно-методическое пособие / А.Г. Егоров, В.Г. Виткалов, Г.Н. Уполовникова, И.А. Живоглядова − Тольятти: ТГУ, 2012. – 135 с.

7. Положение о выпускной квалификационной работе. – Тольятти: ТГУ. – 2014.

8. Визильтер Ю. В. Обработка и анализ изображений в задачах машинного зрения: Курс лекций и практических занятий. – М.: Физматкнига, 2010. – 672 с.

9. Клейнберг Дж., Тардос Е. Алгоритмы: разработка и применение. Классика Computers Science / Пер. с англ. Е. Матвеева. – СПб.: Питер, 2016. – 800 с.

10. Гонсалес Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. – Москва : Техносфера, 2012. – 1104 с.

11. И. С. Грузман Цифровая обработка изображений в информационных системах / Грузман И.С., Киричук В.С., Косых В.П., Перетягин Г.И., Спектор А.А.: Учебное пособие.- Новосибисрк: Изд-во НГТУ, 2002. - 352 с.

12. Н. Красильников Цифровая обработка 2D- и 3D-изображений / Красильников Н.: Отдельное издание. - БХВ-Петербург, 2011. - 608 с.

Электронные ресурсы

13. Image Processing Toolbox– Matlab and Toolboxes [Электронный ресурс]. - Электрон. дан. – [2017]. – Режим доступа : http://matlab.exponenta.ru/imageprocess/book2/48.php.

14. Машинное зрение: понятия, задачи и области применения [Электронный ресурс]. - Электрон. дан. – [2017]. - Режим доступа : http://rusnauka.com/25_SSN_2009/Informatica/51050.doc.htm.

15. Системы компьютерного зрения: современные задачи и методы [Электронный ресурс]. - Электрон. дан. – [2017]. – Режим доступа : http://controleng.ru/innovatsii/sistemy-komp-yuternogo-zreniya-sovremenny-e-zadachi-metody/.

16. Class ConvolveOp (Java Platform SE 8) [Электронный ресурс]. - Электрон. дан. – [2017]. – Режим доступа : http://docs.oracle.com/javase/8/docs/api/java/awt/image/ConvolveOp.html.

17. Java DIP - Понимание свертка [Электронный ресурс]. - Электрон. дан. – [2017]. – Режим доступа : http://www.w3ii.com/ru/java_dip/understand_convolution.html.

18. Image Processing Toolbox– Matlab and Toolboxes [Электронный ресурс]. - Электрон. дан. – [2017]. – Режим доступа : http://matlab.exponenta.ru/imageprocess/book3/11/fspecial.php.

19. Теоретические основы [Электронный ресурс]. - Электрон. дан. – [2017]. – Режим доступа : http://statistica.ru/theory/klasterizatsiya-metod-k-srednikh/.

Литература на иностранном языке

20. Beyerer J. Automated Visual Inspection: Theory, Practice and Applications / J. Beyerer, P. Fernando, F. Christian. – Springer Berlin Heidelberg, 2016. – 798 p.

21. James L. Pro JavaFX 2/ L. James, G. Weiqi, C. Stephen, I. Dean, V. Johan. – Apress, 2012. – 641 p.

22. Giuseppe M., Gian L. Image Analysis and Processing – ICIAP 2011: Crop Detection through Blocking Artefacts Analysis / M. Giuseppe, L. Gian. - Springer-Verlag Berlin Heidelberg, 2011. – 714 p.

ПРИЛОЖЕНИЕ А

Код алгоритма кластеризации k-means

```
package app.service;

import app.AppException;

import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Random;

public class KMeans
{
    private Image sourceImage;           // Исхожное изобажений
    private Image bufferedImage;         // Изображение в кластерах
    private Image outputImage;           // Выходное изображение
    private int[][] pixel_assignments;
    private int WHITE = -1;

    public Image getOutputImage() {
        return outputImage;
    }

    public Image getSourceImage() {
        return sourceImage;
    }

    public Image getBufferedImage() {
        return bufferedImage;
    }

    // Конструктор по файлу
    public KMeans(File file)
    {
        try
        {
            sourceImage = new Image(file);
            kmeansHelper();
        }
        catch(IOException e)
        {
            AppException.Throw(e);
        }
    }

    // Конструктор по изображению
    public KMeans(BufferedImage image)
    {
            sourceImage = new Image(image);
            kmeansHelper();
    }

    private void kmeansHelper()
    {
        int width = sourceImage.getWidth();
        int height = sourceImage.getHeight();
        int [][] rgb = new int[width][height];

        // Создаем заготовку для нового изображения
        BufferedImage buffered = new BufferedImage(width, height,
sourceImage.getBufferedImage().getType());
```

```
        Graphics2D graphics = buffered.createGraphics();
        graphics.drawImage(sourceImage.getBufferedImage(), 0, 0, width, height,
null);

        for(int w = 0; w < width; w++)
        {
            for(int h = 0; h < height; h++)
            {
                rgb[w][h] = buffered.getRGB(w, h);
            }
        }
        // Вызываем метод "алгоритм", который обновляет значения rgb
        algorithm(rgb);

        // Записываем новые значения rgb в изображение
        for(int w = 0; w < width; w++)
        {
            for(int h = 0; h < height; h++)
            {
                buffered.setRGB(w, h, rgb[w][h]);
            }
        }
        // Записываем изображение
        bufferedImage = new Image(buffered);

        // Заранее создаем выходное изображение
        outputImage = new Image(sourceImage.getBufferedImage());
    }

    private void algorithm(int[][] rgb)
    {
        int NUMBER_OF_CLUSTERS = 2;
        int[] clusters_x = new int[NUMBER_OF_CLUSTERS];
        int[] clusters_y = new int[NUMBER_OF_CLUSTERS];
        int[] clusters = new int[NUMBER_OF_CLUSTERS];
        Random rand = new Random();

        // Выбираем первоначальные кластеры
        for(int i = 0; i < clusters.length; i++)
        {
            boolean contains_duplicate = true;
            // В первый кластер записываем центральную точку изображения
            if(i == 0)
            {
                // Центр фото
                clusters[i] = rgb[rgb.length / 2][rgb[0].length / 2];
                clusters_x[i] =  rgb.length;
                clusters_y[i] =  rgb[0].length;
            }
            // В остальные кластеры записываем рандомную точку отлич. от 1
кластера
            else
            {
                do
                {
                    int random_x, random_y;
                    random_x = rand.nextInt(rgb.length);
                    random_y = rand.nextInt(rgb[0].length);
                    clusters_x[i] =  random_x;
                    clusters_y[i] =  random_y;

                    for(int j = 0; j < i; j++)
                    {
                        if(j == i - 1 && clusters[j] != rgb[random_x][random_y])
```

```
                        {
                            // Если 2 кластер и цвета не совпадают, то условие
выполнено
                            clusters[i] = rgb[random_x][random_y];
                            contains_duplicate = false;
                        }
                        else if(clusters[j] == rgb[random_x][random_y])
                        {
                            // Выходим из цикла for т.к. найден дубликат
                            // Пробуем снова, с новыми случ. значениями
                            j = i;
                        }
                    }
                } while(contains_duplicate);
            }
        }

        for(int i = 0; i < clusters.length; i++)
        {
            System.out.println("r: " + getRed(clusters[i]) + ", g: " +
getGreen(clusters[i]) + ", b: " + getBlue(clusters[i]));
        }

        // Группируем пиксели изображения с центрами кластеров
        // pixel assignments (by their index)
        pixel_assignments = new int[rgb.length][rgb[0].length];
        int[] num_assignments = new int[NUMBER_OF_CLUSTERS];

        // Cluster sums for current cluster values (represented by index)
        int[] alpha_sum = new int[NUMBER_OF_CLUSTERS];
        int[] red_sum = new int[NUMBER_OF_CLUSTERS];
        int[] green_sum = new int[NUMBER_OF_CLUSTERS];
        int[] blue_sum = new int[NUMBER_OF_CLUSTERS];

        // Кол-во итераций до схождения. Больше 100 ставить не стоит
        int max_iterations = 100;
        int num_iterations = 1;
        int[] center_iterations = new int[NUMBER_OF_CLUSTERS];


        while(num_iterations <= max_iterations)
        {
            // Clear number of assignments list first
            for(int i = 0; i < clusters.length; i++)
            {
                num_assignments[i] = 0;
                alpha_sum[i] = 0;
                red_sum[i] = 0;
                green_sum[i] = 0;
                blue_sum[i] = 0;
            }

            // Go through all pixels in rgb
            for(int i = 0; i < rgb.length; i++)
            {
                for(int j = 0; j < rgb[0].length; ++j)
                {
                    // Set min_dist initially to infinity (or very large number
that
                    // wouldn't appear as a distance anyways)
                    double min_dist = Double.MAX_VALUE;
                    int cluster_index = 0;
                    // compare instance's RGB value to each cluster point
                    for(int k = 0; k < clusters.length; k++)
```

```
                    {
                        float a_dist = (getAlpha(rgb[i][j]) -
getAlpha(clusters[k])) / 255.f;
                        float r_dist = (getRed(rgb[i][j]) - getRed(clusters[k]))
/ 255.f;
                        float g_dist = (getGreen(rgb[i][j]) -
getGreen(clusters[k]))  / 255.f;
                        float b_dist = (getBlue(rgb[i][j]) -
getBlue(clusters[k]))  / 255.f;
                        //float len = (clusters_x[k] - i) / ((float) rgb.length)
+ (clusters_y[k] - j) / ((float) rgb[0].length);
                        float dist = (float) Math.sqrt(a_dist * a_dist + r_dist
* r_dist + g_dist * g_dist + b_dist * b_dist);
                        if(dist < min_dist)
                        {
                            min_dist = dist;
                            cluster_index = k;
                        }
                    }
                    // Assign pixel to cluster
                    pixel_assignments[i][j] = cluster_index;
                    num_assignments[cluster_index]++;
                    // Add pixel's individual argb values to respective sums for
use
                    // later
                    alpha_sum[cluster_index] += getAlpha(rgb[i][j]);
                    red_sum[cluster_index] += getRed(rgb[i][j]);
                    green_sum[cluster_index] += getGreen(rgb[i][j]);
                    blue_sum[cluster_index] += getBlue(rgb[i][j]);
                }
            }

            // update previous assignments list
            for(int i = 0; i < clusters.length; i++)
            {
                int avg_alpha = (int) ((double) alpha_sum[i] / (double)
num_assignments[i]);
                int avg_red = (int) ((double) red_sum[i] / (double)
num_assignments[i]);
                int avg_green = (int) ((double) green_sum[i] / (double)
num_assignments[i]);
                int avg_blue = (int) ((double) blue_sum[i] / (double)
num_assignments[i]);
                clusters[i] = ((avg_alpha & 0x000000FF) << 24) | ((avg_red &
0x000000FF) << 16) | ((avg_green & 0x000000FF) << 8) | ((avg_blue &
0x000000FF));
            }

            if(num_iterations == 1)
            {
                System.arraycopy(clusters, 0, center_iterations, 0,
clusters.length);
            }
            else
            {
                int counter = 0;
                for(int i = 0; i < clusters.length; i++)
                {
                    if(clusters[i] == center_iterations[i])
                        counter++;
                }

                if(counter == NUMBER_OF_CLUSTERS)
                    break;
```

```
                else
                    System.arraycopy(clusters, 0, center_iterations, 0,
clusters.length);
            }
            num_iterations++;
        }

        // update RGB array
        for(int i = 0; i < rgb.length; i++)
        {
            for(int j = 0; j < rgb[0].length; ++j)
            {
                if(pixel_assignments[i][j] == 0) {
                    //rgb[i][j] = clusters[pixel_assignments[i][j]];
                }
                else
                    rgb[i][j] = WHITE;
            }
        }

        for(int i = 0; i < clusters.length; i++)
        {
            System.out.println("r: " + getRed(clusters[i]) + ", g: " +
getGreen(clusters[i]) + ", b: " + getBlue(clusters[i]));
        }
    }

    public void removeBackground(BufferedImage image)
    {
        outputImage = new Image(image);
        int width = outputImage.getBufferedImage().getWidth();
        int height = outputImage.getBufferedImage().getHeight();
        int[][] rgb = new int[width][height];

        for(int w = 0; w < outputImage.getBufferedImage().getWidth(); w++)
        {
            for(int h = 0; h < outputImage.getBufferedImage().getHeight(); h++)
            {
                rgb[w][h] = outputImage.getBufferedImage().getRGB(w, h);
            }
        }

        // update RGB array
        for(int i = 0; i < rgb.length; i++)
        {
            for(int j = 0; j < rgb[0].length; ++j)
            {
                if(pixel_assignments[i][j] == 0) {
                    //rgb[i][j] = clusters[pixel_assignments[i][j]];
                }
                else
                    rgb[i][j] = 16777215;
            }
        }

        // Записываем новые значения rgb в изображение
        for(int w = 0; w < width; w++)
        {
            for(int h = 0; h < height; h++)
            {
                outputImage.getBufferedImage().setRGB(w, h, rgb[w][h]);
            }
        }
    }
```

```
    public void crop(BufferedImage image)
    {
        outputImage = new Image(image);
        crop();
    }

    public void crop()
    {
        outputImage.crop(WHITE);
        int up = outputImage.getUp(),
                down = outputImage.getDown(),
                left = outputImage.getLeft(),
                right = outputImage.getRight();
        System.out.println(up);
        System.out.println(down);
        System.out.println(left);
        System.out.println(right);
    }

    // HELPER FUNCTIONS - to get individual R, G, and B values
    private static int getRed(int pix)
    {
        return (pix >> 16) & 0xFF;
    }

    private static int getGreen(int pix)
    {
        return (pix >> 8) & 0xFF;
    }

    private static int getBlue(int pix)
    {
        return pix & 0xFF;
    }

    private static int getAlpha(int pix)
    {
        return (pix >> 24) & 0xFF;
    }
}
```

ПРИЛОЖЕНИЕ Б

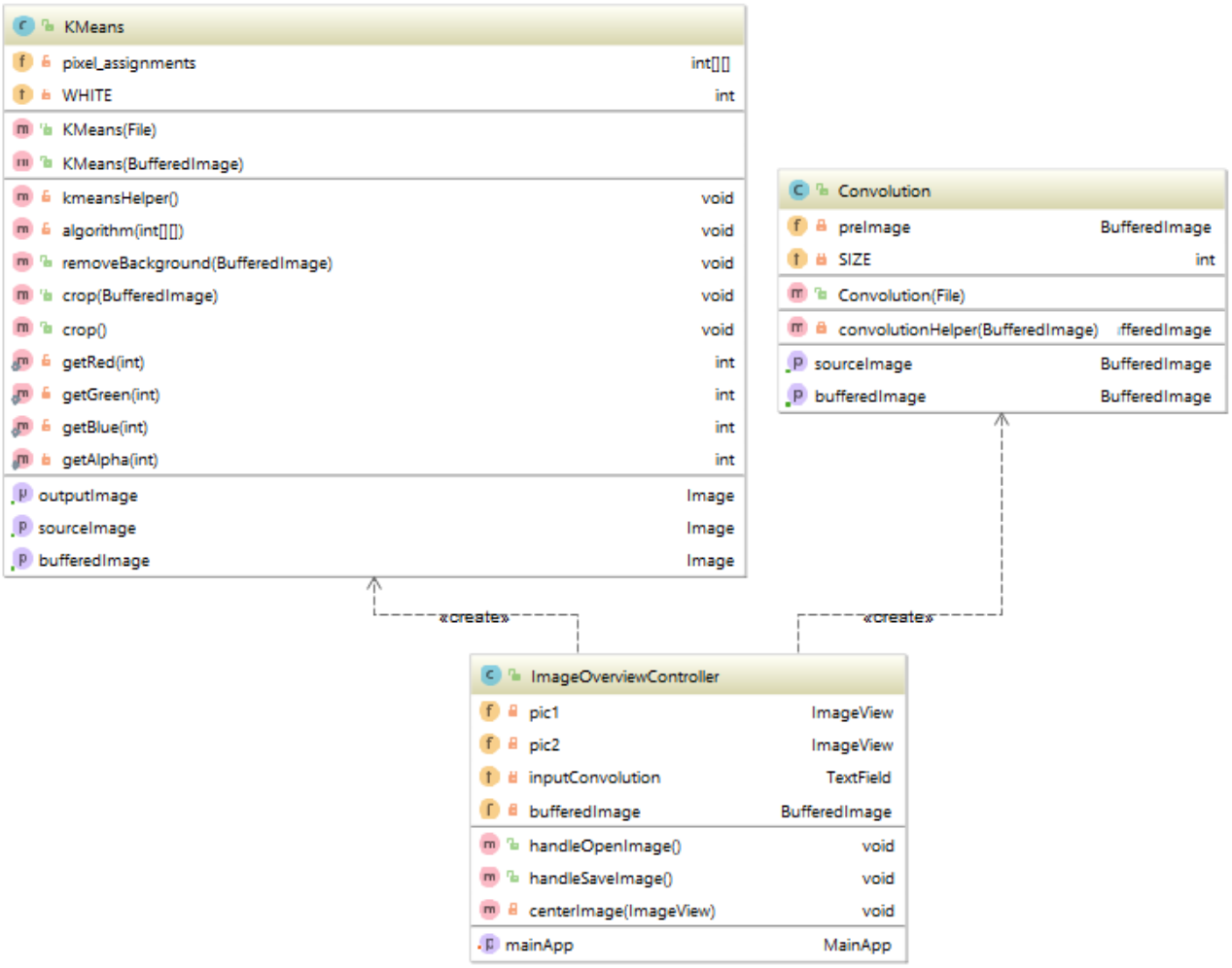
Код алгоритма свертки

```
package app.service;

import app.AppException;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.awt.image.BufferedImageOp;
import java.awt.image.ConvolveOp;
import java.awt.image.Kernel;
import java.io.File;
import java.io.IOException;

public class Convolution
{
    private BufferedImage sourceImage;          // Исхожное изобажений
    private BufferedImage outputImage;          // Размытое изображение
    private BufferedImage preImage;             // 1 раз размытое
    private int SIZE = 60;                      // Размер матрицы

    public BufferedImage getBufferedImage()
    {
        return outputImage;
    }

    public BufferedImage getSourceImage() {
        return sourceImage;
    }

    public Convolution(File file)
    {
        try {
            sourceImage = ImageIO.read(file);

            outputImage = convolutionHelper(sourceImage);
        } catch (IOException e) {
            AppException.Throw(e);
        }
    }

    private BufferedImage convolutionHelper(BufferedImage sourceImage) {

        int width = sourceImage.getWidth();
        int height = sourceImage.getHeight();

        // Создаем заготовку для нового изображения
        outputImage = new BufferedImage(width, height, sourceImage.getType());
        preImage = new BufferedImage(width, height, sourceImage.getType());

        float[] matrix = new float[SIZE];
        for (int i = 0; i < SIZE; i++)
            matrix[i] = 1.0f/(float) SIZE;
        // y
        BufferedImageOp op = new ConvolveOp( new Kernel(1, SIZE, matrix),
ConvolveOp.EDGE_NO_OP, null );
        op.filter(sourceImage, preImage);
        // x
        op = new ConvolveOp( new Kernel(SIZE, 1, matrix), ConvolveOp.EDGE_NO_OP,
null );
        op.filter(preImage, outputImage);
```

```
            return outputImage;
        }
    }
```

ПРИЛОЖЕНИЕ В

Диаграмма классов алгоритмов и их вызовов



ПРИЛОЖЕНИЕ Г

Диаграмма классов вызова программы