## 4 ВЕРЕФИКАЦИЯ И ОПЫТНАЯ ЭКСПЛУАТАЦИЯ РАЗРАБОТАННОГО ПРОГРАММНОГО КОМПЛЕКСА

* 1. **Интерфейс пользователя программного комплекса**

На рисунке 4.1 приведён граф вариантов взаимодействия с мобильным клиентом, который описывает функциональное назначение разработанного программного средства, взаимоотношение и зависимости между группами вариантов использования и пользователем.

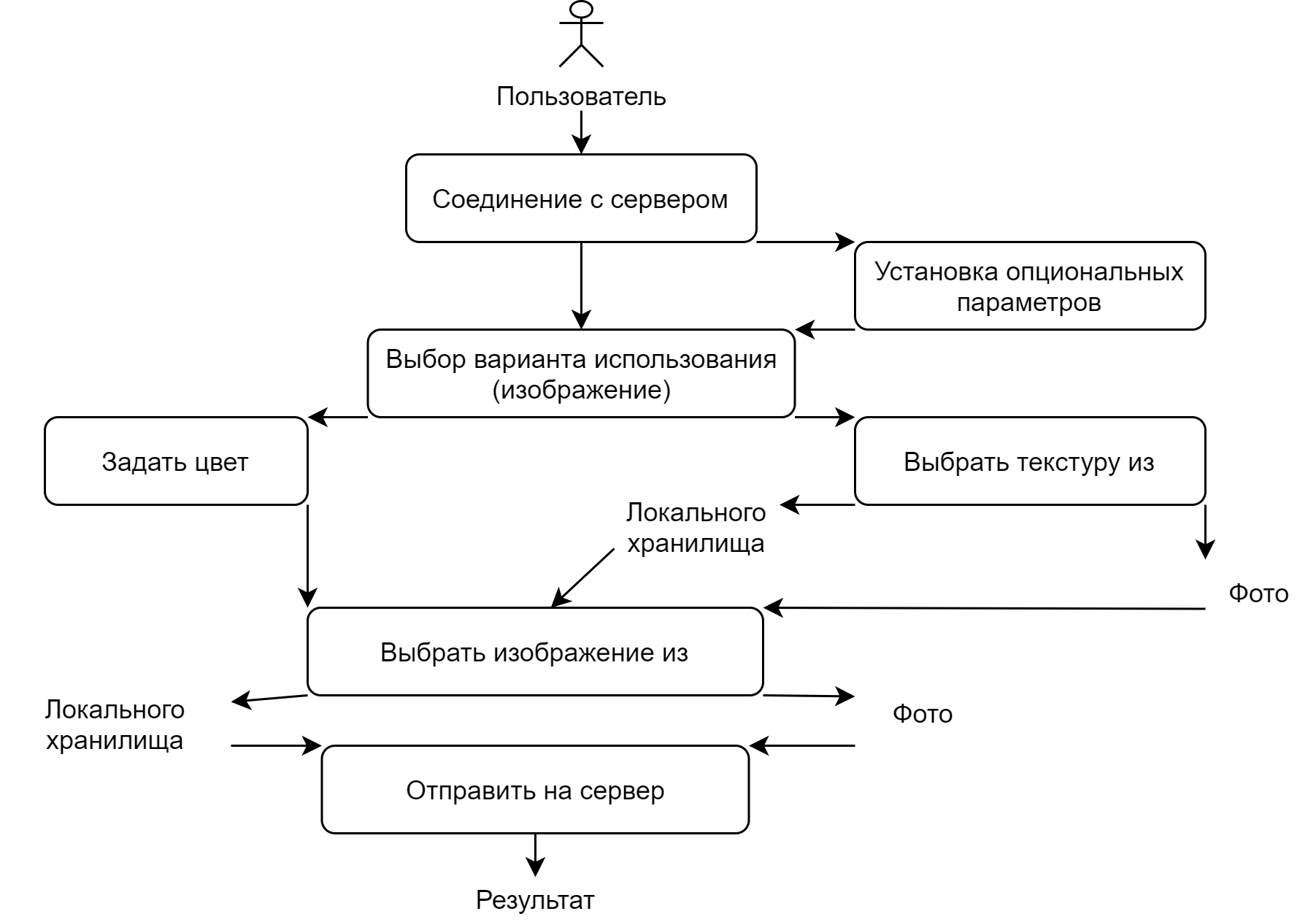


Рисунок 4.1 – Граф вариантов взаимодействия с клиентским приложением

Варианты использования пользовательского интерфейса разработанного мобильного клиента полностью отражают доступный для изменения функционал разработанного сервера.

Как видно из графа выше, взаимодействие с клиентским мобильным приложением состоит из нескольких этапов:

* соединение с сервером (подразумевает ввод необходимых *IP*-адреса и порта);
* конфигурация параметров;
* установка цвета либо текстуры (цвет задаётся кодом в формате трёх цветовой модели *RGB,* текстура выбирается из имеющихся фотографий на мобильном устройстве или через снимок фотокамеры);
* установка изображения (если не включён режим реального времени);
* запрос отправки данных на сервер;
* получение результата (результат в виде сегментированного окрашенного изображения будет отражён на пользовательском интерфейсе мобильного клиента).

В этап конфигурации параметров входит:

* выбор типа скрипта (нейронная сеть с архитектурой *ResNet*50, нейронная сеть с архитектурой *MobileNet* или скрипт с компьютерным зрением без нейронной сети);
* выбор типа клиента (режим реального времени или режим изображений);
* дополнительные параметры (размер изображения поданного на вход нейронной сети и другие отладочные коэффициенты).

Если этап конфигурации параметров (кроме выбора типа клиента) был пропущен – параметры останутся заданными по умолчанию, но есть возможность их изменить в любой момент времени через кнопку «меню» приложения.

Граф для варианта использования пользовательского интерфейса разработанного мобильного приложения в режиме реального времени отображён на рисунке 4.2.

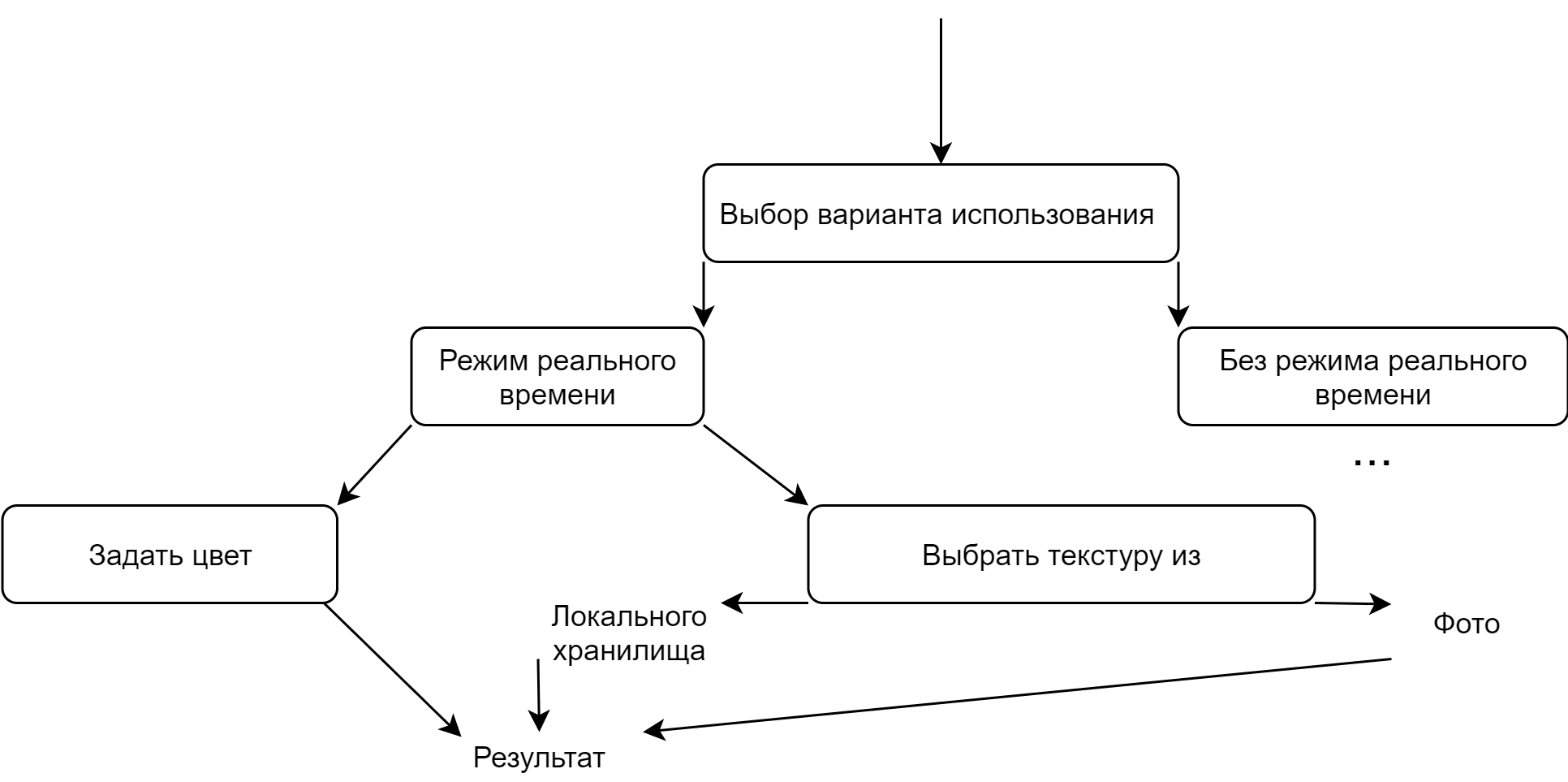


Рисунок 4.2 – Граф вариантов взаимодействия с клиентским приложением в режиме реального времени

Как видно из графа выше – кадры захватываются сразу из камеры мобильного устройства и непрерывно отправляются на сервер, так же как и результат с сервера непрерывно возвращается на клиентское мобильное приложение и отражается на экране графического интерфейса.

В сравнении с графом вариантов использования в режиме без реального времени, граф с режимом реального времени значительно меньше.

Преимуществами данного режима, помимо очевидных, также является изменение цвета или текстуры «на ходу», то есть без остановки транслирования кадров на сервер. Для этого достаточно выбрать из графической панели «Выбор цвета» другой цвет или текстуру и нажать на соответствующее изображение.

На рисунке 4.3 представлены снимки экранов мобильного приложения «*WallsDetecter*».

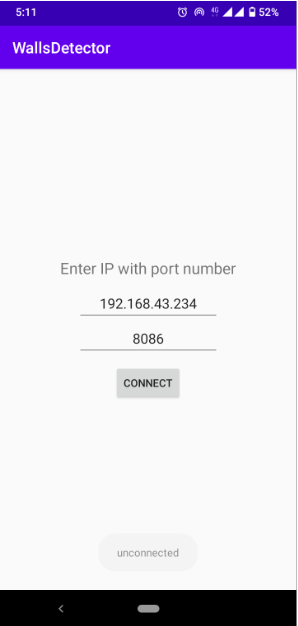
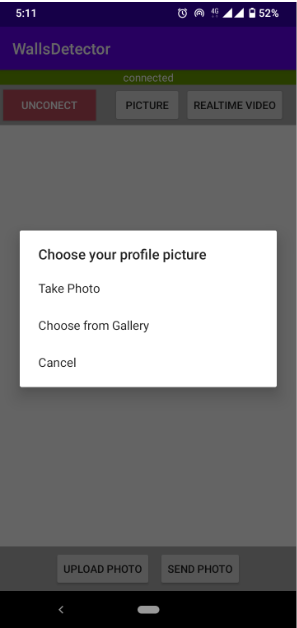
  

Рисунок 4.3 – Основные рабочие окна мобильного приложения «*WallsDetecter*»

Слева на рисунке 4.3 изображено окно подключения. Если подключение прошло успешно, экран перейдёт в рабочую область. Часть рабочей области выделена под отображение рабочих окон. Она является пустой до тех пор, пока не будет выбран тип клиента, нажатием одной из двух соответствующих клавиш.

Кнопка «*unconnected*» отвечает за отключение клиента. Если тип клиента не был выбран, но кнопка «*unconnected*» нажата – сервер закончит работу с безымянным клиентом и перейдёт в режим прослушивания новых подключений.

После выбора типа клиента, на сервер будет отправлено соответствующее сообщение, а приложение откроет один из двух фрагментов: для отправки фотографий или для работы в режиме реального времени.

Тип клиента выбирается нажатием одной из двух кнопок «*picture*» или «*realtime* *video*».

Клик по кнопке «*realtime* *video*» приводит к отображению экрана с камерой и нижней панели инструментов. Нижняя панель инструментов содержит стандартные цвета и несколько стандартных текстур. Отправка и отображение полученного результата с сервера в данном варианте использования происходит автоматически. Таким образом, пользователь может наблюдать за локализацией и окрашиванием стен в реальном времени, а также менять цвет или текстуру окраски в любой момент, не прерывая работу приложения.

Клик по кнопке «*picture*» приводит к отображению экрана с областью отображения картинки и с двумя кнопками: импорт и отправка изображения. Клик по кнопке «*upload photo*» в данном экране открывает панель выбора вариантов загрузки изображения. Здесь пользователь может загрузить готовое изображение из галереи или сделать снимок нового. На рисунке 4.3 (по центру) отображён вид экрана со списком выбора действий для импорта изображения.

Нажатие на кнопку «*send photo*» отправляет изображение на сервер, где происходит дальнейшая обработка. Результат обработки расположен на рисунке 4.3 справа.

В зависимости от размера отправляемого изображения время обработки может варьироваться. Также данный показатель зависит и от качества соединения, которое, в свою очередь, влияет на скорость передачи данных.

Графический интерфейс пользователя со стороны сервера ограничен консольным выводом. На консоль выводятся в основном логи, отражающие происходящие на данный момент работы сервера события, и дополнительные параметры.

На рисунке 4.4 представлен серверный консольный интерфейс после запуска.

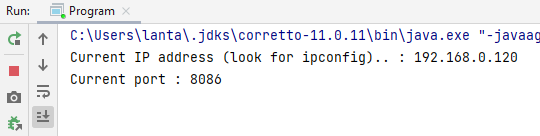


Рисунок 4.4 – Консольный интерфейс серверного приложения при запуске

Сервер начинает работу с отображения в консольном окне *IP*-адреса, на котором он запущен, и порта. Сервер прослушивает любые подключения из вне, находящиеся в той же локальной сети, что и он сам.

На рисунке 4.5 представлен один цикл принятия, обработки и отправки обратно изображения от первого типа клиента.

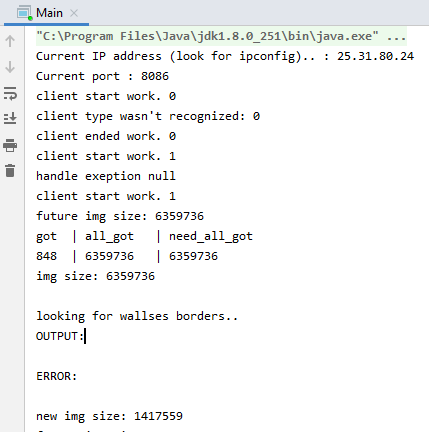


Рисунок 4.5 – Цикл работы сервера с первым типом клиента

Подключение и отключение клиента также регистрируются выводом на консоль информации о типе клиента. На рисунке выше отражено подключение первого типа клиента, получение сервером размерности передаваемого изображения, строка загрузки изображения. Где первое число – количество полученный байт за текущую итерацию, второе число – количество принятых байт всего, третья число – необходимое количество байт, которое совпадает с размерностью изображения.

Дальше следует запуск скрипта и отображение его ошибок, если таковые имели место быть. После подсчитывается размер обработанного изображения, после чего, то отправляется обратно клиенту.

Завершение работы символизируется значением ноль, после получения которого сервер прекращает работу с данным клиентом.

* 1. **Отладка и валидация результатов работы программного комплекса**

Для отслеживания корректности передаваемых данных и общих процессов в целом, как в мобильном приложении, так и в серверном предусмотрена текстовая отчётность хода передачи. На сервере логирование происходит с выводом на консольный интерфейс. Мобильное приложение записывает происходящие процессы как в лог-файл, так и выводит на консоль.

Пример логов при подключении к серверу и отправке типа клиента представлен на рисунке 4.6.

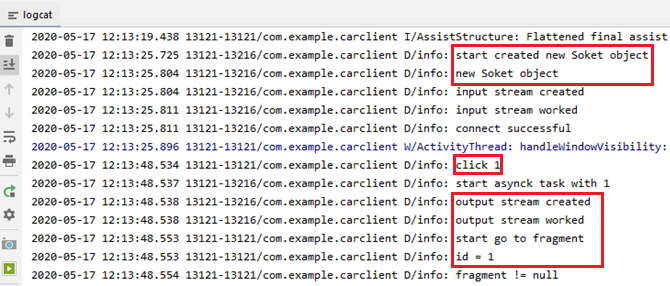


Рисунок 4.6 – Вывод логов при подключении к серверу

Для точного отображения происходящих операций, логи представлены до и после создания объекта подключения. Так же регистрируется создание и работа объектов классов, ответственных за отправление и приём сообщений.

На изображении 4.7 представлены логи, выводящиеся при завершении работы с сервером.

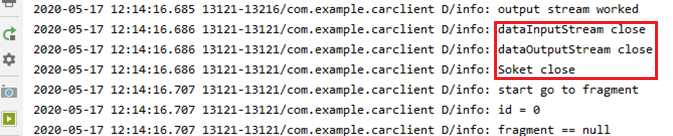


Рисунок 4.7 – Вывод логов при отключении от сервера

Логирование помогает отследить места происхождения ошибок или предотвратить их возможное появление, а также наладить стабильность и корректность работы системы приложений.

Программный комплекс построен таким образом, что на каждом этапе работы с изображением или кадром можно получить текущее состояние последнего. Данный метод отлично подходит для систем, работающих с графической информацией, что позволяет точно определить качество работы методов, и при необходимости, скорректировать параметры.

На рисунке 4.8 представлена *DFD-*модель серверного приложения с отображением ключевых состояний исходного графического образца.

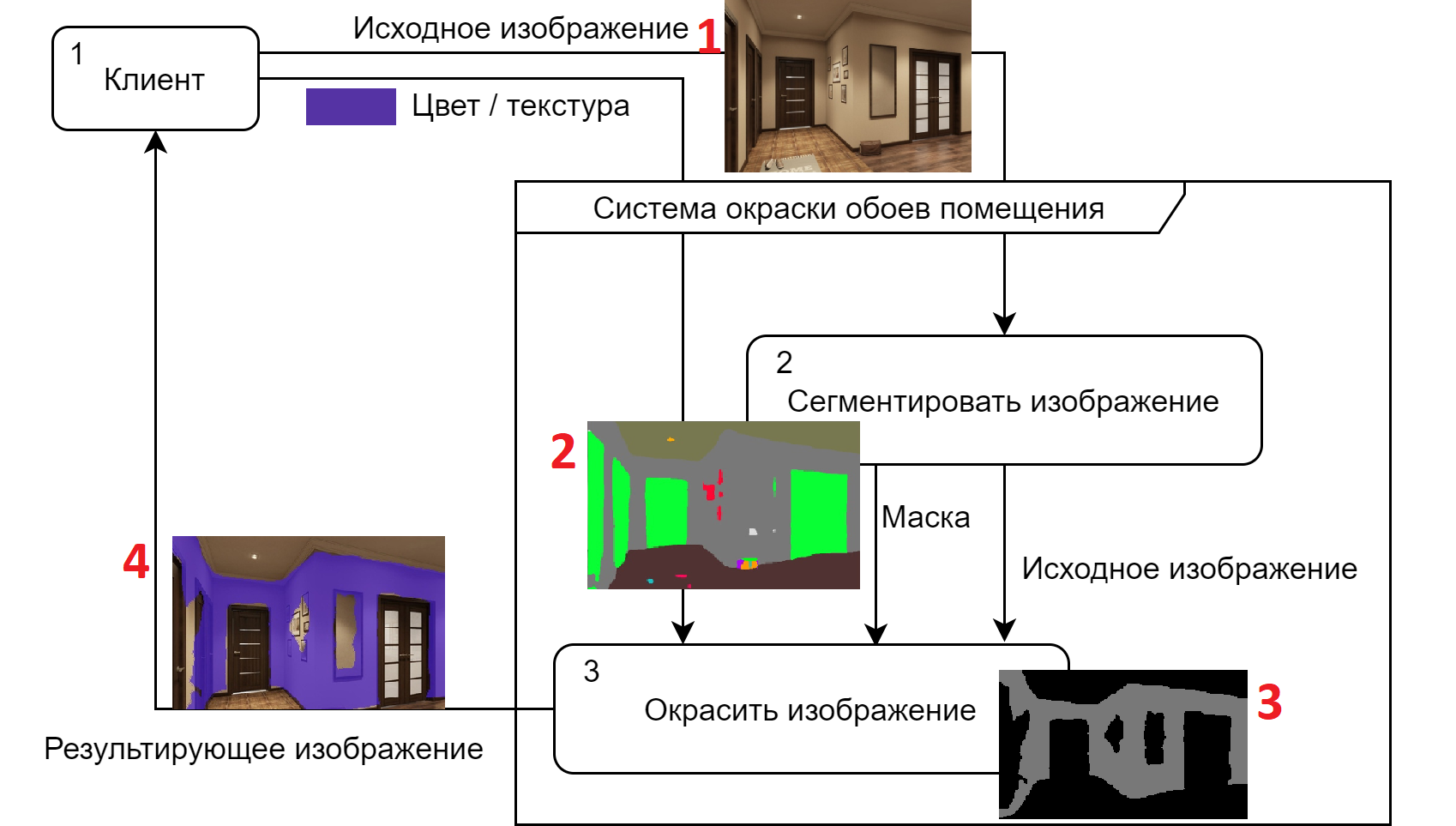


Рисунок 4.8 – *DFD-*модель серверного приложения с ключевыми состояниями исходного графического образца

Ключевые состояния сегментируемого, а затем, окрашиваемого изображения отмечены цифрами во временной очерёдности.

На вход системы подаётся исходное изображение (на рисунке 4.8 отмечено единицей) и цвет (в конкретном рассматриваемом случае). На исходном изображении, прошедшем через этап сегментирования, выделяется маска основных областей изображения (стены, двери, потолок, пол, растения) – на рисунке 4.8 выделено номером два. Далее, этап окраски выделяет маску стены (номер три на рисунке 4.8), добавляет к ней альфа канал и совмещает с исходным изображением. Результат возвращается пользователю (номер четыре на рисунке 4.8).

Таким образом, система подбора цвета и текстуры обоев для финишной отделки помещения справляется с поставленной задачей, а именно, даёт пользователю возможность экстраполировать будущий внешний вид помещения, при выбранном цвете краски для стен или орнаменте обоев, за счёт выявления стен на изображении и подстановки соответствующего цвета или текстуры.

Помимо основной задачи, также представлен ряд удобных дополнений, таких как возможность корректировки точности и качества распознавания стен и возможность отслеживать результат окраски в режиме реального времени.

* 1. **Исследование и анализ программного комплекса**

Для исследования качества сегментации стен на изображениях было проведено несколько экспериментов между тремя алгоритмами, решающими данную задачу:

* нейронная сеть с архитектурой *ResNet*50;
* нейронная сеть с архитектурой *MobileNet*;
* методы компьютерного зрения без нейронных сетей.

Первый эксперимент включает в себя замер времени работы алгоритмов при входном изображении размерностью 1000 пикселей, 510 пикселей, 360 пикселей, 310 пикселей и 220 пикселей.

Второй эксперимент заключается в сравнении качества сегментации трёх алгоритмов при заданных выше значениях размерности входных изображений.

В таблице 4.1 представлены средние показатели времени исполнения каждого из трёх алгоритмов в секундах, при разных размерностях входных изображений.

Таблица 4.1 – Время исполнения алгоритмов компьютерного зрения при разных размерностях обрабатываемого изображения

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Алгоритм / Размер изображения, пиксели | 1000 | 510 | 360 | 310 | 220 |
| *ResNet*50 | 31.7255 | 8.7604 | 4.5548 | 3.3633 | 1.7663 |
| *MobileNet* | 10.2384 | 2.6914 | 1.4496 | 1.0121 | 0.65571 |
| Без нейронной сети | 0.0929 | 0.0259 | 0.0169 | 0.0139 | 0.00994 |

Как видно из таблицы 4.1 нейронная сеть с архитектурой *ResNet*50 исполняется значительно дольше остальных алгоритмов. Использование подобной архитектуры для сегментации в реальном времени – затруднительно.

Нейронная сеть с архитектурой *MobileNet*, в свою очередь, показывает лучшие результаты времени выполнения, но при этом, также не выдаёт скорость близкую к 0.0833 секундам (которая могла бы обеспечить обработку двенадцати кадров секунду).

Алгоритм без нейронной сети работает значительно быстрее алгоритмов с нейронными сетями и отлично подходит для использования в режиме реального времени в разработанном приложении.

На рисунке 4.9 представлен график, наглядно отображающий зависимость времени исполнения алгоритма от размерности входного изображения.

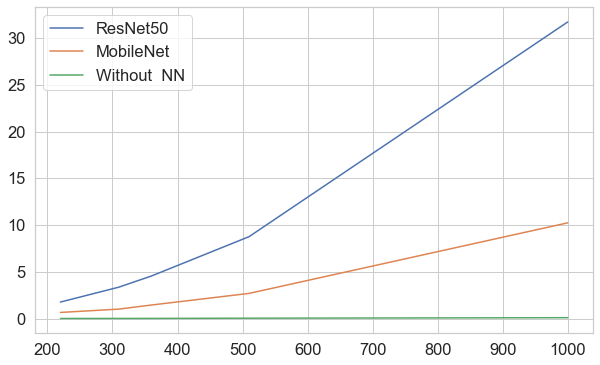


Рисунок 4.9 – Зависимость времени исполнения алгоритмов от размерности входного изображения

Как видно из графика на рисунке 4.9, нейронная сеть с архитектурой *ResNet*50 стремительно увеличивает время обработки после размерности изображения в 500 пикселей по одной из диагоналей. Изображение размерностью в 1000 пикселей обрабатывается данной архитектурой 30 секунд, что крайне долгий результат для использования вне исследовательских целей.

*MobileNet* архитектура хорошо подходит для мобильных устройств за счёт значительно большей скорости обработки кадров. Для обработки кадров в режиме реального времени архитектура *MobileNet* имеет преимущество над *ResNet*50*.*

На рисунке 4.10 отражена разница между точностью сегментации архитектуры *ResNet*50 при изображениях размерностью 1000 пикселей и 220 пикселей.

Рисунок 4.10 – Разница сегментации изображений размерностью 1000 пикселей и 220 пикселей соответственно архитектурой нейронной сети *ResNet*50

На изображении 4.11 видны различия сегментации одного изображения в пяти разных размерностях моделями нейронных сетей *ResNet*50 и *MobileNet*, а также алгоритмом компьютерного зрения без использования нейронной сетисоответственно.

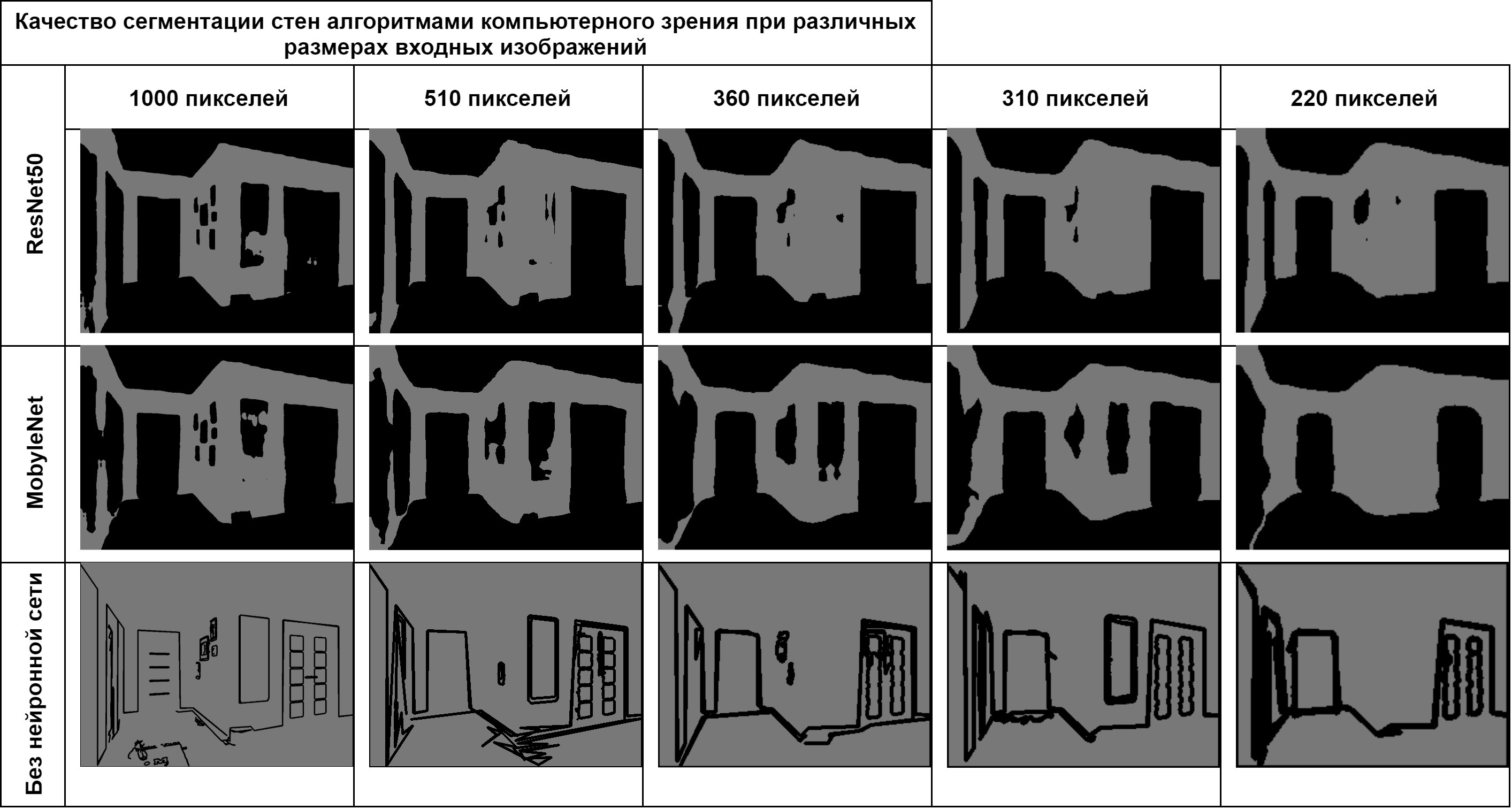


Рисунок 4.11 – Результат сегментации стен алгоритмами компьютерного зрения при различных размерах входного изображения

Как видно из изображения 4.11 – архитектура *ResNet*50более точно сегментирует стены на изображении, в сравнении с двумя другими алгоритмами. При этом, чем выше качество входного изображения, тем подробнее будет сегментация.

Архитектура *MobileNet* уступает по качеству сегментации уступает архитектуре *ResNet*50. *MobileNet* имеет достаточно неровные очертания контуров сегментированной области. Но на максимальной представленной размерности изображения, архитектура выдаёт результат сравнимый с качеством сегментации *ResNet*50.

Алгоритм компьютерного зрения без нейронной сети показал наихудшие результаты по сегментации стен, в сравнении с алгоритмами, использующими нейронные сети. Но, при этом, на больших изображениях он выдавал ровные очертания контуров.

Работа алгоритмов нейронных сетей и алгоритма компьютерного зрения без нейронной сети была протестирована на нескольких типах изображений.

Результаты работы нейронных сетей оказались более качественными в сравнении с результатами работы алгоритма без использования нейронной сети. Как вывод – нейронные сети лучше работают с задачами сегментации изображений, показывая более точные результаты и незначительно проигрывая во времени обработки.