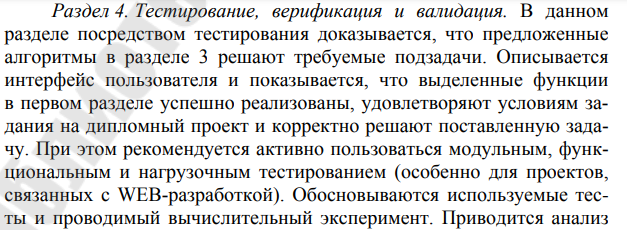
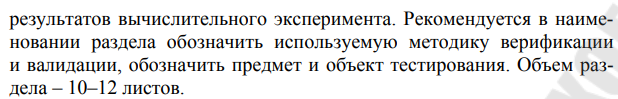
## 4 ВЕРЕФИКАЦИЯ И ОПЫТНАЯ ЭКСПЛУАТАЦИЯ РАЗРАБОТАННОГО ПРОГРАММНОГО КОМПЛЕКСА

* 1. **Описание интерфейса пользователя программного комплекса**

Раздел 4. Тестирование, верификация и валидация. В данном разделе посредством тестирования доказывается, что предложенные алгоритмы в разделе 3 решают требуемые подзадачи. Описывается интерфейс пользователя и показывается, что выделенные функции в первом разделе успешно реализованы, удовлетворяют условиям задания на дипломный проект и корректно решают поставленную задачу. При этом рекомендуется активно пользоваться модульным, функциональным и нагрузочным тестированием (особенно для проектов, связанных с WEB-разработкой). Обосновываются используемые тесты и проводимый вычислительный эксперимент. Приводится анализ результатов вычислительного эксперимента. Рекомендуется в наименовании раздела обозначить используемую методику верификации и валидации, обозначить предмет и объект тестирования. Объѐм раздела 10-12 листов.

Сравнение трёх алгоритмов

Замер времени работы каждого алгоритма

Сравнение качества сегментации каждым алгоритмом

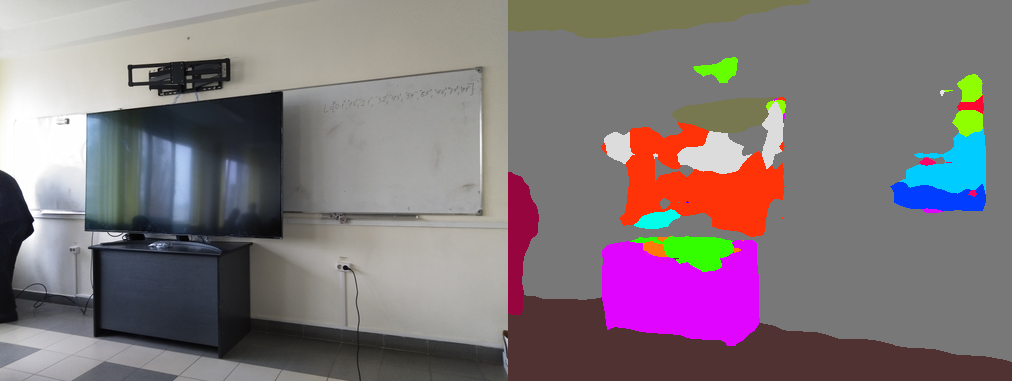


Рисунок 2.11 – Пример результатов работы нейронной сети

На рисунке 2.19 представлены снимки экранов мобильного приложения «*WallsDetecter*».

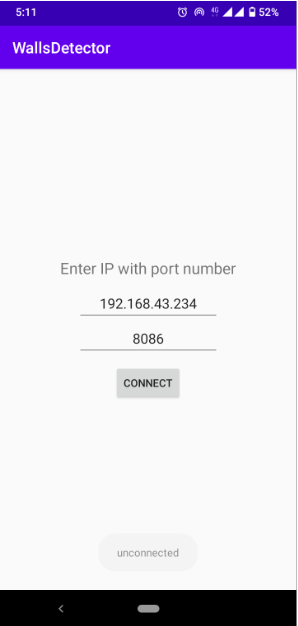
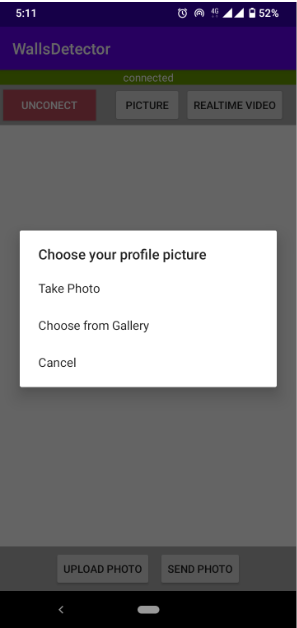
 



Рисунок 2.19 – Основные рабочие окна мобильного приложения

* 1. **Валидация результатов программного комплекса**

*MobileNet* архитектура хорошо подходит для мобильных устройств за счёт значительно большей скорости обработки кадров, при относительно неплохой точности сегментации. Для обработки кадров в режиме реального времени *MobileNet* имеет преимущество над *ResNet.*

На рисунках 2.7 и 2.8 видны различия сегментации моделью *MobileNet* и *ResNet* соответственно.

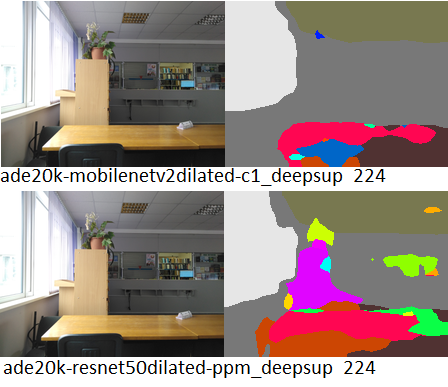


Рисунок 2.7 – Результат сегментации моделью *MobileNet*

Как видно из изображения 2.8 – архитектура *ResNet* более точно и подробно сегментирует изображение, но проигрывает по скорости модели *MobileNet.*

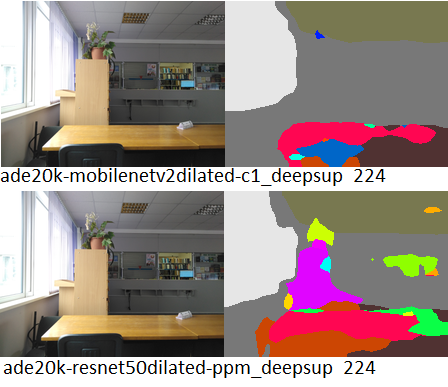


Рисунок 2.8 – Результат сегментации моделью *ResNet*

Работа нейронной сети была протестирована на нескольких изображениях. Результаты работы нейронной сети оказались более качественными в сравнении с результатами работы алгоритма без использования нейронной сети.

Как вывод – нейронная сеть отлично подходит для распознавания стен на изображениях.

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

Уваоораокра

Сервер начинает работу с отображения в консольном окне *IP* адреса, на котором он запущен, и порта. Прослушивает сервер любые подключения из вне.

На рисунке 3.1 представлены снимки экрана мобильного приложения «*WallsDetecter*».

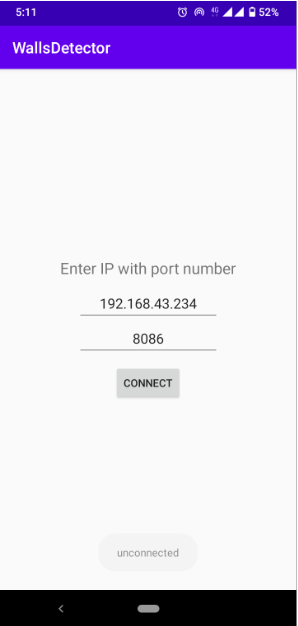
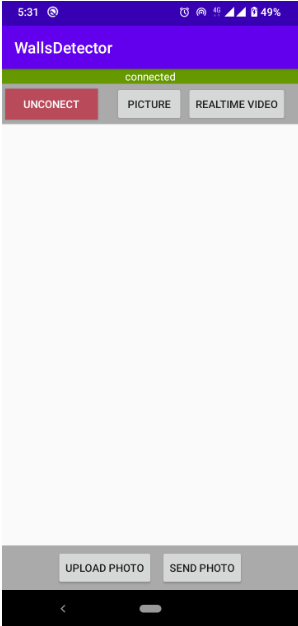
 

Рисунок 3.1 – Основные рабочие окна мобильного приложения

Слева изображено окно подключения. Если подключение прошло успешно, экран перейдёт в рабочую область. Часть рабочей области выделена под отображение фрагментов. 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.

Клик по кнопке «*picture*» (рисунок 3.1 центральная часть) приводит к отображению фрагмента с двумя кнопками импорта и отправления изображения и с областью отображения картинки.

Клик по кнопке «*upload photo*» открывает панель выбора вариантов загрузки изображения. Здесь пользователь имеет выбор – загрузить готовое изображение из галереи или сделать снимок нового.

На рисунке 3.2 представлена работа загрузки изображения. Промежуточные действия фотографирования или поиска в галереи опущены.

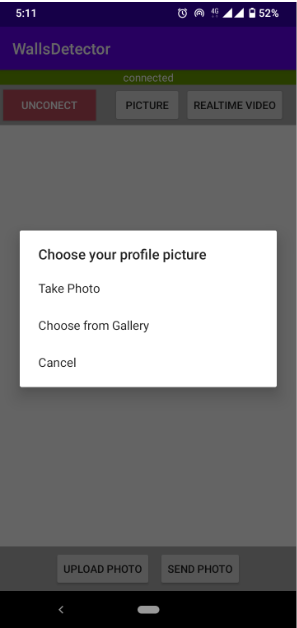
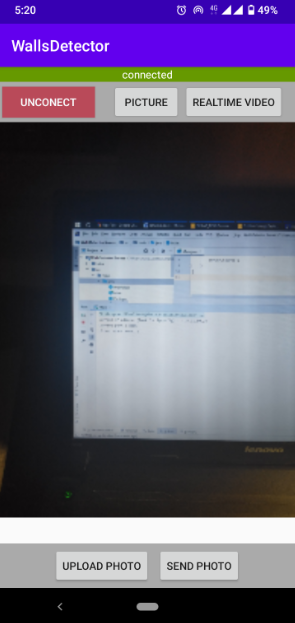
 

Рисунок 3.2 – Работа с функционалом загрузки изображения

Нажатие на кнопку «*send photo*» отправляет изображение на сервер, где происходит дальнейшая обработка.

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.

Для отслеживания корректности передаваемых данных как в мобильном приложении, так и в серверном предусмотрена текстовая отчётность хода передачи. На сервере реализация консольная. Мобильное же приложение записывает происходящие процессы в лог-файл.

Пример логов при подключении к серверу и отправке типа клиента представлен на рисунке 3.3. На изображении 3.4 представлены логи, выводящиеся при завершении работы с сервером.

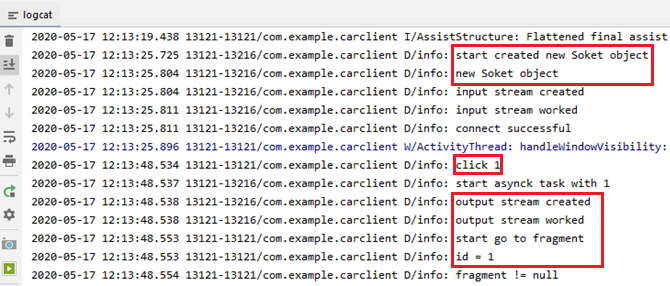


Рисунок 3.3 – Вывод логов при подключении к серверу

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

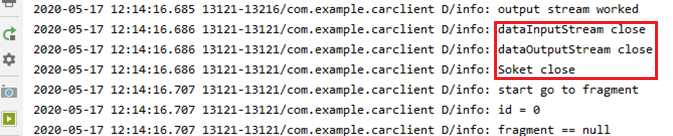


Рисунок 3.4 – Вывод логов при отключении от сервера

Логирование помогает отследить места происхождения ошибок или предотвратить их возможное появление. А также наладить стабильность и правильность в работе приложения.

* 1. **Результат тестирования серверного программного обеспечения**

Со стороны сервера запуск и подключение клиентов отслеживается и отображается консольно.

После запуска сервер выводит строку с *IP* и номером порта, которые необходимо ввести клиентам для подключения. Дальше начинает прослушивать входящие подключения. Как только один из клиентов подключается, сервер выводит в консоль тип подключённого клиента и его *IP*.

На рисунке 3.5 представлен один цикл принятия, обработки и отправки обратно изображения от первого типа клиента.

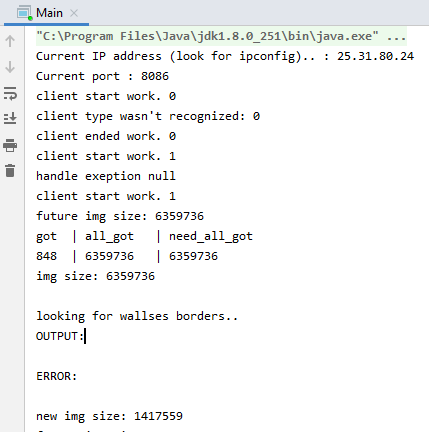


Рисунок 3.5 – Цикл работы сервера с первым типом клиента

На изображении 3.6 представлены варианты распознавания границ средствами библиотеки компьютерного зрения *OpenCv*.

Рисунок 3.6 – Обработанные изображения

Подключение и отключение клиента также регистрируются выводом на консоль информации о сокете 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, совпадает с размерностью изображения.

Дальше следует запуск скрипта, 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.