**ВВЕДЕНИЕ**

В настоящее время рынок информационных технологий активно развивается. Огромные вложения происходят в перспективные проекты, связанные с машинным обучением, решающие задачи как практические, так развлекательные.

С развитием технологий, повседневная жизнь человека обогатилась предметами и инструментами, что были доступны лишь для узкого круга специалистов пару десятилетий назад.

Задачи обнаружения объектов с помощью компьютерного зрения, распознавания тональности человеческой речи, прогнозирования и прочее имеют огромный охват в практическом применении. Именно за счёт того, что программное обеспечение данного рода направленности способно заменить рутинный человеческий труд. Подобные проекты постепенно входят в обиход повседневной жизни. Яркими примерами являются камеры с распознаванием лица в каждом мобильном телефоне, авто-дополнение текста, существующее даже в старых моделях мобильных аппаратов.

Одной из актуальных является задача семантической сегментации помещений на изображениях. Задача требует решение, способное: обнаружить объект в помещении, отнести его к соответствующей группе объектов, сегментировать обнаруженный объект. Объектом может являться стена, пол, потолок или любой другой предмет интерьера.

Подобный инструмент может оказаться полезен в архитектурной сфере, людям, работающим с дизайном интерьеров, конструкторам, архитекторам.

## 1 ОБЗОР И АНАЛИЗ МЕТОДОВ И СРЕДСТВ РАБОТЫ С ИЗОБРАЖЕНИЯМИ

* 1. **Методы анализа и обработки изображений**

Раздел 1. Аналитический обзор существующих методов и 17 средств решения поставленной задачи на основе изучения литературных источников по теме дипломной работы. Примеры решения аналогичных задач, анализ достоинств и недостатков известных решений. Раздел может называться в соответствии с темой дипломной работы и должен содержать постраничные ссылки на все использованные источники. В результате проведенного анализа и в соответствии с требованиями, указанными в задании на дипломную работу, в последнем пункте первого раздела должны быть сформулированы требования к проектируемому программному обеспечению (фактически техническое задание), отражающие цель разработки, требуемый функционал, исходные данные, предполагаемые выходные данные, условия эксплуатации, требования к надѐжности и быстродействию, обоснование выбора средств разработки и используемых технологий. Раздел должен завершаться выводами. Объѐм раздела 12-15 листов.

Поиск элементов на изображениях востребован в сфере анализа данных, а также в развлекательной сфере. Алгоритмы решающие подобную задачу, как правило, основаны на нейронных сетях или общих математических методах.

В некоторых случаях, для распознавания объекта, достаточно программного обеспечения, но в ряде других – необходимо подключать к работе технические возможности. Рассматривая область мобильных приложений, можно заметить, что некоторые приложения используют возможности камеры устройства для улучшения результатов распознавания. В приложениях, работающих с трёхмерным пространством, чья задача анализировать его и, при необходимости, дополнять в реальном времени, не обойтись одним программным обеспечением.

За счёт специализированного технического оснащения камеры в области распознавания трёхмерного пространства лидирует система *IOS*. Существуют приложения на базе *IOS*, решающие задачи поиска предметов в интерьере и взаимодействия с ними. На базе *Android* подобных решений нет, в первую очередь из-за сложности реализации, без сторонней поддержки технического оснащения мобильного устройства.

Существует два основных варианта решения программным путём задачи распознавания трёхмерных объектов на плоских изображениях. Первый – обученная нейронная сеть. Второй – алгоритмы без использования нейронных сетей.

Задачи распознавания объектов на изображениях хорошо решаемы с помощью свёрточных нейронных сетей [3]. Однако отличительная черта проблемы состоит в небольшом количестве особенных характеристик и признаков распознаваемого объекта, что усложняет создание корректной обучающей выборки.

Алгоритмы без использования нейронных сетей не уступают в качестве решения нейронным сетям, обученным на ограниченном количестве данных. Но при этом не требуют значительных ресурсных затрат.

* 1. **Обзор математических алгоритмов *Canny* и Хафа**

Со стороны компьютерного зрения, без использования нейронных сетей, локализация стен может быть осуществлена стандартными методами обработки контуров. Достаточное число математических методов в реализованном виде можно найти в библиотеке компьютерного зрения *OpenCv* [4]*.*

*OpenCV,* как библиотека компьютерного зрения, хорошо подходит для предварительной обработки изображения за счёт развитого функционала и обширной докумментации.

Основными встроенными методами, подходящими для решения задачи, выступают метод *Canny* и метод преобразования Хафа.

Метод *Canny* – метод обнаружения края элементов на изображении.

Метод *Canny* является многоэтапным алгоритмом.

На первом этапе алгоритма происходит подавление шума методом Гаусса. Так как обнаружение контуров чувствительно к шуму. Дальше следует нахождение интенсивности изображения за счёт фильтрации ядром Собеля по вертикали и по горизонтали. *OpenCV* имеет отдельную функцию Собеля, но в совокупности с Canny она работает эффективнее. Этот же этап включает нахождение градиента. После получения величины и направления градиента выполняется полное сканирование изображения для удаления любых нежелательных пикселей, которые могут не составлять края. Для этого на каждом пикселе пиксель проверяется, является ли он локальным максимумом в его окрестности в направлении градиента. После этого этапа на выходе образуется двоичное изображение с тонкими краями.

Завершающим этапом является нахождение пороговое значение гистерезиса. На этом этапе решается, какие из ребер на самом деле являются ребрами, а какие нет. Создаются два значения *maxVal* и *minVal*.

Любые края с градиентом интенсивности больше *maxVal* обязательно будут краями, а те, что ниже *minVal*, обязательно не будут краями. На этом этапе также удаляются небольшие шумы пикселей при условии, что края являются длинными линиями.

Преобразование Хафа – вычислительный алгоритм, применяемый для параметрической идентификации геометрических элементов растрового изображения.

Функции [*HoughLines*](https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga46b4e588934f6c8dfd509cc6e0e4545a) и [*HoughLinesP*](https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga8618180a5948286384e3b7ca02f6feeb) библиотеки *OpenCV*– преобразование Хафа. Предназначены для обнаружения линий на изображении. Чтобы применить преобразование, сначала желательна предварительная обработка края. Для преобразований Хафа линии выражаются в полярной системе координат. В общем случае линию можно обнаружить, определив количество пересечений между кривыми. Чем больше пересекающихся кривых, тем больше точек на линии, представленной этим пересечением. Общий случай даёт возможность определить порог минимального количества пересечений, необходимых для обнаружения линии.

Метод Хафа отслеживает пересечение кривых каждой точки изображения. Если количество пересечений превышает некоторый порог, то он объявляет это как строку с параметрами точки пересечения.

Существуют стандартное и вероятностное преобразование линии Хафа. OpenCV реализует два типа преобразований. На рисунках 2.1 и 2.2 отображена разница между стандартным преобразованием и вероятностным преобразованием соответственно.



Рисунок 2.1 – Результат стандартного преобразования Хафа

Стандартное преобразование Хафа реализовано функцией HoughLines в OpenCV. Вероятностное преобразование линии Хафа является более эффективной реализацией преобразования линии. Он дает в качестве вывода экстремумы обнаруженных линий. В OpenCV данный тип преобразования реализован функцией HoughLinesP.

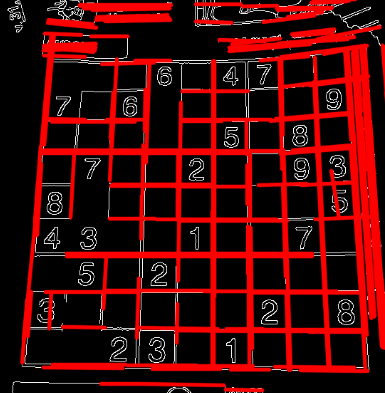


Рисунок 2.2 – Результат вероятностного преобразования Хафа

* 1. **Задача Семантической сегментации изображений**

В настоящее время информационные технологии проникают во все сферы деятельности современного общества: от робототехники и медицины, до игр и повседневного быта. В области анализа изображений остаются актуальными такие задачи как: семантическая сегментация классов материала *(stuff segmentation)* исемантическая сегментация материалов и классов вещей (*panoptic segmentation*) [1]. Где вещи – это объекты определенного размера и формы, которые часто состоят из частей, а классы материалов – это фоновые материалы, которые определяются однородными или повторяющимися узорами мелкомасштабных свойств, но не имеют конкретной или отличительной пространственной протяженности или формы.

Решения данных задач в последующем применяются, например, для анализа окружающей местности робототехникой, в медицинской визуализации, спутниковой съемке, и другом.

Мобильное приложение, реализующее технологию распознавания стен, сможет стать инструментом, облегчающим составление дизайн-проекта на начальном этапе его проектирования. А именно позволит экспериментировать с цветом и текстурой распознанных плоскостей помещения (потолок, стены, пол) в режиме реального времени. Приложение в первую очередь предназначено для упрощения работы архитекторов и дизайнеров интерьера при планировании цветовых палитр в проектах.

На рынке уже существуют инструменты подобного типа. К таковым относится приложение для замера размеров прямоугольных областей, таких как дверные и оконные проёмы. Инструмент основан на компьютерном зрении и технологии *AR* (*Augmented reality*).

Задача распознавания фона, а именно стен, является задачей семантической сегментации материала.

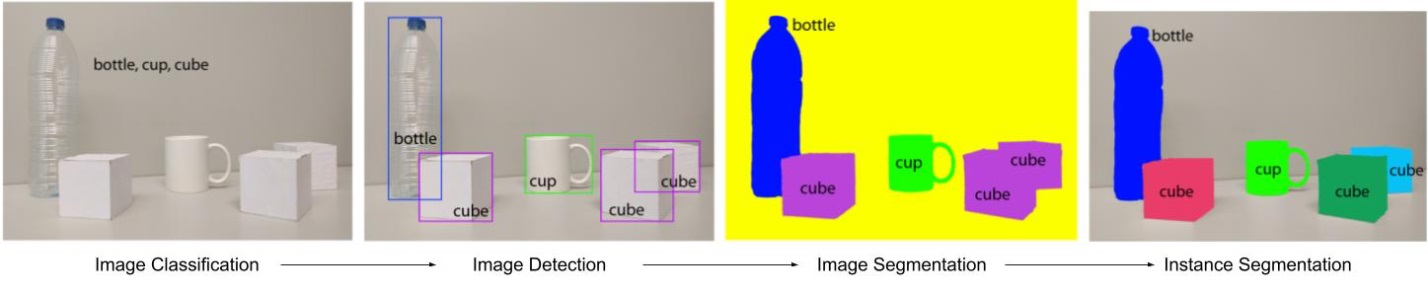
В задачах семантической сегментации каждому пикселю изображения присваивается метка, определяющая принадлежность его к классу. Подобный подход позволяет не просто понять примерную область расположения объекта в кадре, а очертить его контур.

Стены относятся к классам материалов – фоновым материалам, которые определяются однородными или повторяющимися узорами мелкомасштабных свойств, но не имеют конкретной или отличительной пространственной протяженности или формы [2].

Семантическая сегментация – это задача присвоения семантической метки, такой как «дорога», «небо», «человек», «собака», каждому пикселю изображения.

Назначение семантических меток требует точного определения контура объектов и, таким образом, предъявляет гораздо более строгие требования к точности локализации, чем другие задачи распознавания визуальных объектов, такие как [классификация на уровне изображения](https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html) или [обнаружение на уровне ограничивающего прямоугольника](https://research.googleblog.com/2017/06/supercharge-your-computer-vision-models.html).

На рисунке 1.1 представлены типы распознавания объектов на изображениях.



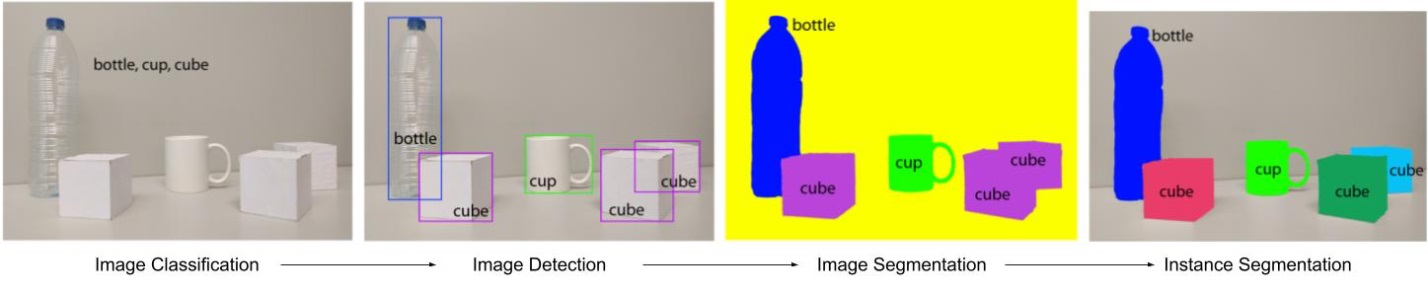


Рисунок 1.1 – Типы распознавания объектов на изображениях

Инстанс-сегментация (*Instance segmentation*) – в отличие от семантической сегментации, в этой задаче каждый объект внутри одного класса выделяется отдельными сегментами. Например, если на изображении пять кошек, две собаки и десять растений, семантическая сегментация просто выделит все области, на которых есть кошки, собаки или растения, не разделяя отдельные объекты внутри каждого класса (определит, что на изображении есть кошки, собаки и растения), в то время как инстанс-сегментация выделит каждую кошку, собаку и растение как отдельный объект.

Паноптическая сегментация (*Panoptic segmentation*) – объединяет задачи семантической и инстанс-сегментации. Также в задаче паноптической сегментации каждому пикселю изображения должна быть присвоена ровно одна метка.

Распознавание стен на кадрах и последующая замена их цвета и текстуры решаемо семантической сегментацией на уровне типа объекта.

* 1. **Нейронные сети для обработки изображений**

В настоящее время проводится огромное количество исследований посвящённых созданию новых моделей нейронных сетей, для решения задач семантической сегментации в том числе.

***1.3.1*** Модели нейронных сетей, созданных для сегментации, разделяют на кодировщик и декодер, где кодеры обычно модифицируются непосредственно из сетей классификации, а декодеры состоят из окончательных сверток и повышающей дискретизации [5].

Дописать про кодер, декодер, почему так и как

Использованная для решения задачи модель, относится к данной категории и состоит из кодировщика и декодера.

### Кодировщик выполняет итерацию по входному предложению по одному токену (например, слову) за раз, на каждом временном шаге выводя вектор «вывода» и вектор «скрытого состояния». Затем скрытый вектор состояния передается на следующий временной шаг, а выходной вектор записывается. Кодер преобразует контекст, который он видел в каждой точке последовательности, в набор точек в многомерном пространстве, которые декодер будет использовать для генерации значимого вывода для данной задачи.

### Декодер генерирует ответное предложение поэтапно. Он использует векторы контекста кодировщика и внутренние скрытые состояния для генерации следующего слова в последовательности. Он продолжает генерировать слова, пока не *выведет EOS\_token* , представляющий конец предложения. В декодер можно встроить [механизм внимания](https://arxiv.org/abs/1409.0473), чтобы помочь ему «обратить внимание» на определенные части ввода при генерации вывода.

При выборе конкретного кодера и декодера был проведён анализ и сравнены общие характеристики моделей. Из числа моделей, показывающих хорошие результаты в качестве сегментации, можно выделить кодировщик *ResNet*(50/101) в связке с декодером *PPM\_deepsup.* Хорошие характеристики скорости распознавания имеет кодировщик *MobileNetV2dilated* в связке с декодером *C1\_deepsup.*

На изображении 2.6 представлено сравнение характеристик выбранных архитектур кодировщика и декодера.

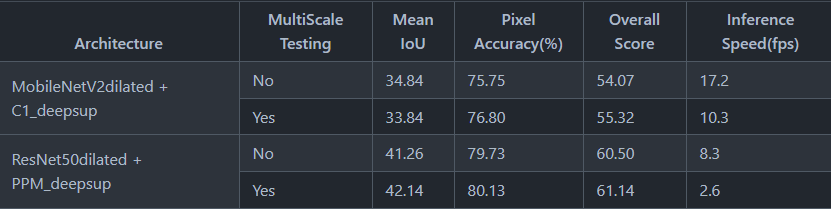


Рисунок 2.6 – Характеристики *ResNet* и *MobileNet*

Подробнее описать происходящее на таблице

Задача распознавания стен относится к задачам *Indoor Object Segmentation* и *Scene Segmentation.*

### Ниже описаны три современные модели для сегментации сцен.

***PSPNet* –** это сеть разбора сцены, которая объединяет глобальное представление с модулем *Pyramid Pooling Module* (*PPM*).

***UPerNet*** – это модель, основанная на *Feature Pyramid Network* (*FPN*) и *Pyramid Pooling Module* (*PPM*). Ему не нужна расширенная свертка, оператор, требующий много времени и памяти. Без наворотов он сравним или даже лучше по сравнению с *PSPNet*, но требует гораздо меньшего времени обучения и меньшего объема памяти графического процессора.

***HRNet* –** это недавно предложенная модель, которая сохраняет представления с высоким разрешением по всей модели без традиционного узкого места.

***1.3.2*** Данные, на которых была обучена сеть имеют важную роль. Количество хороших наборов данных ограничено сложностью нормализации данных. А в случае с задачей сегментации, также и созданием дополнительной разметки на изображениях.

*COCO* (*Common Objects in Context*) – это крупномасштабный набор данных для задач обнаружения и сегментации [6].

*COCO* имеет несколько функций:

– сегментация объекта;

– признание в контексте;

– сегментация пиксельного материала;

– 330 000 изображений (помечено > 200 000);

– 1,5 миллиона экземпляров объектов;

– 80 категорий объектов.

На изображении 2.9 представлен пример размеченных изображений из набора данных *COCO.*



Рисунок 2.9 – Пример изображений набора данных *COCO*

[*ADE20K*](http://groups.csail.mit.edu/vision/datasets/ADE20K/index.html) – это крупнейший набор данных с открытым исходным кодом для семантической сегментации и анализа сцены, выпущенный командой *MIT Computer Vision*[7].

# *ADE20K* состоит из изображений из баз данных *SUN* и *Places*. Изображения полностью аннотированы объектами. Многие изображения также содержат части объектов и под-части объектов. Изображения анонимны – лица и номерные знаки размываются.

Всего для оценки включено 150 семантических категорий, которые включают такие вещи, как небо, дорога, трава, и дискретные объекты, такие как человек, машина, кровать. На изображениях есть неравномерное распределение объектов, имитирующих более естественное появление объектов в повседневной сцене.

Последняя версия набора данных содержит:

– 27 574 изображения (25 574 для обучения и 2000 для тестирования), охватывающих 365 различных сцен;

– 707 868 уникальных объектов из 3 688 категорий, а также их определение и иерархия *WordNet*;

– 193 238 аннотированных частей и под-частей объектов;

– Аннотации многоугольников с атрибутами, временем аннотации, порядком глубины.

На рисунке 2.10 отражено несколько примеров, показывающих изображения, сегментирования объектов:

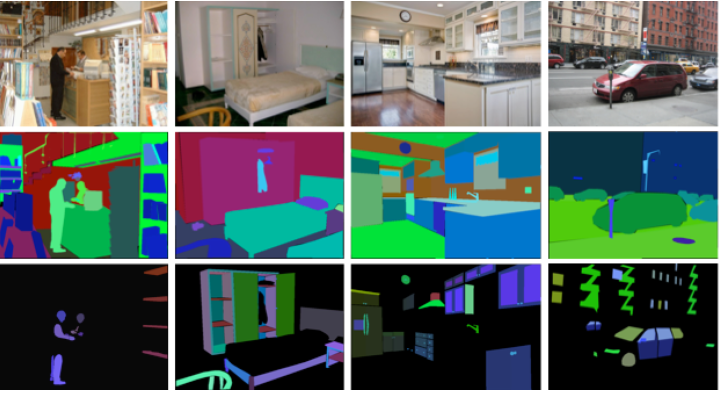


Рисунок 2.10 – Пример изображений набора данных [*ADE20K*](http://groups.csail.mit.edu/vision/datasets/ADE20K/index.html)

Добавить сравнение библиотек

Сформулировать точное описание решаемой задачи.

Для поиска наилучшего результата было написано два скрипта: один решает задачу поиска стен на изображениях методами библиотеки *OpenCv* не включающими нейронные сети, другой – использует обученную нейронную сеть.

Следующим этапом является использование результатов работы нейронной сети на мобильном устройстве. Есть два варианта реализации: разместить нейронную сеть непосредственно на *Android* устройстве, или разместить нейронную сеть на персональном компьютере, организовав клиент-серверное взаимодействие между мобильным устройством и персональным компьютером.

Второе решение выгодно с точки зрения качества распознавания стен. Так как производительность хорошего персонального компьютера превышает мобильную, есть возможность использовать более сложную архитектуру нейронной сети, не теряя во времени распознавания.

Для мобильного устройства была выбрана сеть с моделью кодировщика *MobileNetV2dilated,* моделью декодировщика *C1\_deepsup,* обученная на данных из набора [*ADE20K*](http://groups.csail.mit.edu/vision/datasets/ADE20K/index.html) .

Для использования на персональном компьютере была выбрана сеть с моделью кодировщика *Resnet50dilated,* моделью декодировщика *ppm\_deepsup,* обученная на данных из набора [*ADE20K*](http://groups.csail.mit.edu/vision/datasets/ADE20K/index.html).

## 2 АРХИТЕКТУРА ПРОГРАММНОГО КОМПЛЕКСА ЛОКАЛИЗАЦИИ СТЕН НА ИЗОБРАЖЕНИЯХ

* 1. **Функциональная модель программного комплекса локализации стен на изображениях**

Впрпапо

Раздел 2. Архитектура программного обеспечения. Должны быть выделены и описаны основные функции, реализуемые разрабатываемым программным обеспечением, определены структурные элементы программы, спроектирована внутренняя архитектура и заданы схемы разделения данных. В результате моделирования предметной области должны быть приведены основные сущности и связи между ними, создана информационная модель предметной области. В результате должны быть описаны внутренние и внешние интерфейсы взаимодействия каждой программной части, модель данных и структура базы данных (если требуется). В наименовании раздела рекомендуется указать связь с темой работы и отразить архитектуру, созданного программного обеспечения. Объѐм раздела 10-12 листов.

* 1. **Архитектура компонентов программного комплекса локализации стен на изображениях**

Енпгпг

## 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ ЛОКАЛИЗАЦИИ И ОКРАСКИ СТЕН НА ИЗОБРАЖЕНИЯХ

* 1. **Локализация стен методами компьютерного зрения без использования нейронных сетей**

Раздел 3. Структура и основные алгоритмы созданного программного обеспечения. На основании спроектированной архитектуры выделяются основные блоки (модули) программы, приводится структура программы, описываются необходимые интерфейсы и классы, объясняются алгоритмы основных методов. При этом рекомендуется использовать современные структурные паттерны проектирования. Автоматизируя выделенные функции, с учѐтом технических требований, описывается интерфейс пользователя, дерево достижимости основных функций программы, приводится формат ввода исходных данных и формат получение выходных данных. В наименовании раздела можно отразить содержащуюся в нѐм структуру и основные алгоритмы программного продукта. Объѐм раздела 10-12 листов.

Рассказать подробней о каждом из этапов

Разработка мобильного приложения для подбора цвета и текстур обоев для финишной отделки помещения требует следующих этапов:

* создание алгоритма локализации стен на основе методов компьютерного зрения без использования нейронной сети;
* применение обученной нейронной сети для задачи сегментации стен, её валидация и последующее сравнение результатов с методами компьютерного зрения без использования нейронных сетей;
* разработка мобильного приложения способного использовать алгоритмы, как с нейронной сетью, так и без для обработки входящих изображений;
* создание алгоритма окраски и наложения текстур на сегменты изображения;
* валидация и верификация используемых алгоритмов за счёт вывода результатов работы на графический интерфейс мобильного приложения.

Для поиска наилучшего результата было написано два скрипта: один решает задачу поиска стен на изображениях методами библиотеки OpenCv не включающими нейронные сети, другой – использует обученную нейронную сеть.

Следующим этапом является использование результатов работы нейронной сети на мобильном устройстве. Есть два варианта реализации: разместить нейронную сеть непосредственно на Android устройстве, или разместить нейронную сеть на персональном компьютере, организовав клиент-серверное взаимодействие между мобильным устройством и персональным компьютером.

Второе решение выгодно с точки зрения качества распознавания стен. Так как производительность хорошего персонального компьютера превышает мобильную, есть возможность использовать более сложную архитектуру нейронной сети, не теряя во времени распознавания.

В настоящем проекте библиоткека *OpenCV* решает задачи обработки изображения, а также поиска и классификации на нём прямолинейных контуров. OpenCV имеет ряд полезных методов для решения задачи нахождения контуров.

Обработка входного изображения производится в несколько этапов, каждый из которых, отдаёт результат собственной работы следующему. Принцип конвейерной ленты позволяет добиться наилучшего результата. Гибкость архитектуры позволяет менять местами или добавлять новые методы обработки, тем самым облегчая разработку.

Первый этап представляет первичную обработку изображения и создание маски над ним. Затем следует функция поиска контуров(*findContours*) библиотеки *OpenCV*, что определяет контур объектов на изображении, основываясь на градиентах граничных частей. Основой для первого этапа является изображение, обработанное методом *Canny* библиотеки *OpenCV*. *Canny* – метод обнаружения края элементов на изображении. Является многоэтапным алгоритмом. На первом этапе алгоритма происходит подавление шума методом Гаусса. Так как обнаружение контуров чувствительно к шуму. Дальше следует нахождение интенсивности изображения за счёт фильтрации ядром Собеля по вертикали и по горизонтали. Этот же этап включает нахождение градиента. На выходе образуется чёрно-белое контурное изображение – рисунок 2.3.

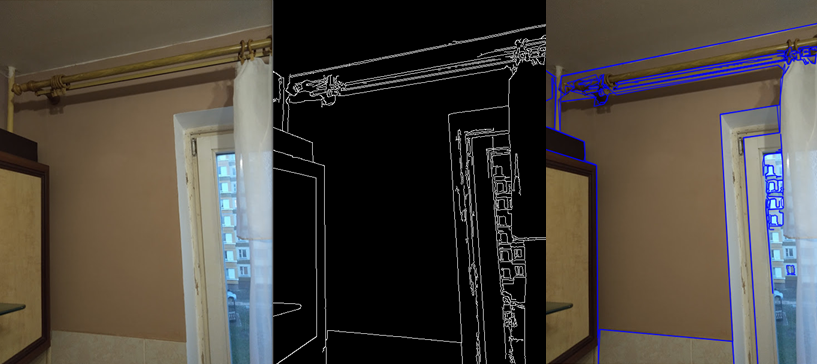


Рисунок 2.3 – Обработка изображения методом *Canny*

Второй этап подразумевает работу непосредственно с самими контурами. На данном этапе происходит отсеивание небольших объектов путём вычисления площадей их контуров. Здесь среднее арифметическое показывает преимущество над медианным средним за счёт достаточного числа небольших шумовых объектов на изображении.

Третий этап работает непосредственно с найденными контурами. На данном этапе идёт сокращение ключевых точек контуров и последующая аппроксимация. Контур приобретает чёткую форму, лишаясь незначительных искривлений (рисунок 2.4).

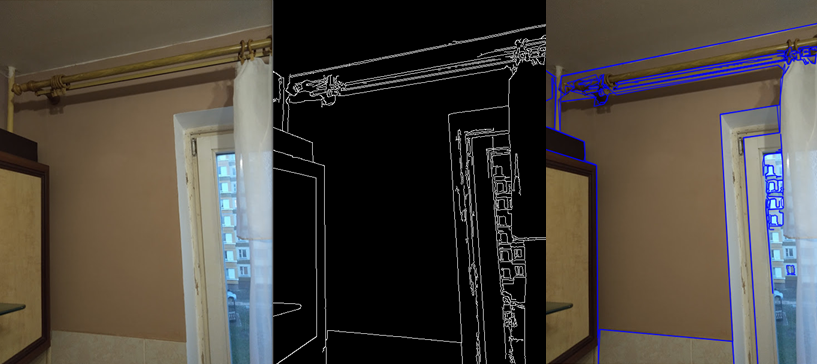


Рисунок 2.4 – Контур изображения после аппроксимации

Четвёртый этап включает функции сортировки контуров на вертикальные и горизонтальные, с учётом отклонения от нормали, а также нахождения точек пересечения двух типов контурных линий. На данном этапе используется функция преобразования Хафа. Преобразование Хафа – вычислительный алгоритм, применяемый для параметрической идентификации геометрических элементов растрового изображения. Чтобы применить преобразование, сначала желательна предварительная обработка края. Для преобразований Хафа линии выражаются в полярной системе координат. В общем случае линию можно обнаружить, определив количество пересечений между кривыми. Чем больше пересекающихся кривых, тем больше точек на линии, представленной этим пересечением. Общий случай даёт возможность определить порог минимального количества пересечений, необходимых для обнаружения линии. Результат представлен на рисунке 2.5.



Рисунок 2.5 – Изображение после обработки методом Хафа

Найденное очертание области графически отображаются на исходном изображении и выводятся пользователю.

Программное обеспечение выполняет задачу распознавания плоскостей на изображениях. Система создана с использованием технологии компьютерного зрения. Распознавание плоскостей на фотографиях является первым этапом в построении системы обнаружения и замены части изображения, в данном случае стены. Потенциально данный инструмент способен облегчить работу дизайнеров интерьера при подборе цветовой гаммы.

Как результат – математические методы без использования нейронных сетей хорошо справляются с точностью определения контуров искомых плоскостей, но не способны отличить стены от других поверхностей, что является существенным недостатком.

* 1. **Локализация стен обученной нейронной сетью**

Для работы с нейронной сетью использовалась библиотека *PyTorch.* За основу взяты модели *MobileNet* и *ResNet,* реализовано переключение между ними.

*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.

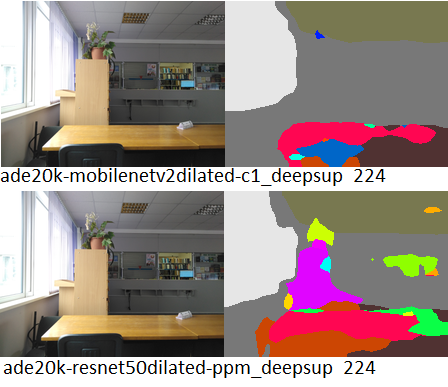


Рисунок 2.7 – Результат сегментации моделью *MobileNet*

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

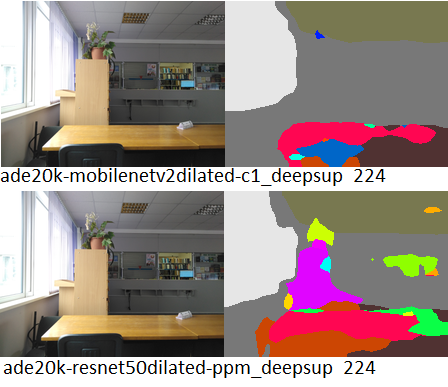


Рисунок 2.8 – Результат сегментации моделью *ResNet*

Скрипт работы с нейронной сетью разбит на несколько секций.

Первая секция – загрузка предварительно сохранённых кодера и декодера. Они и являются основными составляющими обученной нейронной сети и содержат в себе весовые значения.

Далее следует секция объединения кодера и декодера в сегментационный модуль а также переключение модели в режим тестирования методом *eval*.

Следующая часть скрипта посвящена предварительной обработке изображения:

– изменение до размера 510 пикселей по большей стороне;

– конвертация в *RGB*;

– нормализация с помошью параметров *std* и *mean*;

– преобразование в массив и создание тензора на основе данного массива.

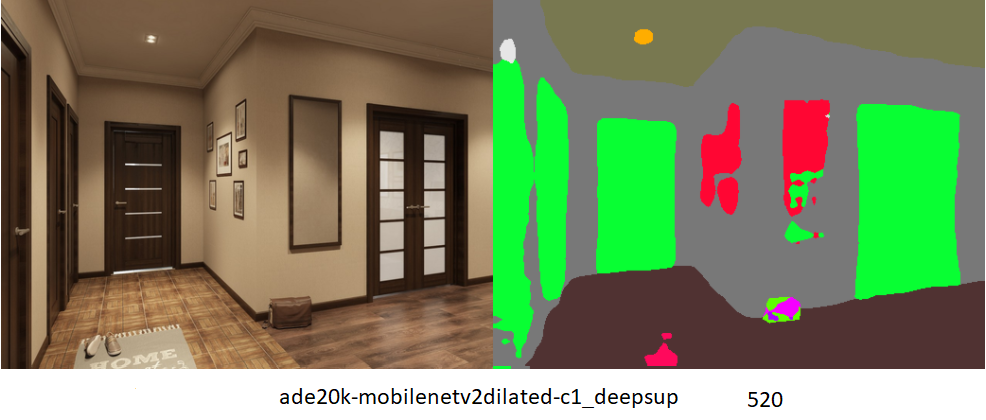
После происходит вызов нейронной сети с флагом no\_*grad* – отключение градиентов.

Завершающая секция производит пост обработку результирующего изображения.

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. Но для достижения эстетически приемлемого результата выходному изображению нужна пост обработка в виде сглаживания контуров. Данная проблема решаема методами библиотеки компьютерного зрения *OpenCv*.

На рисунке 2.11 представлены сегментированные изображения, как результаты работы полученной нейронной сети.



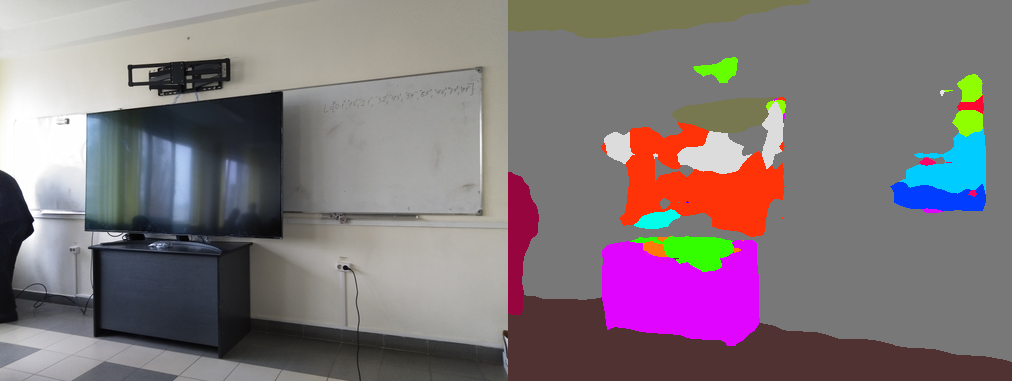


Рисунок 2.11 – Пример результатов работы нейронной сети

* 1. **Программное обеспечение на стороне сервера**

Для применения результатов сегментации кадров в реальных условиях было создано два приложения: сервер и мобильный клиент. Сервер принимает поступающие от клиента изображения, пробрасывает их через нейронную сеть и возвращает исходный результат клиенту.

Оба приложения используют язык программирования java.

Серверное приложение содержит:

–класс сетевого взаимодействия *TCP/IP*;

– класс работы с изображениями, в том числе их чтением и записью;

– нейронную сеть, решающую задачу распознавания контуров на изображениях;

– скрипт взаимодействия с нейронной сетью.

Схема взаимодействия отдельных частей программного комплекса изображена на рисунке 2.12.

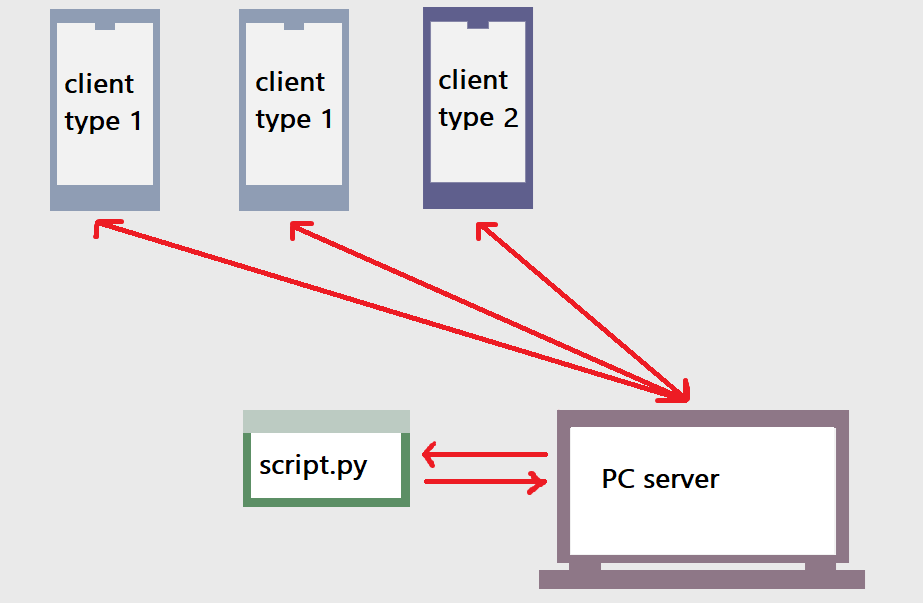


Рисунок 2.12 – Визуализация взаимодействия программного обеспечения

Серверное приложение разворачивается на персональном компьютере пользователя – ПК. Само серверное приложение состоит из двух частей. Первая отвечает за взаимодействие с клиентом – приём и отправку изображений. Вторая – за непосредственную обработку изображения.

После запуска сервер становится в позицию ожидания подключений. Приложение поддерживает многопользовательский режим, за счёт выделения отдельного потока работы для каждого пользователя. После принятия подключения клиента сервер выделяет ему отдельный поток, в котором обрабатывает все его запросы, а в основном потоке продолжает ожидать следующие подключения.

Сервер имеет консольное отображение информации, преимущественно используемое для отладки и тестирования.

Для подключения клиента к серверу необходимо указать *IP*-адрес (*Internet Protocol Address*) и порт. Порт задан постоянной переменной и не изменим при каждом новом запуске.

Существует несколько вариантов узнать *IP*-адрес ПК. Одним из представительных решений может оказаться широковещательная рассылка. Более простой способ – использовать статический *IP*, который можно узнать, вписав команду *ipconfig* в консоль (*Windows*).

На стороне сервера есть четыре класса: *Server*, *ImgHelper*, *PyHelper* и класс, содержащий функцию *main*.

В классе *Server* находятся методы, обрабатывающие входящие подключения от двух типов клиентов. Методы, находящиеся в классе *Server* можно увидеть на рисунке 2.13.

Первый тип клиента работает с изображениями, второй – с видео соответственно. Приняв подключение клиента, сервер создаёт новый поток, в котором проверяет качество связи, узнаёт тип клиента и, далее, обрабатывает соответствующие типу клиента запросы.

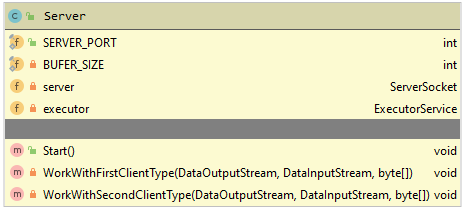


Рисунок 2.13 – Сигнатуры методов класса *Server*

Для передачи информации по сети используется класс *ServerSocket* и его методы *Accept, GetInputStream, GetOutputStream* и *Close*.

Для обработки входящих данных типа *byte* а также их отправки обратно на клиент используются классы *DataInputStream* и *DataOutputStream* соответственно.Вышеупомянутые классы являются обёртками для классов *BufferedInputStream* *BufferedOutputStream*, а те в свою очередь, наследуются от *InputStream* и *OutputStream*. Каждое поколение классов имеет своё преимущество. Задачи решаемые настоящим приложением не требуют низкоуровнего вмешательства в работу сетевого взаимодейcтвия, благодаря чему можно использовать уже готовые методы для работы со строками, числами и массивами байт.

Сервер работает с изображениями, посредством класса *BufferedImage*.

Объект класса *BufferedImage* сериализуется в массив байт для отправки клиенту осуществляется с помощью класса *ImageIO* и его метода *Write*, который принимает *ByteArrayOutputStream* и записывает в него изображение, представленное *BufferedImage.* Затем объект *ByteArrayOutputStream* преобразовывается в массив байт.

При передаче изображения, клиент, сперва, отправляет размерность передаваемого массива байт. А затем, отправляет сам массив, который на стороне сервера принимается порционно.

Существует несколько подходов реализации отправки крупных данных по сети.

Первый подразумевает одно подключение для одной передачи. То есть как только файл, изображение, видео или другой документ были переданы, соединение закрывается. Этот способ надёжен, так как при разрыве соединения серверу, считывающему данные в этот момент, приходит значение-ключ, сигнализирующее непосредственно разрыв соединения, который является и сигналом к завершению передачи файла. Как минус данного подхода – ресурсная затратность.

Второй подход – использовать суррогатный ключ завершения передачи, без разрыва самой связи. Минусом данного способа является сложность составления достаточно надёжного ключа, который бы не совпал с передаваемой частью данных. А также необходимость постоянного отслеживания появления данного ключа от принимающей стороны.

Ещё одним способом является первичная передача данных о размере последующего передаваемого файла принимающей стороне. Плюсом данного способа является относительно несложная реализация и отсутствие разрыва соединения.

Последний способ наиболее подходит для реализации передачи изображений клиентом. Так как разрывать подключение между передачей картинок не удачное решение в виду экономии времени пользователя и ресурсов памяти.

Метод принимает размер будущего изображения, а дальше происходит считывание входящего потока и запись их порциями в *OutputStream* до тех пор, пока размер считанного массива не достигнет заявленного размера клиентом. Затем *OutputStream* с помощью метода *ToByteArray* переводится в массив байт.

Изображения передаются пакетами размерностью с буфер конкретного приложения, как на серверном так и на клиентском программном обеспечении (далее ПО) значение буфера выставлено в 1024\*2. Число выбрано как компромисс между скоростью передачи данных и вероятностью их потери.

Несмотря на размер буфера, размер реально передаваемых данных может отличаться как в большую, так и в меньшую сторону. Данное поведение обусловлено возможными помехами в сетевом взаимодействии и нестабильностью сетевого сигнала в целом.

Вышеупомянутое поведение выставляет ограничение на использование точного подсчёта порций передаваемых данных. Но данная проблема решается используемым механизмом передачи больших файлов, описанным выше.

Класс *ImgHelper* содержит методы работы с изображениями, такие как *SaveImg* и *ReadImg*, которые отвечают за сохранение и считывание файла изображения из дискового пространства.

После принятия файла с клиентского приложения сервер передаёт его на обработку python скрипту, работающему с нейронной сетью. Полученное результирующее изображение отправляется обратно клиенту.

Для работы с python скриптом был создан класс *PyHelper*, имеющий единственный метод *CallPy*. Метод работает с классами *Process* и *Runtime*.

Использование скрипта, как компонента, для решения основной задачи сервера связано с лёгкостью внесения дополнительных изменений в код, без необходимости перезагружать или перекомпилировать сервер.

На изображении 2.14 указана архитектура каталогов.

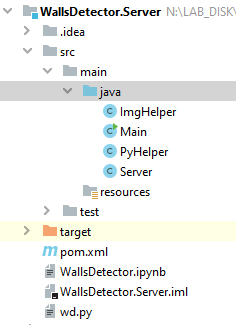


Рисунок 2.14 – Дерево каталогов серверной части ПО

Класс *Main*  содержит аналогичный метод и приводит в запуск классы, хранящиеся в директориях.

Для автоматической загрузки необходимых библиотек и удобной сборки проекта на *java* использовался инструмент *maven*. Файл *pom.xml* содержит настройки сборки проекта и объявления необходимых для работы проекта библиотек.

* 1. **Программное обеспечение на стороне клиента**

Мобильное приложение подразумевает несколько вариантов использования. В зависимости от выбранного варианта сервер будет обрабатывать одни и те же запросы по-разному.

Варианты использования клиентского приложения:

* сегментирование отдельных изображений;
* сегментирование кадров с камеры в реальном режиме времени.

Клиентское приложение содержит:

–класс сетевого взаимодействия *TCP/IP*;

–классы двух основных экранов приложения;

– классы фрагментов приложения.

Первым вариантом использования мобильного клиента является – передача изображений. Импортировать изображение можно выбрав из галереи настоящего устройства или сделав снимок.

Клиент предназначен для разового анализа. Удобен в обработке изображений, сделанных в прошлом, и не требователен к качеству связи и техническим характеристикам мобильного устройства.

Второй тип клиента отвечает за передачу видео, а именно трансляцию последнего на сервер в реальном времени. Ответ от сервера поступает также в реальном времени и отображается на экране приложения.

Данный тип взаимодействия наглядно показывает работу основного алгоритма распознавания.

Платформой разработки был выбран *Android*, как одна из самых распространённых мобильных операционных систем. Языком программирования – *java*.

Для начала обработки изображений сервер должен получить исходный материал в виде отдельного изображения или видео-ролика – разработанный мобильный клиент имеет две вышеперечисленные опции.

Приложение имеет два основных окна: окно подключения (*MainActivity*), которое состоит из двух полей для ввода параметров подключения и кнопки «*connect*», и рабочее окно (*WorkActivity*), которое становится доступным после успешного подключения к серверу. На рисунке 2.15 отображены сигнатуры методов классов *Activity* приложения.

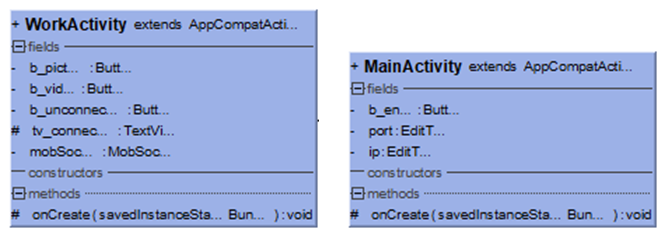


Рисунок 2.15 – Сигнатуры методов классов *WorkActivity* и *MainActivity* соответственно

Как только пользователь ввёл все необходимые данные для подключения и нажал соответствующую кнопку, создаётся новый экземпляр асинхронного класса *MobTask*, который является наследником базового класса *AsynkTask*.

Сигнатуры методов класса *MobTask* находится на рисунке 2.16.

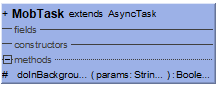


Рисунок 2.16 – Сигнатуры методов класса *MobTask*

Выполнять масштабные операции в основном потоке, работающем с пользовательским интерфейсом, не является хорошей практикой. Операции, такие как: работа с сетью или с базой данных, могут отнимать достаточно большое количество времени. Выполнение подобных операция в основном потоке заблокирует на время их исполнения отклик пользовательского интерфейса. Со стороны пользователя данное явление выглядит как зависание. Это может негативно сказаться на пользовательском опыте. Избежать подобной ситуации можно используя потоки или асинхронное программирование.

Существует ограничение – параллельным потокам запрещён доступ к элементам управления. И если от результата исполнения тяжёлой операции зависит представление пользовательского интерфейса, уместней использовать класс, разработанный специально для этих нужд – *AsynkTask*. Он является абстрактным и требует собственной реализации от разработчика.

Основными методами класса *AsynkTask* можно назвать: *doInBackground* (в нём необходимо реализовать код тяжёлой операции), *onPostExecute* и *onPreExecute* (методы, исполняющиеся до и после операции, в основном потоке, то есть имеющие доступ к интерфейсу пользователя). Так же существуют три реже используемых метода: *onProgressUpdate* (для промежуточного вывода результатов в основном потоке) и *onCanselled* (реализующий код отмены операции). Конструктор *AsynkTask* принимает на вход неограниченное число параметров заданного в объявлении класса типа. Также можно задать типы параметров, возвращаемых при завершении операции или при промежуточных выводах.

Выше описанные составляющие доказывают удобство использования методов данного класса для решения задач с сетевым соединением.

В классе *MobTask* создаётся сокет подключения, принимается сообщение от сервера, подтверждающее успешность подключения и возвращается результат операции в виде булевского значения.

Во время подключения в *MainActivity* создается сокет подключения, позволяющий обмениваться данными между устройствами. Класс, отвечающий за сетевое соединение – *MobSoket*.

Методы и поля класса *MobSoket* представлены на рисунке 2.17.

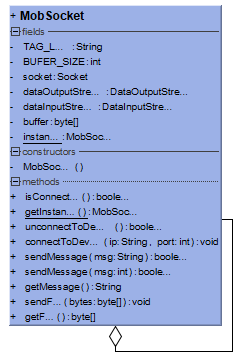


Рисунок 2.17 – Методы и поля класса *MobSocket*

Задача передачи сокета между *Activity* решается паттерном *SingleTone*. Такое решение заменяет глобальную переменную, а также ограничивает создание новых экземпляров класса без необходимости.

Класс *MobSoket*, помимо реализации *SingleTone*, имеет шесть основных рабочих методов, не считая перегрузки: *connectToDevice*, *unconnectToDevice*, *sendMessage* , *getMessage SendFile* и *GetFile*.

В методе *connectToDevice* вызывается метод Connect у объекта сокета. Метод *unconnectToDevice* закрывает все открытые соединения, в том числе и те, что были использованы для передачи или получения данных.

Метод *sendMessage* создаёт или использует объект класса *DataOutputStream*, используемый для отправки сообщения на сервер. И непосредственно отправляет сообщение на сервер.

Метод *getMessage* создаёт или использует объект класса *DataInputStream*, используемый для получения сообщения от сервера в строковом формате.

Также есть два подобных вышеописанным метода *SendFile* и *GetFile*. Они предназначены для отправки объёмных сообщений, как правило, файлов, а в контексте настоящего приложения – изображений. Методы работают с массивом байт и содержат логику характерную для отправки или принятия объёмного объекта.

После успешного подключения к серверу происходит переход на *WorkActivity*, которая располагает тремя кнопками: «*unconnect*», «*picture*» и «realtime\_video».

Кнопка «*unconnect*» отправляет на сервер числовое значение ноль, что является признаком завершения сеанса с пользователем. Следующим действием разрывает соединение, вызывая метод *Unconnect* класса *MobSocket*, который закрывает выходящий и исходщий стримы, а затем и сам сокет. Клиентское же приложение снова переходит в окно подключения.

Подобное мягкое решение разрыва связи создано для создания возможности серверу корректно завершить работу с данным клиентом и обработкой его данных. Так как принудительное завершение работы клиентского приложения пользователем может произойти во время передачи или приёма данных.

Также сервер застрахован от неожиданного разрыва связи. В таком случае он получит ключ со значением минус один, прервёт работу с текущим клиентом, но продолжит работу как с остальными, так и в ожидании новых входящих подключений.

Кнопки «*picture*» и «realtime\_video» отвечают за тип клиента, при их нажатии происходит отправка на сервер сообщения с номером типа клиента, а также создаётся и отображается соответствующий фрагмент. Для определения типа фрагмента использован фабричный метод, возвращающий необходимый класс, по числу. Для выполнения операций по отправки сообщений на сервер создан вложенный класс *ClientTypeTask*, наследующий *AsynkTask*. Он принимает входным значением число, идентифицирующее тип клиента. Отправляет его значение на сервер и, в случае успешной отправки, создаёт и открывает соответствующий фрагмент.

Приложение имеет два класса фрагментов: *FragmentPicture* и *FragmentVideo*.

*FragmentPicture* содержит кнопки, отвечающие за импорт и отправку на сервер изображений: «*upload photo*», «*send photo*». А также объект *ImageView*, отвечающий за отображение подгруженного изображения на экране мобильного приложения для пользователя.

На рисунке 2.18 отображена диаграмма класса *FragmentPicture.*

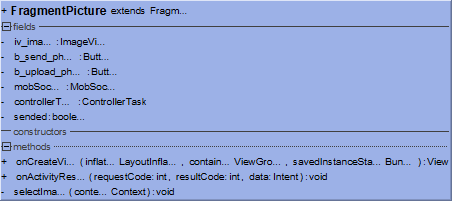


Рисунок 2.18 – Методы и поля класса *FragmentPicture*

Класс *FragmentPicture* содержит вложенный асинхронный класс – *ControllerTask*. Его задачей является асинхронный вызов метода отправки файла, класса *MobSocket*. при нажатии управляющей кнопки, создаётся новый объект класса ControllerTask.

Сервер работает с изображениями формата *jpeg* или *jpg* (название на старых платформах, не поддерживающих формат файла больше трёх символов). Соответственно при импорте из галереи файл конвертируется в необходимый формат перед преобразованием в массив байт. При создании фото – изображения имеют необходимый *jpeg* формат по умолчанию.

После нажатия на кнопку импорт изображения перед пользователем появляется диалоговое окно со списком отзывчивых элементов: *Take Photo, Choose from Gallery, Cancel.*

После выбора изображение передаётся элементу отображения на экране. А затем, при нажатии кнопки отправления, изображение достаётся из него же.

Обработанное изображение снова отображается в элементе *ImageView* и, при необходимости, его можно повторно отправить.

Класс *FragmentVideo*  использует *Camera API* для захвата кадров с камеры в реальном времени.

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

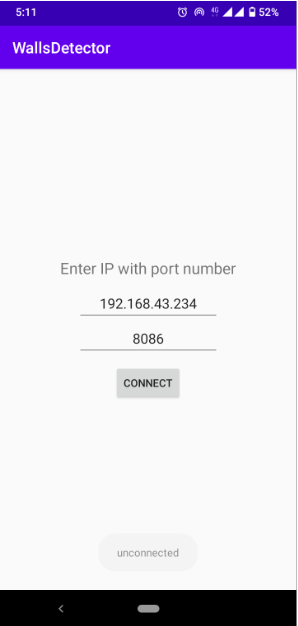
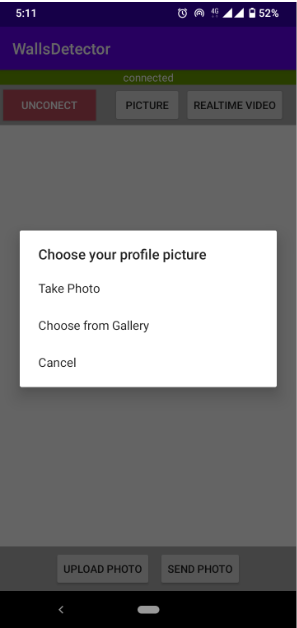
  

Рисунок 2.19 – Основные рабочие окна мобильного приложения

Приложение работает с сервером, которые обрабатывает поступающие на него кадры и возвращает обработанный нейронной сетью результат.

## 4 ВЕРЕФИКАЦИЯ И ОПЫТНАЯ ЭКСПЛУАТАЦИЯ РАЗРАБОТАННОГО ПРОГРАММНОГО КОМПЛЕКСА

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

Уваоораокра

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.

* 1. **Валидация результатов программного комплекса**

уваоораокра

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

Уваоораокра

**ЗАКЛЮЧЕНИЕ**

Разработанное программное обеспечение выполняет задачу распознавания стен на изображениях. Система создана по принципу клиент-сервер, где клиентом выступает мобильное устройство, а сервер стационарен и предназначен для развёртывания на персональном компьютере.

В процессе разработки первоначально был проведен аналитический обзор технологий, в области компьютерного зрения. Были реализованы два метода решающих поставленную задачу, один из которых включал нейронную сеть. Аналитическим образом выявлены преимущества нейронных сетей для решения поставленной задачи.

Язык программирования на стороне сервера – *java*, был выбран как единое решение для разработки мобильного клиента и консольного сервера. Язык программирования *python* выбран как инструмент для решения задачи сегментации изображений.

Результатом разработки стали мобильное приложение и консольное серверное приложение, а также обученная нейронная сеть.

Система приложений предназначена для решения задач связанных с ремонтом. Приложение рассчитано как на стандартных пользователей, так и для профессионалов в области дизайна и архитектуры.

Данный инструмент способен облегчить работу дизайнеров интерьера при подборе цветовой гаммы.

Несомненным преимуществом данной системы приложений является отсутствие идентичного программного обеспечения на рынке мобильных приложений *Android*.

К недостаткам можно отнести скорость и относительно небольшую точность распознавания краёв стен. Но данные проблемы решаемы совершенствованием модели нейронной сети и использованием пост обработки выходного результата.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. [Holger C.](https://arxiv.org/search/cs?searchtype=author&query=Caesar%2C+H) COCO-Stuff: Thing and Stuff Classes in Context / [Holger C.](https://arxiv.org/search/cs?searchtype=author&query=Caesar%2C+H), [Jasper U.](https://arxiv.org/search/cs?searchtype=author&query=Uijlings%2C+J) // Sebastopol: O’Reilly Media. – 2018. – 2 с.
2. Structured3D: большой фотореалистичный набор данных для структурированного 3D-моделирования [Электронный ресурс] – Режим доступа: [https://arxiv.org/abs/1908.00222 – Дата доступа 19.04.2021](https://arxiv.org/abs/1908.00222%20–%20Дата%20доступа%2019.04.2021).
3. Сегментация объекта с помощью Mask R-CNN [Электронный ресурс] – Режим доступа: [https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46 – Дата доступа 20.04.2021](https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46%20–%20Дата%20доступа%2020.04.2021).
4. OpenCv. Компьютерное зрение с открытым исходным кодом. [Электронный ресурс] – Режим доступа: https://docs.opencv.org – Дата доступа 20.04.2021.
5. Semantic Understanding of Scenes through ADE20K Dataset. B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso and A. Torralba. International Journal on Computer Vision (IJCV), 2018.
6. COCO – это крупномасштабный набор данных [Электронный ресурс] – Режим доступа: <https://cocodataset.org/#home> – Дата доступа 20.04.2021.
7. Zhou, B., Zhao, H., Puig, X., Xiao, T., Fidler, S., Barriuso, A., & Torralba, A. (2019). Semantic understanding of scenes through the ade20k dataset. International Journal of Computer Vision, 127(3), 302-321.