МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение

высшего образования

**«Северный (Арктический) федеральный университет имени М.В. Ломоносова»**

Высшая школа информационных технологий и автоматизированных систем

(наименование высшей школы / филиала / института )

**КУРСОВОЙ ПРОЕКТ**

|  |  |
| --- | --- |
| По дисциплине: | Глубокое обучение. Продвинутый уровень (модуль) |
|  | (учебная / производственная) |
| На тему: | Система распознавания товаров для кассы самообслуживания |
|  |  |

|  |  |
| --- | --- |
|  | Выполнил обучающийся:  Гордейчик Александр Сергеевич |
|  | (Ф.И.О.) |
|  | Направление подготовки:  09.04.02 Информационные системы и технологии |
|  | (код и наименование) |
|  | Курс: 2 |
|  | Группа: 151266 |
|  | Руководитель:  Васендина И. С., доцент, к.т.н. |
|  | (Ф.И.О. руководителя, должность / уч. степень / звание) |

Архангельск 2024

Оглавление

[Введение 3](#_Toc164971693)

[1 Техническое задание на разработку 4](#_Toc164971694)

[1.1 Введение 4](#_Toc164971695)

[1.2 Основания для проведения работ 4](#_Toc164971696)

[1.3 Плановые сроки начала и окончания работ по созданию системы 4](#_Toc164971697)

[1.4 Источники и порядок финансирования 5](#_Toc164971698)

[1.5 Назначение системы 5](#_Toc164971699)

[1.6 Требования к функциональным характеристикам 5](#_Toc164971700)

[1.7 Требования к модели машинного обучения 6](#_Toc164971701)

[1.8 Требования к надежности и условиям эксплуатации 6](#_Toc164971702)

[1.9 Требования к информационной и программной совместимости 7](#_Toc164971703)

[2 Модель машинного обучения 8](#_Toc164971704)

[2.1 Поиск и сравнение наборов данных 8](#_Toc164971705)

[2.2 Используемый набор данных 9](#_Toc164971706)

[2.3 Подготовка данных 11](#_Toc164971707)

[2.4 Модель машинного обучения 12](#_Toc164971708)

[2.5 Оценка использования кластера САФУ для обучения 13](#_Toc164971709)

[2.6 Обучение перовой модели 15](#_Toc164971710)

[2.7 Обучение второй модели 16](#_Toc164971711)

[3 Реализация кода для программы. 18](#_Toc164971712)

[3.1 Интерфейс программы 18](#_Toc164971713)

[3.2 Первый модуль, графическая оболочка 19](#_Toc164971714)

[3.3 Аналитическая часть, распознавание объектов 21](#_Toc164971715)

[4 Демонстрация работы 23](#_Toc164971716)

[Заключение 25](#_Toc164971717)

[Список использованных источников 26](#_Toc164971718)

[Приложение А (обязательное) Листинг приложения 27](#_Toc164971719)

Введение

В последнее время в сфере компьютерной визуализации и машинного обучения были сделаны значительные успехи в разработке алгоритмов, которые позволяют программам распознавать разные объекты, людей, животных и многое другое на изображениях и видео. Эта технология находит широкое применение в различных областях, включая обеспечение безопасности, мониторинг, создание автономных автомобилей, медицину и другие. В этом отчёте рассмотрена разработка программы, использующая модели машинного обучения для обнаружения немаркируемых товаров на кассах самообслуживания.

1. Техническое задание на разработку
   1. Введение

Использование моделей машинного обучения для распознавания товаров на кассах самообслуживания является важным моментом для повышения эффективности работы магазинов. Во-первых, это позволяет значительно сократить риск кражи и мошенничества, так как модели машинного обучения способны анализировать поведение покупателей и выявлять подозрительные действия. Например, попытки пронести неоплаченный товар или изменить его количество. Во-вторых, такое решение помогает ускорить процесс оформления покупок, так как система может автоматически распознавать товары, исключая необходимость повторного сканирования или ручной проверки. Это особенно актуально в условиях, когда магазины сталкиваются с нехваткой персонала. Использование машинного обучения позволяет минимизировать участие человека в процессе контроля и повысить общую эффективность работы магазина.

* 1. Основания для проведения работ

Основанием для исполнения работ по созданию приложения распознавания людей на фото является учебный план, направления подготовки 09.04.02 Информационные системы и технологии, по дисциплине «Интеграция моделей глубокого/машинного обучения в программные решения».

Факторы выполнения работ со стороны магазинов:

- повышение эффективности работы магазина;

- улучшение клиентского опыта;

- соответствие современным тенденциям;

- снижение уровня мошенничества;

- адаптация к изменяющемся условиям рынка.

* 1. Плановые сроки начала и окончания работ по созданию системы

Работы по разработке программы выполняются в рамках двух этапов:

1. Разработка пользовательской программы на языке Python

2. Обучение и интеграция модели машинного обучения.

Работы выполняются последовательно.

Срок выполнения работ 05.02.2024 по 20.04.2024

* 1. Источники и порядок финансирования

Разработка программы не предусматривает источников финансирования и реализуется на основе имеющихся ресурсов.

* 1. Назначение системы

Функциональное назначение системы распознавания товаров с помощью моделей машинного обучения на кассах самообслуживания заключается в автоматизации процесса идентификации товаров, что позволяет ускорить процесс оформления покупок и снизить вероятность ошибок, связанных с человеческим фактором. Это достигается благодаря использованию алгоритмов компьютерного зрения и машинного обучения, которые позволяют распознавать товары с видеопотока и определять их тип.

Эксплуатационное назначение системы заключается в повышении эффективности работы касс самообслуживания и улучшении качества обслуживания покупателей. Система позволяет сократить время ожидания покупателей в очереди, уменьшить нагрузку на кассиров и персонал магазина, а также повысить точность и скорость обработки товаров. Кроме того, в будущем, система может использоваться для анализа покупательских предпочтений и оптимизации ассортимента товаров в магазине.

* 1. Требования к функциональным характеристикам

Приложение для распознавания немаркируемых товаров должно иметь следующие выполняемые функции и элементы:

- способность распознавать заданный магазином перечень немаркируемых товаров;

- способность распознавать руки для исключения изменения веса товара пользователем;

- получать стоимость и изображение товаров из базы данных, выводить данную информацию на экран;

- выводить на экран список покупок пользователя;

- выводить на экран каталок товаров;

- предоставлять простой и интуитивно понятный интерфейс для пользователя кассы самообслуживания.

* 1. Требования к модели машинного обучения

Необходимо использовать две модели машинного обучения для распознавания:

- немаркируемых товаров;

- рук человека.

Необходимо обеспечить следующие условия.

Качество данных. Модель должна быть обучена на качественных данных, которые соответствуют поставленной задаче.

Размер данных. Для обучения модели необходимо достаточное количество данных.

Разделение данных. Необходимо разделить данные на обучающую, валидационную и тестовую выборки.

Выбор алгоритма. Алгоритм машинного обучения должен соответствовать поставленной задаче и свойствам данных.

Оценка качества. После обучения модели необходимо оценить её качество на тестовой выборке.

Достоверность. Модель должна быть достоверной и не допускать ошибок. Для повышения достоверности можно использовать методы контроля качества данных, такие как очистка данных и проверка на аномалии.

* 1. Требования к надежности и условиям эксплуатации

Готовое программное обеспечение будет запускаться на компьютере, установленном в кассу самообслуживания. Необходимо обеспечить отказоустойчивость и бесперебойную работу в течение рабочего дня.

* 1. Требования к информационной и программной совместимости

Разработка программного продукта ведется на языке Python. Используется дополнительное программное обеспечение QT Designer для реализации дизайна программы.

1. Модель машинного обучения
   1. Поиск и сравнение наборов данных

Для поиска набора данных использовался сайт Kaggle.com. В таблице 1 представлены найденные датасеты.

Таблица 1 – Найденные наборы данных

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название | Перевод | Адрес, www.kaggle.com/datasets/... |
| 1 | Fruits and Vegetables Image Recognition Dataset | Набор данных для распознавания изображений фруктов и овощей | kritikseth/fruit-and-vegetable-image-recognition |
| 2 | Fresh and Stale Images of Fruits and Vegetables | Свежие и несвежие изображения фруктов и овощей | raghavrpotdar/fresh-and-stale-images-of-fruits-and-vegetables |
| 3 | Vegetable Image Dataset | Набор данных изображений овощей | misrakahmed/vegetable-image-dataset |
| 4 | Fruits & Vegetables | Фрукты и овощи | jorgebailon/fruits-vegetables |
| 5 | Fruit and Vegetables SSM | Фрукты и овощи SSM | shadikfaysal/fruit-and-vegetables-ssm |
| 6 | Fruits and Vegetables dataset | Набор данных о фруктах и овощах | muhriddinmuxiddinov/fruits-and-vegetables-dataset |
| 7 | Vegetables Dataset for Object Detection | Набор данных овощей для обнаружения объектов | ayyuce/vegetables |
| 8 | Fruits & Vegetable Detection for YOLOv4 | Обнаружение фруктов и овощей для YOLOv4 | kvnpatel/fruits-vegetable-detection-for-yolov4 |
| 9 | Green Finder | Зеленый Искатель | tobiek/green-finder |

В таблице 2 представлена сравнительная таблица наборов данных.

Таблица 2 – Сравнение наборов данных

|  |  |  |  |
| --- | --- | --- | --- |
| № | Количество классов | Изображений на класс | Характеристика |
| 1 | 36 | 120 | Изображения многих овощей показаны в естественной природе |
| 2 | 6 | 320-2300 | Сравнение свежих и не свежих овощей, большой разброс по классам |

Окончание таблицы 2.

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 15 | 1400 | Наиболее приближен к реальным изображения овощей, с которыми покупатель взаимодействует в магазине |
| 4 | 7 | 934-976 | Изображения продуктов в бытовых условиях – много других объектов на изображениях |
| 5 | 30 | 600 | В наборе множество графических изображений |
| 6 | 10 | 570-643 | Сравнение свежих и несвежих овощей и фруктов, повторяющиеся изображения |
| 7 | - | Всего 619 | Датасет для детекции объектов, изображения содержать несколько классов сразу |
| 8 | 14 | 328 | Набор данных с продуктами в пакетах |
| 9 | 110 | 850 | Изображения в крайне низком качестве |

* 1. Используемый набор данных

Для выполнения задания выбран набор данных Vegetable Image Dataset [1]. (Набор данных изображений овощей). Данные опубликованы на сайте Kaggle.com в 2020 году пользователем M Israk Ahmed с лицензией CC BY-SA 4.0. Данная лицензия позволяет свободно использовать и изменять материалы с указанием авторства.

В наборе данных содержатся овощи, которые встречаются по всему миру: фасоль, горькая тыква, баклажаны, брокколи, капуста, перец стручковый, морковь, цветная капуста, огурец, папайя, картофель, тыква, редис и помидоры.

Всего используется 21000 изображений из 15 классов, каждый класс содержит 1400 изображений размером 224х224 и в формате \*.jpg. Набор данных разделен на 70 % для обучения, 15 % для проверки и 15 % для целей тестирования.

Пример данных показан на рисунке 1.



Рисунок 1 – Пример данных первого датасета

Набор данных изображений рук собран самостоятельно. Содержит 50 изображений рук в разных положениях. Пример изображений для датасета на рисунке 2.

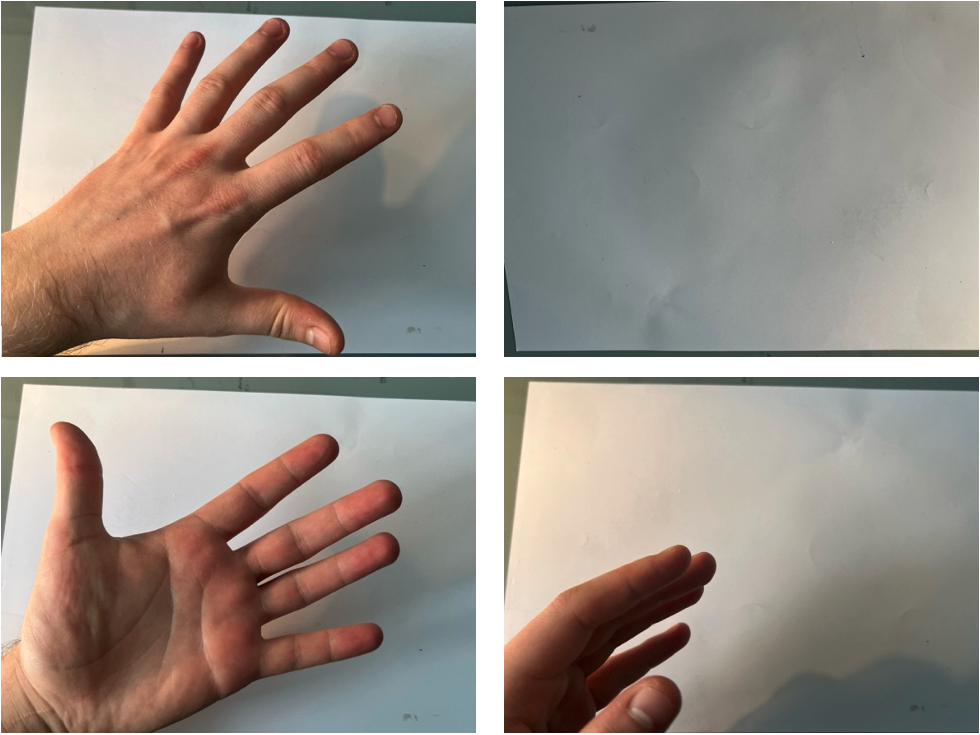


Рисунок 2 – Пример данных второго датасета

* 1. Подготовка данных

Набор данных с изображениями овощей и фруктов является полностью подготовленным и не требует какой-либо дополнительной обработки. Все изображения распределены по трём основным папкам: test, train и validation. Каждая из этих папок, в свою очередь, содержит ещё 15 подпапок, в которых находятся непосредственно сами изображения..

Чтобы исключить ситуацию, в которой покупатель придерживает взвешиваемый продукт для уменьшения веса, после чего его стоимость рассчитывается неправильно, а также для недопущения неточности идентификации, возникающей при перекрытии объекта руками клиента магазина, к приложению был подключён датасет, содержащий изображения рук. Так как в свободном доступе подходящих для решения поставленной задачи датасетов не было, он был собран самостоятельно с использованием смартфона.

Всего было сделано 100 фотографий: 50 снимков кисти руки в различных положениях, наиболее вероятных при размещении овощей на продуктовых весах, и в противовес в датасете присутствуют 50 снимков поверхности в допустимом для взвешивания состоянии, то есть без рук в кадре.

Исходные изображения имеют размер около 3 Мб, разрешение 4032 × 3024 пикселей, геопривязку, формат HEIF. Встроенными функциями ОС были произведены следующие действия:

- уменьшение разрешения снимка;

- сжатие;

- смена формата;

- удаление метаданных.

Итоговые изображения имеют размер около 150 Кб, разрешение 960 × 1280, формат jpeg. Данные разделены по пакам на 60 % для обучения, 20 % для проверки и 20% для целей тестирования. Итоговая иерархия приставлена на рисунке 3.

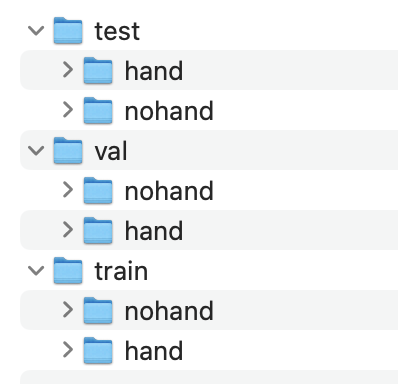


Рисунок 3 – Иерархия датасета c изображениями рук

Итоговый архив с изображениями весит 13 Мб.

* 1. Модель машинного обучения

Для распознавания используется Yolo8 [2]. Модель YOLOv8 от Ultralytics выделяется благодаря своей высокой точностью и скоростью обнаружения объектов в реальном времени. Это делает её особенно ценной для приложений, требующих быстрого и точного распознавания объектов, например, в автономном вождении. YOLOv8 также предлагает гибкую архитектуру, позволяющую разработчикам настраивать модель под свои конкретные нужды. Кроме того, модель обладает расширенными возможностями анализа изображений, включая сегментацию и прогнозирование классов, что позволяет ей обнаруживать не только объекты, но и их характеристики, такие как цвет и текстура.

В данном случае используется модель для классификации изображений yolov8n-cls.pt. Для классификации изображений с помощью YOLOv8, модель сначала обучается на большом наборе данных, содержащем изображения с соответствующими метками классов. После обучения модель способна классифицировать новые изображения, определяя, к какому классу принадлежит каждый объект на изображении.

Процесс классификации включает в себя следующие шаги:

Загрузка модели. Сначала необходимо загрузить предварительно обученную модель YOLOv8.

Преобразование изображения. Изображение должно быть преобразовано в формат, который модель может обработать.

Прогнозирование. Изображение подается на вход модели, которая предсказывает классы объектов на изображении.

Интерпретация результатов. Результаты прогнозирования интерпретируются, чтобы определить, к каким классам принадлежат объекты на изображении.

* 1. Оценка использования кластера САФУ для обучения

Для подключения к кластеру используется ide VScode. Через удаленный репозиторий по SSH соединению произведено подключение. Это позволяет пользоваться кластером через графическую оболочку, как показано на рисунке 4.

Создана папка Jupyter-server. Выполнены команды:

source /home/apps/intel/oneapi/setvars.sh

python3 -m venv venv

source venv/bin/activate

Создан файл ipynb. Установлено расширение для работы Jupyter ноутбуков, рисунок 5.

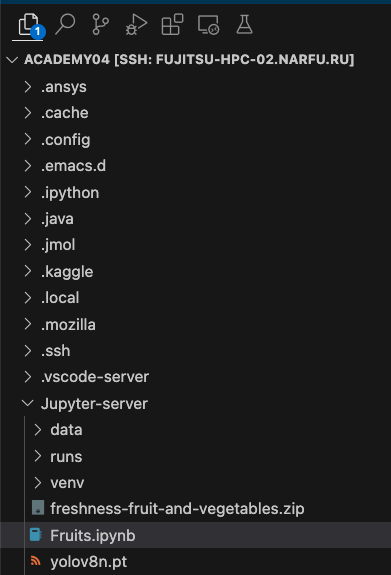


Рисунок 4 – графический доступ к кластеру

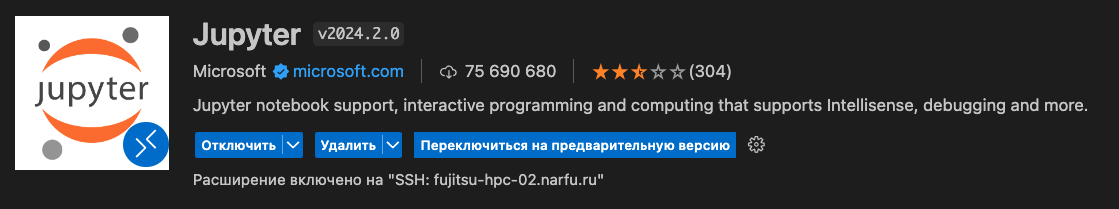


Рисунок 5 – Расширение Jupyter

Обучение одной эпохи на кластере заняло 23 минуты 35 секнуд, как показано на рисунке 6.

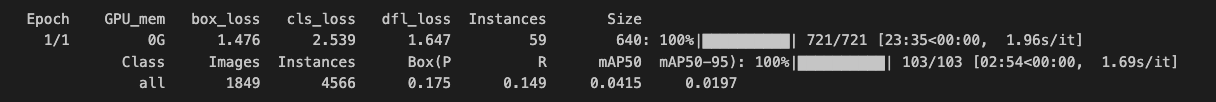


Рисунок 6 – Обучение эпохи на кластере

На локальной машине вычисления заняли 15 минут 28 секунд, рисунок 7

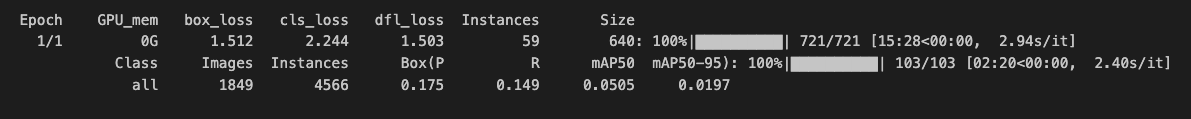


Рисунок 7 – Обучение на локальной машине

Обучение одной эпохи на платформе google colab [3] c графическим ускорителем заняло менее одной минуты, рисунок 8.

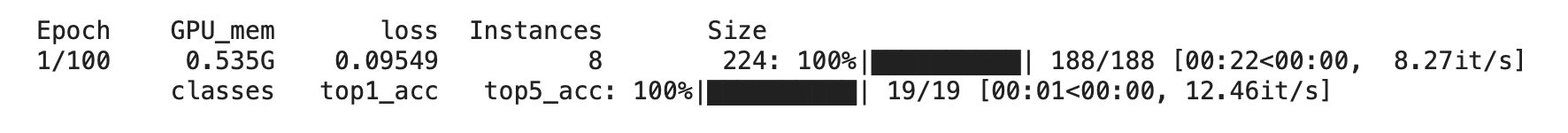


Рисунок 8 – Обучение на кластере

В ходе проверки производительности Google colab показал наилучшие результаты. Данный сервис имеет множество преимуществ. Google Colab доступен бесплатно, необходим только аккаунт. Возможность интеграции с Google Drive. Google Colab позволяет использовать графические процессоры (GPU) и тензорные процессоры (TPU), что значительно ускоряет процесс обучения моделей, особенно тех, которые требуют больших вычислительных мощностей. Кроме того, платформа поддерживает популярные библиотеки машинного обучения.

Однако, имеется не совсем очевидная система выделения объёма ресурсов. Доступ к TPU может прекратиться в любой момент на неопределённый срок.

* 1. Обучение первой модели

Для обучения модели в Google colab необходимо загрузить набор данных в среду выполнения. Так как данные расположены на Kaggle, их можно скачать с использованием API. Для этого необходимо в настройках аккаунта Kaggle создать уникальный токен пользователя. Полученный файл kaggle.json загрузить в среду выполнения Google colab.

После подключения к среде выполнения с аппаратным ускорителем T4 GPU и загрузке kaggle.json необходимо выполнить следующие команды, указанные в листинге 1.

Листинг 1 – Скачивание датасета

! pip install kaggle

! mkdir ~/.kaggle

! cp /content/kaggle.json ~/.kaggle/

! chmod 600 ~/.kaggle/kaggle.json

! kaggle datasets download misrakahmed/vegetable-image-dataset

! unzip /content/vegetable-image-dataset.zip

Данная последовательность устанавливает модуль kaggle, позволяющий скачивать датасеты. Создает папку в корне и помещает туда файл токена. После происходит скачивание и распаковка набора данных.

Для обучения модели необходимо установить пакет ultralytics. Содание модели происходит через команду YOLO. Процесс запуска обучения представлен в листинге 2.

Листинг 2 – Обучение первой модели

!pip install ultralytics

from ultralytics import YOLO

model = YOLO('yolov8n-cls.pt')

results = model.train(data='/content/Vegetable\_Images', epochs=10, imgsz=224)

Результаты обучения представлены на рисунке 9.

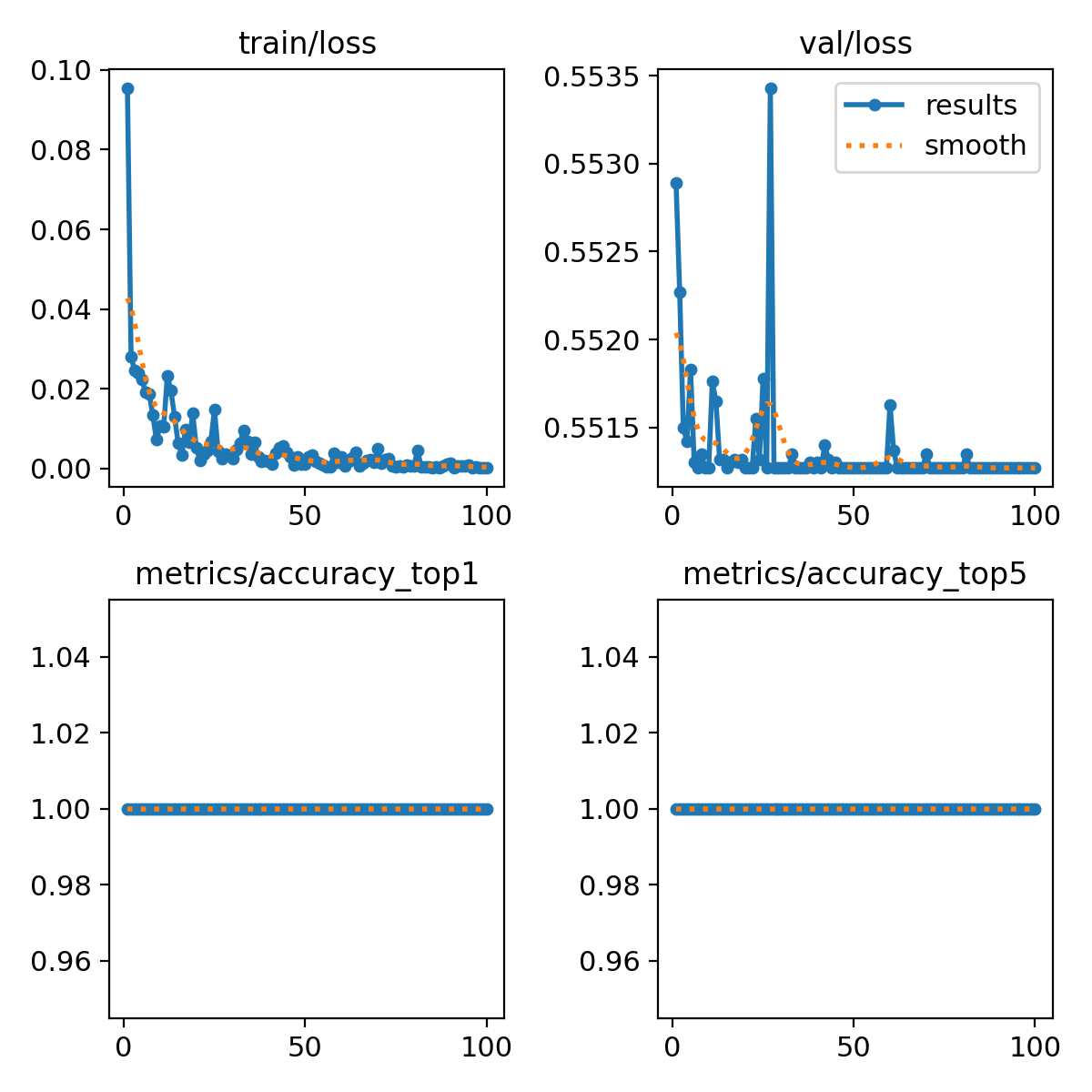


Рисунок 9 – результаты обучения первой модели

По графику видно, что ошибка на тренировочном и обучающем датасетах падает.

* 1. Обучение второй модели

Для обучения модели распознавания рук необходимо загрузить созданный архив в среду выполнения. Обучение модели происходит схоже с предыдущим вариантом (листинг 3).

Листинг 3 – Обучение второй модели.

!pip install ultralytics

from ultralytics import YOLO

! unzip /content/hand.zip

model = YOLO('yolov8s-cls.pt')

results = model.train(data='/content/hand', epochs=50, imgsz=64)

Результат обучения представлен на рисунке 10. По нему также видно, что ошибка падает как на обучающем, там и на тестовом датасете.

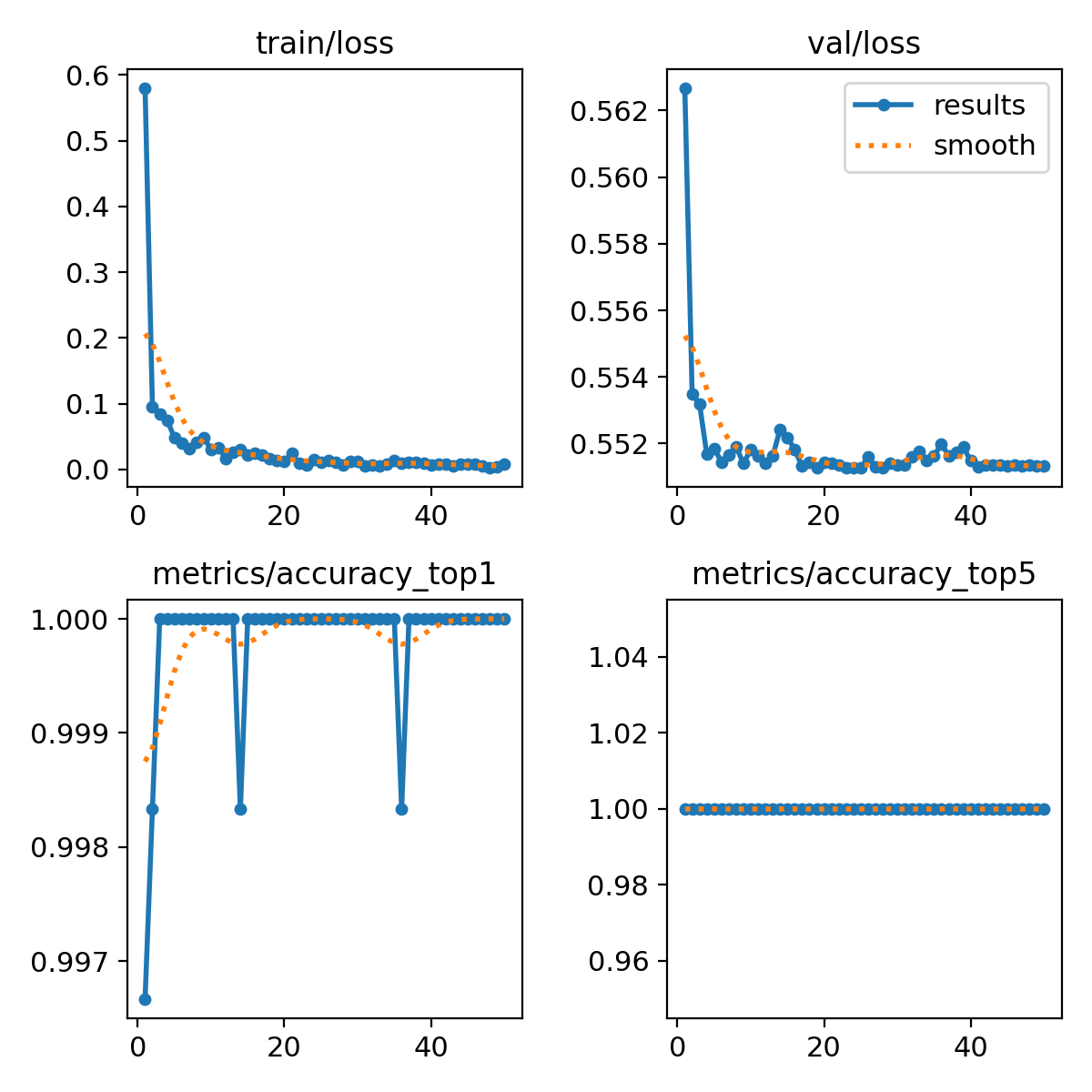


Рисунок 10 – результаты обучения первой модели

1. Реализация кода для программы.
   1. Интерфейс программы

Программа распознавания людей разработана в программе Qt Designer [4]. Рабочий интерфейс представлен на рисунке 11.

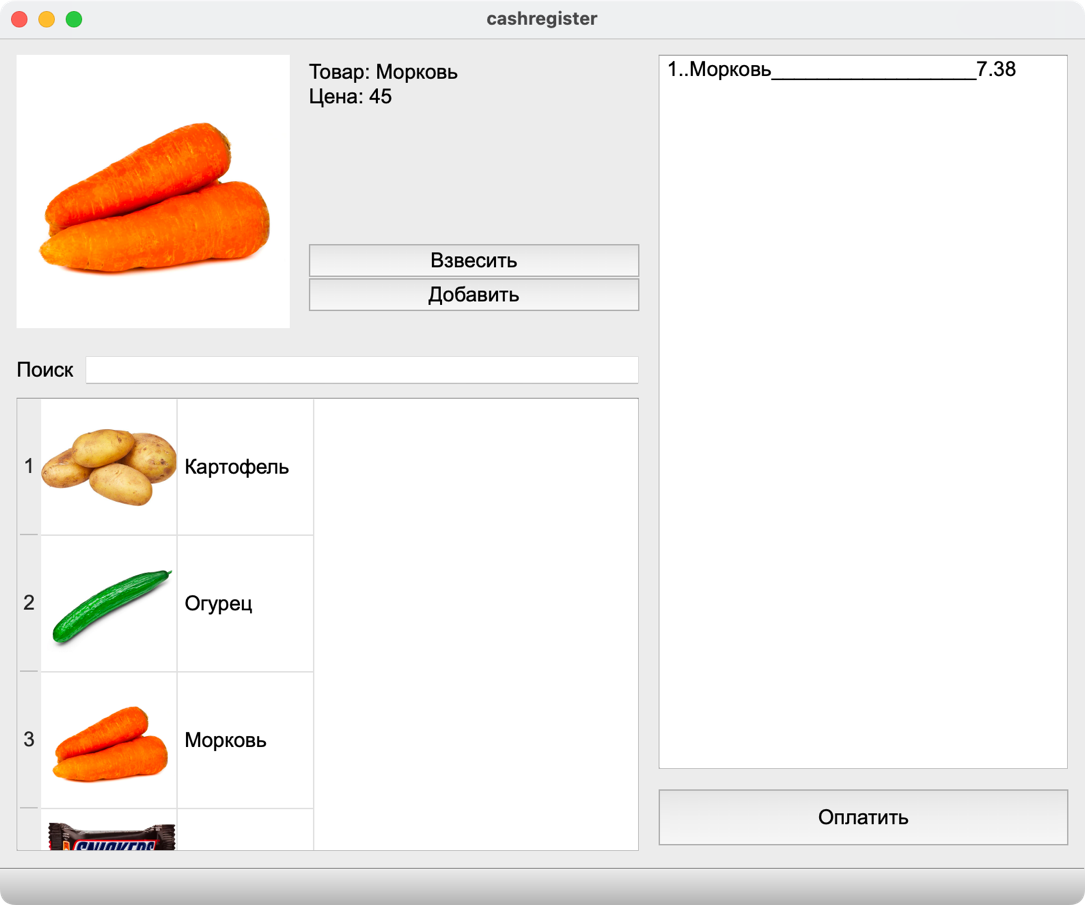


Рисунок 11 –интерфейс программы.

Структурно интерфейс состоит из следующих частей:

- окно распознанного товара;

- панель с информацией о распознанном товаре;

- кнопка «взвесить» для определения товара;

- кнопка «добавить» для добавления продукта в список покупок;

- каталог продуктов и поиск;

- список покупок и кнопка «оплатить».

Для удобства разработки и отладки системы присутствует техническое окно с отображением видеопотока, рисунок 12.

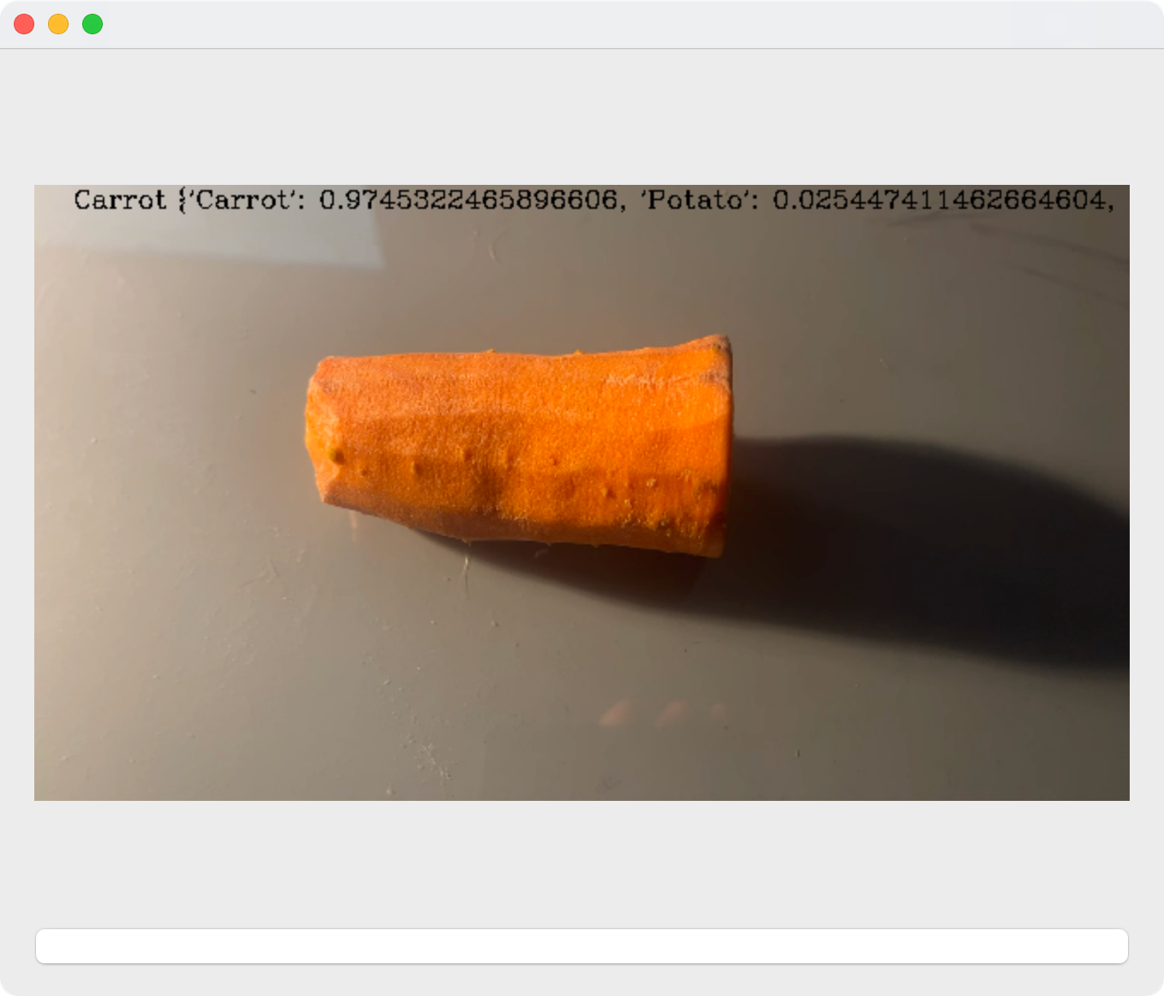


Рисунок 12 – Вспомогательное окно

На видеопотоке отображается распознанный класс и вероятность отнесения объекта на фото к данному классу. Кнопка в нижней части экрана переключает режим работы с real-time в разовое распознавание.

Программа стоит из двух разработанных модулей.

* 1. Первый модуль, графическая оболочка

Первый модуль реализует работу графического интерфейса. Создается новый класс, который реализует работу главного окна. В методе \_\_init\_\_ происходит подключение кнопок «взвесить» и «добавить» к методам, добавление продуктов на экран, создание вспомогательного окна. код инициализации представлен в листинге 4.

Листинг 4 – Инициализация программы

def \_\_init\_\_(self):

super(MainWindow, self).\_\_init\_\_()

self.setupUi(self)

self.info\_window = Info()

Окончание листинга 4

self.setWindowTitle('cashregister')

self.table\_products.insertColumn(0)

self.table\_products.insertColumn(0)

self.table\_products.setHorizontalHeaderLabels(['Товар'])

self.table\_products.horizontalHeader().hide()

self.bt\_add\_product.clicked.connect(self.add\_product\_to\_buy\_lust)

self.bt\_buy.clicked.connect(self.buy)

self.bt\_decode.clicked.connect(self.detect)

Product.load\_data()

self.add\_products\_in\_table()

self.info\_window.show()

Создание вспомогательного окна приведено в листинге 5.

Листинг 5 – Вспомогательное окно

class Info(QtWidgets.QWidget):

"""

Класс для отображения инфо окна

"""

def \_\_init\_\_(self,):

super().\_\_init\_\_()

layout = QtWidgets.QVBoxLayout()

self.label = QLabel(self)

self.label.setFixedSize(640, 480)

self.work\_online = QtWidgets.QPushButton()

layout.addWidget(self.label)

layout.addWidget(self.work\_online)

self.setLayout(layout)

self.cap = cv2.VideoCapture(CAMERA)

# Создание таймера

self.timer = QtCore.QTimer(self)

self.timer.timeout.connect(self.img\_update)

self.work\_online.clicked.connect(self.change\_work)

При нажатии кнопки «взвесить» происходит вызов функции detect, (листинг 6). Основные вычисления происходят в функции img\_update (листинг 7), отображении продукта на экране в функции clicked\_product (листинг 8).

Листинг 6 – Распознавание объектов

def detect(self):

data = self.info\_window.img\_update()

self.clicked\_product(name=data)

Листинг 7 – Функция img\_update

def img\_update(self):

"""

Обновить изображение

"""

# получение кадра

ret, frame = self.cap.read()

# поиск объекта

data = self.Detect\_obj(YOLOmodel,YoloHand,frame)

# вывод изображения на экран

color\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

h, w, ch = color\_frame.shape

img = QImage(color\_frame.data, w, h, ch \* w, QImage.Format\_RGB888)

scaled\_img = img.scaled(640, 480, QtCore.Qt.KeepAspectRatio)

self.label.setPixmap(QPixmap.fromImage(scaled\_img))

return(data)

Листинг 8 – Функция clicked\_product

def clicked\_product(self, name = None):

"""

Отобразить продукт на превью

"""

if name == None:

print(self.sender().objectName())

data = Product.get\_product\_by\_name(self.sender().objectName())

else:

data = Product.get\_product\_by\_name(name)

print()

pix = QPixmap()

pix.loadFromData(data.image\_data)

self.bt\_best\_product.setIcon(pix)

self.bt\_best\_product.setIconSize(QtCore.QSize(200,200))

self.product\_mass = random.random().\_\_round\_\_(3)

self.product\_name = data.name\_ru

self.product\_sum = round(self.product\_mass\*data.price,3)

label = f"""Товар: {data.name\_ru}\nЦена: {data.price}"""

self.label\_best\_product.setText(label)

* 1. Аналитическая часть, распознавание объектов

Второй модуль реализует распознавание объектов на видеопотоке. Используются две модели машинного обучение Yolo8. В заголовке файла создаются необходимые переменные (листинг 9).

Листинг 9 – Необходимые переменные

CAMERA = 0

# Пороги срабатывания для моделей

detection\_threshold = 0.95

detection\_threshold\_hand = 0.9

Окончание листинга 9

# работа в real-time

work\_online = False

Загрузка моделей

YOLOmodel = YOLO('models/best.pt')

YoloHand = YOLO('models/best\_hand.pt')

При вызове функции img\_update используется главная функция распознавания объектов Detect\_obj (Листинг 10). В ней последовательно выполняется распознавание с помощью объектов YOLOmodel и YoloHand.

Листинг 10 – Распознавание объектов

def MY2(self,model,model\_hand,image):

result\_value = {}

result\_hand\_value = {}

# Вывод результатов

# Распознавание рук

results\_hand = model\_hand(image, device='mps')

for result\_hand in results\_hand:

value, product = torch.sort(result\_hand[0].probs.data,descending = True)

print(value,product)

max\_product = result\_hand.names[int(product)]

max\_score = float(value)

if max\_score > detection\_threshold\_hand:

cv2.putText(image,f" {max\_product} {max\_score}",(int(200), int(100)),cv2.FONT\_HERSHEY\_COMPLEX, 1.5 ,thickness = 2, color=(0,0,0))

# Распознавание овощей

results = model(image, device='mps')

for result in results:

value, product = torch.sort(results[0].probs.data,descending = True)

for item in zip(value, product):

result\_value[result.names[int(item[1])]] = float(item[0])

max\_product = max(result\_value, key=result\_value.get)

max\_score = result\_value[max\_product]

if max\_score > detection\_threshold:

cv2.putText(image,f" {max(result\_value, key=result\_value.get)}",(int(20), int(40)),cv2.FONT\_HERSHEY\_COMPLEX, 1.5 ,thickness = 2, color=(0,0,0))

cv2.putText(image,f" {result\_value}",(int(200), int(40)),cv2.FONT\_HERSHEY\_COMPLEX, 1.5 ,thickness = 2, color=(0,0,0))

return max\_product

1. Демонстрация работы

После запуска программа имеет вид как на рисунке 13.

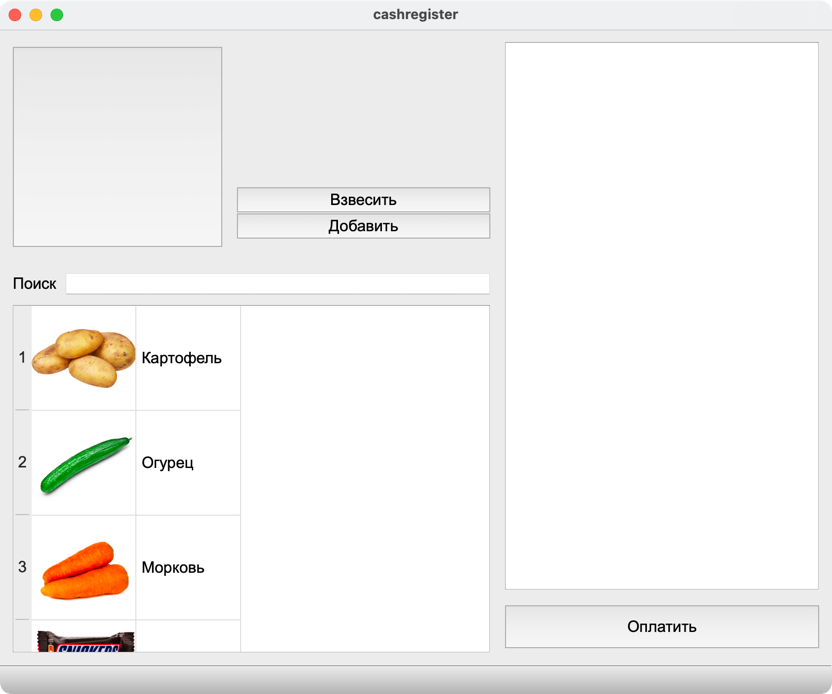


Рисунок 13 – стартовое окно программы

В поле зрения камеры помещается продукт. Предполагается, что это весы кассового аппарата самообслуживания. Изображение основного и вспомогательного экрана после нажатия на кнопку «взвесить» показано на рисунке 14.

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

Рисунок 14– Взвешивание картофеля

На вспомогательном экране отображается вероятность определения картофеля. В данном случае, нейросеть на 100% уверена, что на фото картофель. Его изображение помещается на главный экран. Производится запрос к базе данных, для получения изображения и цены товара. При нажатии кнопки «добавить» товар помещается в список покупок справа.

На рисунке 15 показан товар в момент помещения в поле зрения камеры. Система определяет наличие руки и отображает это на экране.

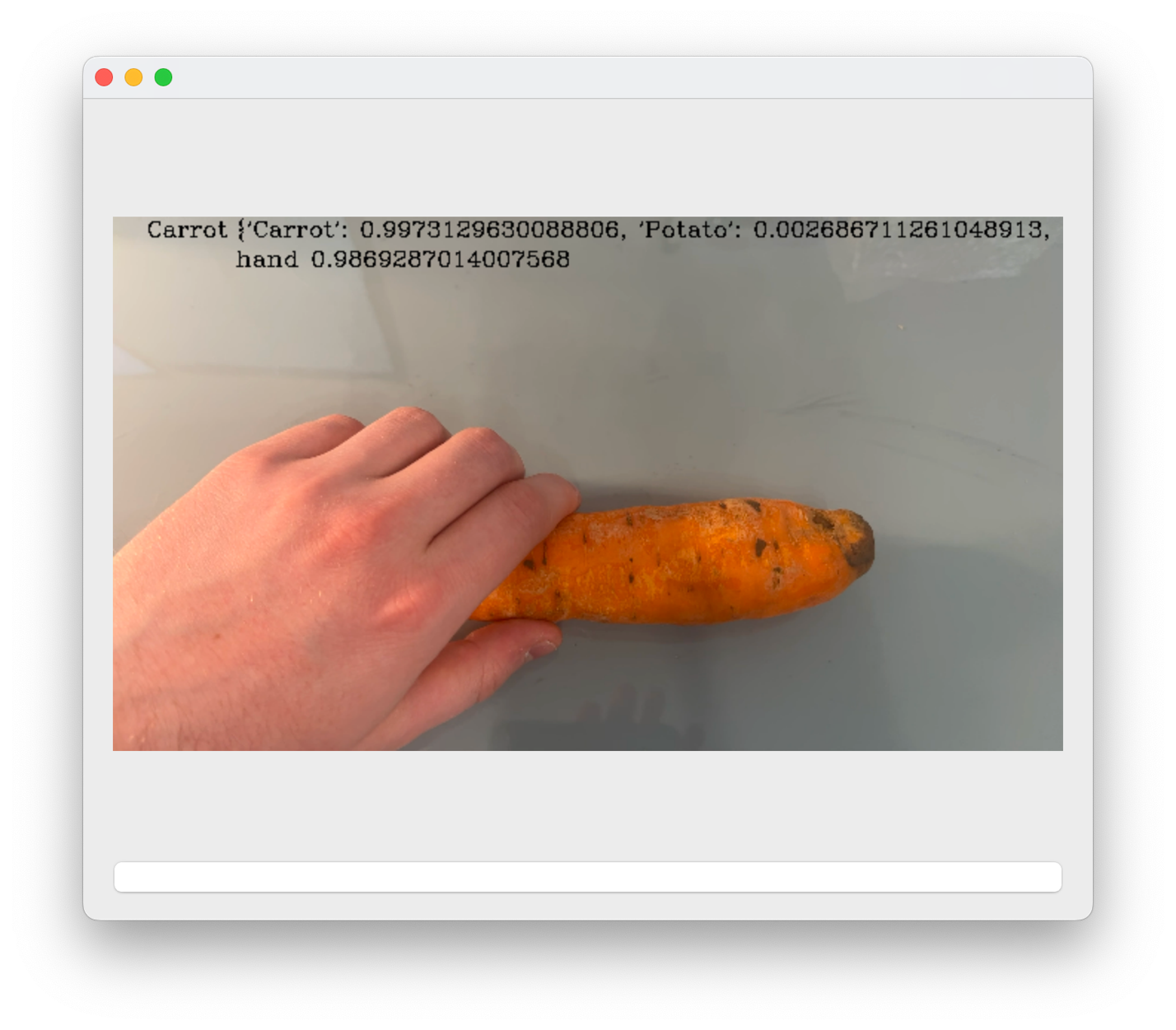


Рисунок 15 – Определение руки.

После удаления руки из кадра возможно добавление продукта в лист покупок.

Кнопка «оплатить» выводит в терминал общую сумму покупки и очищает список покупок, имитируя процесс оплаты.

Заключение

В ходе работы над проектом было реализовано приложение на языке Python, c использованием двух моделей машинного обучения Yolo8. Использовался набор данных с 15 классами овощей и самостоятельно собранный с изображениями рук. Приложение определяет покупаемый товар и отображает информацию о нем.

Благодаря автоматизации процесса идентификации товаров, сокращается время ожидания в очереди, уменьшается нагрузка на персонал и повышается точность обработки товаров.

Внедрение системы распознавания товаров на кассах самообслуживания является перспективным направлением развития розничной торговли, способствующим повышению конкурентоспособности и удовлетворению потребностей современного покупателя.

Список использованных источников

1 vegetable-image-dataset [Электронный ресурс] / Kaggle. – [San Francisco]: Kaggle Inc, – Режим доступа: https://www.kaggle.com/datasets/misrakahmed/vegetable-image-dataset (дата обращения 15.02.2024). – Заглавие с экрана.

2 Yolo8 [Электронный ресурс]:[офиц. сайт]/ Introducing Ultralytics YOLOv8– Электрон. дан. – [Калифорния] : [-2024]. –Режим доступа: https://docs.ultralytics.com/models/yolov8/, свободный (дата обращения : 10.04.2024). – Загл. с экрана.

3 Colab [Электронный ресурс]:[офиц. сайт]/ Google Colaboratory – Электрон. дан. – [Vienna] : [2015-2024]. –Режим доступа: https://colab.research.google.com/?hl=ru, свободный (дата обращения : 18.02.2024). – Загл. с экрана.

4 Qt Designer Download [Электронный ресурс]:[офиц. сайт]/ fman build system – Электрон. дан. – [Vienna] : [2016-2024]. –Режим доступа: https://build- system.fman.io/qt-designer-download, свободный (дата обращения : 10.03.2024). – Загл. с экрана.

Приложение А  
(обязательное)  
Листинг приложения

import sys

import random

from PySide6 import QtWidgets, QtCore

from PySide6.QtGui import QPixmap

from QT import mainWindow

from src.Product import Product

from detect import Info

class MainWindow(QtWidgets.QMainWindow, mainWindow.Ui\_MainWindow):

product\_mass = 0

product\_name = ''

product\_sum = 0

All\_sum = 0

def \_\_init\_\_(self):

super(MainWindow, self).\_\_init\_\_()

self.setupUi(self)

self.info\_window = Info()

self.setWindowTitle('cashregister')

self.table\_products.insertColumn(0)

self.table\_products.insertColumn(0)

self.table\_products.setHorizontalHeaderLabels(['Товар'])

self.table\_products.horizontalHeader().hide()

self.bt\_add\_product.clicked.connect(self.add\_product\_to\_buy\_lust)

self.bt\_buy.clicked.connect(self.buy)

self.bt\_decode.clicked.connect(self.detect)

Product.load\_data()

self.add\_products\_in\_table()

self.info\_window.show()

def add\_products\_in\_table(self):

"""

Добавить продукт в таблицу

"""

for one\_product in Product.product\_list:

self.table\_products.insertRow(0)

self.table\_products.setRowHeight(0,100)

# self.table\_products.insertColumn(0)

pix = QPixmap()

pix.loadFromData(one\_product.image\_data)

bt = QtWidgets.QPushButton(icon=pix,text='')

bt.setIconSize(QtCore.QSize(100,100))

bt.setObjectName(one\_product.name)

bt.clicked.connect(self.clicked\_product)

self.table\_products.setCellWidget(0,0,bt)

self.table\_products.setItem(0, 1, QtWidgets.QTableWidgetItem(str(one\_product.name\_ru)))

def clicked\_product(self, name = None):

"""

Отобразить продукт на превью

"""

if name == None:

print(self.sender().objectName())

data = Product.get\_product\_by\_name(self.sender().objectName())

else:

data = Product.get\_product\_by\_name(name)

print()

pix = QPixmap()

pix.loadFromData(data.image\_data)

self.bt\_best\_product.setIcon(pix)

self.bt\_best\_product.setIconSize(QtCore.QSize(200,200))

self.product\_mass = random.random().\_\_round\_\_(3)

self.product\_name = data.name\_ru

self.product\_sum = round(self.product\_mass\*data.price,3)

label = f"""Товар: {data.name\_ru}\nМасса: {self.product\_mass}\nЦена: {data.price}\nСумма: { self.product\_sum}"""

label = f"""Товар: {data.name\_ru}\nЦена: {data.price}"""

self.label\_best\_product.setText(label)

numbers\_iterator = iter(range(1,999))

def add\_product\_to\_buy\_lust(self):

"""

Добавление продукта с список покупок

"""

label = str.ljust(str(next(self.numbers\_iterator)),3,'.')

label += str.ljust(str(self.product\_name),25,'\_')

label += str(self.product\_sum)

self.list\_products\_buy.addItem(label)

self.All\_sum += self.product\_sum

def buy(self):

"""

Кнопка купить

"""

print(f'Куплено продуктов на {self.All\_sum}')

self.All\_sum = 0

self.list\_products\_buy.clear()

self.numbers\_iterator = iter(range(1,999))

def detect(self):

"""

Произвести детектирование

"""

d = self.info\_window.img\_update()

print(d)

self.clicked\_product(name=d)

def main():

app = QtWidgets.QApplication(sys.argv)

main = MainWindow()

main.show()

sys.exit(app.exec())

if \_\_name\_\_ == '\_\_main\_\_':

main()

import sys

from PySide6 import QtWidgets, QtCore

from PySide6.QtGui import QPixmap, QResizeEvent, QImage, QFont

from PySide6.QtWidgets import QLabel

from ultralytics import YOLO

import cv2

import torch

CAMERA = 0

detection\_threshold = 0.95

detection\_threshold\_hand = 0.9

work\_online = False

cl = ['Carrot','Cucumber', 'Potato']

cl\_dict = {i: c for i, c in enumerate(cl)}

YOLOmodel = YOLO('/Users/aleksandrgordejcik/Yandex.Disk-banahikk.localized/masters/samsung/train/train3/weights/best.pt') # load a pretrained model (recommended for training)

YoloHand = YOLO('/Users/aleksandrgordejcik/Yandex.Disk-banahikk.localized/masters/samsung/cashregister/models/best\_hand.pt') # load a pretrained model (recommended for training)

# YoloHand = YOLO()

class Info(QtWidgets.QWidget):

"""

Класс для отображения инфо окна

"""

def \_\_init\_\_(self,):

super().\_\_init\_\_()

layout = QtWidgets.QVBoxLayout()

self.label = QLabel(self)

self.label.setFixedSize(640, 480)

self.work\_online = QtWidgets.QPushButton()

layout.addWidget(self.label)

layout.addWidget(self.work\_online)

self.setLayout(layout)

self.cap = cv2.VideoCapture(CAMERA)

# Создание таймера

self.timer = QtCore.QTimer(self)

self.timer.timeout.connect(self.img\_update)

self.work\_online.clicked.connect(self.change\_work)

def MY2(self,model,model\_hand,image):

result\_value = {}

result\_hand\_value = {}

# Вывод результатов

# results = model.predict(image, device='mps')

results\_hand = model\_hand(image, device='mps')

for result\_hand in results\_hand:

value, product = torch.sort(result\_hand[0].probs.data,descending = True)

print(value,product)

max\_product = result\_hand.names[int(product)]

max\_score = float(value)

if max\_score > detection\_threshold\_hand:

cv2.putText(image,f" {max\_product} {max\_score}",(int(200), int(100)),cv2.FONT\_HERSHEY\_COMPLEX, 1.5 ,thickness = 2, color=(0,0,0))

results = model(image, device='mps')

for result in results:

value, product = torch.sort(results[0].probs.data,descending = True)

for item in zip(value, product):

result\_value[result.names[int(item[1])]] = float(item[0])

max\_product = max(result\_value, key=result\_value.get)

max\_score = result\_value[max\_product]

if max\_score > detection\_threshold:

cv2.putText(image,f" {max(result\_value, key=result\_value.get)}",(int(20), int(40)),cv2.FONT\_HERSHEY\_COMPLEX, 1.5 ,thickness = 2, color=(0,0,0))

cv2.putText(image,f" {result\_value}",(int(200), int(40)),cv2.FONT\_HERSHEY\_COMPLEX, 1.5 ,thickness = 2, color=(0,0,0))

return max\_product

My\_data = {}

detections = []

# for index, r in enumerate(result.boxes.data.tolist()):

# x1, y1, x2, y2, score, class\_id = r

# x1 = int(x1)

# x2 = int(x2)

# y1 = int(y1)

# y2 = int(y2)

# class\_id = int(class\_id)

# if score > detection\_threshold:

# detections.append([x1, y1, x2, y2, score])

# score = str(round(score,3))[2:4]

# text = cl\_dict[class\_id]

# cv2.putText(image,f" {text}",(int(x1), int(y1-10)),cv2.FONT\_HERSHEY\_COMPLEX, 1.5 ,thickness = 2, color=(0,0,0))

# cv2.rectangle(image, (int(x1), int(y1)), (int(x2), int(y2)),(0.4, 0.5, 0.5), 3)

# tracker.update(frame, detections)

# for track in tracker.tracks:

# bbox = track.bbox

# x1, y1, x2, y2= bbox

# name = f'{x1}/{y1}/{x2}/{y2}'

# track\_id = track.track\_id

# color = (colors[track\_id % len(colors)])

# cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)),color, 3)

# cv2.putText(frame,str(track\_id),(int(x1), int(y1-10)),cv2.FONT\_HERSHEY\_COMPLEX, 1.5 ,thickness = 2, color=color)

return None

def img\_update(self):

"""

Обновить изображение

"""

ret, frame = self.cap.read()

data = self.MY2(YOLOmodel,YoloHand,frame)

# Reading frame in gray scale to process the pattern

gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Reading the image in RGB to display it

color\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

# Creating and scaling QImage

h, w, ch = color\_frame.shape

img = QImage(color\_frame.data, w, h, ch \* w, QImage.Format\_RGB888)

scaled\_img = img.scaled(640, 480, QtCore.Qt.KeepAspectRatio)

# Emit signal

self.label.setPixmap(QPixmap.fromImage(scaled\_img))

return(data)

def change\_work(self):

"""

Смена работы с непрерывной на разовое измерение

"""

if self.work\_online:

self.work\_online = False

self.timer.stop()

else:

self.work\_online = True

self.timer.start()

def main():

app = QtWidgets.QApplication(sys.argv)

main = Info()

main.show()

sys.exit(app.exec())

if \_\_name\_\_ == '\_\_main\_\_':

main()