МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**АНАЛИЗ АЛГОРИТМОВ МАШИННОГО ОБУЧЕНИЯ И НЕЙРОННЫХ СЕТЕЙ ДЛЯ РЕШЕНИЯ ЗАДАЧИ КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ МАГАЗИННЫХ ТЕЛЕЖЕК**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К. А. Корнилов

(подпись)

Направление подготовки 02.03.02 — «Фундаментальная информатика и\_\_\_\_\_

(код, наименование)

информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ курс\_\_\_\_\_3\_\_\_\_\_

Направленность (профиль)\_\_\_\_Математическое и программное обеспечение компьютерных технологий\_\_\_\_

Научный руководитель

канд. физ.-мат. наук \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О. Н. Лапина

(подпись, дата)

Нормоконтролер

преподаватель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е. А. Нигодин

(подпись, дата)

Краснодар

2024

**РЕФЕРАТ**

Курсовая работа 72 с., 3 ч., 52 рис.,0 табл., 8 источн., k прил.

МАШИННОЕ ОБУЧЕНИЕ, АЛГОРИТМЫ МАШИННОГО ОБУЧЕНИЯ, НЕЙРОННЫЕ СЕТИ, СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ, КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ, PYTHON

Объектом исследования в данной работе являются алгоритмы машинного обучения и архитектуры нейронных сетей, использующиеся для классификации изображений.

Цель работы: разработка программы, в которой решается задача классификации изображения магазинных тележек с помощью алгоритма машинного обучения или нейронной сети, показавших наивысшую точность работы.

Методологическая основа исследования включает в себя эмпирический метод, сбор и анализ статистической информации, анализ полученных зависимостей, синтез выводов.

В результате работы была разработана программа, способная классифицировать изображения магазинных тележек с помощью алгоритма, показавшего наилучший результат.

Научная новизна работы заключается в создании собственной программы, способной решать задачу классификацию изображений с помощью конкретного алгоритма.

В результате решения поставленных задач создан набор данных, необходимый для обучения алгоритмов и нейронных сетей, и написана на основе наиболее точного алгоритма программа для классификации изображений магазинных тележек, принимающая в качестве входа результат решения задачи распознавания объектов с помощью алгоритма YOLO из предыдущей курсовой работы. Проведённые экспериментальные исследования показали эффективность данной программы в решении поставленной цели.

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc166773550)

[1 Теоретическая информация 7](#_Toc166773551)

[1.1. Машинное обучение 7](#_Toc166773552)

[1.1.1. Понятие машинного обучения 7](#_Toc166773553)

[1.1.2. Алгоритмы машинного обучения для решения задачи классификации.... 8](#_Toc166773554)

[1.1.2.1. Алгоритм k – ближайших соседей 8](#_Toc166773555)

[1.1.2.2. Алгоритм случайного леса 9](#_Toc166773556)

[1.1.2.3. Машина опорных векторов 11](#_Toc166773557)

[1.1.2.4. Алгоритм градиентного бустинга 13](#_Toc166773558)

[1.2. Нейронные сети 14](#_Toc166773559)

[1.2.1. Основные понятия теории искусственных нейронных сетей 14](#_Toc166773560)

[1.2.1.1. Понятие искусственной нейронной сети и ее составляющих 14](#_Toc166773561)

[1.2.1.2 Общие понятия в обучении нейронной сети 17](#_Toc166773562)

[1.2.1.3. Понятие сверточной нейросети 19](#_Toc166773563)

[1.2.2. Свёрточные нейронные сети для решения задачи классификации изображений 20](#_Toc166773564)

[1.2.2.1. Нейронная сеть Alexnet 20](#_Toc166773565)

[1.2.2.2. Нейронная сеть Inception 22](#_Toc166773566)

[1.2.2.3. Нейронная сеть DenseNet 25](#_Toc166773567)

[1.2.2.4. Нейронная сеть YOLO 27](#_Toc166773568)

[2. Программная реализация 29](#_Toc166773569)

[2.1. Инструменты для разработки 29](#_Toc166773570)

[2.2. Обучение алгоритмов 30](#_Toc166773571)

[2.3. Разработка программы 31](#_Toc166773572)

[3. Исследовательская часть 36](#_Toc166773573)

[3.1. Оценка эффективности работы обученных алгоритмов и нейронных сетей 36](#_Toc166773574)

[3.1.1. Общее описание метода оценки эффективности работы алгоритмов и нейронных сетей…… 36](#_Toc166773575)

[3.1.2. Оценка эффективности алгоритмов машинного обучения 37](#_Toc166773576)

[3.1.2.1. Оценка алгоритма k – ближайших соседей 37](#_Toc166773577)

[3.1.2.2. Оценка алгоритма случайного леса 40](#_Toc166773578)

[3.1.2.3. Оценка машины опорных векторов 42](#_Toc166773579)

[3.1.2.4. Оценка алгоритма градиентного бустинга 45](#_Toc166773580)

[3.1.3. Оценка эффективности работы нейронных сетей 47](#_Toc166773581)

[3.1.3.1. Оценка эффективности нейронной сети AlexNet 47](#_Toc166773582)

[3.1.3.2. Оценка эффективности нейронной сети Inception v3 51](#_Toc166773583)

[3.1.3.3. Оценка эффективности нейронной сети DenseNet 53](#_Toc166773584)

[3.1.3.4. Оценка эффективности нейронной сети YOLO V8 56](#_Toc166773585)

[3.2. Оценка эффективности написанной программы 59](#_Toc166773586)

[Заключение 60](#_Toc166773587)

[Список использованных источников 61](#_Toc166773588)

[Приложение А Архитектура YOLOV8 62](#_Toc166773589)

[Приложение Б Код предобработки изображений для алгоритмов машинного обучения…. 63](#_Toc166773590)

[Приложение В Код предобработки данных для AlexNet, DenseNet, Inception v3………… 64](#_Toc166773591)

[Приложение Г Код реализации полного цикла обучения для моделей PyTorch….. 65](#_Toc166773592)

[Приложение Д Код главного окна 67](#_Toc166773593)

[Приложение Е Код окна настроек 69](#_Toc166773594)

[Приложение Ж Датасет для обучения 72](#_Toc166773595)

[Приложение З Полный код приложения 73](#_Toc166773596)

# ВВЕДЕНИЕ

Актуальность данной работы состоит в рассматривание задачи классификации изображений объектов, нахождение решения которой является востребованным не только в качестве отдельной задачи, но и в качестве части более сложной составной задачи. В настоящее время существуют множество задач, процессов, работ, в которых на определённых этапах необходимо классифицировать изображения объектов и на основании полученного результата принимать определенные решения. Чаще всего подобные задачи решаются с помощью непосредственного участия человека. Однако такое решение далеко не всегда является эффективным из-за особенностей самих людей. Поэтому эта задача нуждается в автоматизации, способной наиболее эффективно ее выполнять. Из всего вышеуказанного следует, что разработка программного обеспечения для классификации изображений объектов довольно востребована.

Основная цель работы – разработка программы, решающей задачу классификации изображения магазинных тележек, на основе наиболее точного алгоритма машинного обучения или нейронной сети и определение эффективности работы данной программы.

Для реализации поставленной цели предполагается решить следующие задачи:

* изучить алгоритмы машинного обучения, наиболее часто применяющиеся для задачи классификации изображений;
* изучить базовые понятия теории нейронных сетей, решающих задачу классификации изображений;
* провести ряд экспериментов для определения эффективности работы выбранных алгоритмов и нейронных сетей;
* разработать программу, предназначенную для решения задачи классификации изображений объектов с помощью наиболее точного из предложенных алгоритмов;
* провести ряд экспериментов для определения эффективности работы данной программы;

Объектом исследования в данной работе является алгоритмы машинного обучения и архитектуры нейронных сетей, использующихся для классификации изображений объектов.

Предметом исследования является качество работы алгоритмов машинного обучения и нейронных сетей при классификации изображений объектов (согласно цели - магазинных тележек).

Информационной базой исследования являются результаты работы программы. Методологическая основа исследования включает в себя эмпирический метод (многократный запуск программы на разных изображениях и получение результатов её работы), сбор и анализ статистической информации (подсчёт среднего значения точности в классификации изображений), анализ полученных зависимостей, синтез выводов.

Научная новизна работы заключается в создании собственного приложения, способного решать задачу классификации изображений объектов.

Теоретическая и практическая значимость работы состоит в широком применении алгоритмов машинного обучения и нейронных сетей для решения задачи классификации изображений магазинных тележек. Решение данной задачи может применяться в дальнейшем для создания различных автоматических систем в магазинах. Также результаты этой работы могут быть использованы людьми с различными уровнями знаний в области классификации изображений для изучения данной темы.

# Теоретическая информация

# Машинное обучение

# Понятие машинного обучения

Машинное обучение ­ представляет собой направление искусственного интеллекта, в основании которого лежит разработка алгоритмов и статистических моделей, позволяющих выполнять задачи без явных инструкций. Подобные алгоритмы и модели разрабатываются с целью обработки данных и выполнения различных прогнозов более простыми способами чем построение модели в ручном режиме.

В машинном обучении обычно создается некоторая математическая модель для исследования данных с последующим постепенным улучшение качества принимаемых решений на новых данных. В основании механизма улучшения модели лежит ее обучение. Оно заключается в том, что пользователь предоставляет алгоритму множество уже решенных задач в виде пар объект-ответ. Алгоритм на основании данных пар находит способ получения ответа по объекту, либо в случае отсутствия ответов для объектов модель в процессе обучения самостоятельно формирует ответы для объекта на основании найденных закономерностей.

Подобный метод работы позволяет находить сложные зависимости между входными и выходными данными.

Существует два основных вида машинного обучения:

* ­обучение с учителем, которое состоит в том, чтобы на маркированных тренировочных данных извлечь модель, которая позволяет делать прогнозы о ранее не встречавшихся или будущих данных;
* обучение без учителя, которое включает моделирование признаков набора данных без каких-либо меток;

# Алгоритмы машинного обучения для решения задачи классификации

Одной из наиболее часто встречающихся задач машинного обучения с учителем является задача классификации. Суть данной задачи заключается в идентификации категориальных меток классов для новый экземпляров данных на основании предыдущих наблюдений.

Существует множество алгоритмов машинного обучения, решающих данную задачу разными подходами. Наиболее часто встречаются попытки решить данную задачу с помощью алгоритмов k-ближайших соседей, случайный лес, машина опорных векторов. Также может использоваться градиентный бустинг. Рассмотрим каждый из алгоритмов подробнее.

# Алгоритм k – ближайших соседей

Алгоритм k – ближайших соседей представляет собой довольно простой, но при этом достаточно эффективный метод решения задач классификации и регрессии.

Среди преимуществ алгоритма выделяют простоту реализации и интерпретации, отсутствие параметров, возможность достаточно точно работать с многомерными данными.

Среди недостатков выделяют низкую скорость работу и большое потребление памяти из-за хранения расстояния между всеми обучающимися и тестовыми образцами, чувствительность к выбросам и шуму, при большом количестве признаков может возникнуть проблема совпадения метрической и смысловой близости объектов.

Работа данного алгоритма разделяется на этапы обучения и классификации.

Во время этапа обучения происходит запоминание всех входных данных и соответствующий каждому экземпляру классов. Также запоминается параметр k – количество ближайших соседей.

Во время этапа классификации на вход поступает новый объект, его класс неизвестен. Алгоритм находит k – ближайших соседей к данному объектов. Классом нового объекта становится тот класс, который встречается чаще всего среди k – ближайших соседей.

В качестве меры близости между объектами используются различные метрики.

Среди них чаще всего применяются Евклидово расстояние, которое описывается формулой (1.1):

(1.1)

где

p, q – векторы значений, между которыми вычисляется расстояние;

,– k–ые значения векторов p и q.

Другой часто применяемой мерой является Манхэттенское расстояние, которое описывается формулой (1.2):

(1.2)

где

p, q – векторы значений, между которыми вычисляется расстояние;

,– k–ые значения векторов p и q.

# Алгоритм случайного леса

Алгоритм случайного леса представляет собой ансамблевый алгоритм, которые основывается на построении множества деревьев решений.

Деревья решений представляют структуру из “листьев” и “веток”. На “ветках” дерева решения записаны признаки, от которых зависит целевое значение, в “листьях” записаны значения целевого значения, а в остальных узлах – признаки, на основании которых происходит разделение выборки. Чтобы классифицировать новый случай, надо спуститься по дереву до листа и выдать соответствующее значение. Пример дерева решений представлен на рисунке 1.

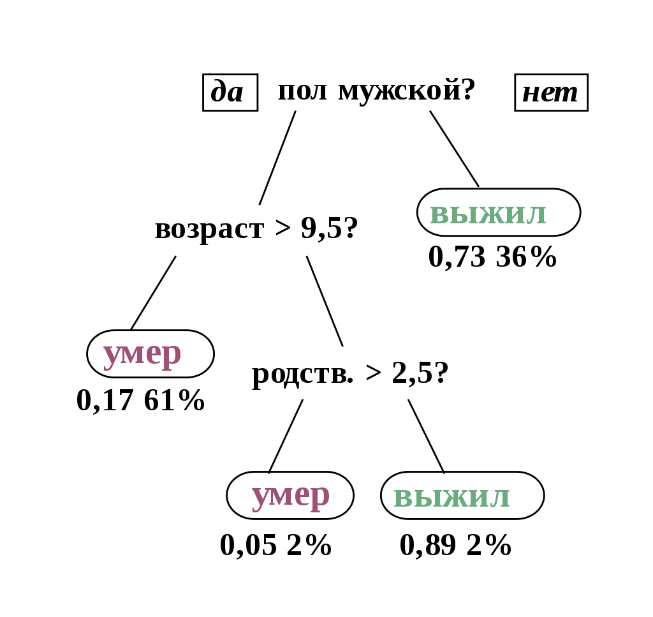


Рисунок 1 – Пример дерева решений

Среди преимуществ данного алгоритма выделяют достаточно высокую точность, возможность работать с большим количеством признаков, возможность работать с пропущенными данными.

Среди недостатков же выделяют высокую стоимость вычислений при работе с большими данными, возможность наличия проблем интерпретации полученного разделения, медленную скорость работы при большом количестве деревьев, отсутствие фактического контроля над построением деревьев.

Работа алгоритма случайного леса делится 4 на этапа.

1. Генерация случайного подмножества размера N из общего тренировочного набора данных.
2. Построение деревьев решений на основании случайных m признаков. При этом на каждом этапе выбирается лучший из признаков для построения разбиения. Данное действия может осуществляться на основании множества критериев, например, критерия Джини.
3. Дерево строится до полного исчерпания подвыборки. Затем происходит отсечение ветвей дерева в тех местах, где это не приводит к возрастанию ошибки.
4. Полученные деревья используются на новых данных. Результатом классификатора является наиболее часто встречающиеся значение класса по всем построенным деревьям.

# Машина опорных векторов

Другим довольно популярным методом решения для задачи классификации является метод опорных векторов (машина опорных векторов).

Идея данного алгоритма заключается в построение для пространства данных плоскости, разделяющей данные на классы. Алгоритм работает в предположении, что чем больше расстояние (зазор) между разделяющей гиперплоскостью и объектами разделяемых классов, тем меньше будет средняя ошибка классификатора.

Обычно данный алгоритм применяется для линейной бинарной классификации. Бинарность предполагает наличие только двух выходных классов. А линейность предполагает, что значение метки класса зависит от линейной комбинации признаков.

Но есть также его вариации для классификации на 2 более классов. Например, могут строиться гиперплоскости по принципу 1 класс и все остальные объекты. Другим вариантом может быть построение плоскостей между всеми возможными парами.

Также есть модификация алгоритма для решения задачи нелинейной классификации. В данной модификации к входным данным применяется сначала ядро (отображение, которое переводит пространство начальных данных в такое пространство, где для них уже можно решить задачу линейной классификации).

Пример результата работы алгоритма представлена на рисунке 2.

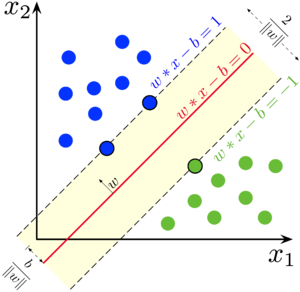


Рисунок 2 – Пример работы машины опорных векторов

Среди преимуществ данного алгоритма обычно выделяют возможность нелинейной классификации с помощью ядер, возможность работать с многомерными данными, достаточно небольшая подверженность переобучению.

Среди недостатков же выделяют проблемы с выбором нужного ядра для нелинейной классификации, достаточно высокая сложность вычислений, сложность интерпретации результатов в некоторых случаях, отсутствие в качестве вывода вероятности принадлежности к определенному классу, сильная зависимость от выбранных при обучении параметров.

Для построения разделяющей плоскости решается уравнение, заданное формулой (1.3). При этом алгоритм пытается найти такое решение, чтобы расстояние от ближайшего объекта до разделяющей плоскости было максимальным.

(1.3.)

где

x – входные данные;

w – вектор нормали к разделяющей плоскости;

b – коэффициент смещения смещения;

Работа алгоритма машины опорных векторов делится 5 на этапа.

1. Считывание данных из тренировочного набора данных.
2. Задача начальных параметров b и w для модели.
3. Вычисление расстояния между всеми точками – данными и текущей плоскостью разделения.
4. Решение задачи оптимизации, при которой минимальное расстояние между плоскостью и данными должно быть максимальным.
5. Полученные значения w и b могут использоваться для определения классов для новых данных.

# Алгоритм градиентного бустинга

Алгоритм градиентного бустинга представляет собой алгоритм машинного обучения, основанный на построение модели предсказания в форме ансамбля, где каждый предсказатель строится последовательно с учетом ошибок предыдущих моделей.

Градиентный бустинг основан на идее, что сумма отклонений от истинного значения должна быть равна нулю. Следовательно, ошибки предыдущих моделей могут быть компенсированы следующими моделями.

Среди преимуществ данного алгоритма выделяют простоту реализации, уменьшение смещения (помогает уменьшить смещение, которое является следствие неточностей в тренировочных данных), применение ансамблевого метода для построения модели (объединяются несколько моделей для повышения точности).

Среди недостатков выделяют сложность интерпретации результатов работы, высокая чувствительность к параметрам (например, шаг изменения параметров), очень высокие требования к ресурсам во время обучения и работы.

Алгоритм работы градиентного бустинга состоит из 5 этапов:

1. Инициализация базовой модели, которая делает первоначальные предсказания.
2. Вычисления остатков(разницы между предсказанным и реальным значением) как значение антиградиента функции ошибки.
3. Построения второй модели, которая будет предсказывать полученные остатки
4. Изменение параметров оригинальной модели на основании параметров второй модели
5. Повторение шагов 2-4 до условия окончания работы
6. Применение полученной обновленной модели к входным данным как обычной модели.

# Нейронные сети

# Основные понятия теории искусственных нейронных сетей

# Понятие искусственной нейронной сети и ее составляющих

Под искусственной нейронной сетью (далее нейронной сетью) обычно понимается математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей – сетей нервных клеток живого организма.

Нейронные сети являются подмножеством алгоритмов машинного обучения. Таким образом, нейронные сети могут находить сложные зависимости между входными и выходными данными, что в свою очередь приводит к возможности получения верного ответа на входных данных, для которых задача раннее не была решена

Хотя нейронные сети и отличаются в большинстве случаев от своих биологических предков, они имеют некоторые общие черты в строении.

Каждая нейронная сеть состоит из множества взаимосвязанных друг с другом процессоров, называемых нейронами. Нейрон представляет собой элемент нейронной сети, вычисляющий определенным образом выходные сигналы на основе входных значений. Он имеет достаточно простую структуру. Пример строения нейрона представлен на рисунке 3.

У него имеются синапсы, в качестве которых выступают связи, по которым поступают входные значения. Причем входные значения могут поступать как из внешней среды (то есть входные данные задачи), так и от других нейронов.

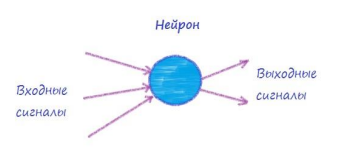


Рисунок 3 – Строение нейрона

Каждая входная связь характеризуется весом, определяющим степень влияния данных, приходящих по этой связи, на состояние нейрона. Если значение веса отрицательно, то связь считается тормозящей. В противном случае, связь считается возбуждающей.

Имеются также у нейрона аксоны, которыми являются его выходы, обеспечивающие передачу выходного сигнала нейрона последующим нейронам.

Непосредственно сам нейрон представляет собой функцию активации с аргументом, являющимся суммой произведений входных значений на их веса.

Аргумент можно описать формулой (1.4):

(1.4)

где

– величина входного значения;

– величина веса связи.

Функция активации характеризует зависимость сигнала на выходе нейрона от входных данных. Чаще всего данная функция является нелинейной. Обычно функция является монотонно возрастающей и находится в области значений [-1,1] или [0,1]. Для некоторых алгоритмов обучения необходимо, чтобы активационная функция была непрерывно дифференцируемой на всей числовой оси.

Часто в качестве функции активации выступает функция leaky ReLu, которая описывается формулой (1.5):

(1.5)

где

P – сумма произведений входных значений на веса их связей.

Совокупность нейронов, расположенных на одном уровне, называется слоем нейронной сети. В общем случае у каждой нейронной сети есть 3 вида слоев.

1. Слой входных данных (Input Layer). Это первый слой нейронной сети, который принимает входные данные. Количество нейронов в этом слое соответствует количеству признаков или размерности входных данных.
2. Скрытые слои (Hidden Layers). Это слои, которые находятся между входным и выходным слоями. Они выполняют вычисления, основанные на входных данных и весах связей, делается это для извлечения признаков и создания более сложных представлений данных. В нейронных сетях может быть любое количество скрытых слоев.
3. Слой выходных данных (Output Layer). Это последний слой нейронной сети, который предсказывает результаты или выходные данные на основе обработанных входных данных. Количество нейронов в этом слое зависит от типа задачи: например, для задачи классификации количество нейронов соответствует количеству классов, а для задачи регрессии - один нейрон.

Совокупность слоев, их строения, количества нейронов в каждом слое и связей между ними формируют архитектуру нейронной сети. Она играет ключевую роль в процессе обучения и работе нейронной сети, так как влияет на ее способность извлекать признаки, обобщать данные и делать предсказания. Пример архитектуры простейшей нейронной сети представлен на рисунке 4.

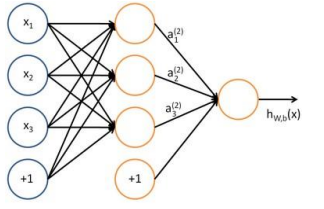


Рисунок 4 – Пример строения простейшей нейронной сети

## **Общие понятия в обучении нейронной сети**

В общем смысле обучение нейронной сети представляет собой процесс вычисления весов нейронов сети на основе предоставленных обучающих данных. Цель обучения нейронной сети заключается в нахождении таких значений весов, которые будут верно характеризовать зависимости между входными и выходными данными и, с использованием которых, нейронная сеть получит правильный результат на входных данных, не участвовавших в обучении.

В общем случае выделяют два вида обучения, включающие в себя множество специальных алгоритмов.

Первым является обучение с учителем, при котором во время обучения имеются и входные данные, подающиеся на вход нейронной сети, и выходные данным, с помощью которых происходит сравнение результата работы нейронной сети на очередном этапе обучения с реальным ответом для текущих входных данных.

Вторым является обучения без учителя. При данном подходе имеются только входные данные, выходные данные, являющиеся верными ответами, отсутствуют. Для изменения весов в этом случае используются специальные алгоритмы.

Из представленных двух видов обучения чаще всего используется обучение с учителем ввиду легкости своей реализации и простоты использования.

Для обучения нейронной сети с учителем обычно используется алгоритм обратного распространения ошибки. В общем случае в нем выделяется 5 этапов.

1. Инициализация весов. Веса нейронов инициализируются случайными значениями перед началом обучения.
2. Прямое распространение. Входные данные подаются на входной слой нейронной сети, и происходит проход данных через слои сети. Каждый нейрон выполняет вычисления на основе своих входных данных и функции активации и передает результаты следующему слою. Процесс прямого распространения выполняется до выходного слоя, где генерируются предсказания сети.
3. Вычисление ошибки. Cгенерированные предсказания сравниваются с целевыми значениями в обучающих данных, и вычисляется ошибка.
4. Обратное распространение ошибки. Ошибка обратно распространяется через сеть, начиная с выходного слоя и двигаясь назад к входному слою. В каждом слое обновляются веса нейронов на основе ошибки и градиента функции потерь. Градиент указывает направление наиболее быстрого убывания ошибки, и на его основе обновляются веса нейронов.
5. Повторение процесса. Шаги 2-4 повторяются для каждого обучающего примера в наборе данных. Этот процесс называется эпохой. Обычно обучение выполняется на нескольких эпохах для достижения лучших результатов.

## **Понятие сверточной нейросети**

Свёрточная нейронная сеть – специальная архитектура искусственных нейронных сетей, предложенная Яном Лекуном в 1988 году. Основная идея данной архитектуры сетей состоит в том, чтобы использовать одни и те же части нейронной сети для работы с разными маленькими, локальными участками входов.

Основным отличием сверточных нейронных сетей от других архитектур является наличие новых специальных слоев, используемых для извлечения признаков различной степени важности из входных данных.

Основным видом новых слоев являются сверточные слои. Сверточные слои состоят из набора фильтров (ядер), которые применяются к входным данным. Фильтры выполняют операцию свертки. Последовательное применение свертки с помощью фильтров различного размера и количества позволяет извлекать признаки разного уровня сложности и абстрактности.

Операция свертки состоит из трех шагов.

* 1. Определение фильтра. Фильтр представляет собой небольшую матрицу весов, которая определяет, какие признаки будут извлечены из входных данных. Фильтр может быть различного размера и содержать различные значения весов.
  2. Применение фильтра. Фильтр применяется к входным данным путем перемещения его по всей области входных данных c заданным размером шага. На каждом шаге вычисляется свертка между фильтром и соответствующей областью входных данных. Свертка выполняется путем поэлементного перемножения значений фильтра и соответствующих значений входных данных, а затем суммирования результатов.
  3. Создание карты признаков. После применения фильтра ко всем областям входных данных, результаты сверток объединяются в карту признаков.

После последнего шага алгоритма свертки выполняется еще одна новая операция – пуллинг. Он выполняется еще одним новым видом слоев, которыми являются пулинговые слои или слои объединения. Пулинговые слои выполняют уменьшение размерности данных путем объединения информации из соседних областей. Наиболее распространенной операцией пулинга является операция максимального пулинга (max pooling), которая выбирает максимальное значение из каждой области. Пулинговые слои помогают уменьшить количество параметров в сети и обеспечить инвариантность к малым трансляциям объектов в изображении.

# Свёрточные нейронные сети для решения задачи классификации изображений

# Нейронная сеть Alexnet

Одной из самых простых в плане своей архитектуры и достаточно точных в плане работе сверточных нейронных сетей для решения задачи классификации изображений является AlexNet. Данная нейросеть была опубликована в 2012 году Алексеем Крижевским. В том же году она заняла первое место на соревнование ImageNet, решив задачу классификации изображений на одноименном датасете с ошибкой всего в 15 %,что для того времени стало одним из лучших результатов.

Также данная сеть популяризировала некоторые приемы, которые сейчас являются стандартными для глубоких нейронных сетей. В частности, использование функции ReLu, добавление в нейронные сети слоев DropOut (которые при обучения случайным образом обнуляют связи), применение аугментации к тренировочным данным, применение перекрывающий max pooling.

AlexNet имеет достаточно простую архитектуру, представленную на рисунке 5. Она содержит всего восемь слоев с весовыми коэффициентами. Первые пять из них являются сверточными, а остальные 3 полносвязными. Relu применяется после каждого сверточного и полносвязного слоя. Дропаут применяется перед первым и вторым полносвязными слоями.

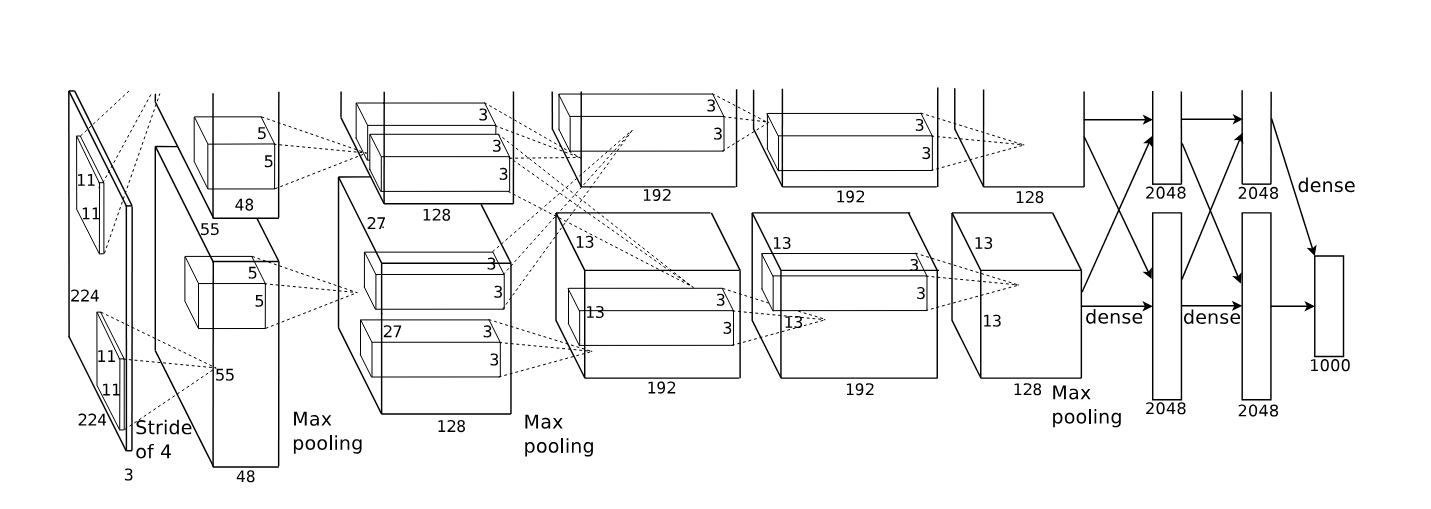


Рисунок 5 – Архитектура сети AlexNet

Стоит отметить, что ядра сверточных слоев связаны по–особенному. Ядра второго, четвертого и пятого сверточных слоев связаны только с теми картами ядра в предыдущем слое, которые находятся на одном и том же графическом процессоре. Ядра третьего сверточного слоя связаны со всеми картами ядер второго слоя.

Выходные данные пропускаются через функцию потерь softmax, представленной формулой (1.6.) для формирования распределения вероятностей 1000 меток классов.

(1.6.)

где

– вектор вещественных чисел в интервале [0,1];

– значение вектора z размерности k.

# Нейронная сеть Inception

Другой популярной нейросетью является Inception. Впервые идеи данной сеть появилась в 2014 году в виде сети GoogLeNet, развитием которой и является Inception. На тот момент была достигнута точность в 80 на наборе данных ImageNet. Основной идеей данной нейросети стал рост в ширину с сохранением относительно небольшого числа весов (у третьей версии их 25 млн против 60 млн у AlexNet).

Фактически построение данной сети основано на следующих 4 принципах.

1. Очень много сигналов близки друг к другу в пространстве (т.е. в соседних "пикселах"), и этим можно пользоваться, чтобы делать свертку меньшего размера.
2. Для эффективного использования ресурсов нужно увеличивать и ширину, и глубину сети. Увеличение только глубины сети даст меньше выигрыша в точности, чем увеличение и ширины, и глубины.
3. Слои с резким уменьшением размерности входных данных в начале сети могут привести к ухудшению обучаемости сети.
4. "Широкие" слои быстрее обучаются, что особенно важно на высоких уровнях.

Увеличение ширины сети основано на разветвление вариантов обработки очередных входных данных. Изначально такое разветвление состояло из следующих вариантов: свертка с ядром размера 1x1, свертка с ядром 3x3, свертка с ядром 5x5, пулинг с окном 3x3. При поступлении очередных входных данных происходил проход по каждой из веток и формирование ответа на основании всех полученных результатов. Пример подобного “Inception” блока представлен на рисунке 6.

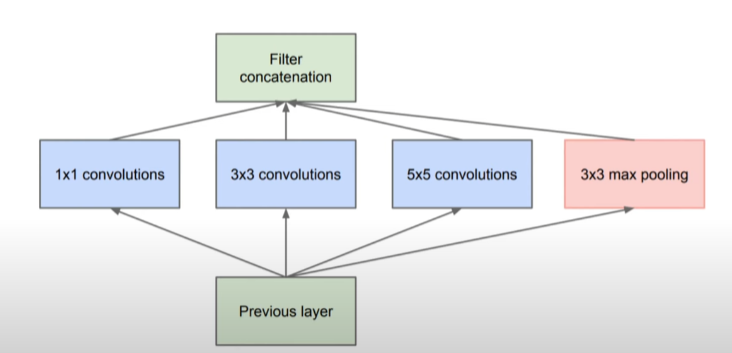


Рисунок 6 ­ Пример Inception блока из Inception v1

В более поздних версиях нейронной сети увеличилось число слоев до 42 и были оптимизированы блоки Inception. Вместо применения затратной по времени и ресурсам свертки с ядрам ширины 3x3 и более начали применять последовательно свертки размерностью 1xN. Пример оптимизированного блока представлен на рисунке 7.

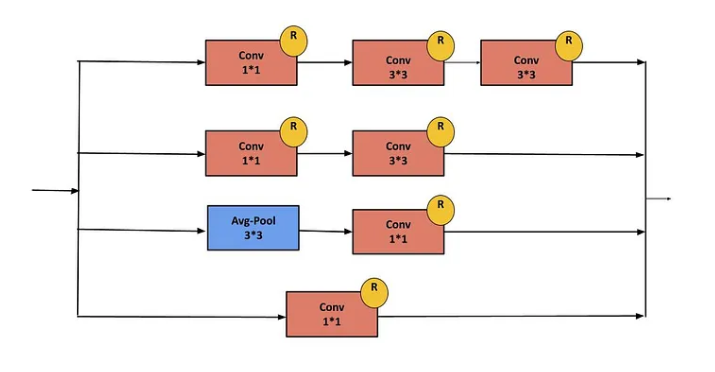


Рисунок 7 ­ Пример оптимизированного Inception блока

Подобные эксперименты с блоками привели к повышению точности и скорости работы нейронной сети. Общий вид архитектуры нейронной сети Inception v3 представлен на рисунках 8 – 9.

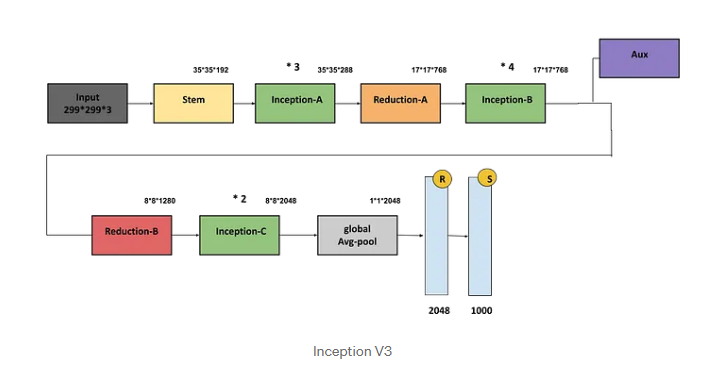


Рисунок 8 – Общая схема архитектуры Inception v3

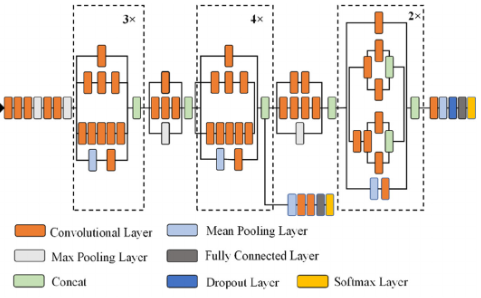


Рисунок 9 – Подробная версия архитектуры Inception v3

# Нейронная сеть DenseNet

При обучение многих нейронных сетей есть вероятность возникновения опасных ситуаций, сильно влияющих на качество обучаемой модели. Одними из самых опасных ситуаций являются градиентные затухания и всплески.

Градиентное затухание – ситуация, при которой градиент функции потерь становится слишком маленьким по значению, что практически приводит к неизменению параметров более ранних слоев модели.

Градиентный всплеск – ситуация, при которой значение градиента функции потерь становится слишком большим, что приводит к нарушению траектории обучения модели.

Для решения подобных проблем была предложена модель DenseNet. Идея этой модели заключается в том, что точность модели можно повысить, если добавить прямую связь между слоями, не соединенными непосредственно. Перед поступлением данных на новый слой происходит “конкатенирование” карт признаков предыдущих слоёв. Обратной стороной применения подобной тактики является увеличение затрачиваемых ресурсов на обучение и работу сети. Пример подобных “Dense” блоков представлен на рисунке 10.

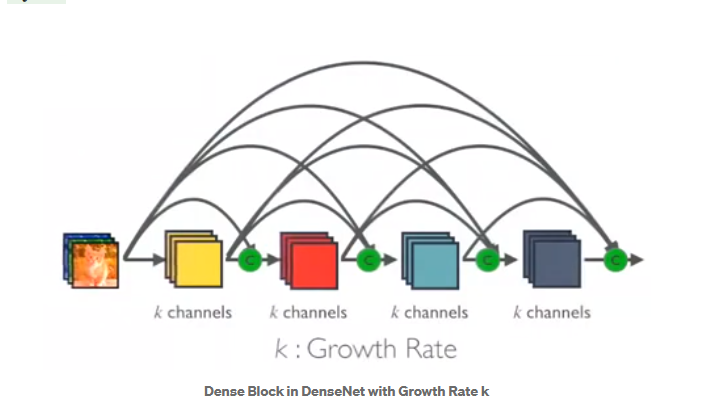


Рисунок 10 – Пример “Dense” блока

Однако у данной идеи построения сети существует 1 серьезная проблема: конкатенировать карты признаков можно только в том случае, если они имеют одинаковые размеры по вертикали и горизонтали, следовательно, необходимо сохранять размеры карт признаков на протяжение всей работы сети. Но данная идея полностью противоречит основам сверточных нейронных сетей. Поэтому для осуществления свертки данных в сети вводят переходные блоки, располагающиеся между Dense блоками, которые и осуществляют свертку данных. Пример подобной реализации представлен на рисунке 11. Примеры же архитектуры рабочей сети, построенной по данному принципу представлены на рисунке 12.

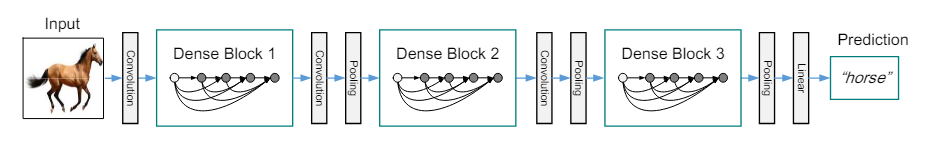


Рисунок 11 – Пример реализации архитектуры DenseNet из 3 “Dense” блоков

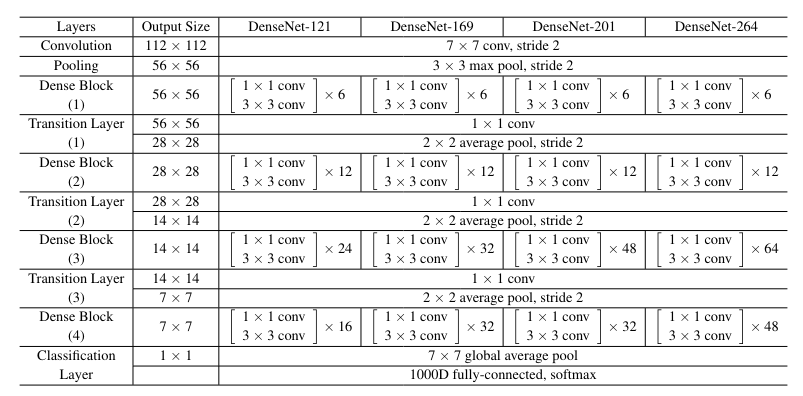


Рисунок 12 – Описание архитектуры рабочих версий сети DenseNet

# Нейронная сеть YOLO

Нейронная сеть YOLO изначально разрабатывалась для задачи распознавания объектов на изображении. В основании ее работы лежит алгоритм YOLO (You only look once). Особенностью работы данного алгоритма является то, что он за один проход изображения через нейросеть определяет и ограничивающие объекты коробки, и классы объектов внутри данных коробок. Данная особенность выделяет его на фоне других алгоритмов решения данной задачи, которым требовалось два прохода изображения через сеть: первый проход – для определения ограничивающих коробок, второй проход – для определения классов объектов.

По своему механизму работы, согласно словам разработчиков, версия нейронной сети для решения задачи распознавания объектов и решения задачи классификации изображений различается лишь тем, что во втором случае влияние определенных ограничивающих коробок практически не влияет на итоговый класс изображения (и, как следствие, они отсутствуют в итоговом ответе).

Архитектура последней версии нейронной сети по данному алгоритму использует сверточную нейронную сеть, которую можно разделить на две основные части: позвоночник (backbone) и голову (head). Изображения полной архитектуры сети представлено в приложении A.

Позвоночник представляет собой совокупность слоев свертки и пулинга.

Данная часть нейронной сети отвечает за извлечение основных признаков объектов на изображении.

Голова YOLOv8 состоит из нескольких сверточных слоев, за которыми следует ряд полносвязных слоев. Эти слои отвечают за прогнозирование ограничивающих прямоугольников (bounding boxes), оценки объектности (objectness scores) и вероятности классов для объектов, обнаруженных на изображении.

Одной из ключевых особенностей YOLOv8 является использование механизма самоконтроля в голове сети. Этот механизм позволяет модели сосредоточиться на разных частях изображения и устанавливать важность различных признаков в зависимости от их применимости к задаче.

# Программная реализация

# Инструменты для разработки

Для реализации приложения был выбран язык Python 3.10. Для реализации интерфейса, алгоритмов машинного обучения и нейронных сетей были использованы дополнительные библиотеки.

1. Roboflow: данная библиотека предоставляет инструментарий для создания и загрузки датасетов изображений.
2. Ultralytics: данная библиотека содержит реализацию и инструменты работы с нейросетью, основанной на алгоритме YOLO. С помощью данной библиотеки можно достаточно просто переобучить нейросеть на алгоритме YOLO для распознавания нужного объекта.
3. Sklearn: данная библиотека предоставляет реализацию огромного количества алгоритмов машинного обучения, метрик, предобработок данных и инструментов взаимодействия с ними.
4. PyTorch: данный фреймворк предоставляет огромное количество инструментов для работы и построения нейронных сетей, содержит в себе реализации множества моделей нейронных сетей.
5. PyQt5: данный фреймворк предоставляет инструменты для создания кроссплатформенных приложений с графическим интерфейсом. Позволяет создать удобный интерфейс для взаимодействия пользователя с моделью
6. Pathlib: данная библиотека предназначена для взаимодействия с файловой системой с помощью средств языка Python.
7. Matplotlib: библиотека для построения графиков на языке Python. Является реализацией фреймворка MatPlot.
8. CV2: это open source библиотека компьютерного зрения, которая предназначена для анализа, классификации и обработки изображений. С помощью нее можно довольно просто работать с изображениями и видео-файлами.

# Обучение алгоритмов

Для исследования эффективности работы алгоритмов был составлен набор данных на 2300 изображений с использованием сайта Roboflow. Набор данных загружается в среду разработки PyCharm c использованием библиотеки Roboflow для дальнейшего использования для обучения.

Перед обучением алгоритмов машинного обучения (k – ближайших соседей, машина опорных векторов, случайный лес, градиентный бустинг) проводится преобразование загруженных изображений с использованием библиотеки sklearn. Все изображение приводятся к размерам 32x32,64x64,128x128,224x224, и происходит нормализация значений цвета пикселя в пределах всех значений конкретного цвета. Код данных преобразований представлен в приложении Б.

Полученные наборы данных подаются на вход реализациям указанных методов машинного обучения, входящий в библиотеку sklearn.

Аналогичным образом организована предобработка данных для нейронных сетей. Наборы данных преобразуются встроенными инструментами фреймворка pytorch к нужному размеру и нормализуются. Затем они подаются на вход описанным в разделе теории реализациям нейронных сетей AlexNet, DenseNet, Inception v3 с помощью встроенного для pytorch класса DataLoader. Код предобработок и примеров класса DataLoader представлены в приложении В.

Для реализации Yolo предобработка не требуется, ее реализации в библиотеке Ultralytics подается на вход только путь до файлов набора данных.

Для нейронных сетей AlexNet, DenseNet, Inception v3 написан полный цикл обучений с использования инструментов Pytorch. В данном цикле используются реализации функции потерь перекрестной энтропии, а также алгоритмов оптимизации стохастического градиентного спуска и Адама из этого же фреймворка. Код реализации данного цикла обучения представлен в приложении Г.

Для нейронной сети Yolo использовался встроенный интерфейс для вызова уже реализованного цикла обучения нейронной сети.

Для наглядной оценки качества работы алгоритмов используется реализация матрицы ошибок (матрица, строки и столбцы которой представляют собой классовые метки; при предсказании класса для объекта увеличивается на 1 ячейка с индексами (номер верного класса, номер предсказанного класса)) в библиотеке sklearn. Ей в качестве параметров передаются предсказанные по тестовой части набора данных классы объектов, верные метки классов для объектов и названия классов. Также используется библиотека MatplotLib для построения графиков общей точности работы алгоритмов.

# Разработка программы

Для исследования эффективности работы лучших нейронных сетей и алгоритмов в полевых условиях была доработана программа из первой курсовой работы, чтобы они могла решать задачу классификации изображений магазинных тележек с использованием языка Python, фреймворка PyQt5 и PyTorck, а также с использованием библиотеки Ultralytics.

Программа принимает на вход изображение. Сначала на данном изображении решается задача распознавания объектов с помощью нейронной сети Yolo, обученной в прошлой курсовой работе. Затем на основании результатов ее работы из исходного изображения вырезаются изображения, содержащие объекты, похожие на тележки. Полученные новые изображения подаются на вход алгоритму или нейронной сети, решающей задачу классификации изображений. Результатом работы программы выступает исходное изображение, на котором выделены магазинные тележки с учетом работы первой и второй моделей.

На рисунке 13 представлено основное окно программы. В этом окне пользователь может выбрать файл с изображением или видео, на котором будет производиться работа алгоритма. Выбор файла осуществляется либо введением полного пути до него, либо нажатием на кнопку страницы с закладкой. Выбор папки осуществляется также либо введением полного пути до нее, либо нажатием на кнопку с папкой. Пользователь может выбрать обработку с сохранением результатов, либо показ видео без сохранения. Во втором случае при выборе видео - файла будет происходить работа алгоритма в реальном времени.

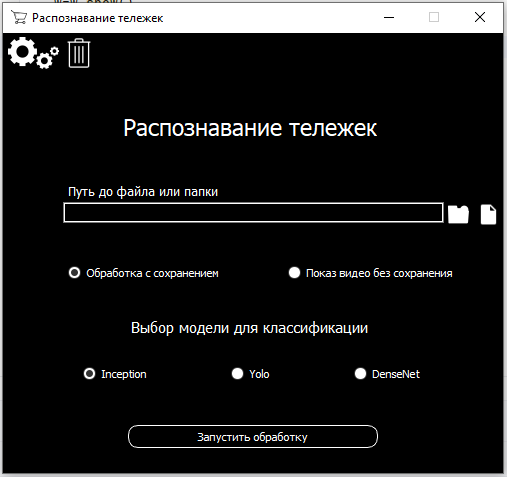


Рисунок 13 – Главное окно программы

При нажатии на шестеренку в левом верхнем углу программы будет открыто окно настроек. В данном окне пользователь может выбрать веса нейронной сети, которые будут использоваться при решение задачи распознавания объектов. Также пользователь может указать минимальное значение точности, при достижении которого объект будет выделен на итоговом результате, и значение наложения (intersection over union), которое определяет, какая часть ограничивающих прямоугольников будет выбрана при их пересечении, разрешение изображения, которое будет использоваться при его обработке. Для решения же задачи классификации пользователь может указать минимальное значение вероятности принадлежности к классу тележек, при достижении которого тележка будет выделяться на итоговом результате. Окно настроек представлено на рисунке 14.

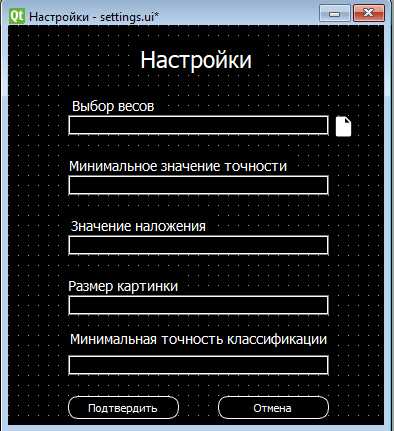


Рисунок 14 – Окно настроек

При нажатии на кнопку “запустить обработку” будет начата обработка выбранных изображений и видео - файлов. По окончанию работы программы будет открыто окно в проводнике, где для каждого изображения будет содержаться директории с каждым этапом работы программы. Пример итоговой директории представлен на рисунке 15.

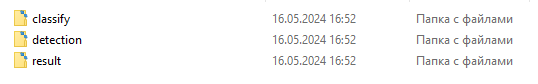


Рисунок 15 – Пример итоговой директории для 1 изображения

На рисунках 16 – 18 представлены примеры результатов работы на каждом этапе программы.



Рисунок 16 – Пример результата распознавания объектов



Рисунок 17 – Пример результатов классификации объектов



Рисунок 18 – Пример итогового результата

# Исследовательская часть

# Оценка эффективности работы обученных алгоритмов и нейронных сетей

# Общее описание метода оценки эффективности работы алгоритмов и нейронных сетей

В качестве исследовательской части перед написанием приложения была проведена оценка точности работы выбранных алгоритмов машинного обучения и нейронных сетей после их обучения на собранном наборе данных при различных параметрах, влияющих на обучение и точность работы. Данные параметры являются полностью разными для алгоритмов машинного обучения. В качестве общего “параметра” для них выступает размер входных изображений (32x32,64x64,128x128,224x224). Для нейронных сетей же общими изменяемыми параметрами выступает размер выборки (16,32,64,128), подаваемой на вход нейронной сети, а также количество эпох обучения.

Точность работы алгоритмов машинного обучения оценивается как доля правильный меток классов к общему числу меток.

Аналогичным образом оценивается точность работы нейронных сетей. Также для нейронных сетей высчитывается среднее значение функции потерь перекрестной энтропии, описанной формулой (1.7) на каждой эпохе обучения.

(1.7)

где

M – количество различных меток классов ;

N – количество объектов, по которым считается ошибка;

– метка класса j у объекта i;

– вероятность метки класса j у объекта i.

На основании полученных результатов обучения были построены графики зависимости итоговой точности работы алгоритмов и нейронных сетей от параметров обучения. Также были построены матрицы ошибок для тестовых выборок. На основании полученных результатов также было построено приложении для классификации изображений магазинных тележек.

# Оценка эффективности алгоритмов машинного обучения

# Оценка алгоритма k – ближайших соседей

Первым алгоритмом для оценки был выбран алгоритм k – ближайших соседей. Он является достаточно простым и эффективным алгоритмом для решения задачи классификации.

Данный алгоритм в качестве основного свободного параметра имеет количество соседей, по которому определяется класс нового объекта. В ходе оценки точности работы алгоритма перебираются количество соседей от 1 до 30 для каждого из разрешений. Количество соседей достаточно сильно влияет на общую точность работы модели. Пример такого влияния представлен на рисунке 19 для размера изображения 32x32.

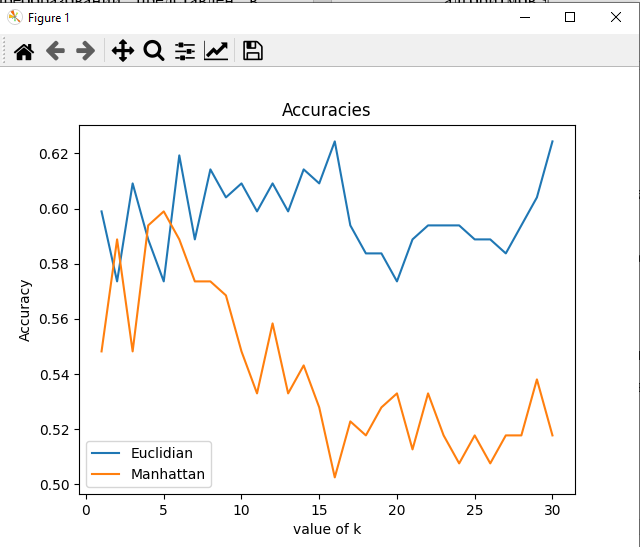


Рисунок 19 – Пример зависимости точности работы алгоритма от количества соседей для определения класса объекта

В качестве второго параметры выступает метрика расстояния между объектами. В данном случае проводится оценка для Эвклидова расстояния и для Манхэттенского расстояния.

В результате обучения было получено, что для всех представленных размеров изображений точность работы алгоритма находится в районе 60 %. А максимальное достигнутое значение точности было равно 66 %. Данные результаты являются достаточно плохими и показывают, что алгоритм k – ближайших соседей не подходит для эффективного решения задачи классификации изображения магазинных тележек. График изменения зависимости точности работы алгоритма от размерности изображения представлен на рисунке 20. Также на рисунках 21 и 22 представлены матрицы ошибок лучших итераций данного алгоритма для Эвклидова и Манхэттенского расстояний соответственно.

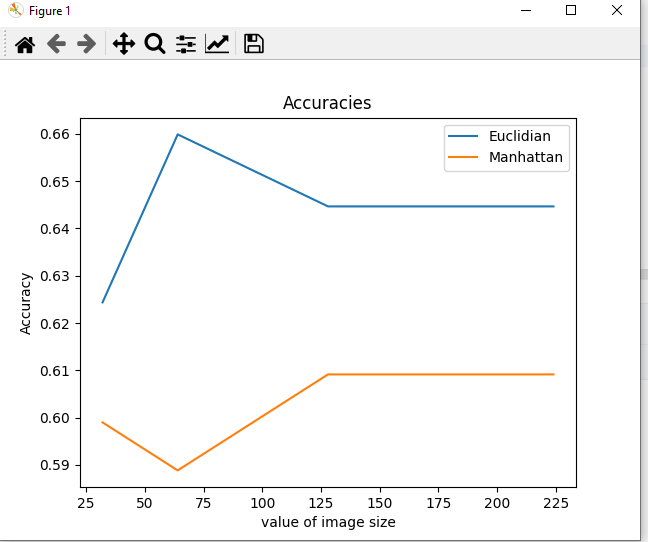


Рисунок 20 – Результат измерения точности работы алгоритма для изображений разных размеров

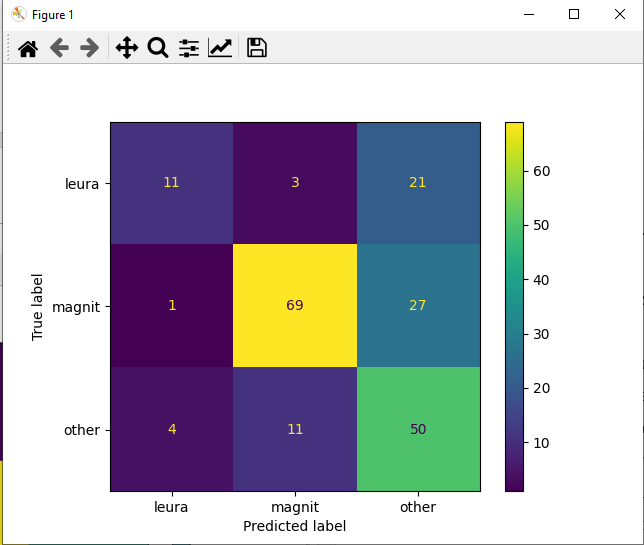


Рисунок 21 – Матрица ошибок лучшей итерации для Эвклидова расстояния

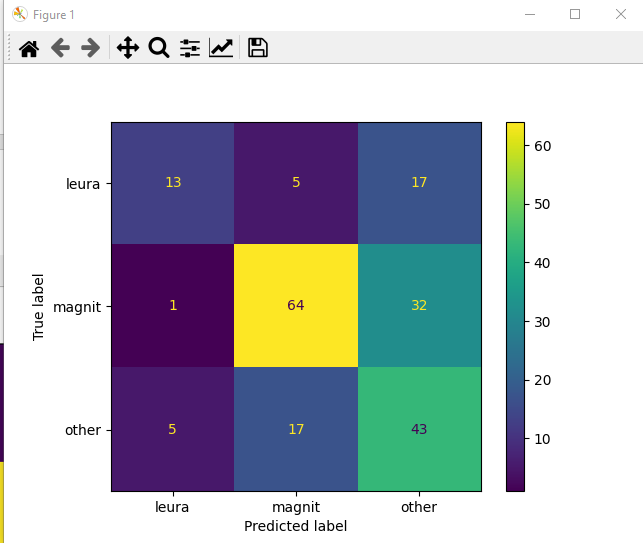


Рисунок 22 – Матрица ошибок лучшей итерации алгоритма для Манхэттенского расстояния

Также были получены графики зависимости лучшего количества соседей для определения класса объекта от разрешения входящего изображения. Как видно по графику на рисунке 23 для Манхэттенского расстояния наилучшим результатом оказалось 2 соседа. Для Эвклидова расстояния данный параметр различается для всех разрешений.

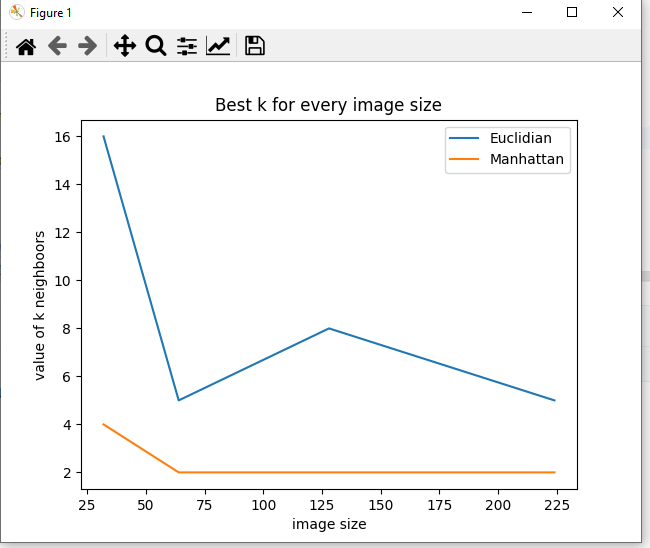


Рисунок 23 – Графики зависимостей лучшего количества ближайших соседей от размеров входных данных

# Оценка алгоритма случайного леса

Вторым алгоритмом был выбран алгоритм случайного леса. Он является ансамблевым алгоритмом (то есть он строит несколько моделей). Основными параметрами в данной модели является количество построенных деревьев и критерий разделения выборки. В процессе проверки эффективности строятся 100,200,300,400,500,600,700,800,900,1000 деревьев без ограничения ширины и максимальной глубины. В качестве критерия разделения выборки был критерий Джини, описывающийся формулой (1.8).

(1.8)

где

T – текущий узел;

– вероятность класса j в узле T.

В результате проведенных экспериментов было выяснено, что представленном наборе данных алгоритм имеет максимальную точность в 70 % при размере изображения 224x224. Данный результат лучше, чем у алгоритма k – ближайших соседей, но все еще не является достаточно хорошим, поскольку мы имеем ошибку в 30 % случаев. Результаты измерения точности для разных размерностей изображения представлены на рисунке 24.

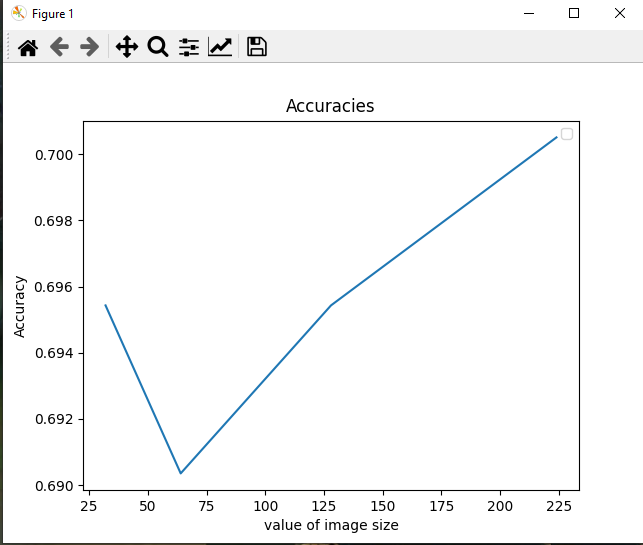


Рисунок 24 – Результаты измерения точности алгоритма для разных размерностей изображения

На рисунке 25 представлена матрица ошибок лучшей итерации алгоритма.

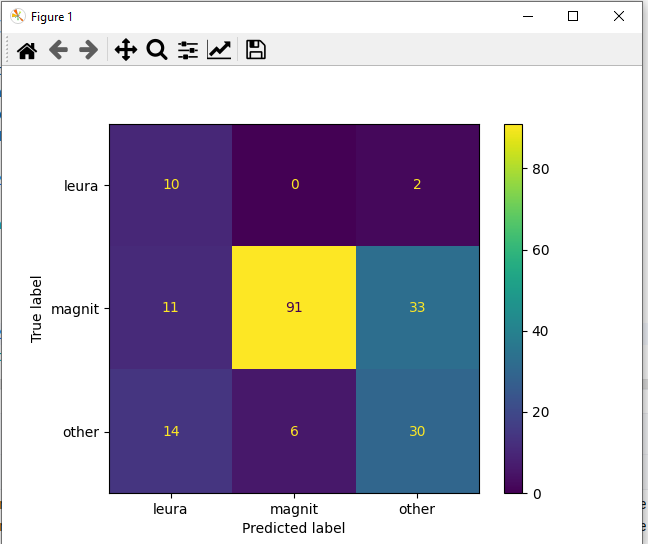


Рисунок 25 – Матрица ошибок лучшей итерации алгоритма

Также было вычислено лучшее число деревьев для каждого из размеров изображения. Результаты вычисления зависимости представлены на рисунке 26. Как видно с увеличением размера изображения для более качественной классификации требуется увеличение числа построенных деревьев.

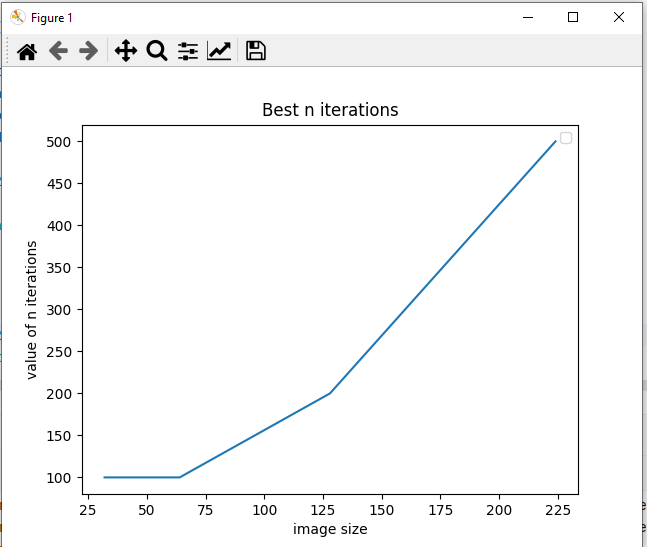


Рисунок 26 – Зависимость числа построенных деревьев от размера изображения

# Оценка машины опорных векторов

Третьим алгоритмом была выбрана машина опорных векторов. Хотя данный алгоритм и не является самым гибким в плане построения классификации объектов и эффективным с точки зрения затрачиваемых ресурсов, он все – таки является довольно точным и предоставляет возможность построения нелинейной классификации объектов с помощью построения линейной классификации и ядер.

В качестве перебираемых параметров выступают ядра (выбраны полиномиальное ядро и ядро rbf), а также коэффициент C (0.01,0.1,1,10,100) (отвечающий за “строгость решения”, то есть определяет соотношения между гибкостью классификации и ее строгостью по отношению к тренировочным данным).

В результате проведения оценки эффективности работы алгоритма было получено, что на предложенном наборе данных точность алгоритма находится в пределах 60 – 67 %, что является хуже, чем у случайного леса. Учитывая данное обстоятельство, а также то, что данный алгоритм более затратен по ресурсам, чем случайный лес, можно сделать вывод о нецелесообразности использования данного алгоритма для решения поставленной задачи. Результаты измерения точности представлены на рисунке 27.

Также была построена матрица ошибок для лучшей итерации алгоритма. Она представлена на рисунке 28.

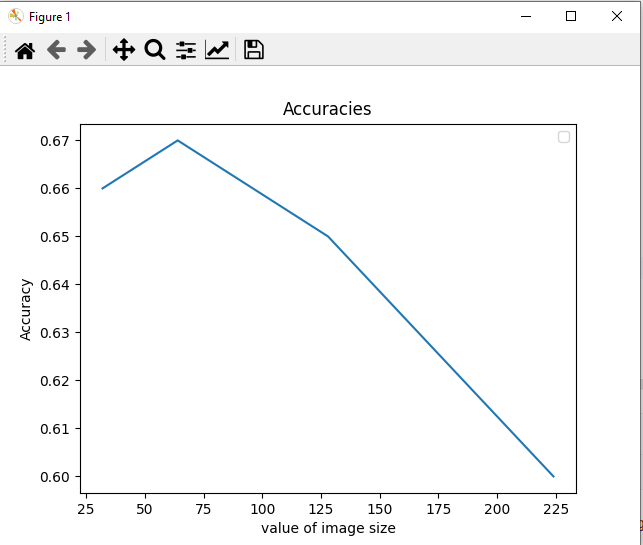


Рисунок 27 – Результаты измерения точности

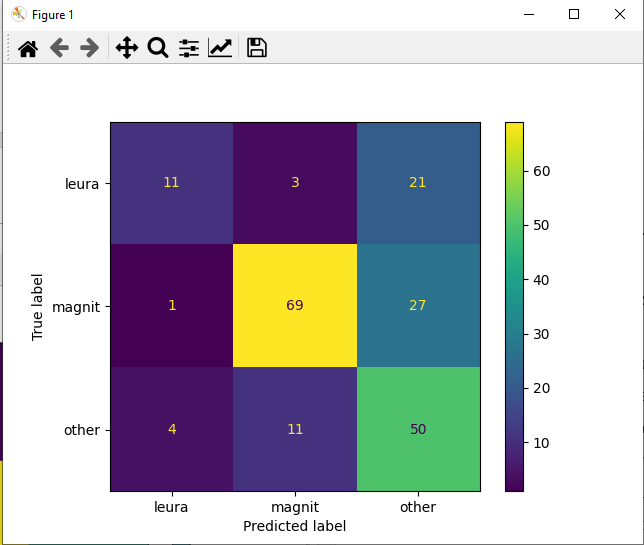


Рисунок 28 – Матрица ошибок для лучшей итерации алгоритма

Также была проведена оценка лучших параметров для реализации алгоритмов на изображениях разных размеров. Результаты измерений представлены на рисунка 29 – 30.

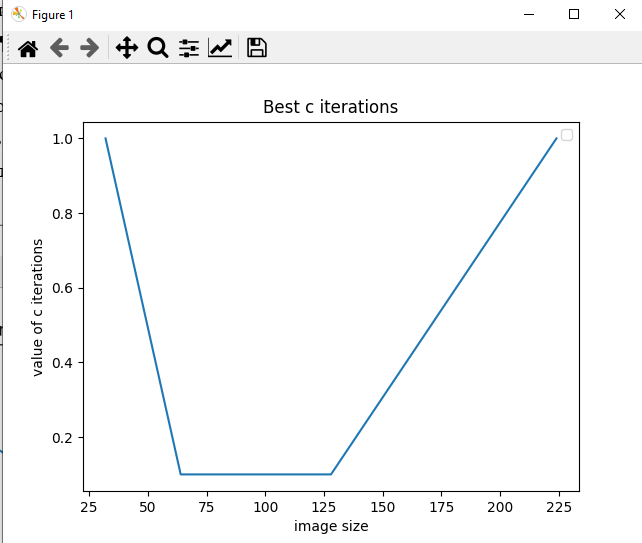


Рисунок 29 – Результаты измерения параметра C для разных размеров изображений

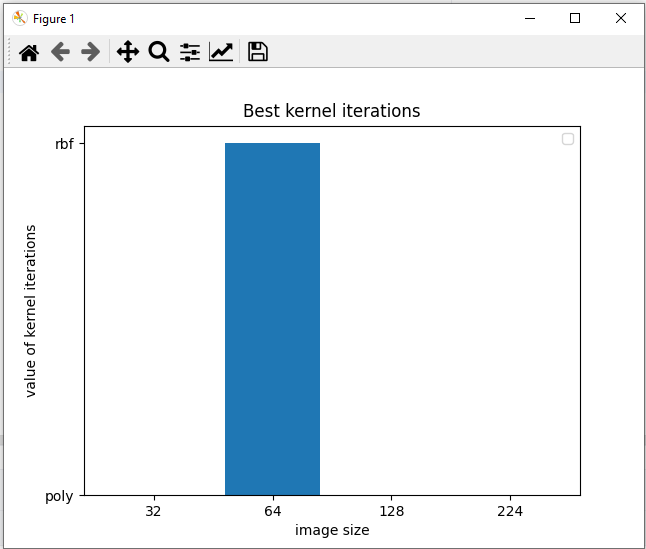


Рисунок 30 – Результат измерения лучшего ядра для изображений разного размера

# Оценка алгоритма градиентного бустинга

Третьим алгоритмом выступает алгоритм градиентного бустинга. Данный алгоритм предполагает построение 1 модели, которая на каждой итерации алгоритма корректируется с помощью построения дополнительной модели.

В качестве свободных параметров выступает количество итераций исправления модели и шаг обучения.

В качестве функции потерь для построения вспомогательных моделей при оценке эффективности была выбрана функция перекрестной энтропии, описываемая формулой (1.7).

В результате проведения оценки эффективности работы алгоритма было получено, что на предложенном наборе данных точность алгоритма находится в пределах 60 – 65 %, что хуже точности случайного леса. Таким образом, для решения поставленной задачи лучше использовать алгоритм случайного леса. Результаты измерения точности представлены на рисунке 31.

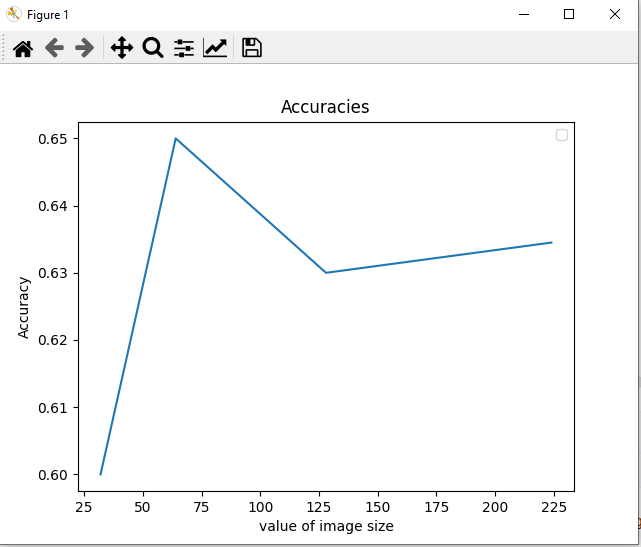


Рисунок 31 – Зависимость точности обучения алгоритма от размера входных данных

Также была проведена оценка количества итераций работы алгоритма для изображений разных размеров. Результаты оценки представлены на рисунке 32. Согласно полученным данным максимальное число итераций на предложенных разрешениях не превысило 250.

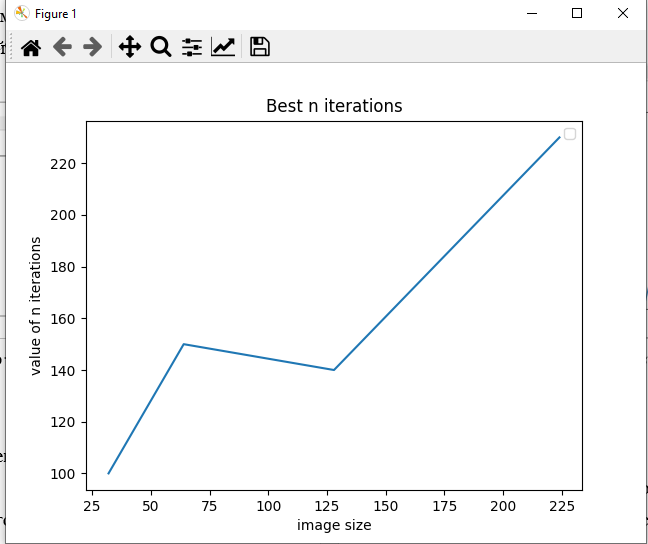


Рисунок 32 – Зависимость числа итераций работы алгоритма от размеров входных данных

Также была построена матрица ошибок для лучшей итерации алгоритма. Она представлена на рисунке 33.

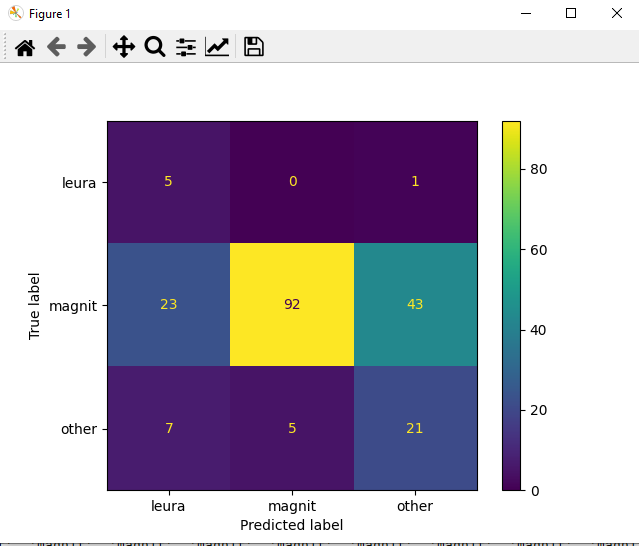


Рисунок 33 – Матрица ошибок лучшей итерации алгоритма

# Оценка эффективности работы нейронных сетей

# Оценка эффективности нейронной сети AlexNet

Первой нейронной сетью для решения поставленной задачи была выбрана AlexNet. Данная сеть имеет достаточно простую архитектуру и при этом является достаточно эффективной.

Для данной нейронной сети было проведено обучения при размере обучающей выборки 16,32,64 и 128 фотографий.

Для обновления весов при обучения использовался в первом случае алгоритм стохастического градиентного спуска с параметрами 0.001 и 0.01 шаг обучения (шаг изменения значений весов), значение коэффициента инерции momentum 0.9 (позволяется учитывать предыдущие изменения весов). Итоговая формула изменения весов может быть описана формулой (1.9)

(1.9)

где

– значение коэффициента инерции;

– шаг градиента (обучения);

– значение градиента функции (функции потерь) в точке ;

Для обновления весов во втором случае использовался алгоритма Адама с шагом обучения 0.001, который является более эффективным по сравнению с алгоритмом стохастического градиентного спуска за счет учитывания большего числа параметров и условий.

Для данной нейронной сети было проведено обучения при размере обучающей выборки 16,32,64 и 128 фотографий.

Для обновления весов при обучения использовался в алгоритм Адама со значением шага обучения 0.001.

В результате измерений было получено, что наивысшая точность равна 85 % для алгоритма Адама при размере выборки в 128 изображений. Для алгоритма стохастического градиента спуска же наивысшая точность 82 % при размере выборки 16. Данные результаты уже лучше, чем у алгоритмов машинного обучения. Результаты измерений представлены на рисунке 34.

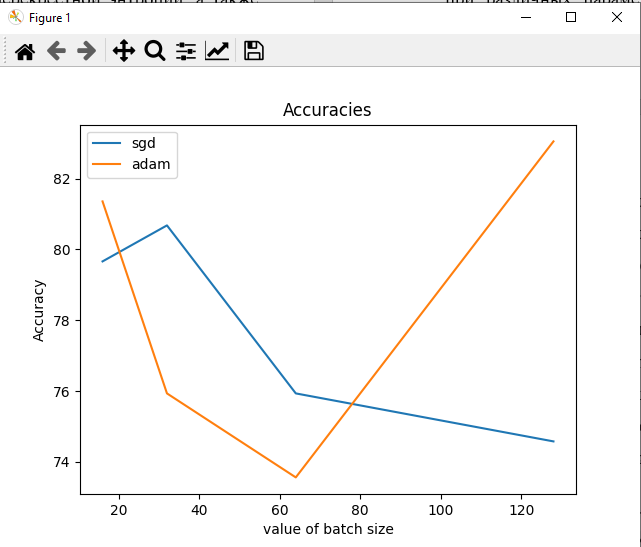


Рисунок 34 – Результаты измерения точности

Также были построены матрицы ошибок лучших итераций для двух алгоритмов. Результаты построения представлены на рисунка 35 и 36.

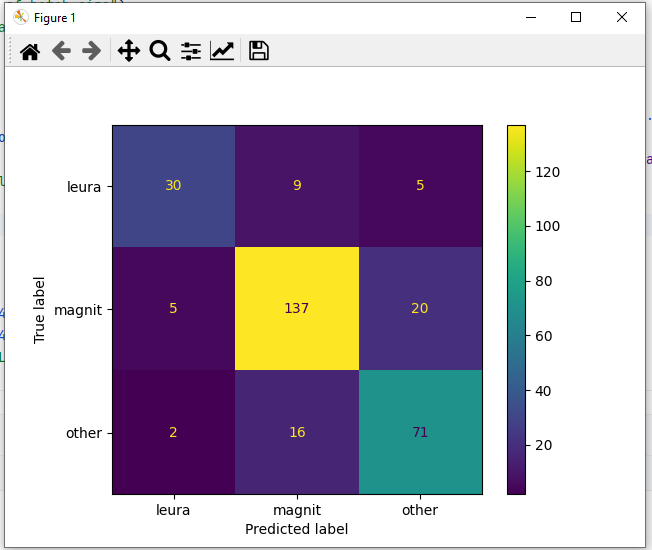


Рисунок 35 – Результаты построения матрицы ошибок для алгоритма градиентного спуска

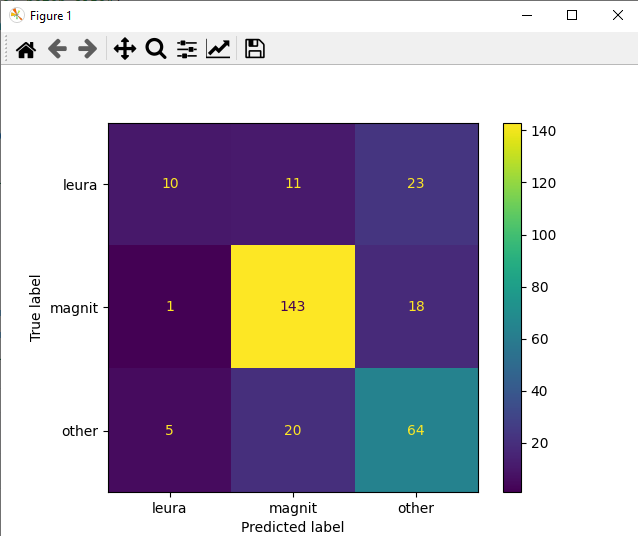


Рисунок 36 – Результаты построения матрицы ошибок для алгоритма Адама

Также были проведены измерения значения функции ошибки на тестовом наборе данных от количества эпох обучения и от размера обучающей выборки. Результаты измерения представлены на рисунка 37 и 38. Как видно по графику, увеличение количества эпох обучения способствует уменьшению значения функции ошибки. Увеличение размера обучающей выборки в данном случае также способствует уменьшению значения функции ошибки

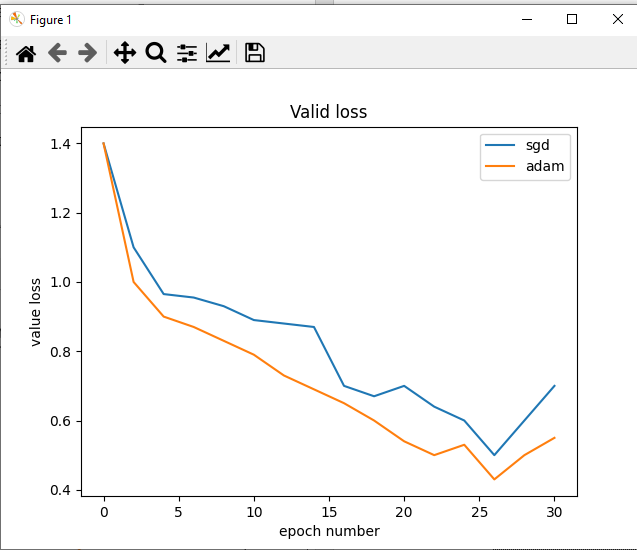


Рисунок 37 – График зависимости значения функции ошибки от количества эпох обучения

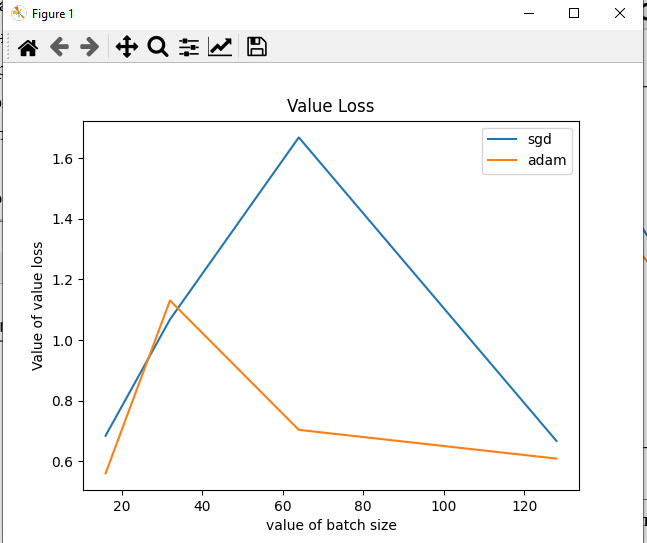


Рисунок 38 – График зависимости значения функции ошибки от размера обучающей выборки

# Оценка эффективности нейронной сети Inception v3

Второй нейронной сетью для обучения была выбрана Inception v3. Данная нейронная сеть включается в себя разветвляющиеся блоки, которые проводят несколько разных преобразований над данными.

Для данной нейронной сети были проведены оценки эффективности при размере обучающей выборки 16,32,64,128.

В результате проведения оценки точности работы было получено, что наибольшая точность в 86 % достигается при размере обучающей выборки в 32 изображения. Результаты оценки точности работы представлены на рисунке 39.

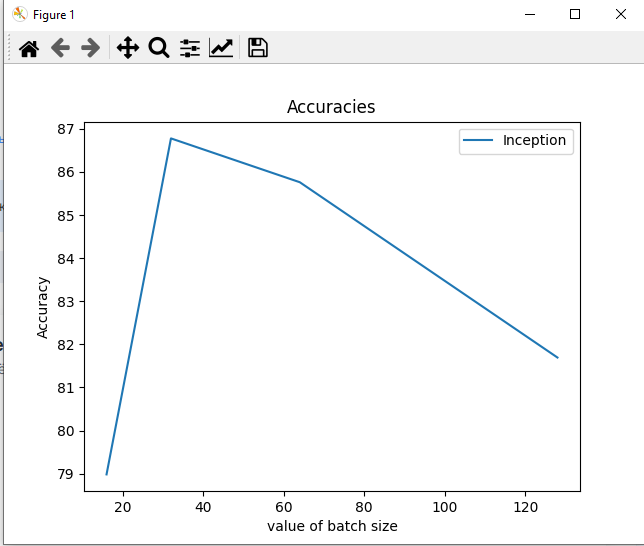


Рисунок 39 – График зависимости точности работы нейронной сети от размеров обучающей выборки

Также была построена матрица ошибок лучшей итерации нейронной сети. Результаты построения представлены на рисунке 40.

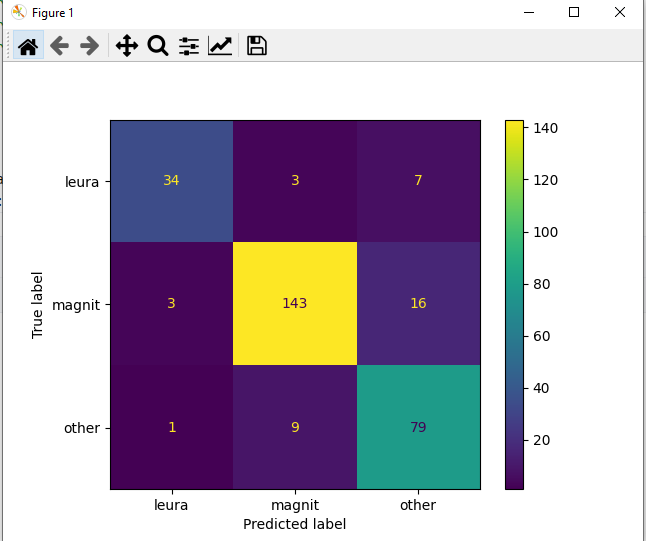


Рисунок 41 – Результаты построения матрицы ошибок лучшей итерации нейронной сети

Также были выведены зависимости значений функции ошибки от количества эпох обучения и от размеров обучающей выборки. Результаты изучения зависимостей представлены на рисунка 42 и 43. Согласно графику увеличения числа эпох обучения до 25 способствует уменьшению значения функции ошибки. Но изменение размера обучающей выборки может как уменьшить значение функции ошибки, так и увеличить.

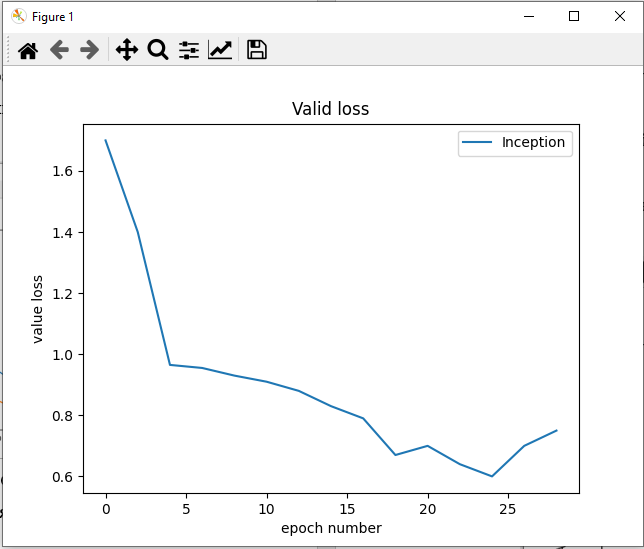


Рисунок 42 – График зависимости значения функции ошибки от количества эпох обучения

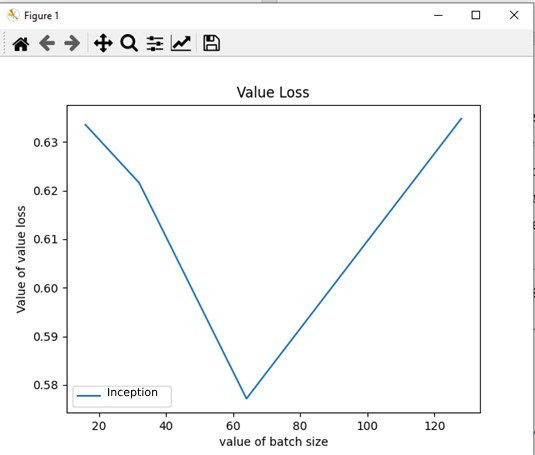


Рисунок 43 – График зависимости значения функции ошибки от размеров обучающей выборки

# Оценка эффективности нейронной сети DenseNet

Третьей нейронной сетью для обучения была выбрана DenseNet. Главной ее особенностью является наличие блоков, в котором каждый “подблок” получает на вход результаты работы всех предыдущих таких же подблоков. Подобная архитектура позволяет бороться с явлением затухание градиента и градиентного всплесков.

Для данной нейронной сети было проведено обучения при размере обучающей выборки 16,32 и 64 фотографий на версии сети из 121 слоя и при размере обучающей выборки 16 и 32 на версии сети из 201 слоя.

Для обновления весов при обучения использовался в алгоритм Адама со значением шага обучения 0.001.

В результате проведения оценок эффективности было выяснено, что для реализации нейронной сети из 121 слоя наивысшая точность равна 95 % при размере выборки в 32 изображения. Для реализации же из 201 слоя наивысшая точность равна 92 % для того же размера выборки. Результаты оценки точности представлены на рисунке 44.

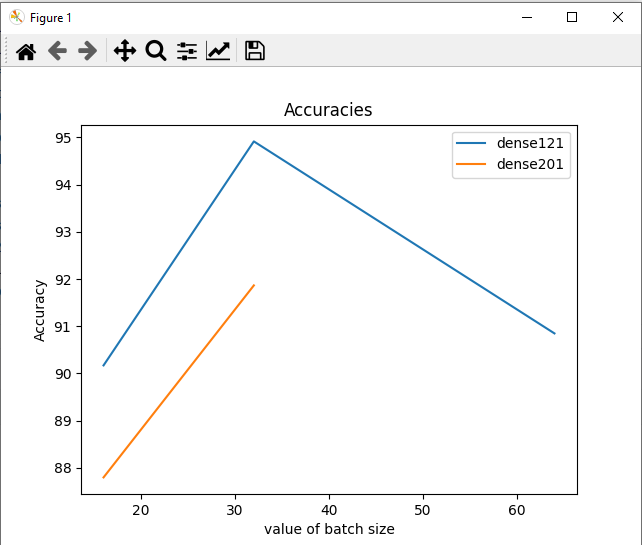


Рисунок 44 – График зависимости точности работы от размера обучающей выборки

Также были построены матрицы ошибок для лучших итераций обеих моделей. Результаты построения матрицы ошибок представлены на рисунках 45 и 46.

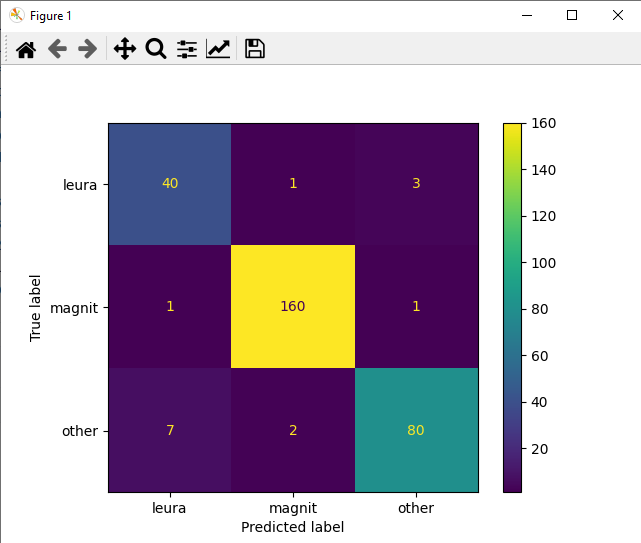


Рисунок 45 – Матрица ошибок для модели из 121 слоя

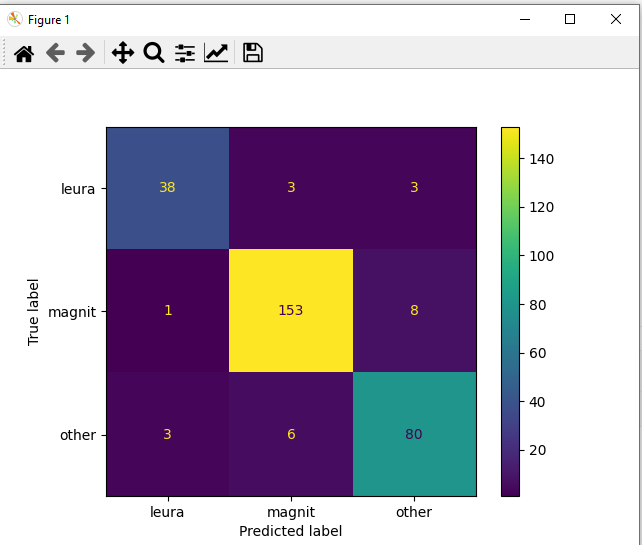


Рисунок 46 – Матрица ошибок для модели из 201 слоя

Также были выведены зависимости значения функции потерь от количества эпох обучения и от размера обучающей выборки. Результаты измерений представлены на рисунка 47 и 48. Как видно по графику, увеличение количества эпох обучения способствует уменьшению значения функции ошибки. Изменение же размера обучающей выборки как увеличивает, так и уменьшает значение функции ошибки.

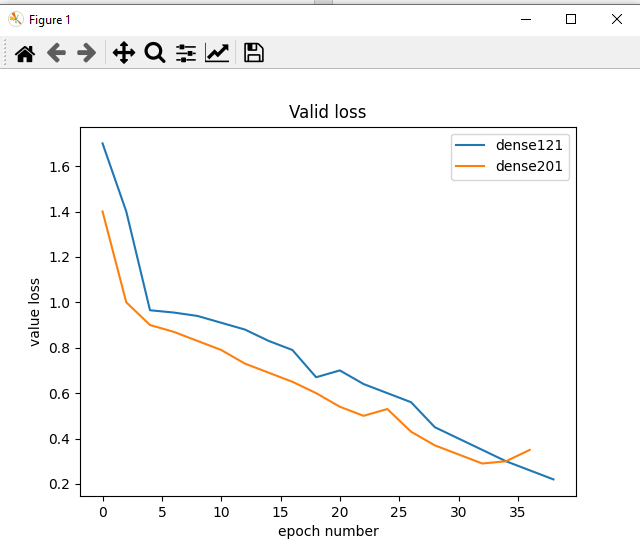


Рисунок 47 – График зависимости значения функции потерь от количества эпох обучения

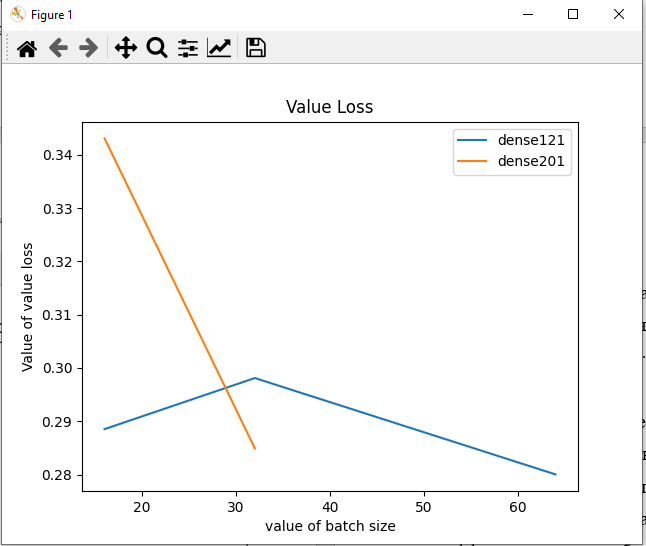


Рисунок 48 – График зависимости значения функции потерь от размера обучающей выборки

# Оценка эффективности нейронной сети YOLO V8

В качестве последней нейронной сети была выбрана YOLO. Она имеет достаточно сложную, но эффективную архитектуру.

Для данной нейронной сети было проведено обучения при размере обучающей выборки 16,32,64 фотографий с разрешением 224.

В результате оценки точности работы модели было получено, что максимальная точность в 95 % достигнута на размере обучающей выборки в 32 изображения. Результаты измерений представлены на рисунке 49.

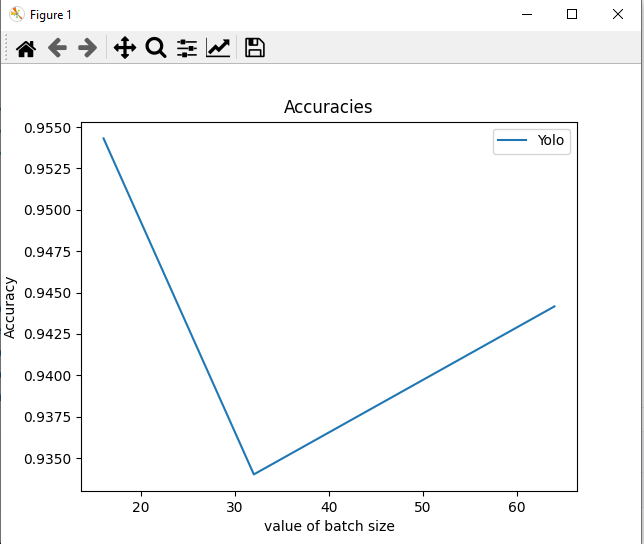


Рисунок 49 – График зависимости точности работы от размера обучающей выборки

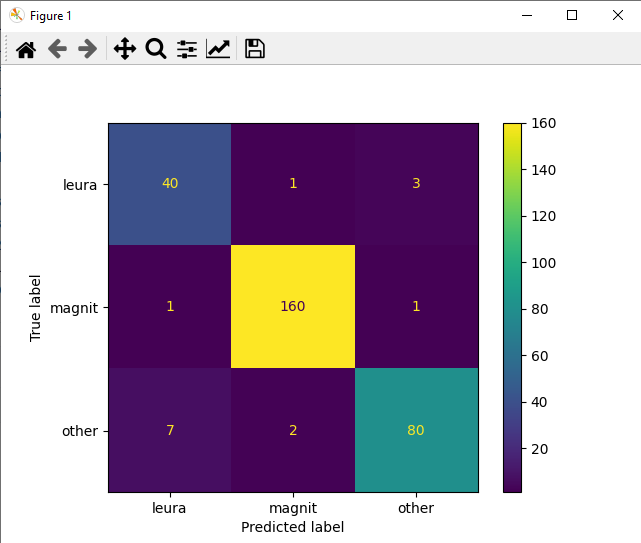


Рисунок 50 – Результат построения матрицы ошибок лучшей из итераций модели

Также была проведена оценка зависимости значения функции потерь от количества эпох обучения и от размера обучающей выборки. Согласно полученным графика на рисунка 51 и 52 увеличение обоих параметров в целом ведет к уменьшению обоих значений

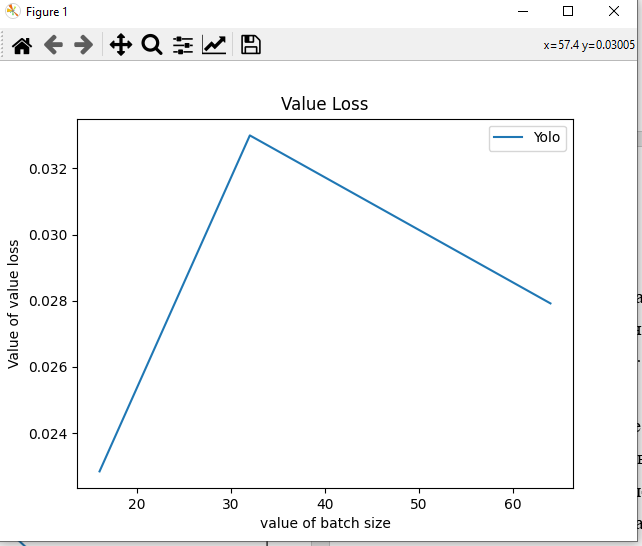


Рисунок 51 – График зависимости значения функции ошибки от размеров обучающей выборки

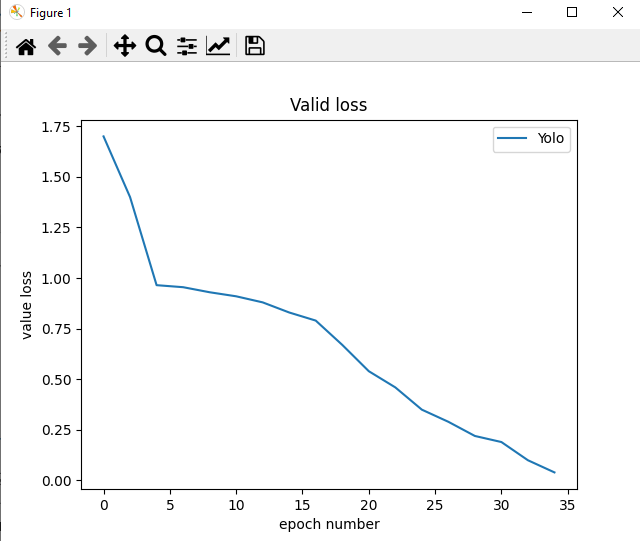


Рисунок 52 – График зависимости значения функции ошибки от количества эпох обучения

# Оценка эффективности написанной программы

Поскольку согласно предыдущей части наиболее точными оказались модели DenseNet, Inception и Yolo, то в качестве алгоритмов для классификации изображений в программе используются именно они.

За счет высокой точности данных моделей программа является достаточно эффективной, определяя и оставляя на изображении тележки с точностью до 95%. Однако имеются случаи снижения точности работы программы из – за ошибок, совершаемых еще на первом этапе работы, так как в некоторых случаях может распознаться не совсем верно ограничивающая рамка тележки (например, когда друг за другом стоят несколько тележек).

Также имеет место быть проблема времени работы программы. Поскольку необходимо сначала прогнать изображение через первую сеть, а затем и через вторую, на это тратится достаточно много времени и ресурсов устройства. Для эффективной работы данной программы в режиме реального времени понадобятся достаточно мощное по характеристикам устройство.

# ЗАКЛЮЧЕНИЕ

Задача классификации изображений объектов имеет множество возможностей применений на практике. Поэтому понимание принципов ее решения и разработка на их основе собственных программ являются актуальными.

В данной работе были изучены одни из наиболее популярных и эффективных алгоритмов решения данной задачи. Реализована программа для решения задачи классификации изображений магазинных тележек, использующая нейронные сети DenseNet, Inception, YOLO. Произведено исследование эффективности работы данной программы. По результатам экспериментальных исследований удалось выяснить, что данная программа достаточно эффективна, хотя и имеет свои недостатки в плане разработки и последующей работы, вызванных принципами работы нейронных сетей.

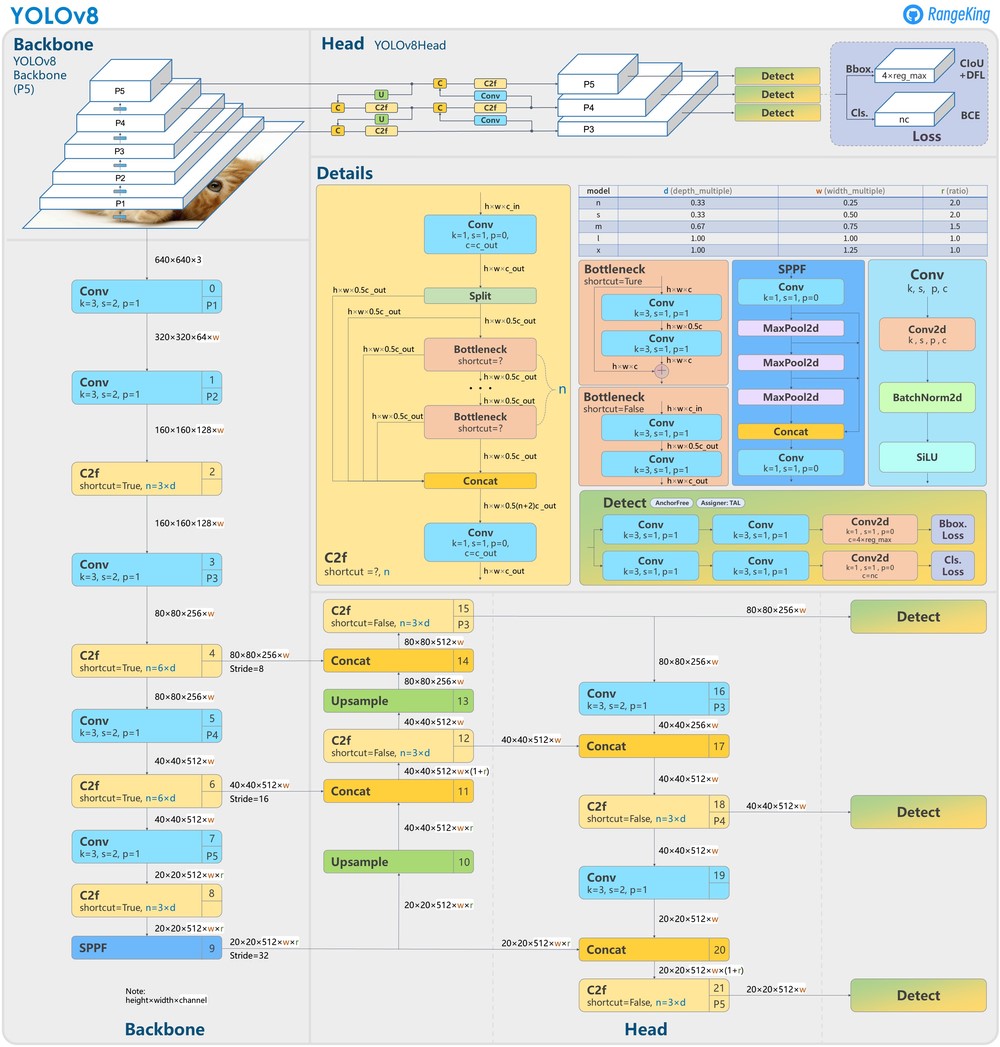
Разработанная программа может в дальнейшем использоваться как основа для системы слежения за тележками, либо быть частью более сложной программы, определяющей на основании фотографии очереди необходимость открытия дополнительной кассы.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Шолле, Франсуа Глубокое обучение на Python / Франсуа Шолле. – Санкт-Петербург : Питер, 2018. – 400 с. – ISBN 978-1617294433. – Текст : непосредственный.
2. Николенко, С. Глубокое обучение / С. Николенко, А. Кадурин, Е. Архангельская. – Санкт-Петербург : Питер, 2018. – 480 с. – ISBN 978-5-496-02536-2. – Текст : непосредственный.
3. Документация по архитектуре нейронной сети YOLO : [сайт]. – 2023. – URL: https://mmyolo.readthedocs.io/en/latest/recommended\_topics/ algorithm\_descriptions/yolov8\_description.html (дата обращения: 10.11.2023). – Текст : электронный.
4. Машинное обучение : [сайт]. – 2023. – URL: http://www. machinelearning.ru/wiki/index.php?title=%D0%97%D0%B0%D0%B3%D0%BB%D0%B0%D0%B2%D0%BD%D0%B0%D1%8F\_%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0 (дата обращения: 04.12.2023). – Текст : электронный.
5. Документация по нейронной сети YOLO : [сайт]. – 2023. – URL: https://docs.ultralytics.com (дата обращения: 04.11.2023). – Текст : электронный.
6. Krizhevsky, A. One weird trick for parallelizing convolutional neural networks : A. Krizhevsky. – Текст: электронный. – 2014. – URL: https://arxiv.org/pdf/1404.5997 (Дата обращения: 13.05.2024).
7. Huang, G. Densely Connected Convolutional Networks / G. Huang, Z. Liu, L. van der Maaten, K.Q. Weinberger. – Текст: электронный. – 2016. – URL: https://arxiv.org/pdf/1608.06993 (Дата обращения: 13.05.2024).
8. Szegedy C. Rethinking the Inception Architecture for Computer Vision / C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens. – Текст: электронный. – 2015. – URL: https://arxiv.org/pdf/1512.00567 (Дата обращения: 13.05.2024).

## **ПРИЛОЖЕНИЕ А**

**Архитектура YOLOV8**



## **ПРИЛОЖЕНИЕ Б**

**Код предобработки изображений для алгоритмов машинного обучения**

def create\_array(path, x, y, class\_name, classes=[], cluster\_param=0):  
 data=[]  
 for entry in glob.glob(path):  
 img = np.array(mpimg.imread(entry))  
 img1 = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  
 img2 = cv2.resize(img1, (x, y))  
 img2 = sklearn.preprocessing.normalize(img2.reshape(x\*y,3)).reshape(x,y,3)  
 if cluster\_param==1:  
 img2 = clusterize(img2,x,y)  
 if cluster\_param==2:  
 img2=np.vstack((img2.reshape([x,y,3]),clusterize(img2,x,y)))  
 img3 = np.reshape(img2,[img2.shape[0]\*img2.shape[1]\*3],order="F")  
 data.append(img3)  
 classes.append(class\_name)  
 return data,classes  
def resiz(x,y,cluster\_param=0):  
 classes=[]  
 data\_leura,classes= create\_array(r"C:\Users\kiril\Desktop\Curs2\TrolleysClassification-3\val\leura\_trolley\\*",x,y,"leura",cluster\_param=cluster\_param,classes=classes)  
 data\_magnit,classes = create\_array(r"C:\Users\kiril\Desktop\Curs2\TrolleysClassification-3\val\magnit\_trolley\\*",x,y,"magnit",cluster\_param=cluster\_param,classes=classes)  
 other,classes = create\_array(r"C:\Users\kiril\Desktop\Curs2\TrolleysClassification-3\val\other\\*",x,y,"other",cluster\_param=cluster\_param,classes=classes)train\_data = data\_leura+data\_magnit+other  
 print(len(train\_data),len(classes))  
 classes\_test=[]  
 data\_leura\_test,classes\_test = create\_array(r"C:\Users\kiril\Desktop\Curs2\TrolleysClassification-3\val\leura\_trolley\\*",x,y,"leura",cluster\_param=cluster\_param,classes=classes\_test)  
 data\_magnit\_test,classes\_test = create\_array(r"C:\Users\kiril\Desktop\Curs2\TrolleysClassification-3\val\magnit\_trolley\\*",x,y,"magnit",cluster\_param=cluster\_param,classes=classes\_test)  
 other\_test,classes\_test = create\_array(r"C:\Users\kiril\Desktop\Curs2\TrolleysClassification-3\val\other\\*",x,y,"other",cluster\_param=cluster\_param,classes=classes\_test)  
 test\_data=data\_leura\_test+data\_magnit\_test+other\_test  
def clusterize(img2,x,y):  
 img3 = img2.reshape(x\*y,3)  
 n = 60  
 k\_means = KMeans(n\_clusters=n,n\_init='auto')  
 model = k\_means.fit(img3)  
 centroids = k\_means.cluster\_centers\_  
 labels = k\_means.labels\_  
 res\_labels = centroids[labels]  
 result\_image = res\_labels.reshape(x, y, 3)  
 result\_image = result\_image.astype(np.uint8)  
 return np.round(result\_image)

## **ПРИЛОЖЕНИЕ В**

**Код предобработки наборов данных для AlexNet,DenseNet,Inception v3**

*#alexnet transforms  
 transform = transforms.Compose(  
 [transforms.Resize(256),  
 transforms.CenterCrop(224),  
 transforms.ToTensor(),  
 transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])  
  
 #inception v3 transforms  
 transform = transforms.Compose([  
 # transforms.Resize(299),  
 # transforms.CenterCrop(299),  
 # transforms.ToTensor(),  
 # transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),  
 # ])  
  
 #denseNet transforms* transform = transforms.Compose([  
 transforms.Resize(256),  
 transforms.CenterCrop(224),  
 transforms.ToTensor(),  
 transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),  
])  
 training\_set = torchvision.datasets.ImageFolder(r'C:\Users\kiril\Desktop\Curs2\TrolleysClassification-3\train',  
 transform=transform)  
 validation\_set = torchvision.datasets.ImageFolder(r'C:\Users\kiril\Desktop\Curs2\TrolleysClassification-3\val',  
 transform=transform)training\_loader = torch.utils.data.DataLoader(training\_set, batch\_size=128, shuffle=True, num\_workers=4)  
 validation\_loader = torch.utils.data.DataLoader(validation\_set, batch\_size=128, shuffle=False, num\_workers=4)

## **ПРИЛОЖЕНИЕ Г**

**Код реализации полного цикла обучения для моделей PyTorch**

import os  
from datetime import datetime  
import torch  
import torchvision  
import torchvision.transforms as transforms  
from torch.autograd import Variable  
from torch.utils.tensorboard import SummaryWriter  
def train\_one\_epoch(model,optimizer,device,loss\_fn):  
 model.train()  
 train\_loss = 0.0  
 total\_correct = 0  
 total\_samples = 0  
 for i, (images, labels) in enumerate(training\_loader):  
print(f"Выборка {i + 1}")optimizer.zero\_grad()images = Variable(images.to(device))  
 labels = Variable(labels.to(device))outputs = model(images)  
 \_, predicted = torch.max(outputs, 1)  
 total\_correct += (predicted == labels).sum().item()  
 total\_samples += labels.size(0)loss = loss\_fn(outputs, labels)loss.backward()optimizer.step()  
 train\_loss += loss.item() \* images.size(0)  
 accuracy = 100 \* total\_correct / total\_samples  
 return train\_loss, accuracy  
def testAccuracy(model,loss\_fn):  
 model.eval()  
 valid\_loss = 0.0  
 device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")  
 total\_correct = 0  
 total\_samples = 0  
 with torch.no\_grad():  
 for (images, labels) in validation\_loader:  
images = Variable(images.to(device))  
 labels = Variable(labels.to(device))outputs = model(images)  
 \_, predicted = torch.max(outputs.data, 1)  
 total\_correct += (predicted == labels).sum().item()  
 total\_samples += labels.size(0)loss = loss\_fn(outputs, labels)  
 valid\_loss += loss.item()\*images.size(0)  
 accuracy = 100 \* total\_correct / total\_samples  
 return valid\_loss, accuracy  
def main\_train(name, model,loss\_fn,optimizer,EPOCHS=100,check\_point\_epochs=4):device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")  
 print(device)  
 model.to(device)timestamp = datetime.now().strftime('%Y%m%d\_%H%M%S')  
 path = './runs/{}\_{}/'.format(name, timestamp)  
 os.makedirs(os.path.dirname(path), exist\_ok=True)  
 os.makedirs(os.path.dirname(path + "/best/"), exist\_ok=True)  
 print(len(validation\_loader.sampler))  
 print(len(training\_loader.sampler))  
 best\_valid\_loss = 1\_000\_000  
 for epoch\_number in range(EPOCHS):  
 print('EPOCH {}:'.format(epoch\_number + 1))  
 train\_loss, train\_accuracy = train\_one\_epoch(device=device, model=model, optimizer=optimizer,  
 loss\_fn=loss\_fn)  
 train\_loss = train\_loss / len(training\_loader.dataset)  
 valid\_loss, valid\_acc = testAccuracy(loss\_fn=loss\_fn, model=model)  
 valid\_loss = valid\_loss / len(validation\_loader.dataset)  
 print(f"Train loss:{train\_loss} Valid loss:{valid\_loss}")  
 print(f'Accuracy train = {train\_accuracy:.2f}%')  
 print(f'Accuracy valid = {valid\_acc:.2f}%')  
 if (epoch\_number % check\_point\_epochs == check\_point\_epochs - 1):  
 torch.save(model.state\_dict(), path + f"{epoch\_number + 1}.pt")  
 if valid\_loss < best\_valid\_loss:  
 best\_valid\_loss = valid\_loss  
 torch.save(model.state\_dict(), path + "/best/best.pt")  
if \_\_name\_\_ == '\_\_main\_\_':  
 *#alexnet* model = torchvision.models.alexnet(weights=None)  
 loss\_fn = torch.nn.CrossEntropyLoss()  
 optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9, weight\_decay=0.0005)  
 *#main\_train("alexnet",model,loss\_fn=loss\_fn,optimizer=optimizer,EPOCHS=100)  
  
 #inception* model = torchvision.models.inception\_v3(init\_weights=False)  
 model.aux\_logits = False  
 loss\_fn = torch.nn.CrossEntropyLoss()  
 optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)  
 *#main\_train("inception",model,loss\_fn=loss\_fn,optimizer=optimizer,EPOCHS=100)  
  
 #denseNet201* model = torchvision.models.densenet121(weights=None)  
 loss\_fn = torch.nn.CrossEntropyLoss()  
 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)  
 main\_train("denseNet", model, loss\_fn=loss\_fn, optimizer=optimizer, EPOCHS=100)

## **ПРИЛОЖЕНИЕ Д**

**Код главного окна**

***Main\_window.py***

import sys  
from PyQt5.QtWidgets import QApplication, QMainWindow  
from PyQt5 import uic  
from view import settings  
from controller import main\_window\_controller  
from images import images\_paths  
import os  
class MainWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.controller = main\_window\_controller.MainWindowController(self)  
  
 def create(self):  
 uic.loadUi(os.path.join(os.path.dirname(\_\_file\_\_), '..\\ui\\main.ui'), self)  
  
 *# Ивент открытия* self.settings.clicked.connect(lambda : self.controller.clicked\_settings())  
 self.dir.clicked.connect(lambda :self.controller.clicked\_dir())  
 self.file.clicked.connect(lambda: self.controller.clicked\_file())  
 self.start.clicked.connect(lambda :self.controller.clicked\_start())  
 self.clearlogs.clicked.connect(lambda :self.controller.clicked\_clear())  
 return self  
 def show(self):  
 super().show()  
 return self  
  
 def closeEvent(self, QCloseEvent):  
 del self.controller  
 sys.exit()

***main\_window\_controller.py***  
from view import settings  
from PyQt5.QtWidgets import QFileDialog, QMessageBox  
from model import yoloModel  
import os  
from pathlib import Path  
import validators  
class MainWindowController():  
 def \_\_init\_\_(self,window):  
 self.\_window = window  
 self.\_model = yoloModel.YoloModel()  
 def clicked\_settings(self):  
 sets = settings.Settings(self.model)  
 sets = sets.create()  
 sets.show()  
  
 @property  
 def window(self):  
 return self.\_window  
 @window.setter  
 def window(self,new):  
 self.\_window = new  
  
 @property  
 def model(self):  
 return self.\_model  
 @model.setter  
 def model(self,new):  
 self.\_model = new  
  
  
  
 def clicked\_dir(self):  
 dialog = QFileDialog()  
 dir = dialog.getExistingDirectory(self.window,'Open file','/home')  
 self.window.Input.setText(dir)  
 def clicked\_file(self):  
 dialog = QFileDialog()  
 fname = dialog.getOpenFileName(self.window,'Open file','/home')[0]  
 self.window.Input.setText(fname)  
  
 def clicked\_start(self):  
 f = Path(self.window.Input.text())  
 if (f.is\_file() or f.is\_dir() ) and self.window.Input.text() != '':  
 if self.window.img.isChecked():  
 self.model = self.model.create\_model(self.window.Input.text(), 1)  
 print(1)  
 elif self.window.video.isChecked():  
 self.model = self.model.create\_model(self.window.Input.text(), 2)  
 print(2)  
 self.model.process(self.window.Input.text())  
  
 else:  
 m = QMessageBox(1, "Выбор файлов", "Неверно выбрано значение")  
 m.setStyleSheet("background-color:white")  
 m.exec\_()  
  
 def clicked\_clear(self):  
 with open(os.path.join(os.path.dirname(\_\_file\_\_), '..\logs\logs.txt'), 'w') as file:  
 pass  
 m = QMessageBox(1, "Очистка логов", "Успешное очищено")  
 m.setStyleSheet("background-color:white")  
 m.exec\_()

## **ПРИЛОЖЕНИЕ Е**

**Код окна настроек**

***Settings.py***

from PyQt5.QtWidgets import QMainWindow, QFileDialog  
from PyQt5 import uic  
from images import images\_paths  
import os  
from controller import settings\_controller  
class Settings(QMainWindow):  
 def \_\_init\_\_(self,model):  
 super().\_\_init\_\_()  
 self.controller = settings\_controller.SettingsController(self,model)  
  
 def create(self):  
 uic.loadUi(os.path.join(os.path.dirname(\_\_file\_\_), '..\\ui\\settings.ui'), self)  
  
 self.weight\_but.clicked.connect(lambda :self.controller.choose\_weights())  
 self.cancel.clicked.connect(lambda : self.controller.close())  
 self.accept.clicked.connect(lambda: self.controller.change\_model())  
  
 self.weights.setText(self.controller.model.path\_weights)  
 self.conf.setText(str(self.controller.model.conf))  
 self.iou.setText(str(self.controller.model.iou))  
 self.imgsz.setText(str(self.controller.model.imgsz))  
  
 return self  
  
 def show(self):  
 super().show()  
 return self

***Settings\_controller.py***

from view import settings  
from PyQt5.QtWidgets import QFileDialog  
from PyQt5.QtWidgets import QMessageBox  
  
from pathlib import Path  
  
from model import yoloModel  
  
  
class SettingsController():  
 def \_\_init\_\_(self, window, model):  
 self.\_window = window  
 self.\_model = model  
  
 @property  
 def window(self):  
 return self.\_window  
 @window.setter  
 def window(self,new):  
 self.\_window = new  
  
 @property  
 def model(self):  
 return self.\_model  
 @model.setter  
 def model(self,new):  
 self.\_model = new  
  
  
 def choose\_weights(self):  
 dialog = QFileDialog()  
 fname = dialog.getOpenFileName(self.window, 'Open file', '/home','pt(\*.pt)')[0]  
 self.window.weights.setText(fname)  
  
 def change\_model(self):  
  
 k = 0  
  
 w\_path = self.window.weights.text()  
  
 file = Path(w\_path)  
 if not file.is\_file() and w\_path!='':  
 m = QMessageBox(1, "Выбор весов", "Неверно выбрано значение")  
 m.setStyleSheet("background-color:white")  
 m.exec\_()  
 else:  
 k = k+1  
  
 if k == 1:  
 conf = self.window.conf.text()  
 try:  
 if float(conf) >= -1:  
 k = k+1  
 except Exception:  
 m = QMessageBox(1, "Выбор минмального значения точности", "Неверно выбрано значение",)  
 m.setStyleSheet("background-color:white")  
 m.exec\_()  
  
 if k == 2:  
 iou = self.window.iou.text()  
 try:  
 if float(iou) >= -1:  
 k = k+1  
 except Exception:  
 m = QMessageBox(1, "Выбор значения наложения", "Неверно выбрано значение")  
 m.setStyleSheet("background-color:white")  
 m.exec\_()  
  
 if k == 3:  
 imgsz = self.window.imgsz.text()  
 try:  
 if int(imgsz) >= -1:  
 k = k+1  
 except Exception:  
 m = QMessageBox(1, "Выбор разрешения", "Неверно выбрано значение")  
 m.setStyleSheet("background-color:white")  
 m.exec\_()  
 if k == 4:  
 conf = float(conf)  
 iou = float(iou)  
 imgsz = int(imgsz)  
 self.model.change\_parameters(imgsz, w\_path, conf, iou)  
 self.close()  
  
 def create\_model(self):  
 self.model = yoloModel.YoloModel()  
  
  
 def close(self):  
 self.window.close()

## **ПРИЛОЖЕНИЕ Ж**

**Датасет для обучения**

Ссылка на датасет на сайте roboflow: https://universe.roboflow.com/kirill-kornilov-kn3yx/shopping-trolley-kn5tj

## **ПРИЛОЖЕНИЕ З**

**Полный код приложения**

Ссылка на github с программой: https://github.com/Murlocki/Curs