|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ *ИУ-КФ «Информатика и управление»*

КАФЕДРА *ИУ3-КФ «Системы автоматического управления*

*и электротехника»*

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ***

***НА ТЕМУ:***

Разработка модуля сегментации изображений для мобильной автономной платформы.

Студент  *УТС.Б-81*   *2.06.2020* *Д.С.Трушков*

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель ВКР  *2.06.2020* *М.О.Корлякова*

(Подпись, дата) (И.О.Фамилия)

Нормоконтролер  *2.06.2020*  *А.В.Финошин*

(Подпись, дата) (И.О.Фамилия)

*2020 г.*

**Аннотация**

Темой выпускной квалификационной работы является разработка модуля сегментации изображений для мобильной автономной платформы.

Выпускная квалификационная работа содержит 78 страниц, 58 рисунков, 9 таблиц, 8 листов плакатов формата А1.

В рамках работы был проведён анализ основных методов сегментации изображений, определены их достоинства, недостатки и реализуемость на мобильной автономной платформе.

Разработаны алгоритмы, реализующие выбранные методы сегментации. Алгоритмы разработаны как на языке программирования MatLab и python. Был разработан алгоритм для обучения модели, произведён подбор параметров сети в широком диапазоне значений. В конечном результате, был поставлен эксперимент, показывающий работоспособность модели.

Приводится оценка точности и потерь при обучении модели. Приведена стоимостная оценка разработки модели автоматической калибровки системы технического зрения для анализа положения автономной платформы.

Содержание

[Введение 5](#_Toc43660330)

[1. Глава 1. Научно-исследовательская часть 6](#_Toc43660331)

[1.1. Описание мобильной платформы и среды 6](#_Toc43660332)

[1.2. Анализ подходов к сегментации изображения 6](#_Toc43660333)

[1.3. Классические подходы к сегментации 7](#_Toc43660334)

[1.3.1. Сегментация уровня серого 7](#_Toc43660335)

[1.3.2. Марковские случайные поля 8](#_Toc43660336)

[1.4. Методы глубокого обучения 10](#_Toc43660337)

[1.4.1. Основные понятия и теория нейросетей 11](#_Toc43660338)

[1.4.2. Проблемы обучения нейронных сетей 15](#_Toc43660339)

[1.4.3. Модельные архитектуры 18](#_Toc43660340)

[1.4.4. VGG-16 19](#_Toc43660341)

[1.4.5. U-net 20](#_Toc43660342)

[1.4.6. Модель тирамису 22](#_Toc43660343)

[1.4.7. MultiScale методы 23](#_Toc43660344)

[1.5. Возможность реализации на мобильной платформе 24](#_Toc43660345)

[2. Глава 2. Проектно-конструкторская часть 27](#_Toc43660346)

[2.1. Описание мобильной платформы 27](#_Toc43660347)

[2.1.1. Составляющие автономной мобильной платформы 27](#_Toc43660348)

[2.1.2. Характеристики автономной платформы 28](#_Toc43660349)

[2.1.3. Взаимодействие модулей автономной платформы 30](#_Toc43660350)

[2.2. Программное обеспечение 31](#_Toc43660351)

[2.3. Реализация базовых алгоритмов сегментации 34](#_Toc43660352)

[2.3.1. Сегментация по пороговому уровню 34](#_Toc43660353)

[2.3.2. Сегментация по активному контуру 35](#_Toc43660354)

[2.3.3. Сегментация по движению относительно фона 37](#_Toc43660355)

[2.4. Реализация сегментации на основе глубокого обучения 41](#_Toc43660356)

[2.5. Формирование набора данных для доучивания 45](#_Toc43660357)

[2.6. Тестирование модели 48](#_Toc43660358)

[2.6.1. Влияние параметров модели на результат сегментации 48](#_Toc43660359)

[2.6.2. Гиперпараметры и характеристики сети 59](#_Toc43660360)

[2.7. Анализ результатов работы модели 61](#_Toc43660361)

[2.8. Экономическая часть 66](#_Toc43660362)

[2.8.1. Экономическое обоснование дипломного проекта 66](#_Toc43660363)

[2.8.2. Организация и планирование работ 66](#_Toc43660364)

[2.8.3. Определение стоимости специального оборудования 67](#_Toc43660365)

[2.8.4. Расчет основной заработной платы 67](#_Toc43660366)

[2.8.5. Выводы по экономическому разделу дипломного проекта 69](#_Toc43660367)

[2.9. Охрана труда 69](#_Toc43660368)

[2.10. Экологическая безопасность 72](#_Toc43660369)

[Заключение 75](#_Toc43660370)

[Список использованных источников 76](#_Toc43660371)

[Приложение А 78](#_Toc43660372)

[Приложение Б 79](#_Toc43660373)

[Приложение В 84](#_Toc43660374)

[Приложение Г 86](#_Toc43660375)

[Приложение Д 86](#_Toc43660376)

# Введение

В современном обществе существует тенденция к как можно большей автоматизации различных технических систем. Растет число автономных мобильных систем, действующих в одной с человеком среде. Таким модулям необходимы устройства и алгоритмы как для позиционирования в среде, так и для безопасного движения. Самый распространённый и доступный способ получить информацию о сцене для автономного модуля – использовать камеру. Однако её использование влечёт ряд проблем интерпретации поступающего видеопотока: необходимо иметь высокопроизводительные бортовые вычислители и значительный объем памяти для хранения и обработки кадров.

Основные задачи обработки видеопотока можно разделить на следующие подвиды: сегментация изображений, распознавание и детектирование отдельных объектов, определение расстояний до объектов сцены. Кроме того, обработка видеоряда не является конечной целью работы, т.к. это только предварительный этап для построения входных данных для системы управления движением мобильного робота.

Таким образом, задача анализа видеопотока в бортовой системе управления должна быть решена достаточно быстро, чтобы успеть выработать и исполнить управление. Итак, стоит задача сформировать быстрый и простой механизм обработки видеопотока бортовой камеры [1].

Основной целью данной работы является разработка программы для сегментации изображений. Для этого было произведено следующее:

1. Изучение подходов к методам переноса обучения в моделях обработки изображений.
2. Разработка модели переноса для сегментации.
3. Формирование процедуры доучивания.
4. Анализ результатов и сравнение с результатами ручной сегментации.
5. Оценка вычислительной сложности алгоритмов.

# Глава 1. Научно-исследовательская часть

## Описание мобильной платформы и среды

На мобильной автономной платформе установлен микрокомпьютер Raspberry Pi 3 Model B+ и микроконтроллер ATmega1281. Платформа движется в помещении, на которое не налагается никаких ограничений с точки зрения находящихся в ней объектов, освещения и т.д. Необходимо разработать систему (интеллектуальный датчик), позволяющую роботу, двигаясь внутри сцены, проводить её сегментацию, разделяя изображения на 3 класса: пол, стены, объекты. В начале необходимо определить понятие сегментации и рассмотреть её возможные методы.

## Анализ подходов к сегментации изображения

По определению сегментация — выделение областей, однородных по какому-либо критерию, например по яркости. Однако, в данной работе подразумевается семантическая сегментация, которая обозначает выделение по критерию принадлежности к определённой группе (объекты, пол, стены).

Семантическая сегментация — это процесс присвоения каждому пикселю изображения своей метки, что в свою очередь, резко отличается от классификации, где одна метка присваивается всему изображению. Семантическая сегментация рассматривает несколько объектов одного класса как один объект. Случайная сегментация, в свою очередь, обрабатывает несколько объектов одного класса как отдельные объекты. Как правило, это более сложный процесс.

Обзор задачи семантической сегментации:

Целью семантической сегментации является точная классификация и сегментация каждого объекта, представленного на изображении. Семантическая классификация состоит из трех основных шагов:

1. Первоначально происходит выделение всех классов, присутствующих на обрабатываемом изображении.

2. Следующим шагом является сегментация / обнаружение, которые предоставляют не только сущности классов, но и информацию относительно пространственного расположения этих классов.

3. На последнем шаге достигается точный вывод. Создаются метки для каждого пикселя. Эти метки указывают на принадлежность определенного пикселя к сущности класса.

Результат задачи семантической сегментации, интерпретированный в изображение, понятное человеку, показан на рис. 1.

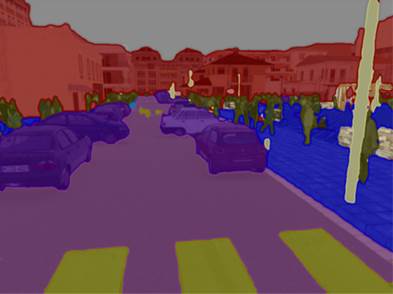


Рисунок 1 – Результат задачи семантической сегментации

## Классические подходы к сегментации

Исторически первыми, ещё до появления глубокого обучения, для сегментирования изображений использовались большое количество методов обработки изображений. Рассмотрим некоторые из этих методов ниже.

### Сегментация уровня серого

Простейшая форма семантической сегментации, включающая в себя жестко закодированные правила или свойства, которым должен соответствовать пиксель, чтобы ему была присвоена конкретная метка. Правило, такое как интенсивность уровня серого, может быть оформлено в свойствах самого пикселя. Одним из таких методов, который использует эту технику, называется алгоритм разделения и слияния. Этот алгоритм рекурсивно разбивает изображение на подчасти до тех пор, пока метку нельзя уже будет назначить, а затем объединяет смежные части с одной и той же меткой. Также к методам сегментации по интенсивности уровня серого можно отнести сегментацию по пороговому уровню, сегментацию по активному контуру и сегментацию по движению относительно фона. Данные методы будут реализованы в проектно-конструкторской части.

Проблема этого подхода в том, что правила должны быть жестко закодированы. Кроме того, чрезвычайно сложно работать со сложными классами, к примеру людьми, которые состоят не только из информации об уровне серого. Следовательно, методы извлечения признаков и оптимизации сложных классов необходимы для правильного изучения представлений.

Таким образом, данный вариант решения проблемы сегментации изображений не подходит для данной задачи в связи с отсутствием возможности прописать жёсткую логику для всех сцен, в которых может передвигаться автономная платформа.

### Марковские случайные поля

В случае, когда сегментируется изображение с помощью обучения модели назначать класс на пиксель, и в случае, когда модель не идеальна, мы можем получить размытые результаты сегментации, которые невозможны по своей природе (например, пиксели собаки, смешанные с пикселями кошки, как показано на рисунке 2(a-d). Более реалистичная сегментация показана на рисунке 2(d).

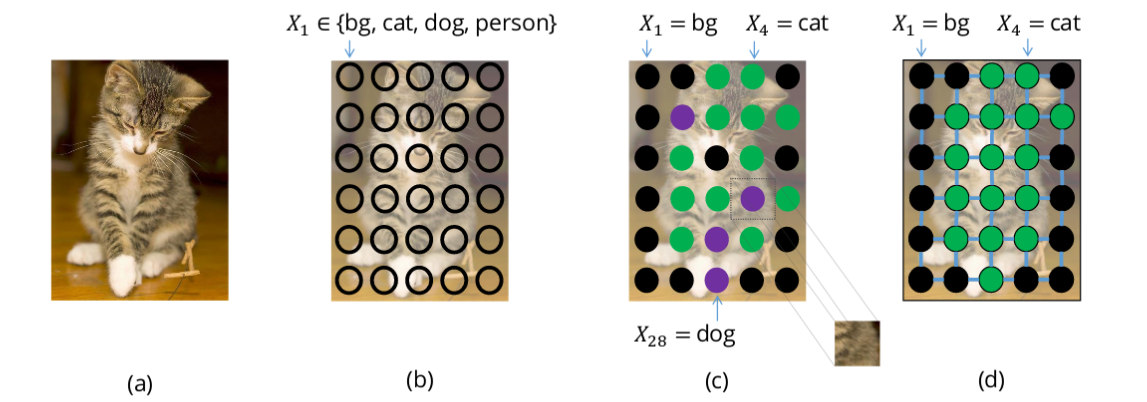


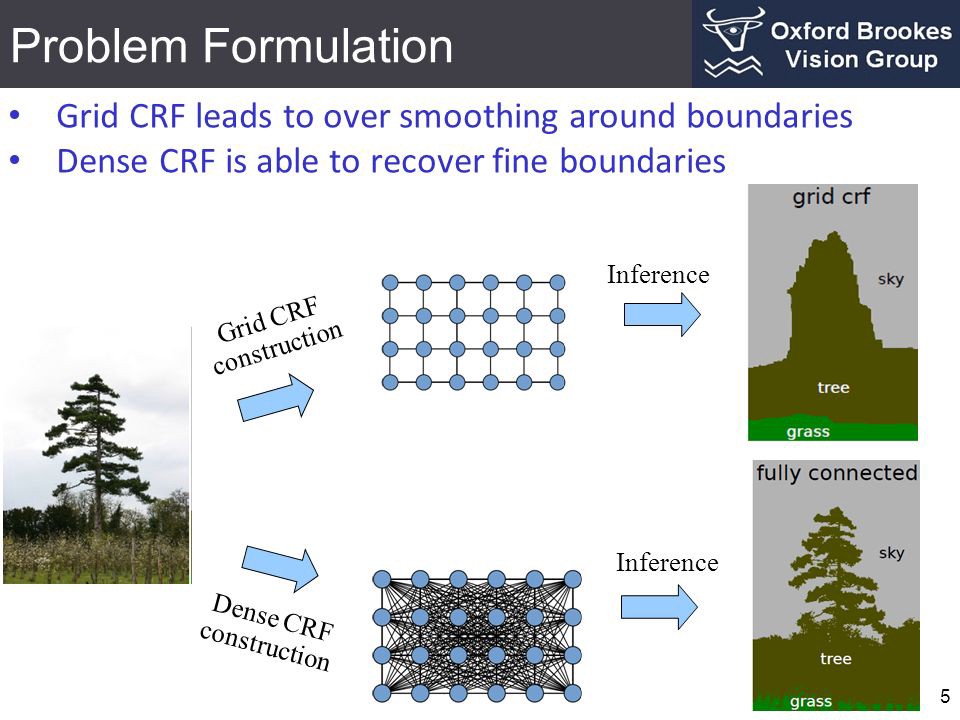
Рисунок 2 – Пиксели с меткой «собака», смешанные с пикселями с меткой «кошка»

Этого можно избежать, рассматривая предыдущее соотношение между пикселями, например тот факт, что объекты являются непрерывными и, как следствие имеют одинаковые метки. Чтобы смоделировать эти отношения, мы используем марковские случайные поля (CRF).

Марковским случайным полем или Марковской сетью называют графовую модель, которая используется для представления совместных распределений набора нескольких случайных переменных [2].

Каждый пиксель в изображении связан с конечным набором возможных состояний. В нашем случае метки назначения представляют собой набор возможных состояний. Цена присвоения состояния (или метки u) одному пикселю (x) называется его одинарной стоимостью. Чтобы смоделировать отношения между пикселями, мы также учитываем стоимость присвоения пары меток (u,v) паре пикселей, (x,y) известную как парная стоимость. Мы можем рассмотреть пары пикселей, которые являются его непосредственными соседями (Grid CRF), или мы можем рассмотреть все пары пикселей в изображении (Dense CRF)(рис. 3).

* Grid CRF приводит к чрезмерному сглаживанию границ.
* Dense CRF способен восстанавливать мелкие границы.

  
  
Рисунок 3 – Иллюстрация работы Grid CRF и Dense CRF

Сумма одинарной и попарной стоимости всех пикселей известна как энергия (или стоимость / потеря) CRF. Это значение может быть минимизировано для получения хорошего результата в сегментации.

На сегодняшний день именно метод CRF является наиболее популярным и точным способом извлечения объектов из текста. Например, он был реализован в проекте Стэндфордского университета Stanford Named Entity Recognizer. Так же этот метод успешно реализован для разных видов лингвистической обработки текста [2].

Данный метод на практике хорошо работает в задачах анализа текста и другой неграфической информации. Для случая автономной мобильной платформы, где необходима быстрая и лёгкая сегментация в реальном времени CRF не подойдёт своей сложностью, в связи с размером получаемых деревьев.

## Методы глубокого обучения

Глубокое обучение значительно упростило процесс выполнения семантической сегментации и даёт результаты впечатляющего качества. В этом разделе произведём анализ популярных модели архитектур и функций потерь, используемых для глубокого обучения.

### Основные понятия и теория нейросетей

*Нейронная сеть* (также искусственная нейронная сеть, ИНС) — математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма (рисунок 4). Это понятие возникло при изучении процессов, протекающих в мозге, и при попытке смоделировать эти процессы.

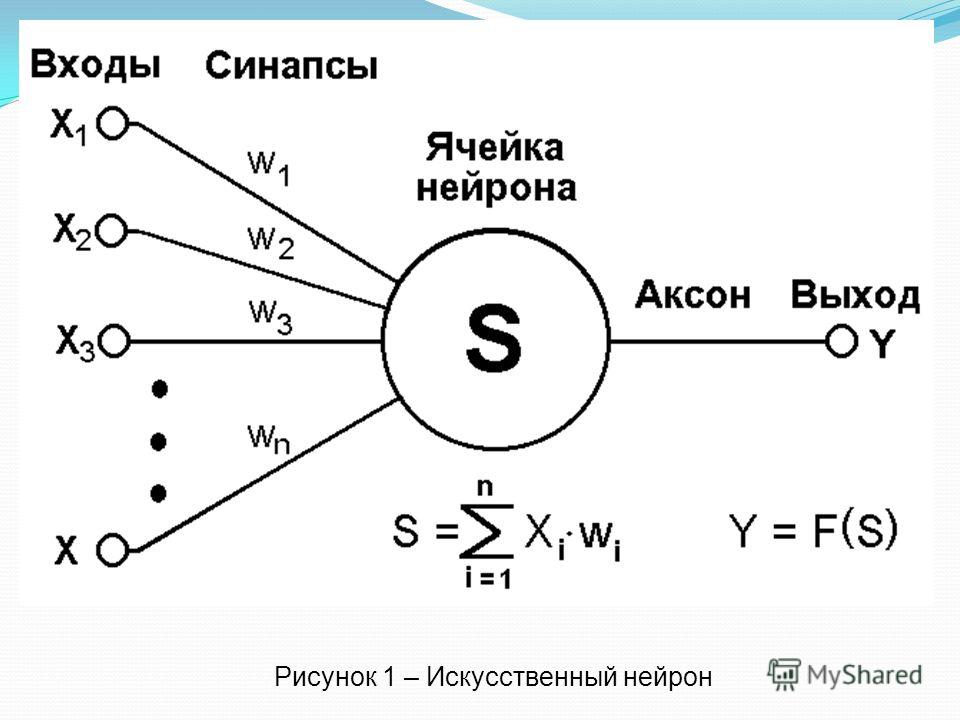


Рисунок 4 – Модель нейрона в нейронной сети

*Метод обратного распространения ошибки* (англ. backpropagation) — метод вычисления градиента, который используется при обновлении весов многослойного перцептрона. Это итеративный градиентный алгоритм, который используется с целью минимизации ошибки работы многослойного перцептрона (нейронной сети) и получения желаемого выхода.

Основная идея этого метода состоит в распространении сигналов ошибки от выходов сети к её входам, в направлении, обратном прямому распространению сигналов в обычном режиме работы. Схема сети представлена на рисунке 5.

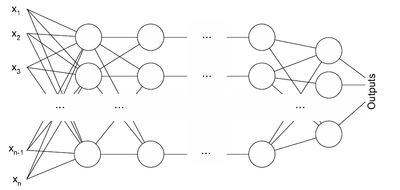
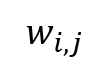
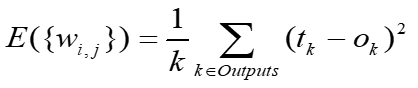
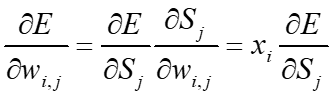
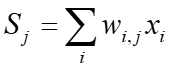


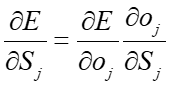
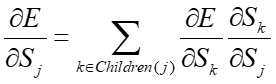
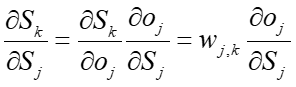
Рисунок 5 – Схема сети

*  - множество входов сети.
*  - множество выходов сети
* Обозначим вес, стоящий на ребре, соединяющий i-й и j-й узлы. А через  - выход i-го узла. Если нам известен обучающий пример (правильные ответы  ), то функция ошибки, полученная по методу наименьших квадратов будет равна:



Для модификации весов используем метод стохастического градиентного спуска  , где  задаёт скорость движения. Производная вычисляется следующим образом:

, где 

Если j-й узел на последнем уровне, то  , иначе  , где Children(j) - выход j-го узла, а .

Учитывая, что  - аналитическая поправка, вычисленная для узла следующего уровня, обозначим её как 

Поскольку мы научились вычислять поправку для узлов последнего уровня и выражать поправку для узла более низкого уровня через поправки более высокого, можно уже писать алгоритм. Именно из-за этой особенности вычисления поправок алгоритм называется алгоритмом обратного распространения ошибки (backpropagation).

Таким образом, для всех узлов можно вычислить поправку 

*Сверточный слой* нейронной сети представляет из себя применение операции свертки к выходам с предыдущего слоя, где веса ядра свертки являются обучаемыми параметрами (рисунок 6).

*Свертка* (англ. convolution) — операция над парой матриц  (размера ) и  (размера ), результатом которой является матрица  размера . Каждый элемент результата вычисляется как скалярное произведение матрицы  и некоторой подматрицы  такого же размера (подматрица определяется положением элемента в результате). То есть, .

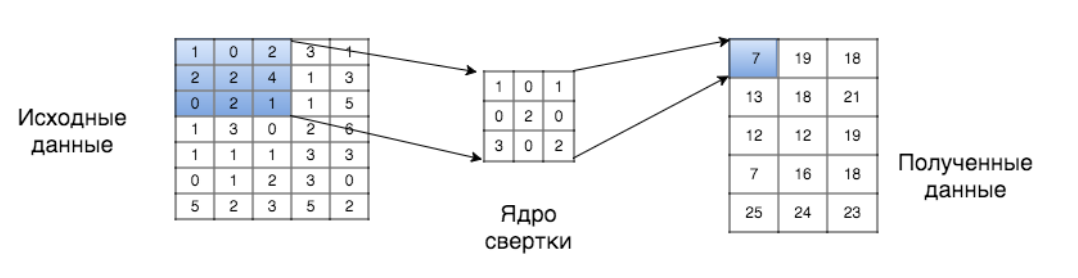


Рисунок 6 – Операция свёртки

*Операция пулинга* (иначе подвыборки, субдискретизации) представляет собой нелинейное уплотнение карты признаков, при этом группа пикселей (обычно размера 2×2) уплотняется до одного пикселя, проходя нелинейное преобразование (рисунок 7). Наиболее употребительна при этом функция максимума. Преобразования затрагивают непересекающиеся прямоугольники или квадраты, каждый из которых ужимается в один пиксель, при этом выбирается пиксель, имеющий максимальное значение. Операция пулинга позволяет существенно уменьшить пространственный объём изображения.



Рисунок 7 – Операция пулинга (maxpooling)

*Развёртывающую нейросеть* можно представить как свёрточную нейросеть, которая использует такие же компоненты (фильтрация, пулинг), но наоборот, так что вместо отображения пикселов для признаков она делает противоположное. Для изучения конкретной активации свёрточной нейросети, мы устанавливаем все остальные активации в этом слое на ноль и пропускаем карты признаков как входящие параметры к присоединённому слою развёртывающей нейросети. Потом мы успешно производим

1) анпулинг;

2) исправление;

3) фильтрацию, чтобы восстановить активность в нижнем слое, который породил выбранную активацию. Потом процедура повторяется до тех пор, пока мы не дойдём до исходного пиксельного слоя.

*Residual neural network (ResNet)* – является искусственной нейронной сетью (ИНС), которая основывается на принципе, основанном на пирамидных клетках в коре головного мозга. Остаточные нейронные сети повторяют их, используя пропуск соединений или ярлыки для перехода через некоторые слои. Типичные модели ResNet реализованы с использованием двух- или трехслойных пропусков, которые содержат нелинейности (ReLU) и периодическую нормализацию между ними (рисунок 8). Дополнительная матрица весов может использоваться для изучения весов пропуска. Эти модели известны как HighwayNets. Модели с несколькими параллельными пропусками называются DenseNets.

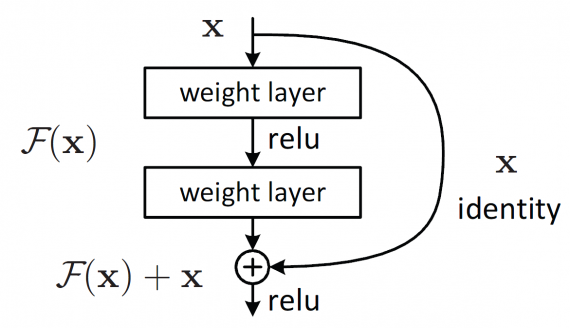


Рисунок 8 – Пропуск слоёв в резидуальной сети

Одной из причин пропуска слоев является попытка избегания проблемы затухания градиентов путем повторного использования активаций из предыдущего слоя, пока соседний слой не узнает свои веса. Во время обучения веса адаптируются для приглушения уровня выше по течению и усиливают пропущенный ранее уровень.

Пропуск эффективно упрощает сеть, используя меньшее количество уровней на начальных этапах обучения. Это ускоряет процесс обучения, уменьшая влияние исчезающих градиентов, поскольку существует меньше слоев, через которые можно распространяться. Затем сеть постепенно восстанавливает пропущенные слои по мере изучения пространства объектов. К концу обучения, когда все слои расширены, он остается ближе к многообразию решений и, таким образом, учится быстрее. Нейронная сеть без остаточных частей исследует больше пространства функций. Это делает его более уязвимым для возмущений, которые заставляют его покидать многообразие, и требует дополнительных данных обучения для восстановления.

### Проблемы обучения нейронных сетей

Хотя основной алгоритм обучения многослойных нейронных сетей открыт уже достаточно давно и весьма хорошо исследован, подход к обучению нейронных сетей в данный момент не строго регламентирован.

Исследователи вынуждены пробовать множество разнообразных архитектур сетей, обучать каждую из них в отдельности и затем выбирать ту, которая наилучшим образом решает поставленную задачу.

Однако, выбор архитектуры сети еще не решает проблемы, так как после задания архитектуры нейронной сети ее все равно необходимо обучать. По своей сути нейронная сеть является универсальным аппроксиматором. Это означает, что в процессе настройки она не вычисляет целевую функцию, а как бы лишь подбирает внутренний набор функций, при сложении которых образуется функция, выдающая на выходе ряд значений, напоминающий исходный ряд, предъявленный ей в процессе обучения (аппроксимационный полином). Отсюда следует вывод, что выходные данные работающей нейронной сети всегда будут содержать ошибку, причем величина этой ошибки никогда заранее не известна. Известно только, что в процессе обучения данная ошибка, возможно, будет уменьшена до некоторого приемлемого уровня. Рабочая точка нейросетевой системы в процессе обучения скользит по поверхности ошибок по направлению к глобальному минимуму целевой функции. Причем, в силу неровности рельефа поверхности ошибок, сеть может застрять в локальном минимуме очень далеко от ожидаемого глобального. Если склон локального минимума достаточно крут, а шаг обучения слишком мал, чтобы рабочая точка выкатилась на его край, наступает состояние, называемое параличом сети, при котором сеть на обучающей выборке дает недостаточно точные результаты, а обучение при этом все равно останавливается.

Существует также состояние, когда ошибка обучения начинает беспорядочно колебаться (осциллировать) и сеть входит в состояние переобучения (рис. 9). Это соответствует слишком точной аппроксимации обучающих данных.

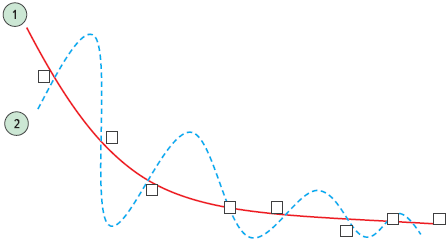


Рисунок 9 – Состояние переобучения: 1 — истинная зависимость; 2 — слишком точная аппроксимация полиномом.

Это так называемое «проклятие размерности». В общем случае входные данные содержат шум и некоторое количество малозначащей информации. При попытке заставить сеть разобрать все это по кластерам без предварительной подготовки входных данных мы рискуем получить неверные обобщения. С другой стороны, если внутренняя структура данных нам неизвестна (задача кластеризации в чистом виде), мы в подобной ситуации вынуждены увеличивать число входных элементов сети. Необходимое число входных нейронов быстро возрастает с увеличением размерности кластеризуемого пространства (приблизительно как ), а вслед за увеличением числа входных нейронов неизбежно изменяется и число нейронов скрытого слоя, иначе процесс кластеризации может сопровождаться потерями значимой информации. Это усложняет процесс обучения нейронной сети и делает его менее предсказуемым.

Таким образом, для успешного решения задачи обучения нейронной сети необходимо:

* Правильно выбрать структуру сети (то есть количество слоев и количество нейронов в каждом слое). Выбор подходящей структуры сети должен соответствовать характеру решаемой задачи.
* Правильно выбрать параметры обучения, в частности шаг обучения сети, норму обучения, количество обучающих примеров, сам алгоритм обучения.
* Правильно подготовить входные данные. Желательно, чтобы перед началом обучения исследователь в общих чертах представлял себе структуру входных данных, отфильтровал шум, желательно также избавиться от малозначащих составляющих входных данных. В некоторых случаях предварительная обработка данных может предусматривать линейное преобразование, выделяющее из всей массы входных данных некоторый набор наиболее значимых направлений или векторов (метод главных компонент, Bishop, 1995). Иногда такое преобразование можно поручить дополнительной нейронной сети, содержащей в скрытом слое меньше нейронов, чем во входном или выходном слоях, что позволяет понижать размерность входных данных. То есть сжимать их.

### Модельные архитектуры

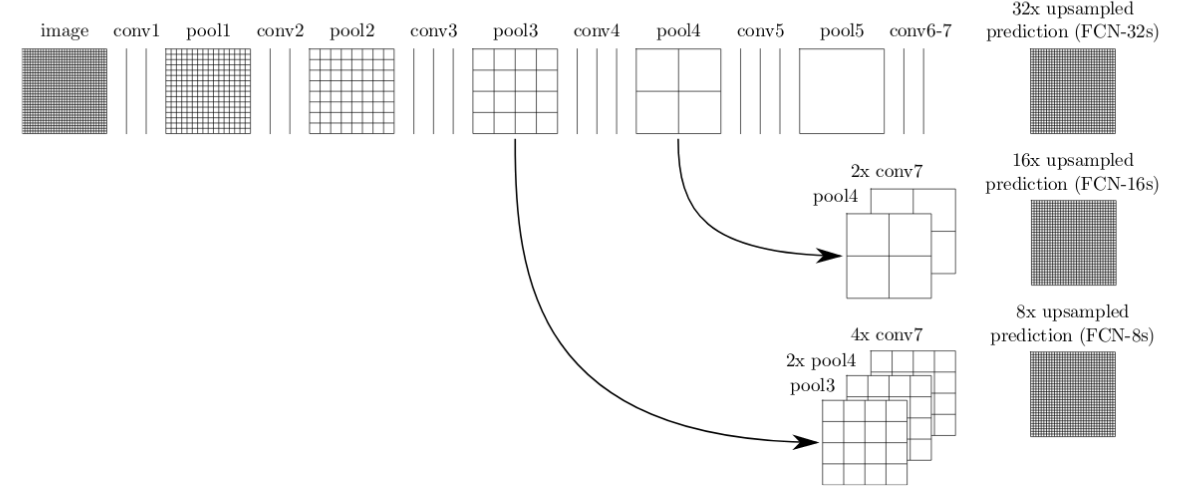
Одной из самых простых и популярных архитектур, используемых для семантической сегментации, является полностью сверточная сеть (FCN)[3]. В [статье](https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf) FCN для семантической сегментации авторы используют FCN, чтобы уменьшить входное изображение до меньшего размера (получая больше каналов) через серию сверток (рис. 10). Этот набор сверток обычно называют кодером. Затем кодированный выход подвергается дискретизации с помощью билинейной интерполяции или серии транспонированных сверток. Этот набор транспонированных сверток обычно называется декодером.  
  


Рисунок 10 – Downsampling и Upsampling в FCN

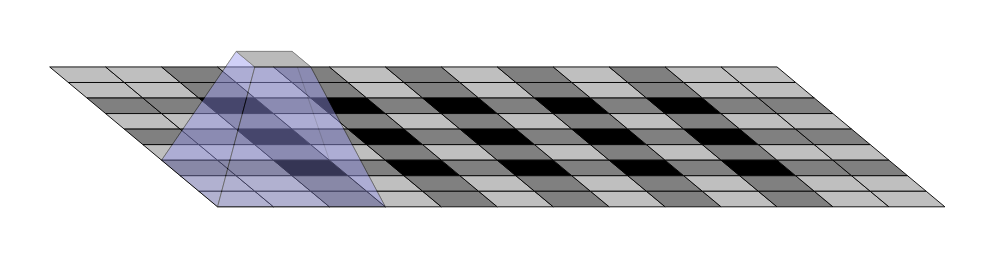
Эта базовая архитектура, которая несмотря на свою эффективность, имеет ряд недостатков. Одним из таких недостатков является наличие артефактов шахматной доски (рис. 11) из-за неравномерного перекрытия выходных данных операции транспонирования-свертки.  


Рисунок 11 – Формирование шахматных артефактов

Другим недостатком является плохое разрешение на границах из-за потери информации в процессе кодирования.

В этой связи было предложено несколько решений для улучшения качества работы базовой модели FCN. Ниже приведены некоторые из популярных решений, которые оказались эффективными:

### VGG-16

VGG16 — модель сверточной нейронной сети, предложенная K. Simonyan и A. Zisserman из Оксфордского университета в статье “Very Deep Convolutional Networks for Large-Scale Image Recognition” [4]. Модель достигает точности 92.7% — топ-5, при тестировании на ImageNet в задаче распознавания объектов на изображении. Этот датасет состоит из более чем 14 миллионов изображений, принадлежащих к 1000 классам.

VGG16 — одна из самых знаменитых моделей, отправленных на соревнование ILSVRC-2014. Она является улучшенной версией AlexNet, в которой заменены большие фильтры (размера 11 и 5 в первом и втором сверточном слое, соответственно) на несколько фильтров размера 3х3, следующих один за другим (рис. 12). Сеть VGG16 обучалась на протяжении нескольких недель при использовании видеокарт NVIDIA TITAN BLACK.

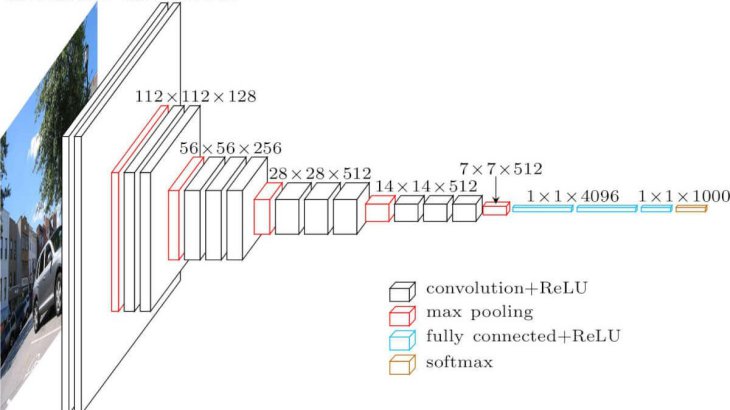


Рисунок 12 – Архитектура VGG-16

К сожалению, сеть VGG имеет два серьезных недостатка:

Очень медленная скорость обучения.

Сама архитектура сети весит много (появляются проблемы с диском и пропускной способностью).

### U-net

[U-Net](https://arxiv.org/abs/1505.04597) является обновлением до простой архитектуры FCN [5]. Оно имеет пропускаемые соединения с выхода блоков свертки на соответствующий вход блока транспонированной свертки, на том же уровне (рис. 13).

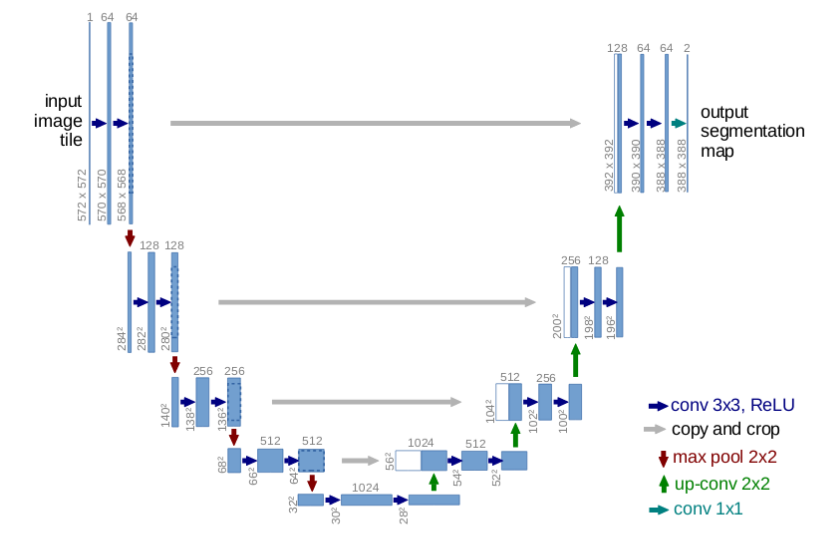


Рисунок 13 – Структура сети U-Net

Этот пропуск соединений позволяет градиентам «протекать» лучше и предоставляет информацию из нескольких масштабов изображения. Информация более крупных масштабов (верхних уровней) может помочь модели лучше проводить классификацию. Информация из более мелких масштабов (более глубоких слоев) может помочь сегменту модели лучше локализоваться.

Эта сеть универсальна и может использоваться для любой разумной задачи сегментации изображений. Высокая точность достигается при условии надлежащего обучения, подходящего датасета и правильного выбора времени обучения.

Таким образом, данная модель может быть использована на автономной мобильной платформе, поскольку обладает достаточной лёгкостью и скоростью (всего сеть содержит 23 свёрточных слоя), однако серьёзной проблемой оказывается обучение модели, так как для достаточной точности необходимо большое количество размеченных примеров, а составление датасета трудоёмкая задача.

### Модель тирамису

[Модель тирамису](https://arxiv.org/abs/1611.09326) (рис. 14) похожа на сеть U-Net, за исключением того факта, что она использует блоки свертки и транспонированной сверток, как это реализовано в [DenseNet](https://arxiv.org/pdf/1608.06993.pdf). Блок состоит из нескольких слоев сверток, где карты характеристик всех предыдущих слоев используются в качестве входных данных для всех последующих слоев. Получающаяся сеть чрезвычайно эффективна по параметрам и может получать доступ к функциям из более старых уровней.

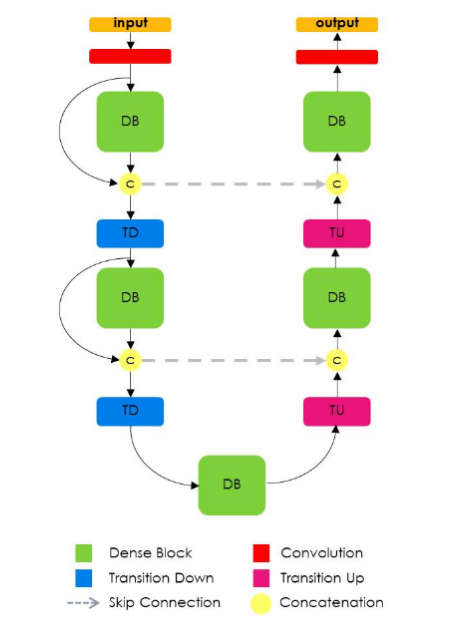


Рисунок 14 – Сеть Тирамису

Недостатком этого метода является то, что из-за характера операций конкатенации в нескольких средах ML, он не очень эффективно использует память графического процессора.

Данный метод является определённой модификацией предыдущего, однако сильные стороны данного метода для поставленной задачи являются не такими критичными, как его недостатки. Поскольку графический процессор автономной мобильной платформы имеет малые вычислительные мощности (для используемой Raspberry Pi 3 Model B+ 2-ядерный VideoCore IV, с частотой 3D GPU - 300 МГц, видео GPU - 400 МГц) данный подход оказывается не применим к поставленной задачи.

### MultiScale методы

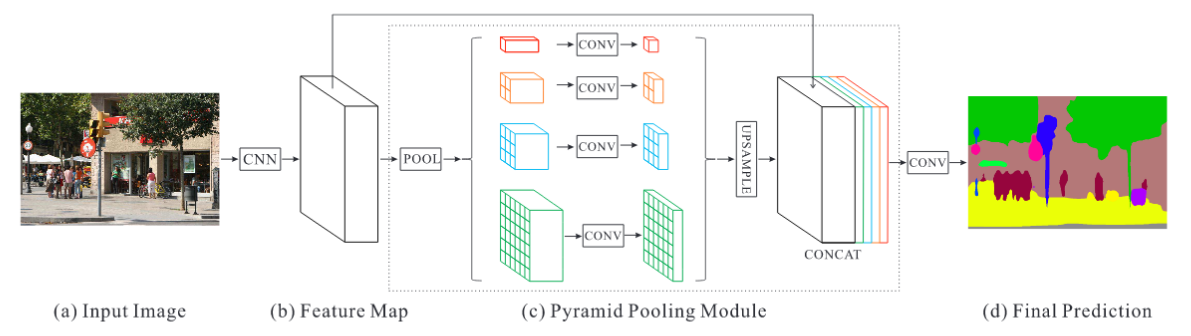
Некоторые модели глубокого обучения имеют методы для использования информации разных масштабов. Например, [сеть разбора фона](https://arxiv.org/pdf/1612.01105.pdf) Pyramid ([PSPNet](https://arxiv.org/pdf/1612.01105.pdf)) (рис. 15) выполняет операцию объединения с использованием четырех различного размера ядер и переходит к выходной карте возможностей CNN, такой как ResNet[6]. Затем сеть увеличивает размер всех выходных данных пула и карты выходных объектов CNN с помощью билинейной интерполяции и объединяет их все по оси канала. Окончательная свертка выполняется на этом каскадном выходе для генерации прогноза.  
  


Рисунок 15 – Сеть разбора фона Pyramid (PSPNet)

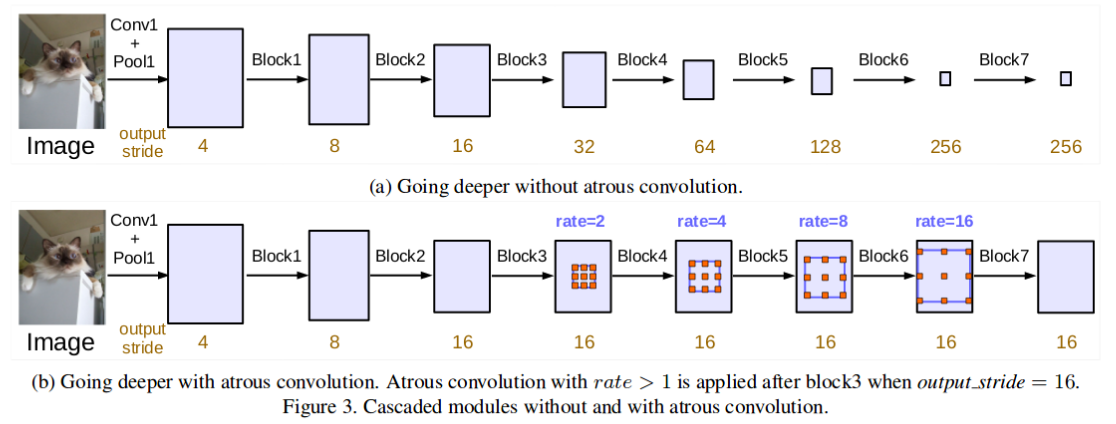
Atrous (Dilated) Convolutions (рис. 16) представляют собой эффективный метод объединения функций нескольких масштабов без значительного увеличения количества параметров. Регулируя степень расширения, у того же фильтра значения его веса распределяются дальше в пространстве. Это позволяет ему изучать более глобальную информацию.  
  


Рисунок 16 – Cascaded Atrous Convolutions

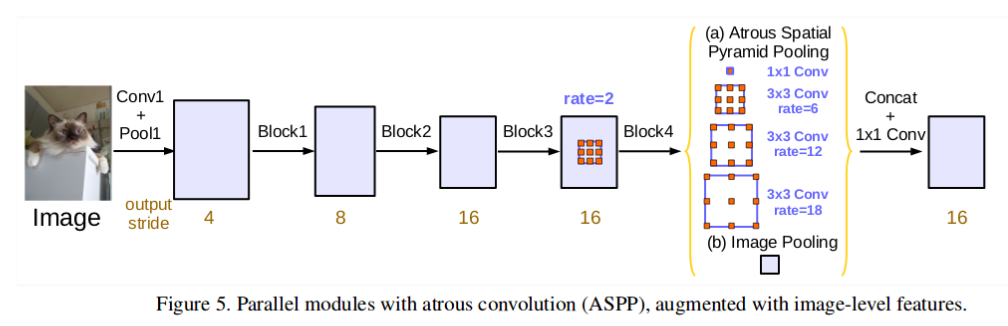
[DeepLabv3](https://arxiv.org/pdf/1706.05587.pdf) использует Atrous (рис. 17) свертки с различными скоростями растяжений для сбора информации различных масштабов, без потери в размере изображения [7]. Они экспериментируют с использованием свертки Atrous каскадным образом (как показано выше), а также параллельно в форме объединения пространственных пирамид Atrous (как показано ниже).  
  


Рисунок 17 – Параллельные модули с Atrous свёрткой, дополненная функциями уровня изображения

## Возможность реализации на мобильной платформе

Сегментация на основе классических методов не может дать удовлетворительные результаты только в случае контрастных объектов на однородном (желательно белом) фоне, что невозможно в рамках стоящей задачи сегментации сцены в процессе движения автономной мобильной платформы, поскольку в реальных условиях обеспечить постоянство цвета фона и освещения не представляется возможным. Таким образом, необходимо найти оптимальный метод на основе использования глубокого обучения, отвечающий требованиям реализуемости на мобильной платформе и скорости выполнения.

Произведём сравнительный анализ классических и нейросетевых моделей по основным критериям (таблица 1):

Таблица 1 – Сравнение классических и нейросетевых моделей

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Среднее время сегментации одного изображения, сек | Занимаемый объём, Мб | Accuracy |
| Сегментация по пороговому уровню | 0,02 | – | 0,9223 |
| Сегментация по активному контуру | 1,75 | – | 0,8761 |
| Сегментация по движению относительно фона | 0,5 | – | 0,9735 |
| Модель VGG16  (19 слоёв) | 0,019 | 56,19 | >0,6 |
| Модель Xception  (132 слоёв) | 0,038 | 79,95 | - |
| Модель ResNet50  (175 слоёв) | 0,039 | 90,45 | - |
| Модель DenseNet121  (427 слоёв) | 0,078 | 27,94 | - |

Несмотря на то, что классические методы показывают большую точность, они неприменимы в нашей задаче, поскольку производят исключительно бинарную классификацию.

На основании таблицы 1 можно сделать вывод, что оптимальным для нашей задачи будет модель VGG16, которая является наиболее быстрой, по сравнению с другими моделями и в то же время не занимает слишком много места.

Реализация выбранной моделей на мобильной платформе возможна, поскольку она отвечает всем требованиям устанавливаемого программного обеспечения, однако только в результате эксперимента возможно определить скорость сегментации на мобильной платформе.

Таким образом, необходимо разработать сеть для сегментации изображений на основе классических методов, а также сети VGG16, и произвести подбор её параметров в широком диапазоне.

Далее рассмотрим технические характеристики мобильной платформы и проведём ряд экспериментов для определения наиболее подходящей к поставленной задаче модели сегментации.

# Глава 2. Проектно-конструкторская часть

## Описание мобильной платформы

### Составляющие автономной мобильной платформы

Для выполнения вышепоставленных задач была использована мобильная платформа на основе микрокомпьютера Raspberry Pi3 и микроконтроллера AtMega1281, представленная на рисунке 18:



Рисунок 18 – Составляющие автономной платформы Robot4

Микрокомпьютер Raspberry Pi 3 Model B+ - это функциональная сборка, оптимальная для построения автоматизированных систем благодаря своей компактности и большому объему оперативной памяти. Он снабжен контроллерами Wi-Fi и Bluetooth и может работать с внешними устройствами ввода-вывода, в том числе с камерой. Микрокомпьютер Raspberry Pi служит для отправки команд на AtMega1281. К нему подключена камера для определения положения в пространстве.

Микроконтроллер AtMega1281 является высокопроизводительным восьмиразрядным микроконтроллером на базе микросхемы AVR RISC и имеет 54 линии ввода-вывода общего назначения, 32 рабочих регистра общего назначения, счетчик реального времени, 2 восьмиразрядных таймера-счетчика и 4 шестнадцатиразрядных.

Ввод-вывод данных на AtMega1281 организован с помощью портов USART и COM-портов управляющих устройств. Данный микроконтроллер имеет два порта USART, один из которых предназначен для управления с Raspberry Pi, а второй используется для передачи данных через Bluetooth или Wi-Fi.

### Характеристики автономной платформы



Рисунок 19 – Внешний вид Raspberry Pi 3

На рисунке 19 представлен внешний вид микрокомпьютера Raspberry Pi 3, а в таблице 2 – его характеристики. Проанализировав их, можно удостовериться в том, что данный микрокомпьютер способен справиться с задачей обработки видеопотока в реальном времени.

Таблица 2 – Характеристики Raspberry Pi 3 B+

|  |  |
| --- | --- |
| **Система на кристалле (SoC)** | Broadcom BCM2837B0 (CPU + GPU + RAM) |
| **Процессор** | 64-битный четырёхъядерный ARMv8 Cortex-A53 процессор с тактовой частотой 1.4 ГГц; 16 КБ cache L1 и 512 КБ cache L2 |
| **Графический процессор** | Двухъядерный процессор (GPU) VideoCore IV® (3D GPU @ 300 МГц, видео GPU @ 400 МГц) поддерживает стандарты OpenGL ES 2.0, OpenVG, MPEG-2, VC-1 и способен кодировать, декодировать и выводить Full HD-видео (1080p, 30 FPS, H.264 High-Profil) |
| **ОЗУ** | 1 ГБ SDRAM LPDDR2 |
| **Ethernet** | 10/100/1000 Мбит Gigabit Ethernet (через USB 2.0) (контроллер LAN7515 — USB 2.0 Hub и Ethernet) |
| **Wi-Fi/Bluetooth** | 2.4 ГГц и 5 ГГц IEEE 802.11.b/g/n/ac WI-FI и Bluetooth 4.2 Low Energy (BLE), обеспечиваемые микросхемой Cypress CYW43455 |
| **Видео вход** | 1 x CSI-2 для подключения камеры по интерфейсу MIPI |
| **USB-порты** | 4 порта USB 2.0 через USB hub в Microchip LAN7515 |
| **Периферия** | 40 портов ввода-вывода общего назначения (GPIO), UART (Serial), I²C/TWI, SPI с селектором между двумя устройствами; пины питания: 3,3 В, 5 В и земля. |
| **ОС** | Ubuntu, Debian, Fedora, Arch Linux, Gentoo, RISC OS, Android, Firefox OS, NetBSD, FreeBSD, Slackware, Tiny Core Linux, Windows 10 IOT |

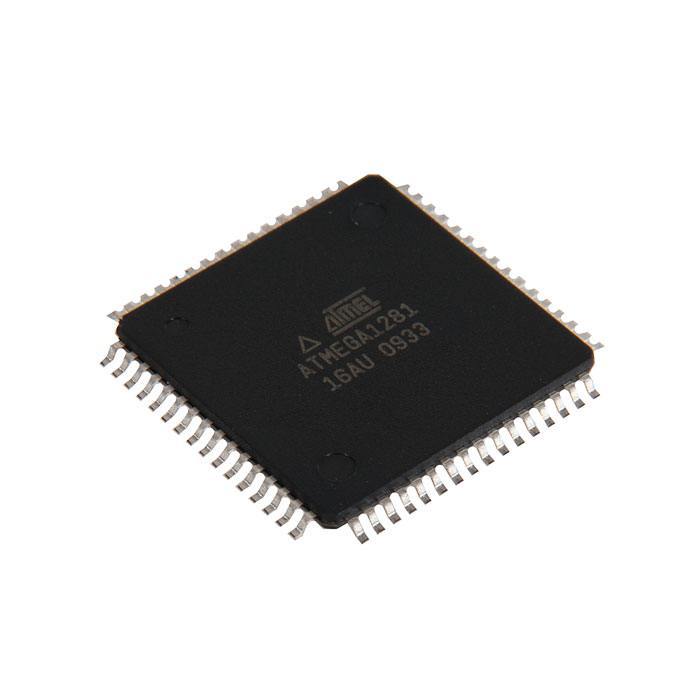


Рисунок 20 – Внешний вид микрочипа Atmega1281

Таблица 3 – Характеристики Atmega1281

|  |  |
| --- | --- |
| **ЦПУ: Ядро** | [AVR](http://catalog.gaw.ru/index.php?page=document&id=21423) |
| **ЦПУ: F,МГц** | от 0 до 16 |
| **Память: Flash,КБайт** | 128 |
| **Память: RAM,КБайт** | 8 |
| **Память: EEPROM,КБайт** | 4 |
| **I/O (макс.),шт.** | 54 |
| **Таймеры: 8-бит,шт** | 2 |
| **Таймеры: 16-бит,шт** | 4 |
| **Таймеры: Каналов ШИМ,шт** | 9 |
| **Таймеры: RTC** | Да |
| **Интерфейсы: UART,шт** | 2 |
| **Интерфейсы: SPI,шт** | 1 |
| **Аналоговые входы: Разрядов АЦП,бит** | 10 |
| **Аналоговые входы: Каналов АЦП,шт** | 8 |
| **Аналоговые входы: Быстродействие АЦП,kSPS** | 76.9 |
| **Аналоговые входы: Аналоговый компаратор,шт** | 1 |
| **VCC,В** | от 1.8 до 5.5 |
| **ICC,мА** | 16 |
| **TA,°C** | от -40 до 85 |
| **Корпус** | [TQFP-64](http://catalog.gaw.ru/index.php?page=document&id=15217) MLF (VQFN) 64 |

На рисунке 20 представлено изображение чипа AtMega1281. Его характеристики можно видеть в таблице 3.

Следует отметить, что подход с использованием AtMega1281 и Raspberry Pi 3 в связке с камерой является наиболее дешевым из существующих.

Помимо этого, рассматривается вариант добавления какого-либо датчика для более точной корректировки движения работа.

### Взаимодействие модулей автономной платформы

|  |  |
| --- | --- |
|  |  |

Рисунок 21 – Внешний вид автономной мобильной платформы Robot 4.

Снизу – Atmega1281, необходимая для управления колёсами робота. Сверху – Raspberry Pi 3, служащая для формирования сигнала управления при помощи камеры.

Внешний вид автономной мобильной платформы представлен на рисунке 21. AtMega1281 служит для управления двигателями колес, получения сигнала обратной связи и обработки внешнего сигнала управления, который подается в виде кода команды. Он представляет из себя один символ, что связано с особенностями приема и передачи данных через USART, обеспечивающий внешнее соединение с устройствами управления.

## Программное обеспечение

Есть несколько библиотек, позволяющих работать с нейронными сетями на языке python, также возможно использовать MatLab, однако он не может быть установлен на мобильную платформу из-за слишком большого размера и того, что не поддерживается его установка на Linux системе.

Самые крупные python библиотеки для машинного обучения это Keras [8], Tensorflow [9] и PyTorch [10].

*Плюсы Tensorflow:*

* Для нее написано большое количество руководств и документации;
* Она предлагает мощные средства мониторинга процесса обучения моделей и визуализации (Tensorboard);
* Она поддерживается большим сообществом разработчиков и техническими компаниями;
* Она обеспечивает обслуживание моделей;
* Она поддерживает распределенное обучение;

*Минусы Tensorflow:*

* Она проигрывает по скорости работы в эталонных тестах, в сравнении с некоторыми другими библиотеками;
* Она имеет более высокий входной порог для начинающих, чем PyTorch или Keras.

*Плюсы PyTorch:*

* Благодаря архитектуре фреймворка, процесс создания модели достаточно прост и прозрачен;
* Режим по умолчанию “define-by-run” – отсылка к традиционному программированию. Фреймворк поддерживает популярные инструменты для дебага, такие как pdb, ipdb или программа PyCharm;
* Он поддерживает декларативный параллелизм данных;
* Он имеет много предварительно обученных моделей и готовых модульных частей, которые легко комбинировать;

*Минусы PyTorch:*

* Недостаточная поддержка моделей;
* Он еще не готов для полноценного выхода, однако дорожная карта к версии 1.0 выглядят действительно впечатляюще;
* Недостаток интерфейсов для мониторинга и визуализации, как TensorBoard – однако он имеет внешнее подключение к Tensorboard.

*Плюсы Keras:*

* Прототипирование действительно быстрое и простое;
* Он достаточно маловесный для построения моделей глубокого обучения для множества слоев;
* Имеет полностью конфигурируемые модули;
* Имеет простой и интуитивно-понятный интерфейс, соответственно, хорош для новичков;
* Имеет встроенную поддержку для обучения на нескольких GPU;
* Может быть настроен в качестве оценщиков для TensorFlow и обучен на кластерах GPU на платформе Google Cloud;
* Поддерживает GPU от NVIDIA, TPU от Google, GPU с Open-CL, такие как AMD.

*Минусы Keras:*

* Может оказаться слишком высокоуровневым и не всегда легко кастомизируется;
* Он ограничен бэкэндами Tensorflow, CNTK и Theano.

В результате анализа библиотек, было решено использовать Keras, поскольку он отличается сравнительно малым занимаемым объёмом, большими возможностями для построения глубоких моделей сетей и предустановленными и предобученными на ImageNet необходимыми нам моделями.

Для выполнения сегментации на мобильной платформе необходимо установить Python версии выше 3.5, и произвести установку библиотек *tensorflow*, *numpy* и *keras*. Сделать это можно через командную строку с помощью команды *pip install*. На Raspberry Pi 3 установлена операционная система Linux, в ней есть предустановленная IDE (интегрированная среда разработки) *Thonny*, в которой можно создавать и запускать python проекты.

Исследование базовых алгоритмов сегментации и вспомогательные операции производились в языке программирования *MatLab* на стационарном компьютере. Глубокое обучение нейросетей производилось в облачном сервисе *Google Colaboratory*, который предоставляет в свободное пользование значительные вычислительные ресурсы для ускорения процесса обучения.

## Реализация базовых алгоритмов сегментации

Реализуем несколько классических методов сегментации, определим область их применимости и оценим возможность применения этих методов на мобильной автономной платформе.

### Сегментация по пороговому уровню

Сегментация в общем случае производится следующим образом: на цветном изображении находятся области, отделяемые резким переходом цвета [11]. Алгоритм нахождения таких областей может отличаться. Для простейшего случая, когда фон светлый или полностью белый достаточно произвести отсечение по яркости (бинаризацию по уровню порога). То есть если цвет выбранного пикселя близок к белому- этот пиксель относится к фону, он будет принимать значение 0. Если же он другого цвета- пиксель относится к объекту, находящемуся на белом фоне, будет заменяться на 1. В результате такой обработки теряется большая часть информации о первоначальном изображении, зато определяются сегментированные области.

В таком случае определить объекты достаточно просто:

Greyim=rgb2gray(RGB);

Greyim=imcomplement(Greyim);

BlackWhiteIm=imbinarize(Greyim,'adaptive','Sensitivity',0.4);

Здесь, RGB- исходное полноцветное изображение.

Результат сегментации представлен на рисунке 22, приложение А.

Greyim- полутоновое изображение, так как функция imbinarize для преобразования в бинарное изображение по уровню яркости работает только с полутоновыми изображениями.

Функция imcomplement нужна только для того, чтобы вместо черного объекта на белом фоне, мы получали на выходе белое изображение на черном фоне.

Параметр функции imbinarize ‘adaptive’ уменьшает влияние углов освещенности, для случаев, когда объект лежит в тени. Параметр ‘Sensitivity’ с коэффициентом 0.4 определяет уровень отсечения цвета.

BlackWhiteIm- полученное бинарное изображение с сегментированными областями.

На практике также необходимо использовать алгоритмы избавления от возможных шумов.

|  |  |
| --- | --- |
|  |  |

А) Б)

Рисунок 22 - А - исходное изображение, Б - изображение, полученное после сегментации по уровню порога

Реализация сегментации по пороговому уровню относительно других возможных вариантов является самой простой и самой быстрой. Accuracy данного метода составило 0,9223 (Приложение Г). В среднем время обработки изображения занимает 0.020 секунд.

### Сегментация по активному контуру

Более сложным и в то же время более универсальным методом является сегментация по активному конуру. Контур на изображении является значительным локальным изменением интенсивности изображения, и как правило, связан с разрывом интенсивности изображения или первой производной интенсивности изображения. В таком случае мы не ограничены цветами фона или объекта, а ограничены только чувствительностью к контрастности объектов на фоне. Данный подход реализует встроенная MatLab функция activecontour.

BlackWhiteIm=activecontour(RGB,mask);

Здесь, RGB- исходное полноцветное изображение.

Результат сегментации представлен на рисунке 23, приложение А.

mask- бинарная матрица того же размера, что и изображение, в которой единицами обозначены области, в которых нужно искать активный контур, а нулями обозначен фон.

BlackWhiteIm- бинарное изображение с белыми объектами на черном фоне.

Скорость выполнения функции activecontour, без учета выбора областей поиска, заметно меньше, чем скорость сегментации по пороговому уровню и в среднем равна 1.750 секунд. Accuracy данного метода составило 0,8761 (Приложение ).

|  |  |
| --- | --- |
|  |  |

А) Б)

Рисунок 23 - А - исходное изображение, Б - изображение, полученное после сегментации по активному контуру

Можно выделить несколько недостатков данного подхода:

1. Метод требует некоторую предварительную обработку изображений, а именно определение областей, в которых находятся объекты.
2. Не ясно каким образом происходит разделение на объект/фон. То есть используемый подход для определения контуров. Возможно, он не является оптимальным и занимает слишком существенное время.
3. Метод не решает проблему появления шумов, соответственно изображение все равно требует пост обработки.

### Сегментация по движению относительно фона

Рассмотренные методы решают задачу сегментации для одного изображения. В динамических системах часто приоритетным является сегментация не столько точная, но медленная, сколько быстрая, но не такая точная. В этом случае мы работаем не с одним кадром (изображением), а с последовательностью кадров. Здесь определение фона, объектов и даже расстояний до объектов производится посредством сравнения нового кадра с предыдущим. Так объект, располагающийся ближе к камере, будет перемещаться быстрее, чем более дальние объекты, как в случае движения объекта, так и в случае движения и камеры, и объекта. В результате этого может быть искажена форма сегментируемой области, но проблема слияния объекта и фона практически пропадает за счет разной освещённости для разных объектов сцены, даже в случае их одинакового цвета и соответственно проблемы сегментации по контуру.

На рис. 24. наглядно видно, что более быстрое, относительно других точек кадра, смещение руки с ручкой, выделенное белым фоном более заметно, относительно смещения головы или более дальних объектов, в результате небольшого движения камерой (приложение А).

|  |  |
| --- | --- |
|  |  |

А) Б)

Рисунок 24 - А - исходный кадр, Б - разница между соседними кадрами видеопотока



Рисунок 25 - Кадр, сегментированный по активному контуру

На рис. 25 хорошо заметна разница, между сегментацией по движению и сегментацией по активному контуру. При сегментировании по активному контуру было утрачено самое главное- форма объекта, которая необходима на этапе распознавания. Даже при условии четкого выделения границ объекта распознавания, мы видим много посторонних шумов и посторонних объектов, от части которых не представляется возможным избавиться даже с помощью классификатора.

В то же самое время нельзя назвать сегментацию по движению идеальным вариантом выполнения задачи. Возникают проблемы, когда объект распознавания оказывается неподвижным, в результате чего он становится как бы «невидимым» для данного подхода. А для достаточно больших объектов возникает проблема, которая называется апертурной проблемой.

Для динамических систем реального времени оценить скорость выполнения сегментации по движению достаточно сложно. Необходимо учитывать не только разрешение камеры, но также скорость записи файла, загрузки в оперативную память устройства двух соседних кадров и работа с ними. На стационарном компьютере выполнение операции сегментации по двум кадрам, уже загруженным в память занимает не более 0.5 секунд. Accuracy данного метода составило 0,9735. Значения на реальной установке могут сильно отличаться в том числе за счёт меньших вычислительных ресурсов.

Таким образом, можно сделать следующие выводы по данным трём классическим методам сегментации:

Сегментация по пороговому уровню является самой быстрой, но имеет ряд ограничений. Она может использоваться, например, в следящих системах для передвижения по заранее нарисованной полосе или же для определения объектов на статичном однородном фоне.

Сегментация по активному контуру, в отличие от предыдущей более универсальна, но в свою очередь имеет ограничения по быстродействию, возможности слияния фона и объекта.

Сегментация по движению кажется наиболее удачной с практической точки зрения, хотя не может быть реализована в отрыве от других в связи с возможными проблемами потери объекта для статического изображения и апертурной проблеме.

## Реализация сегментации на основе глубокого обучения

В результате анализа подходов к сегментации на основе глубокого обучения, было принято решение использовать архитектуру полностью свёрточной нейронной сети, в частности сети U-net. В результате поиска реализаций сети U-net был найден код для реализации и обучения нейронной сети. Код на языке python, представлен в приложении Б.

Поскольку для обучения сети необходимо много размеченных данных, на которых будет происходить обучение. Было решено использовать Sityscape датасет.

Набор данных Cityscapes фокусируется на семантическом понимании городских уличных сцен. Поскольку он создать как раз для обучения определения объектов сцены, пусть даже и не в помещении, но на улице. Суть сегментации остаётся той же самой.

В датасете Cityscapes содержатся 25000 аннотированных изображений, из 50 различных городов Европы, снятых в разные времена года (весна, лето, осень) и в разное время и освещение, что делает обученную на нём модель более гибкой и достигающей приемлемой точности (пример изображений на рисунке 26).

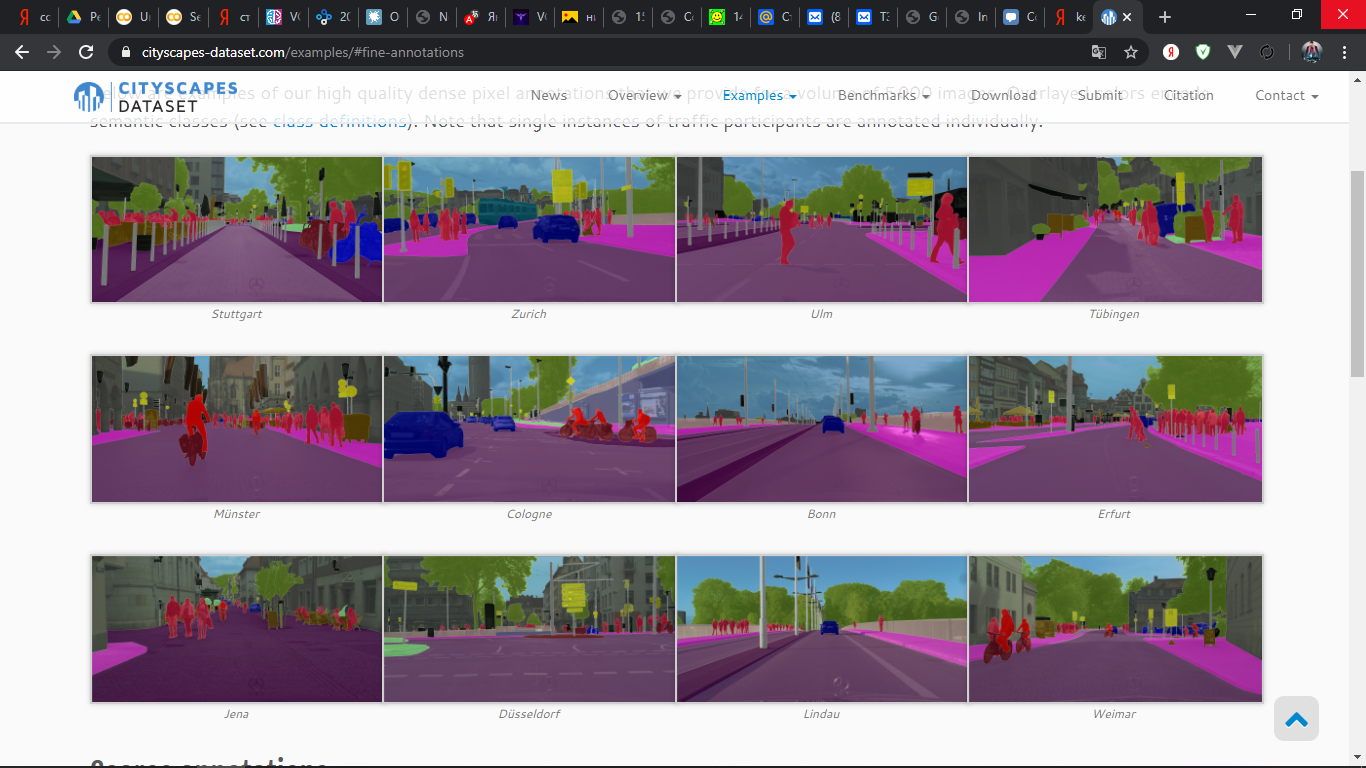


Рисунок 26 – Пример аннотированных изображений в датасете Cityscapes

Однако, структура данной сети не предусматривает дообучение, которое необходимо в данной работе, поскольку хорошо натренированная сеть на сегментацию объектов дорожного движения не будет способна сегментировать объекты, встречающиеся в помещении, где будет двигаться робот, поэтому было принято решение найти U-net подобную сеть с возможностью дообучения и, желательно, обученную, чтобы не производить ресурсоёмкое обучение на своей машине.

В ходе поиска возможных вариантов и изучения библиотек keras и tensorflow, было принято решение воспользоваться стандартной моделью для классификации изображений, представленной в библиотеке keras. А именно использовать VGG-16. Данная сеть, по результатам тестов для задачи классификации даёт результаты лучше многих других возможных сетей. Однако для нашей задачи сегментации изображений она должна быть доработана, к ней необходимо добавить некоторое количество развёрточных слоёв (deconvolution layer), таким образом свёрточная нейронная сеть VGG-16 с deconvolution слоями на выходе будет иметь вид U-net сети (наглядное сравнение на рисунке 27).

|  |  |
| --- | --- |
| А) | image  Б) |

Рисунок 27 – А – модель VGG-16 свёрточной сети Б – модель U-net сети

Преимущество VGG-16 в лёгкости и удобности использования, поскольку в библиотеке keras предусмотрена возможность выгрузки сети с весами, обученными на датасете imagenet (о ней будет сказано ниже), а также удобство в возможности задать слоям параметр layer.trainable = False что позволит «заморозить» первые слои, то есть сделать их не обучаемыми, поскольку операция down sampling в нейросетях позволяет выделять признаки из изображений, что одинаково для сегментирования любых объектов на изображении. Таким образом, используя свёрточную VGG-16 сеть, отсекая несколько последних уровней, отвечающих за классификацию изображений и заменяя их на deconvolution слои можно добиться достаточной точности сегментированных изображений даже на небольшом объёме дообучающего датасета.

Датасет ImageNet - это база данных изображений, организованная в соответствии с иерархией WordNet, в которой каждый узел иерархии представлен сотнями и тысячами изображений. Аннотации на уровне самих изображений показывают наличие или отсутствие объекта данного класса (например, «на картинке имеется тигр» или «на картинке нет тигров»). На уровне объекта в аннотацию включается прямоугольник с координатами видимой части объекта. ImageNet использует вариант семантической сети WordNet для категоризации объектов, которая достаточно детализирована, например, породы собак представлены 120 классами. Каждому узлу сети WordNet сопоставлены сотни или тысячи изображений, но в среднем на 2016 год — около 500 изображений. На август 2017 года в ImageNet 14 197 122 изображения, разбитых на 21 841 категорию (пример категорий на рисунке 28).

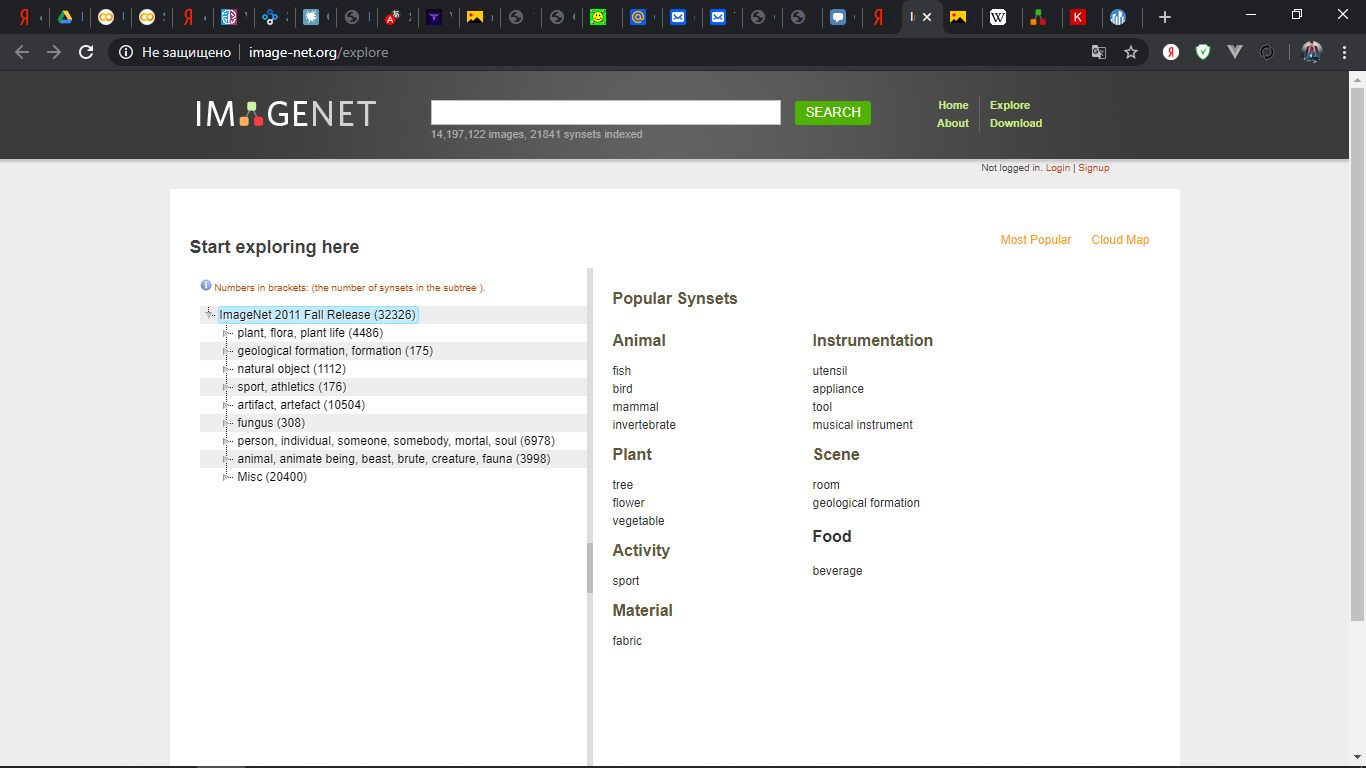


Рисунок 28 – Дерево классов imagenet

В приложении В представлен код на языке python, в результате выполнения которого были получены дальнейшие результаты.

В таблице 4 представлено примерное сравнение классических и нейросетевых моделей:

Таблица 4 – Сравнение классических и нейросетевых моделей

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Среднее время сегментации одного изображения, сек | Занимаемый объём, Мб | Точность сегментации |
| Сегментация по пороговому уровню | 0.02 | - | 0,1068 |
| Сегментация по активному контуру | 1,75 | - | 0,2519 |
| Сегментация по движению относительно фона | 0,5 | - | 0,0511 |
| Модель VGG-16 (19 слоёв) | 0,019 | 56,19 | >0,6 |
| Модель Xception (132 слоёв) | 0,038 | 79,95 | - |
| Модель ResNet50 (175 слоёв) | 0,039 | 90,45 | - |
| Модель DenseNet121  (427 слоёв) | 0,078 | 27,94 | - |

## Формирование набора данных для доучивания

Для дообучения сети с VGG-16 была произведена сегментация кадров, снятых с камеры робота. Сегментация производилась вручную, с помощью приложения, втроенного в среду matlab Image Labeler. Интерфейс данного приложения выглядит следующим образом (рисунок 29):

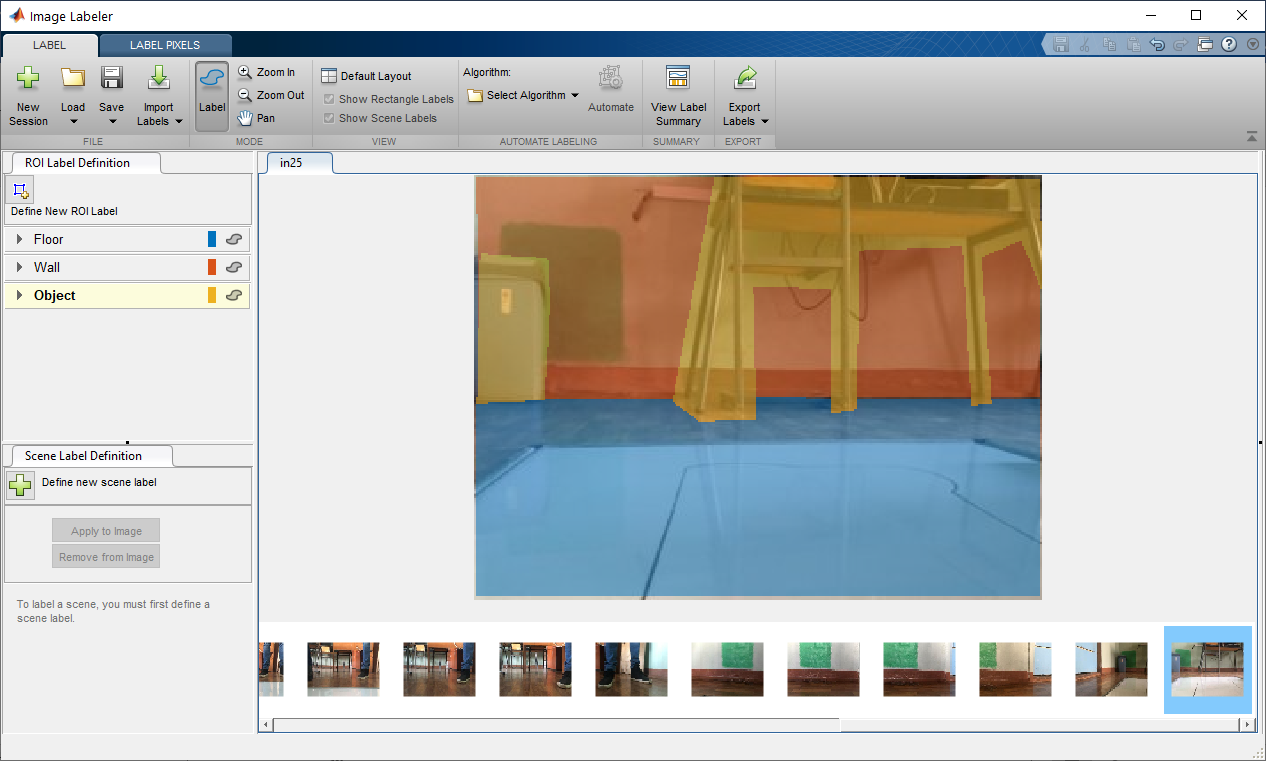


Рисунок 29 – Окно приложения Image Labeler

Производилась сегментация изображений с камеры робота (рисунок 31). Для сцены, в которой двигался робот была произведена сегментация по 3 классам: Пол (floor), стена(wall), объект(object). В результате сегментации случайно взятых изображений были получены сегментированные .bmp файлы, которые, в результате преобразования (операция преобразования реализована на в программной среде matlab, представлена в приложении Г) стали выглядеть следующим образом (рисунки 30, 32):



Рисунок 30 – Одно из сегментированных изображений для дообучения

Таким образом был получен собственный датасет для дообучения

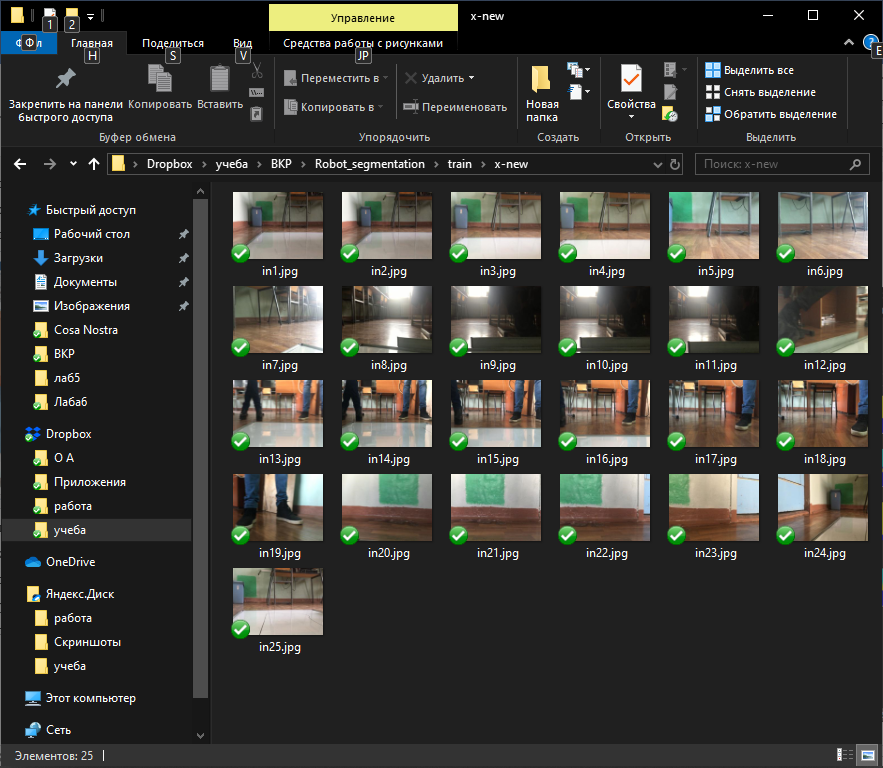


Рисунок 31 – Сегментируемые изображения

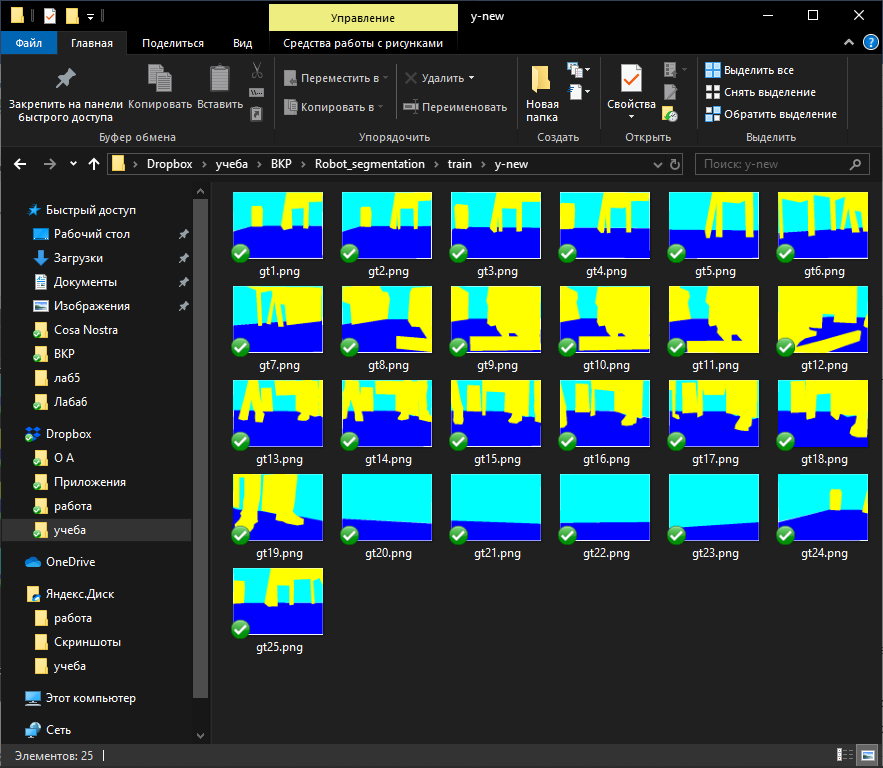


Рисунок 32 – Сегментированные изображения

## Тестирование модели

### Влияние параметров модели на результат сегментации

Обучение производилось на сегментированном вручную датасете из 45 изображений.

Ниже для наглядности показаны результаты сегментации изображения, не входящего в обучающий датасет, представленного на рисунке 33 при изменении различных параметров сети.



Рисунок 33 – Сегментированные изображения

Определим зависимость результата сегментации от изменения количества выходных слоёв:

Для трёх слоёв сети (рисунок 34):

|  |  |
| --- | --- |
|  | (loss=426.4626, accuracy=0.3931) |
|  | (loss=418.0788, accuracy=0.4145) |
|  | (loss=424.0211, accuracy=0.3867) |
|  | (loss=425.7204, accuracy=0.3882) |
|  | (loss=424.5309, accuracy=0.3587) |

А) Б) В)

Рисунок 34 – А- структура выхода сети, Б- вручную сегментированное изображение, В- результат сегментации представленной моделью



Рисунок 35 – Зависимость параметра loss от количества тренируемых параметров сети для 3 выходных слоёв сети

На основании рисунков 34, 35 можно сделать вывод, что результат сегментации, как правило, лучше, когда размер ядра в слое меньше.

Для четырёх слоёв сети (рисунок 36):

|  |  |
| --- | --- |
|  | (loss=296.3184, accuracy=0.6259) |
|  | (loss=295.7778, accuracy=0.5950) |
|  | (loss=297.6985, accuracy=0.5814) |
|  | (loss=294.1549, accuracy=0.5709) |

А) Б) В)

Рисунок 36 – А- структура выхода сети, Б- вручную сегментированное изображение, В- результат сегментации представленной моделью



Рисунок 37 – Зависимость параметра loss от количества тренируемых параметров сети для 4 выходных слоёв сети

На основании рисунков 36, 37 можно сделать вывод, что результат сегментации практически не зависит от размера ядер слоёв сети.

Для пяти слоёв сети (рисунок 38):

|  |  |
| --- | --- |
|  | (loss= 257.3914, accuracy= 0.6355) |
|  | (loss= 253.8228, accuracy= 0.6885) |

А) Б) В)

Рисунок 38 – А- структура выхода сети, Б- вручную сегментированное изображение, В- результат сегментации представленной моделью



Рисунок 39 – Зависимость параметра loss от количества тренируемых параметров сети для 5 выходных слоёв сети

На основании рисунков 38, 39 можно сделать вывод, что ещё один слой сети дал существенный прирост точности, и результат сегментации практически не зависит от размера ядер слоёв сети.

Для шести слоёв сети (рисунок 40):

|  |  |
| --- | --- |
|  | (loss= 250.4580, accuracy= 0.6723) |

А) Б) В)

Рисунок 40 – А- структура выхода сети, Б- вручную сегментированное изображение, В- результат сегментации представленной моделью

Количество тренируемых параметров сети для случая 6 слоёв равно 3931683.

На основании рисунка 40 можно сделать вывод, что добавление ещё одного слоя не ведёт к увеличению точности, относительно структуры сети с 5 слоями.



Рисунок 41 – Зависимость параметра loss от количества тренируемых параметров сети для разного количества выходных слоёв сети

На основании рисунков 34-41 можно сделать вывод, что результат сегментации лучше, когда количество слоёв сети больше. Выберем оптимальный вариант с 5 слоями сети.

Далее определим зависимость результата сегментации от размера обучающего датасета (рисунок 42):

|  |  |
| --- | --- |
| 45 примеров  (loss= 253.2783, accuracy= 0.7399) |  |
| 35 примеров  (loss= 244.6610, accuracy= 0.7313) |  |
| 25 примеров  (loss= 243.1896, accuracy= 0.6458) |  |
| 15 примеров  (loss= 254.1213, accuracy= 0.4778) |  |
| 5 примеров  (loss= 221.4896, accuracy= 0.6242) |  |

А) Б) В)

Рисунок 42 – А- количество изображений в обучающем датасете, Б- вручную сегментированное изображение, В- результат сегментации представленной моделью



Рисунок 43 – Зависимость параметра loss от количества изображений в тренировочном датасете

На основании рисунков 42, 43 можно сделать вывод, что результат сегментации лучше, когда примеров сегментации больше, поскольку значение accuracy больше, не смотря на значение функции потерь, которая становится меньше, поскольку количества примеров не хватает для корректного обучения модели.

Определим зависимость результата сегментации от количества эпох обучения (рисунок 44):

|  |  |
| --- | --- |
| 19 эпох  (loss= 255.3817, accuracy= 0.6965) |  |
| 14 эпох  (loss= 258.3974, accuracy= 0.6935) |  |
| 9 эпох  (loss= 274.4977, accuracy= 0.5657) |  |
| 4 эпохи  (loss= 314.5171, accuracy= 0.5921) |  |

А) Б) В)

Рисунок 44 – А- количество эпох обучения, Б- вручную сегментированное изображение, В- результат сегментации представленной моделью



Рисунок 45 – Зависимость параметра loss от количества эпох обучения

На основании рисунков 44, 45 можно сделать вывод, что результат сегментации лучше, когда обучение длится больше эпох (на определённом этапе точность перестаёт увеличиваться и может возникнуть переобучение).

Также была произведена настройка по использованному оптимизатору (методу обучения). Результаты для 5 слоёв сети и 45 обучающих примеров следующие (рисунок 46):

|  |  |
| --- | --- |
| RMSprop  19 эпох  (loss= 254.5419, accuracy= 0.6235) |  |
| SGD  31 эпох  (loss= 265.8085, accuracy= 0.6482) |  |
| Adam  14 эпох  (loss= 255.8210, accuracy= 0.5744) |  |
| Adadelta  100 эпох  (loss=460.5457, accuracy=0.3414) |  |
| Adagrad  17 эпох  (loss=265.6629, accuracy=0.6328) |  |
| Adamax  16 эпох  (loss=256.9841, accuracy=0.6575) |  |

А) Б) В)

Рисунок 46 – А- условное обозначение оптимизатора, сколько эпох заняло обучение с этим оптимизатором, Б- вручную сегментированное изображение, В- результат сегментации представленной моделью



Рисунок 47 – Зависимость параметра loss от количества эпох обучения

На основании рисунков 46, 47 можно сделать вывод, что лучший результат сегментации показывают оптимизаторы: RMSprop, Adam, Adagrad, Adamax. Однако лучший результат показал оптимизатор RMSprop, на нём и остановим выбор параметров модели.

### Гиперпараметры и характеристики сети

В результате изучения влияния параметров сети на результат сегментации была выбрана следующая конфигурация модели: 5 выходных слоёв, максимально возможное количество обучающих данных (45 изображений), обучение пока не будет возникать переобучение (в среднем 19-20 эпох).

При обучении сети VGG-16 использовалась категориальная перекрёстная (categorical crossentory) функция потерь.

Категориальная перекрёстная энтропия (categorical crossentory):

, где

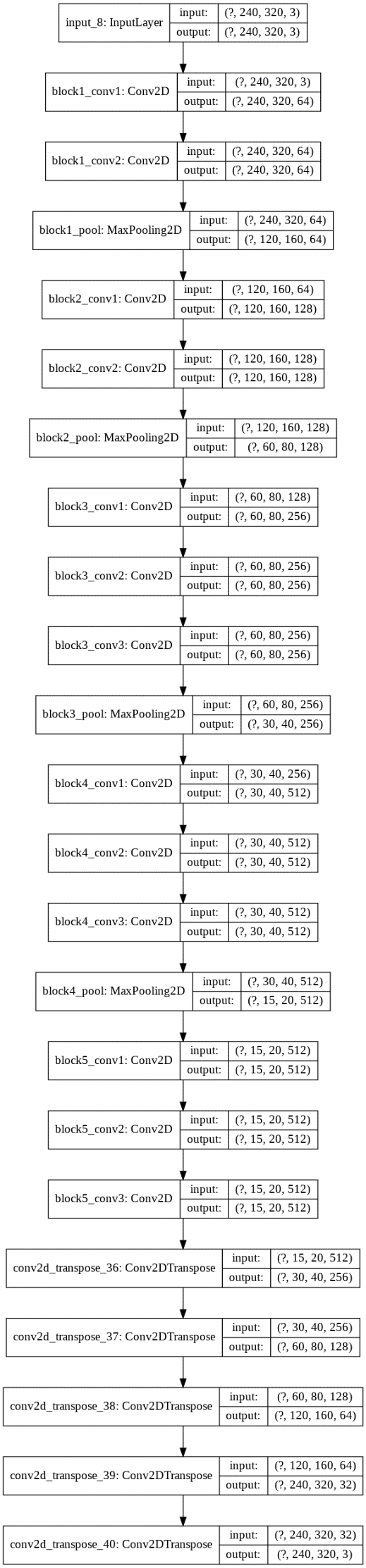
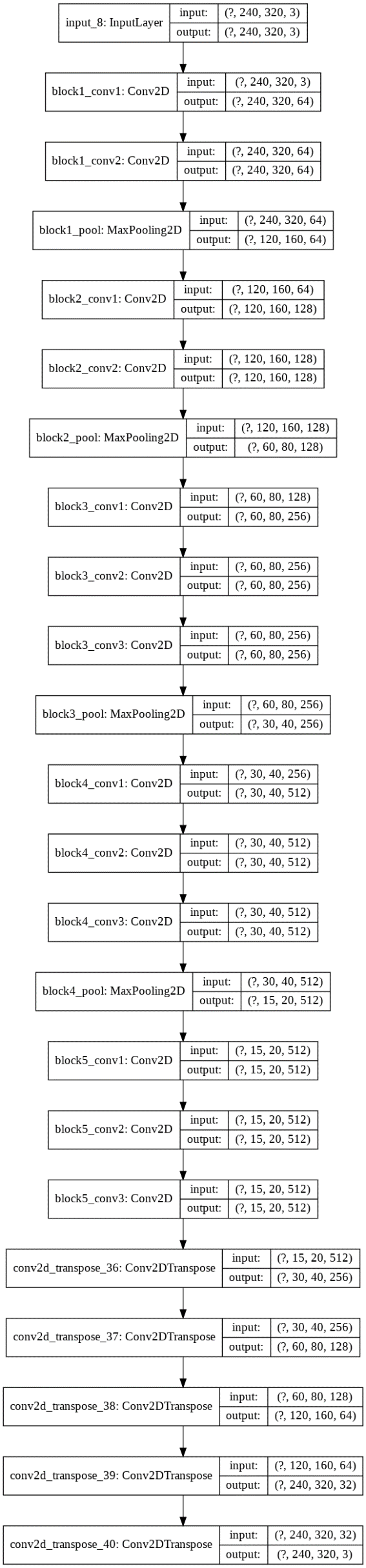
- вероятность прогнозируемого значения,

- истинное значение,

- размер вектора .

Также использовался адаптивный метод обучения RMSprop (root mean square propagation). В основе RMSprop лежит метод градиентного спуска, однако используется только знак градиента, а шаг адаптируется в ходе обучения.

Схема сети выглядит следующим образом (рисунок 48):



А) Б)

Рисунок 48 – Полная схема сети. А- первая часть, Б- вторая часть

Для выбранной модели обучения было произведено 10 перезапусков операции обучения и были получены следующие графики кривой обучения (рисунок 49):



Рисунок 49 – Кривые обучения для 10 реализаций сети

Исходя из приведённых на рисунке 49 графиков, определим ошибку обучения.

Среднее значение loss для 10 реализаций равно , дисперсия составляет , что говорит о достаточно малой ошибке обучения и хорошей устойчивости найденных решений (в процессе обучения нет стремления к локальному минимуму).

## Анализ результатов работы модели

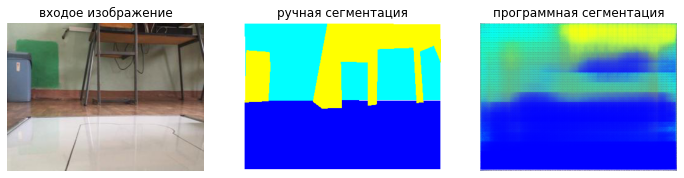
Было произведено несколько экспериментов:

Было произведено обучение сети на датасете из 45 сегментированных изображений, без предварительно обученных на ImageNet весов, в результате чего обучение заняло 11 эпох и кривые обучения для loss и accuracy представлена на рисунке 50:



Рисунок 50 – Кривые обучения для loss и accuracy без предварительно обученных весов

Тестирование сегментации происходило на изображении, не участвующем в обучении для большей достоверности. Результат сегментации модели без предобучения на ImageNet (для изображения, не участвующем в обучении) (рисунок 51):



А) Б) В)

Рисунок 51 – А – входное изображение с камеры робота, Б – вручную сегментированное изображение. В – результат программной сегментации изображений

Значение функции потерь в данном случае равно 300.3045, а точность равна 0.6272.

Также было произведено обучение на том же датасете из 45 сегментированных изображений с использованием предварительно обученных на ImageNet весов, с заморозкой свёрточных слоёв и обучением только deconvolution слоёв. Таким образом обучение заняло 17 эпох эпох и кривые обучения для loss и accuracy представлены на рисунке 52:



Рисунок 52 – Кривые обучения для loss и accuracy c предварительно обученными весами

Тестирование также происходило на изображении из той же сцены, но не участвующем в обучении. Результат сегментации модели, предобученной на ImageNet (для изображения, не участвующем в обучении) (рисунок 53):

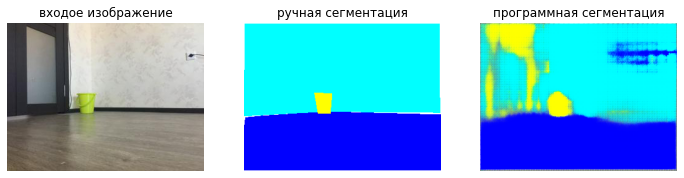


А) Б) В)

Рисунок 53 – А – входное изображение с камеры робота, Б – вручную сегментированное изображение. В – результат программной сегментации изображений

Значение функции потерь в данном случае равно 247.2975, а точность равна 0.6530.

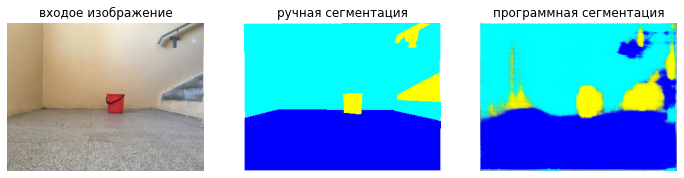
Для полноты эксперимента было произведено тестирование на изображениях из других сцен, также не участвующих в обучении модели. Результат сегментации модели, предобученной на ImageNet (рисуноки 54-57):



А) Б) В)

Рисунок 54 – А – входное изображение с камеры робота, Б – вручную сегментированное изображение. В – результат программной сегментации изображений

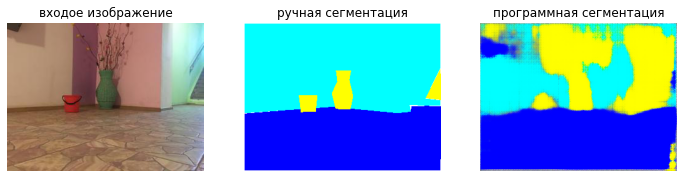
Значение функции потерь в данном случае равно 448.1728, а точность равна 0.7351.



А) Б) В)

Рисунок 55 – А – входное изображение с камеры робота, Б – вручную сегментированное изображение. В – результат программной сегментации изображений

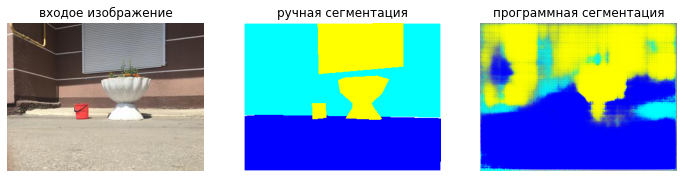
Значение функции потерь в данном случае равно 345.4741, а точность равна 0.8088.



А) Б) В)

Рисунок 56 – А – входное изображение с камеры робота, Б – вручную сегментированное изображение. В – результат программной сегментации изображений

Значение функции потерь в данном случае равно 645.2115, а точность равна 0.7842.



А) Б) В)

Рисунок 57 – А – входное изображение с камеры робота, Б – вручную сегментированное изображение. В – результат программной сегментации изображений

Значение функции потерь в данном случае равно 448.1728, а точность равна 0.7351.

На представленных выше изображениях видно, что результаты сегментации получились достаточно точными, для использования на автономной мобильной платформе, хоть вес полученной модели оказался 124,31 Мб, такая модель может поместиться на Raspberry Pi 3 и быть использована, поскольку операция сегментирования на ней происходит в среднем за 20 мс, что приемлемо, учитывая скорость мобильной платформы и поставленные перед ней задачи.

Стоит отметить, что задача семантической сегментации очень важна для корректной работы робота. Предполагается, что детектирование и распознавание объектов будет происходить не постоянно, а только для объектов, которые были выделены на более быстром и менее ресурсоёмком процессе сегментации, поскольку вычислительные ресурсы микрокомпьютера необходимы для принятия решений роботом, находящимся в сцене.

## Экономическая часть

### Экономическое обоснование дипломного проекта

В дипломном проекте проводится научно-исследовательская работа по разработке модели системы технического зрения для анализа положения автономной платформы. Выполним расчеты затрат на выполнение научно-исследовательской работы (НИР), а именно, на изучение теоретических положений, разработку и реализацию алгоритмов калибровки и позиционирования объекта в сцене.

Научно-исследовательская работа проведена одним инженером-программистом. Затраты, возникающие при выполнении научно-исследовательской работе, включают в себя следующие пункты:

1. затраты по основной заработной плате;
2. затраты по дополнительной заработной плате;
3. затраты по отчислениям на социальное страхование.

Выполненную работу можно разделить на следующие этапы:

1. разработка технического задания;
2. изучение темы;
3. научно-теоретическое исследование;
4. научно-практическое исследование (написание программы);
5. оформление работы.

### Организация и планирование работ

Все работы делятся на три основных этапа:

1. Исследовательская часть (теоретическая база);
2. Конструкторская часть (разработка алгоритмов, написание программ);
3. Настройка, выпуск сопроводительной документации.

Для оптимального планирования работ разрабатывается календарный график, который показывает время начала каждой работы относительно начала проекта и её продолжительность.

Трудоемкость и состав исполнителей определяется по технологии для каждой конкретной работы. Планирование работ представлено в таблице 5.

Таблица 5 – Планирование работ по теме

|  |  |  |  |
| --- | --- | --- | --- |
| **№ п/п** | Перечень работ | Исполнители | Трудоёмкость  работы (дни) |
| 1 | Разработка ТЗ | Инженер-программист | 1 |
| 2 | Изучение темы | Инженер-программист | 3 |
| 3 | Научно-теоретическое исследование | Инженер-программист | 5 |
| 4 | Научно-практическое исследование | Инженер-программист | 5 |
| 5 | Разработка документации | Инженер-программист | 5 |

### Определение стоимости специального оборудования

Расчет стоимости материалов и покупных изделий. На эту статью относится стоимость материалов, покупных изделий, комплектующих и других материальных ценностей, расходуемых непосредственно в процессе выполнения работ по теме.

Для данного проекта используется новое покупное оборудование: аппаратно-вычислительная платформа (компьютер), WEB-камера и среда программирования MATLAB Home R2018a. Смета на приобретение оборудования и ПО представлена в таблице 6.

Таблица 6 – Смета на приобретение оборудования и ПО

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Наименование изделий | Количество,  шт | Стоимость,  руб. | Общая  стоимость  руб. |
| 1 | Аппаратно-вычислительная платформа | 1 | 50 000 | 50 000 |
| 2 | WEB - камера | 1 | 2 000 | 2 000 |
| 3 | MATLAB Home R2018a | 1 | 17 000 | 17 000 |
| **Итого стоимость оборудования:** | | | | **69 000** |

### Расчет основной заработной платы

Заработная плата непосредственных исполнителей рассчитывается исходя из количества исполнителей, их оклада и времени, затраченного на выполнения работ.

Так как календарные сроки проведения работ по проектированию неизвестны, то принято, что в месяце 21 рабочий день.

Заработная плата основного персонала представлена в таблице 7.

Таблица 7 – Заработная плата основного персонала

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Исполнители | Месячный оклад, руб. | Оклад,  руб./день | Количество отработанных дней | Выплаты, руб. |
| 1 | Инженер-программист | 40 000 | 1900 | 19 | 36 100 |
| **Итого выплаты по основной зарплате:** | | | | | **36 100** |

Выплаты по основной заработной плате составят **36 100 руб**.

а) Дополнительная заработная плата по данному виду работ не предусматривается.

б) На настоящий момент, общая сумма страховых платежей составляет 30 процентов от фонда оплаты труда. Из них 22 процента - обязательный пенсионный взнос. В Фонд обязательного медицинского страхования пойдут 5,1 процента, а в Фонд социального страхования - оставшиеся 2,9 процента. Процентная ставка и размер начислений представлены в таблице 8.

Таблица 8 – Начисления на заработную плату

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Наименование начисления на заработную плату -  сумма страховых платежей | Процент | Начисления, руб. |
| 1 | Пенсионный фонд | 22% | 7 942,00 |
| 2 | ФСС - Фонд социального страхования | 2,9% | 1 046,90 |
| 3 | ФОМС - Фонд обязательного медицинского страхования | 5,1% | 1 841,10 |
| **Итого страховых платежей:** | | | 10 830,00 |

в) По данному виду работы расходы по статьям: расходы на научные и производственные командировки; оплата работ, выполняемых сторонними организациями; прочие расходы не предусматривается.

Проведем калькуляцию статей расходов (таблица 9) и рассчитаем общую сумму затрат на разработку модели сегментации изображений.

Таблица 9 – Калькуляция статей расходов

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Наименование статей расхода** | **Затраты, руб.** |
| 1 | Специальное оборудование и ПО | 69 000 |
| 2 | Основная зарплата | 36 100 |
| 3 | Дополнительная зарплата | 7 220 |
| 4 | Сумма страховых платежей | 10 830 |
| 5 | Расходы на научные и производственные командировки | - |
| 6 | Оплата работ, выполняемых сторонними организациями | - |
| 7 | Прочие затраты | - |
|  | **Итого затрат:** | 123 150 |

### Выводы по экономическому разделу дипломного проекта

В данном разделе выпускной квалификационной работы рассчитали стоимость оборудования, необходимого для работы системы, выплаты по основной заработной плате, а также сумму страховых платежей.

Время, необходимое для разработки системы, составляет **19 дней.**

Итоговая стоимость разработки системы сегментирования изображения для автономной платформы составляет **123 150 рублей.**

## Охрана труда

**Характеристика микроклимата на рабочем месте.**

Показателями, характеризующими микроклимат, являются:

* температура воздуха;
* относительная влажность воздуха;

Параметры микроклимата рабочей зоны должны соответствовать требованиям ГОСТ 12.1.005-88. Значения параметров микроклимата:

* *в холодный период*: оптимальная температура воздуха 18-21 °С, допустимая температура: верхняя граница 21 °С , нижняя граница 17°С;
* относительная влажность: оптимальная 40-60%, допустимая 75%.
* *в теплый период*: оптимальная температура воздуха 20-25 °С, допустимая температура: верхняя граница 28 °С , нижняя граница 13°С;
* относительная влажность: оптимальная 40-60%;
* скорость движения воздуха на рабочем месте в холодный и теплый период 0,1м/с.

**Характеристика условий освещения**

Исходные данные:

1. Площадь помещения - 5 кв. м

2. Высота свеса светильника - 1,9 м

3. Тип помещения - Кабинеты

4. Коэффициент запаса - 1,4

5. Освещенность помещения - 300 Люкс

6. Тип светильника - ЛПО 18-(2х40)

Рассчитаем необходимое количество светильников в мультимедийном приложении, разработанном кафедрой «Промышленная экология» (2012г.)

Результат расчета: необходимо установить ***2*** *светильника ЛПО 18-(2х40).*

**Эргономика рабочего места оператора**

Главными элементами рабочего места программиста являются письменный стол и кресло. Основным рабочим положением является положение сидя. Рабочее место для выполнения работ в положении сидя организуется в соответствии с ГОСТ 12.2.032-78.

Рабочая поза сидя вызывает минимальное утомление программиста. Рациональная планировка рабочего места (рисунок 42) предусматривает четкий порядок и постоянство размещения предметов, средств труда и документации. То, что требуется для выполнения работ чаще, расположено в зоне легкой досягаемости рабочего пространства.

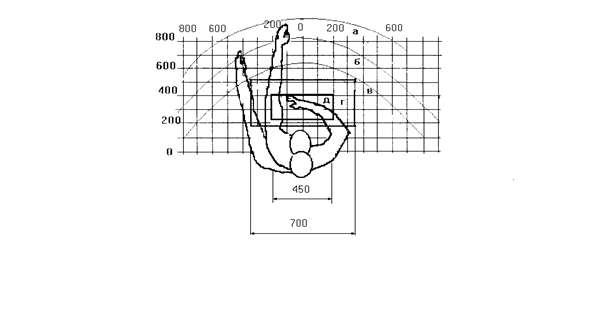


Рисунок 58 – Зоны досягаемости рук в горизонтальной плоскости

 Где:

а *—*зона максимальной досягаемости;

б — зона досягаемости пальцев при вытянутой руке;

в — зона легкой досягаемости ладони;

г — оптимальное пространство для грубой ручной работы;

д *—*оптимальное пространство для тонкой ручной работы.

Рассмотрим оптимальное размещение предметов труда и документации в зонах досягаемости рук:

ДИСПЛЕЙ размещается в зоне а (в центре);

КЛАВИАТУРА — в зоне г/д;

СИСТЕМНЫЙ БЛОК размещается в зоне б (слева);

ПРИНТЕР находится в зоне а (справа);

ДОКУМЕНТАЦИЯ:

1. в зоне легкой досягаемости ладони — **в** (слева) — литература и документация, необходимая при работе;
2. в выдвижных ящиках стола — литература, неиспользуемая постоянно.

 При проектировании письменного стола следует учитывать следующее:

* высота стола должна быть выбрана с учетом возможности сидеть свободно, в удобной позе, при необходимости опираясь на подлокотники;
* нижняя часть стола должна быть сконструирована так, чтобы программист мог удобно сидеть, не был вынужден поджимать ноги;
* поверхность стола должна обладать свойствами, исключающими появление бликов в поле зрения программиста;
* конструкция стола должна предусматривать наличие выдвижных ящиков (не менее 3 для хранения документации, листингов, канцелярских принадлежностей, личных вещей).

Важным элементом рабочего места программиста является кресло. Оно выполняется в соответствии с ГОСТ 21.889-76.

## Экологическая безопасность

**Общая характеристика источников вредного воздействия на среду**

Наиболее опасным фактором, воздействующим на окружающую среду при разработке данного проекта, является наличие люминесцентных ламп в помещении. Эти лампы, при неправильной утилизации оказывают вредное воздействие на окружающую среду. Чтобы снизить вред, оказываемый на окружающую среду, лампы утилизируют.

**Утилизация люминесцентных ламп.**

Ртутные лампы и люминесцентные ртутьсодержащие трубки (далее – ртутьсодержащие лампы) представляют собой газоразрядные источники света, принцип действия которых заключается в следующем: под воздействием электрического поля в парах ртути, закачанной в герметическую стеклянную трубку, возникает электрический разряд, сопровождающийся ультрафиолетовым излучением. Нанесенный на внутреннюю поверхность люминофор преобразует ультрафиолетовое излучение в видимый свет.

Люминесцентные лампы относят к отходам 1 класса опасности, которые являются чрезвычайно опасными отходами. Степень вредного влияния отходов подобного класса опасности на окружающую среду очень высокая. Когда на окружающую среду воздействуют отходы первого класса опасности, то, во-первых она необратимо нарушается, а во-вторых период её восстановления отсутствует.

Одним из наиболее опасных факторов, загрязняющих окружающую среду, является ртуть и её пары. Органолептические свойства паров ртути таковы, что пораженный не замечает, как вдыхает их. При этом поражается центральная нервная и мочевыделительная системы. Ртуть, попавшая в окружающую среду, препятствует нормальному функционированию организма. Длительное нахождение ртути в условиях экосистемы приводит к её деградации.

Правила утилизации и правила работы со ртутью соответствуют нормам: СанПиН 2.1.7.1322-03 «Гигиенические требования к размещению и обезвреживанию отходов производства и потребления»;

СП 4607-88 «Санитарные правила при работе со ртутью, ее соединениями и приборами с ртутным заполнением» (утв. Главным государственным санитарным врачом СССР 04.04.1988).

**Расчет количества образующихся отходов отработанных люминесцентных ламп**

Для освещения помещения рабочей зоны используются люминесцентные лампы типа ЛДЦ-30 в количестве 4 шт.

Согласно методическим рекомендациям «Справочные материалы по удельным показателям образования важнейших видов отходов производства потребления» 1997 г.

Количество отработанных ламп ЛДЦ-30 вычислим по формуле:



где n – количество установленных ламп;

Т – время работы лампы в сутках;

К – количество рабочих дней в году;

Nс.с– норматив срока службы одной люминесцентной лампы ЛДЦ-30.

Вес отработанной лампы составит 190 г.

Помещение для хранения отделено от производственных и жилых помещений, а также защищено от воздействия химических агрессивных веществ, различных атмосферных осадков, влажности и воды, которая может поступать по поверхности или же из-под грунта.

Отработанные люминесцентные лампы складируют в специальной таре в отдельном помещении, закрытом от доступа посторонних лиц до сдачи их на утилизацию.

**Выводы:**

1. Определены оптимальные параметры микроклимата на рабочем месте.
2. Выбрано эргономичное расположение объектов для работы программиста.
3. Определен порядок утилизации люминесцентных ламп.

# Заключение

На выполнение работы были поставлены следующие задачи:

1. Изучение подходов к методам переноса обучения в моделях обработки изображений.
2. Разработка модели переноса для сегментации.
3. Формирование процедуры доучивания.
4. Анализ результатов и сравнение с результатами ручной сегментации.
5. Оценка вычислительной сложности алгоритмов.

В ходе выполнения работы было сделано следующее:

1. Описаны подходы к методам переноса обучения. Была использована платформа Google Colaboratory для обучения сети, использовалась предобученная на датасете ImageNet свёрточная сеть VGG-16.

2. Создан обучающий датасет в программе MatLab Image Laberer для доучивания сети.

3. Разработан модуль для сегментации изображений для мобильной автономной платформы.

4. Точность сегментации полученной модели на произвольной сцене достигает 80%.

5. Время сегментации одного кадра на полученной модели занимает в среднем 20 мс.

Полученная модель сегментации применима на мобильной автономной платформе и может быть выполнена её ресурсами.

# Список использованных источников

1. М.О. Корлякова, Д.С. Трушков, Н.В. Лохмачев, Е.Ю. Корлякова. Анализ моделей распознавания образов в системах технического зрения для мобильных автономных платформ. Наукоемкие технологии в приборо- и машиностроении и развитие инновационной деятельности в вузе: материалы Всероссийской научно-технической конференции, 13 – 15 ноября 2018 г. Т. 2. –Калуга: Издательство МГТУ им. Н. Э. Баумана, 2018. – С. 13–17. (дата обращения: 25.03.2020)
2. Rue, Håvard. Gaussian Markov random fields: theory and applications / Håvard Rue, Leonhard Held. — CRC Press, 2005. — ISBN 1584884320. (дата обращения: 15.03.20).
3. Москалев Н. С. Виды архитектур нейронных сетей // Молодой ученый. — 2016. — №29. — С. 30-34. — URL https://moluch.ru/archive/133/37121/ (дата обращения: 29.03.20).
4. Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition — URL <https://arxiv.org/pdf/1409.1556.pdf> (дата обращения: 15.03.20).
5. Olaf Ronneberger, Philipp Fischer, Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation — URL <https://arxiv.org/pdf/1505.04597.pdf> (дата обращения: 10.03.20).
6. Москалев Н. С. Виды архитектур нейронных сетей // Молодой ученый. — 2016. — №29. — С. 30-34. — URL https://moluch.ru/archive/133/37121/ (дата обращения: 24.03.20).
7. Хайкин С. Нейронные сети: полный курс = Neural Networks: A Comprehensive Foundation. 2-е изд. — М.: Вильямс, 2006. — 1104 с. — ISBN 0-13-273350-1 (дата обращения: 10.03.20).
8. <https://keras.io/> (дата обращения: 25.04.20).
9. <https://www.tensorflow.org/> (дата обращения: 05.03.20).
10. <https://pytorch.org/> (дата обращения: 18.02.20).
11. Шапиро Л., Стокман Дж. Компьютерное зрение. Агентство электронных изданий «Интермедиатор» — URL http://www.bibliorossica.com/book.html?currBookId=21863 (дата обращения: 13.03.20).

## Приложение А

nam='t6';

tic

filename=[nam '.mp4'];

% Счмиывание видео

v = VideoReader(filename);

count=0;

% Параметры видео

x=v.height;

y=v.width;

frames=[];

video1 = readFrame(v);

se\_disk=strel('disk', 4);

for count=1:2

video2 = readFrame(v);

video2 = readFrame(v);

video2 = readFrame(v);

video2 = readFrame(v);

VIDEO=video2-video1;

video1=video2;

VIDEO=VIDEO.^3; %делаем разность между изображениями более контрастной

% Складываем цвет по 3 каналам видео

gfr=((VIDEO(:,:,1)+VIDEO(:,:,2)+VIDEO(:,:,3)));

% Бинаризация изображения

bw=imbinarize(gfr,'adaptive','Sensitivity',0.4);

figure; imshow(bw);

pause(0.1);

bw=imopen(bw, se\_disk);

% Выделаяем области квадратом

figure(1); showrect(bw,bw2rectc(bw));

end

TI=toc

T=TI/t

% Возвращаем признаки объекта

m=bw2props(bw);

[Y,~,~]=myNeuralNetworkFunction(m);

function showrect(fr,matrica)

%Функция выделяет найденные объекты прямоугольниками

%fr - настоящее изображение

%matrica - в строках этой матрицы содержатся координаты левого верхнего х и

%y и их ширина и высота соответственно [x y w h]

imshow(fr);

hold on

[n,~]=size(matrica);

for i=1:n

rectangle('Position',matrica(i,:), 'EdgeColor','r','LineWidth',2 )

end

hold off

end

function [matrica] = bw2rectc(bw)

%Возвращает матрицу с координатами и размером объектов из бинарного

%изображения

%bw - это бинарное изображение для сегментации

%matrica - в строках этой матрицы содержатся координаты левого верхнего х и

%y и их ширина и высота соответственно [x y w h]

matrica=[];

st = regionprops(bw, 'BoundingBox', 'Area','MinorAxisLength' );

[x,~]=size(st);

for i=1:x

if st(i).Area>2000

a=st(i).BoundingBox;

matrica=[matrica; fix(a)];

end

end

end

function [matrica] = bw2props(bw)

%Возвращает матрицу со свойствами(признаками) объекта

%bw - это бинарное изображение для сегментации

%matrica - вид матрицы: [ [x y w h] area eccentricity orientation

%MajorAxisLength MinorAxisLength] (в строках матрицы)

matrica=[];

st = regionprops(bw, 'BoundingBox', 'Area','Eccentricity','Orientation','MajorAxisLength','MinorAxisLength' );

[x,~]=size(st);

for i=1:x

if st(i).Area>2000

bb=st(i).BoundingBox;

a=st(i).Area;

e=st(i).Eccentricity;

o=st(i).Orientation;

Majal=st(i).MajorAxisLength;

Minal=st(i).MinorAxisLength;

matrica=[matrica; [bb a e o Majal Minal]];

end

end

end

## Приложение Б

%tensorflow\_version 1.x

import numpy as np

import matplotlib.pyplot as plt

import os

from PIL import Image

import keras

from keras.models import Model

from keras.layers import Conv2D, MaxPooling2D, Input, Conv2DTranspose, Concatenate, BatchNormalization, UpSampling2D

from keras.layers import  Dropout, Activation

from keras.optimizers import Adam, SGD

from keras.layers.advanced\_activations import LeakyReLU

from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping

from keras import backend as K

from keras.utils import plot\_model

import tensorflow as tf

import glob

import random

import cv2

from random import shuffle

def image\_generator(files, batch\_size = 32, sz = (256, 256)):

  while True:

    #extract a random batch

    batch = np.random.choice(files, size = batch\_size)

    #variables for collecting batches of inputs and outputs

    batch\_x = []

    batch\_y = []

    for f in batch:

        #get the masks. Note that masks are png files

        mask = Image.open(f'annotations/trimaps/{f[:-4]}.png')

        mask = np.array(mask.resize(sz))

        #preprocess the mask

        mask[mask >= 2] = 0

        mask[mask != 0 ] = 1

        batch\_y.append(mask)

        #preprocess the raw images

        raw = Image.open(f'images/{f}')

        raw = raw.resize(sz)

        raw = np.array(raw)

        #check the number of channels because some of the images are RGBA or GRAY

        if len(raw.shape) == 2:

          raw = np.stack((raw,)\*3, axis=-1)

        else:

          raw = raw[:,:,0:3]

        batch\_x.append(raw)

    #preprocess a batch of images and masks

    batch\_x = np.array(batch\_x)/255.

    batch\_y = np.array(batch\_y)

    batch\_y = np.expand\_dims(batch\_y,3)

    yield (batch\_x, batch\_y)

batch\_size = 32

all\_files = os.listdir('images')

shuffle(all\_files)

split = int(0.95 \* len(all\_files))

#split into training and testing

train\_files = all\_files[0:split]

test\_files  = all\_files[split:]

train\_generator = image\_generator(train\_files, batch\_size = batch\_size)

test\_generator  = image\_generator(test\_files, batch\_size = batch\_size)

x, y= next(train\_generator)

plt.axis('off')

img = x[0]

msk = y[0].squeeze()

msk = np.stack((msk,)\*3, axis=-1)

plt.imshow( np.concatenate([img, msk, img\*msk], axis = 1))

def mean\_iou(y\_true, y\_pred):

    yt0 = y\_true[:,:,:,0]

    yp0 = K.cast(y\_pred[:,:,:,0] > 0.5, 'float32')

    inter = tf.count\_nonzero(tf.logical\_and(tf.equal(yt0, 1), tf.equal(yp0, 1)))

    union = tf.count\_nonzero(tf.add(yt0, yp0))

    iou = tf.where(tf.equal(union, 0), 1., tf.cast(inter/union, 'float32'))

    return iou

def unet(sz = (256, 256, 3)):

  #Размер входного сигнала

  x = Input(sz)

  inputs = x

  #down sampling

  f = 8

  layers = []

  for i in range(0, 6):

    x = Conv2D(f, 3, activation='relu', padding='same') (x)

    x = Conv2D(f, 3, activation='relu', padding='same') (x)

    layers.append(x)

    x = MaxPooling2D() (x)

    f = f\*2

  ff2 = 64

  #bottleneck

  j = len(layers) - 1

  x = Conv2D(f, 3, activation='relu', padding='same') (x)

  x = Conv2D(f, 3, activation='relu', padding='same') (x)

  x = Conv2DTranspose(ff2, 2, strides=(2, 2), padding='same') (x)

  x = Concatenate(axis=3)([x, layers[j]])

  j = j -1

  #upsampling

  for i in range(0, 5):

    ff2 = ff2//2

    f = f // 2

    x = Conv2D(f, 3, activation='relu', padding='same') (x)

    x = Conv2D(f, 3, activation='relu', padding='same') (x)

    x = Conv2DTranspose(ff2, 2, strides=(2, 2), padding='same') (x)

    x = Concatenate(axis=3)([x, layers[j]])

    j = j -1

  #classification

  x = Conv2D(f, 3, activation='relu', padding='same') (x)

  x = Conv2D(f, 3, activation='relu', padding='same') (x)

  outputs = Conv2D(1, 1, activation='sigmoid') (x)

  #model creation

  model = Model(inputs=[inputs], outputs=[outputs])

  model.compile(optimizer = 'rmsprop', loss = 'binary\_crossentropy', metrics = [mean\_iou])

  return model

model = unet()

def build\_callbacks():

        checkpointer = ModelCheckpoint(filepath='unet.h5', verbose=0, save\_best\_only=True, save\_weights\_only=True)

        callbacks = [checkpointer, PlotLearning()]

        return callbacks

# inheritance for training process plot

class PlotLearning(keras.callbacks.Callback):

    def on\_train\_begin(self, logs={}):

        self.i = 0

        self.x = []

        self.losses = []

        self.val\_losses = []

        self.acc = []

        self.val\_acc = []

        #self.fig = plt.figure()

        self.logs = []

    def on\_epoch\_end(self, epoch, logs={}):

        self.logs.append(logs)

        self.x.append(self.i)

        self.losses.append(logs.get('loss'))

        self.val\_losses.append(logs.get('val\_loss'))

        self.acc.append(logs.get('mean\_iou'))

        self.val\_acc.append(logs.get('val\_mean\_iou'))

        self.i += 1

        print('i=',self.i,'loss=',logs.get('loss'),'val\_loss=',logs.get('val\_loss'),'mean\_iou=',logs.get('mean\_iou'),'val\_mean\_iou=',logs.get('val\_mean\_iou'))

        #choose a random test image and preprocess

        path = np.random.choice(test\_files)

        raw = Image.open(f'images/{path}')

        raw = np.array(raw.resize((256, 256)))/255.

        raw = raw[:,:,0:3]

        #predict the mask

        pred = model.predict(np.expand\_dims(raw, 0))

        #mask post-processing

        msk  = pred.squeeze()

        msk = np.stack((msk,)\*3, axis=-1)

        msk[msk >= 0.5] = 1

        msk[msk < 0.5] = 0

        #show the mask and the segmented image

        combined = np.concatenate([raw, msk, raw\* msk], axis = 1)

        plt.axis('off')

        plt.imshow(combined)

        plt.show()

#Training

train\_steps = len(train\_files) //batch\_size

test\_steps = len(test\_files) //batch\_size

model.fit\_generator(train\_generator,

                    epochs = 30, steps\_per\_epoch = train\_steps,validation\_data = test\_generator, validation\_steps = test\_steps,

                    callbacks = build\_callbacks(), verbose = 0)

#Testing

!wget http://r.ddmcdn.com/s\_f/o\_1/cx\_462/cy\_245/cw\_1349/ch\_1349/w\_720/APL/uploads/2015/06/caturday-shutterstock\_149320799.jpg -O test.jpg

raw = Image.open('test.jpg')

raw = np.array(raw.resize((256, 256)))/255.

raw = raw[:,:,0:3]

#predict the mask

pred = model.predict(np.expand\_dims(raw, 0))

#mask post-processing

msk  = pred.squeeze()

msk = np.stack((msk,)\*3, axis=-1)

msk[msk >= 0.5] = 1

msk[msk < 0.5] = 0

#show the mask and the segmented image

combined = np.concatenate([raw, msk, raw\* msk], axis = 1)

plt.axis('off')

plt.imshow(combined)

plt.show()

## Приложение В

! git clone https://github.com/lim-eren/CDnet2014.git

from \_\_future\_\_ import print\_function

import glob, os, numpy as np

from keras.preprocessing import image as kImage

def getData(dataset\_dir):

    X\_list= sorted(glob.glob(os.path.join(dataset\_dir, 'x','\*.jpg')))

    Y\_list = sorted(glob.glob(os.path.join(dataset\_dir, 'y' ,'\*.png')))

    X= []

    Y= []

    for i in range(len(X\_list)):

        # Load input image

        x = kImage.load\_img(X\_list[i])

        x = kImage.img\_to\_array(x)

        X.append(x)

        # Load ground-truth label and encode to value 0 and 1

        x = kImage.load\_img(Y\_list[i])

        x = kImage.img\_to\_array(x)

        Y.append(x)

    X = np.asarray(X)

    Y = np.asarray(Y)

    # Shuffle the training data

    idx = list(range(X.shape[0]))

    np.random.shuffle(idx)

    X = X[idx]

    Y = Y[idx]

    return X, Y

import keras

from keras.models import Model

from keras.layers import Deconvolution2D as Deconv2D, Input

import keras.backend as K

import tensorflow as tf

from keras import regularizers

def initModel():

    ### Encoder

    net\_input = Input(shape=(240,320,3))

    vgg16 = keras.applications.vgg16.VGG16(include\_top=False, weights='imagenet', input\_tensor=net\_input)

    #vgg16 = keras.applications.vgg16.VGG16(include\_top=False, weights=None, input\_tensor=net\_input)

    #for layer in vgg16.layers[:17]:

    #  layer.trainable = True

    for layer in vgg16.layers[:17]:

      layer.trainable = False

    x = vgg16.layers[-2].output # 2nd layer from the last, block5\_conv3

    ### Decoder

    x = Deconv2D(256, (3,3), strides=(2,2), activation='relu', padding='same')(x)

    x = Deconv2D(128, (3,3), strides=(2,2), activation='relu', padding='same')(x)

    x = Deconv2D(64, (3,3), strides=(2,2), activation='relu', padding='same')(x)

    x = Deconv2D(32, (3,3), strides=(2,2), activation='relu', padding='same')(x)

    x = Deconv2D(3, (1,1), activation='sigmoid', padding='same')(x)

    model = Model(inputs=vgg16.input, outputs=x)

    model.compile(loss=keras.losses.categorical\_crossentropy, optimizer=keras.optimizers.RMSprop(lr=5e-4), metrics=['accuracy'])

    return model

# load data

dataset\_path = os.path.join('CDnet2014', 'new\_train')

X, Y = getData(dataset\_path)

# init the model

model = initModel()

early = keras.callbacks.EarlyStopping(monitor='val\_loss', min\_delta=1e-4, patience=10)

reduce = keras.callbacks.ReduceLROnPlateau(monitor='val\_loss', factor=0.1, patience=5)

model.fit(X, Y, batch\_size=1, epochs=100, verbose=2, validation\_split=0.2, callbacks=[reduce, early], shuffle=True)

model.save('my\_model.h5')

from keras import backend as K

import tensorflow as tf

import os

dataset\_path = os.path.join('CDnet2014', 'new\_test')

X, Y = getData(dataset\_path)

pred = model.predict(X, verbose=1, batch\_size=1)

import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (12.0, 9.0)

idx = 0 # image index in a range of [0,9]

img = np.empty(3, dtype=object)

img[0] = X[idx]

img[1] = Y[idx]

img[2] = np.around(pred[idx]\*255)

np.around(pred[idx]\*255).shape

title = ['входое изображение', 'ручная сегментация', 'программная сегментация']

for i in range(3):

  plt.subplot(1, 3, i+1)

  plt.imshow(img[i].astype('uint8'))

  plt.axis('off')

  plt.title(title[i])

plt.show()

## Приложение Г

for i=1:25

Im=imread('Label\_'+string(i),'png');

N=label2rgb(Im);

imwrite(N,'gt000'+string(i)+'.png')

end

## Приложение Д

A1=im2bw(imread('31.png')); %считываем изображения

A2=im2bw(imread('PixelLabelData\Label\_3.png')\*255);

A2=imresize(A2,size(A1));

Accur(A1,A2) %вычисляем Accuracy

figure; imshow(A1)

figure; imshow(A2)

function [result] = Accur(A1,A2)

All=size(A1,1)\*size(A1,2);%Полный размер изображения

TP=sum(sum(A1.\*A2));%Считаем true positive

TN=length(find((A1+A2)==0));%Считаем true negative

result=(TP+TN)/All;%Значение accuracy

end