|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  федеральное государственное автономное образовательное учреждение высшего образования  «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ» |

Кафедра №43 «Компьютерных технологий и программной инженерии»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ОТЧЁТ ПО ПРАКТИКЕ  ЗАЩИЩЁН С ОЦЕНКОЙ  Руководитель |  | | | |
| Ст. преподаватель |  |  |  | М.Д. Поляк |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

ОТЧЁТ ПО ПРАКТИКЕ

|  |  |  |
| --- | --- | --- |
| вид практики | производственная | |
| тип практики | по получению профессиональных умений и опыта профессиональной деятельности | |
| на тему индивидуального задания | | Обнаружение и классификация различных блюд (еды) |
| на видео | | | |
|  | | | |

|  |  |
| --- | --- |
| выполнен | Григорьевым Данилой Алексеевичем |
| фамилия, имя, отчество обучающегося в творительном падеже | |

|  |  |  |  |
| --- | --- | --- | --- |
| по направлению подготовки | 09.03.04 |  | Программная инженерия |
|  | код |  | наименование направления |
|  | | | |
| наименование направления | | | |
| направленности | 02 |  | Проектирование программных систем |
|  | код |  | наименование направленности |
|  | | | |
| наименование направленности | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Обучающийся группы № | 4233К |  |  |  | Д.А. Григорьев |
|  | номер |  | подпись, дата |  | инициалы, фамилия |

Санкт–Петербург 2025

Оглавление

[**ЦЕЛЬ РАБОТЫ** 2](#_Toc204173459)

[**ИСХОДНЫЕ ДАННЫЕ И ИХ ОПИСАНИЕ** 2](#_Toc204173460)

[**ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ** 4](#_Toc204173461)

[**ПРАКТИЧЕСКИЙ РАЗДЕЛ** 6](#_Toc204173462)

[**РЕЗУЛЬТАТ** 7](#_Toc204173463)

[**ВЫВОД** 10](#_Toc204173464)

[**Приложение А** 11](#_Toc204173465)

[**Приложение Б** 12](#_Toc204173466)

# **ЦЕЛЬ РАБОТЫ**

Целью работы является разработка программного обеспечения для обнаружения и классификации различных блюд на видео в реальном времени с использованием нейронной сети YOLOv8. В рамках данной цели решаются следующие задачи:

1. Изучение теоретических основ обнаружения объектов и классификации изображений с применением нейронных сетей.
2. Выбор и настройка архитектуры нейронной сети YOLOv8 для задачи классификации блюд.
3. Обучение модели на специализированном наборе данных, содержащем изображения различных блюд.
4. Разработка программного приложения для обработки видео и вывода результатов классификации.
5. Тестирование и оценка точности работы модели, а также её оптимизация для достижения наилучших результатов.
6. Оформление отчётной документации по результатам проделанной работы.

# **ИСХОДНЫЕ ДАННЫЕ И ИХ ОПИСАНИЕ**

Для обучения и тестирования модели использовались следующие данные:

1. **Набор данных изображений блюд**:
   * Обучающая, валидационная и тестовая выборки.
   * Изображения в формате JPG/PNG с разрешением 640x640 пикселей.
   * Аннотации в формате YOLO (текстовые файлы с координатами bounding box и метками классов).



Рисунок 1- Исходное изображение



Рисунок 2 – Маска детекции изображения

1. **Классы блюд**:
   * pizza, hamburger, ice cream, french fries.
2. **Дополнительные данные**:
   * Аугментация данных (вращение, изменение яркости, контрастности) для увеличения разнообразия обучающей выборки.
   * Использование предобученных весов yolov8n.pt для ускорения обучения.

# **ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ**

YOLO — это однопроходный (one-stage) детектор объектов, который обрабатывает изображение за один проход через нейронную сеть. Алгоритм делит изображение на сетку (например, S×S клеток), где каждая клетка отвечает за предсказание объектов, центры которых попадают в неё. Для каждой клетки сетки YOLO предсказывает несколько bounding box (ограничивающих рамок), каждый из которых включает координаты (x, y — центр, w, h — ширина и высота), уверенность (confidence) в наличии объекта и вероятности классов.

YOLO использует заранее заданные anchor boxes (якорные рамки) для улучшения предсказания формы объектов. После получения предсказаний применяется Non-Maximum Suppression (NMS), чтобы убрать дублирующиеся и слабовероятные рамки, оставляя только наиболее точные.

Ключевые особенности:

* Однопроходная архитектура (высокая скорость).
* Использование anchor boxes для улучшения детекции.
* NMS для фильтрации избыточных предсказаний.

YOLO объединяет задачи локализации (поиск bbox) и классификации в единую нейросеть, что делает его быстрым и эффективным для реального времени.

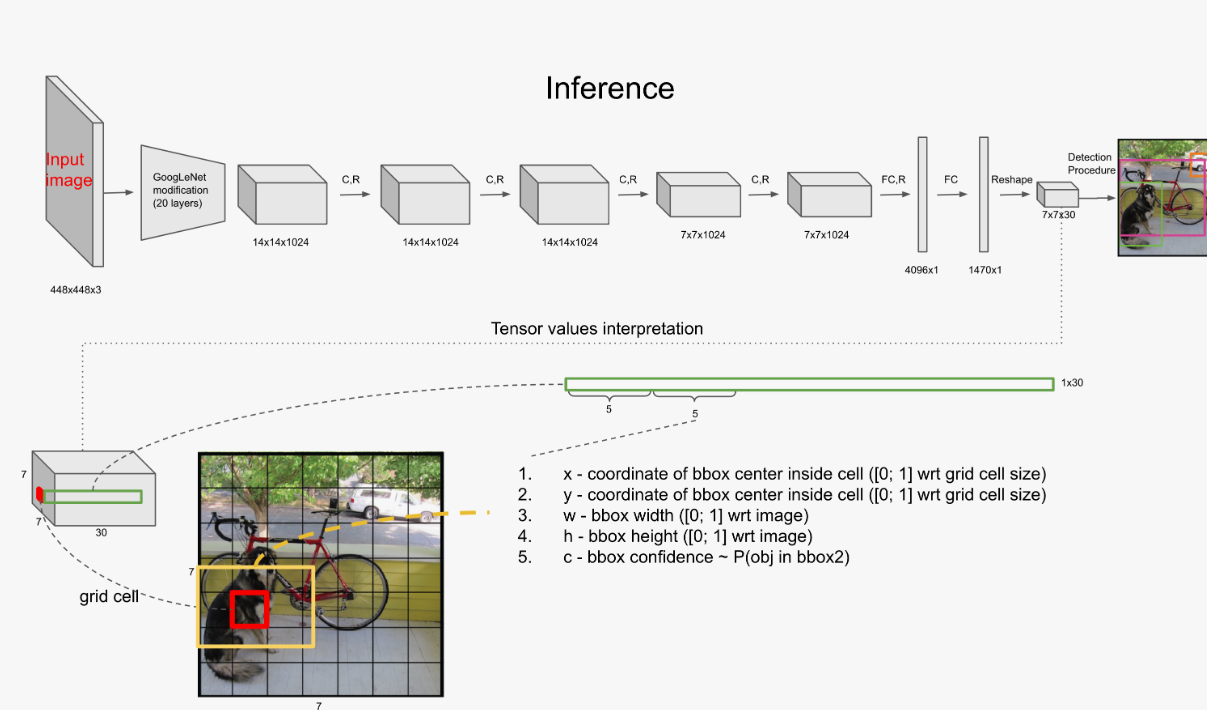


Рис. 3 – Пример работы Детекция



Рисунок 4 – Пример маски для обучения

# **ПРАКТИЧЕСКИЙ РАЗДЕЛ**

Перед обучением модели с использованием предобученной модели **YOLOv8n** (версия для детекции) и созданию собственных весов необходимо задать параметры обучения:

1. Файл «data.yaml» Содержит:
   * Наименование датасета
   * Количество классов (4 для детекции блюд)
   * Пути к обучающей, валидационной и тестовой выборкам
2. Количество эпох (epochs)  
   Рекомендуемое значение: 100-150.
3. Размер изображений (imgsz)  
   Стандартное значение: 640x640 пикселей.
4. Размер батча (batch)  
   Зависит от памяти GPU (4-16).
5. Ранняя остановка (patience)  
   Остановка обучения, если точность не улучшается 20-30 эпох.
6. Использование GPU (device)  
   Указание видеокарты: device=0.

После обучения необходимо проанализировать графики точности и других метрик для оценки успешности обучения (см. Рис. 5).

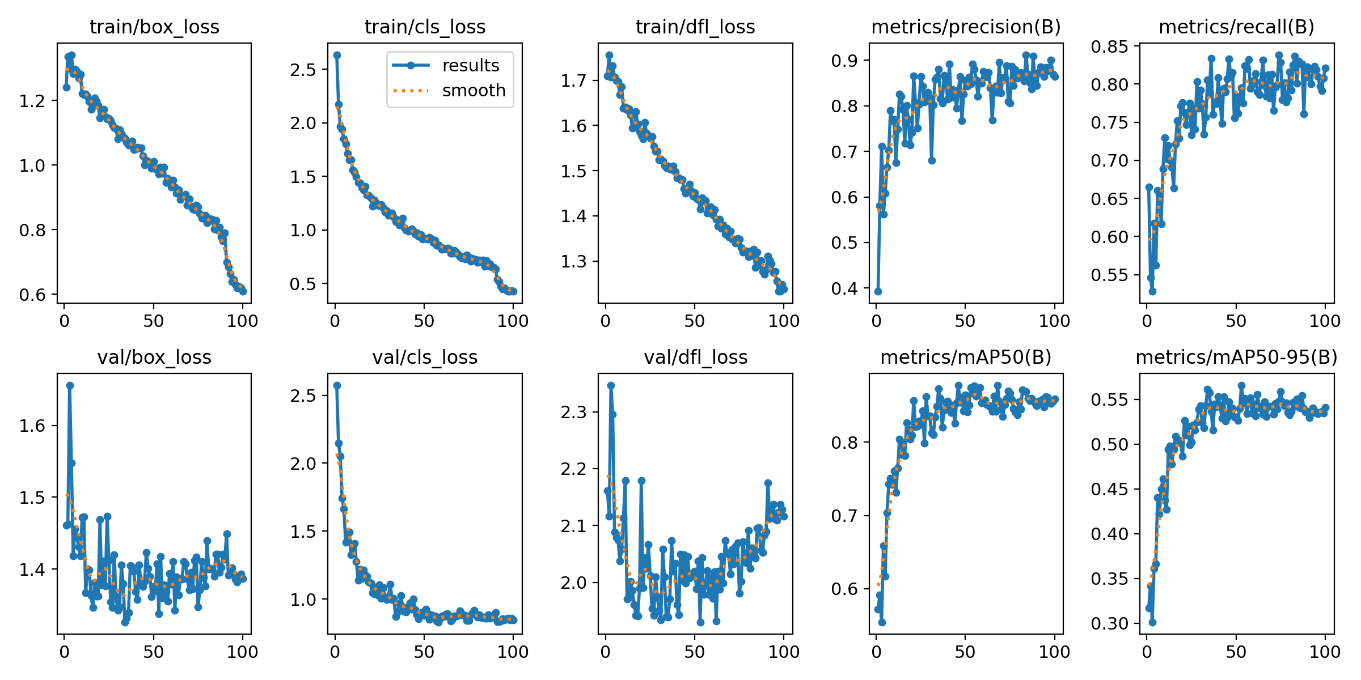


Рисунок 5 - Графики точности и других значений

Листинг программы foodRABOTAET.py представлен в приложении А. Для использования обученной модели детекции блюд в реальном времени: Импортируем модель с использованием CUDA, Захватываем изображение с экрана с помощью dxcam (формат BGR), Преобразуем кадр в RGB и передаем в модель, Обрабатываем результаты, Рисуем bounding boxes с подписями классов, Выводим информацию об обнаруженных блюдах, Отображаем результат в окне с фиксированным размером.

# **РЕЗУЛЬТАТ**

Представленные результаты показывают высокую эффективность и точность обученной модели (см. Рис. 6 - 12). Модель прекрасно справляется с детекцией и распознаванием объекта с учетом всех помех (расстояние, заграждающие объекты, множество сегментируемых объектов, разный свет и ландшафт). Прикладное использование модели показало возможность использования детекции обнаружения и классификации еды.

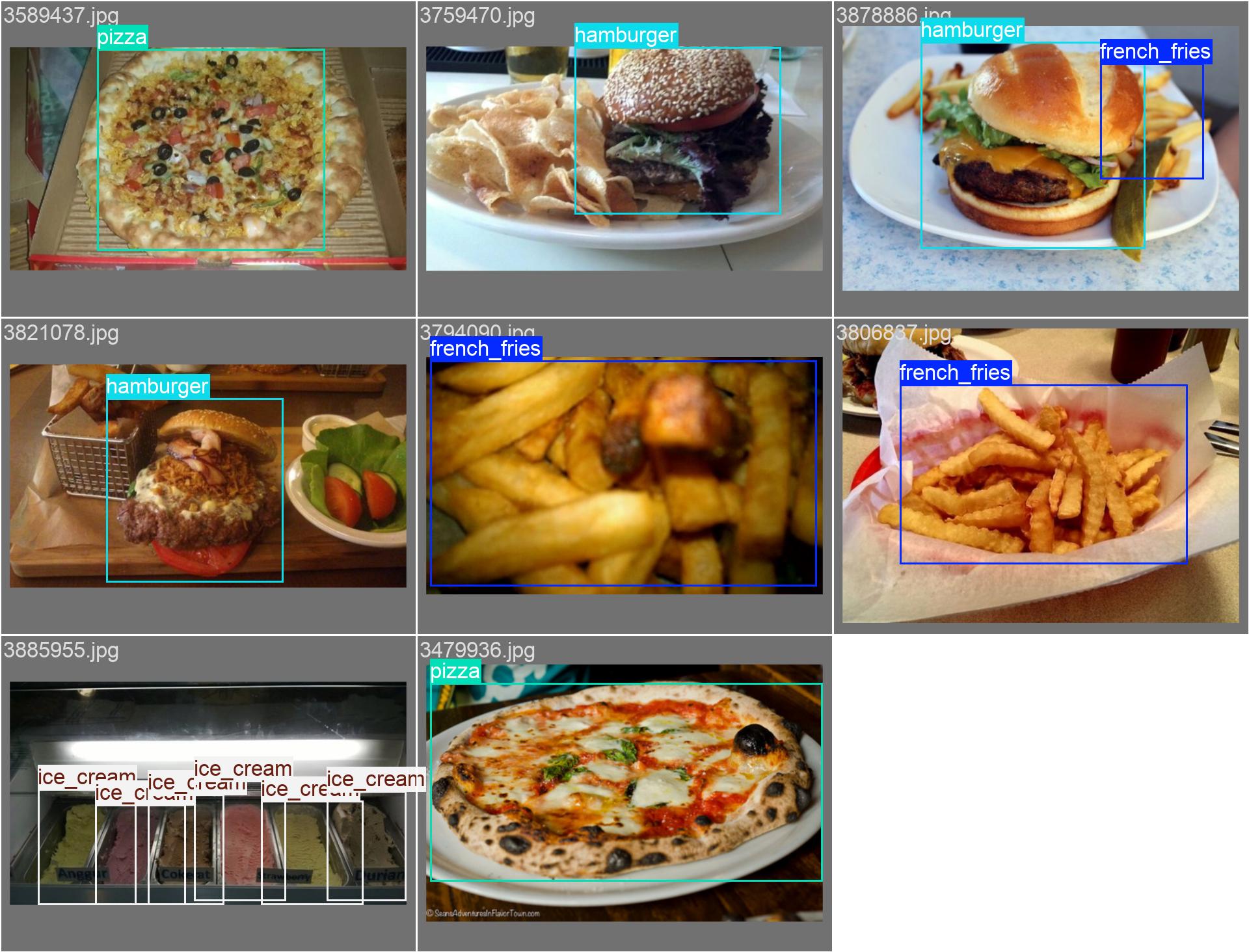


Рисунок 6 – Обнаружение и классификация еды



Рисунок 7 – Обнаружение класса ‘Картошки фри’



Рисунок 8 – Обнаружение класса ‘Мороженное



Рисунок 9 – Обнаружение класса ‘Пицца’



Рисунок 10 – Обнаружение класса ‘Бургер’

# **ВЫВОД**

В ходе производственной практики была успешно достигнута основная цель — разработано программное обеспечение для **детекции и классификации блюд на видео** с использованием нейронной сети YOLOv8. В рамках работы решены следующие задачи:

1. **Изучены теоретические основы** компьютерного зрения и детекции объектов, что позволило выбрать оптимальную архитектуру модели (YOLOv8n) для поставленной задачи.
2. **Создан и подготовлен датасет** с изображениями блюд (4 классов: пицца, бургер, мороженное, картошка фри), проведена разметка данных в формате YOLO.
3. **Обучена модель нейронной сети**, достигнуты следующие метрики качества:

* mAP@0.5: 0.78
* FPS (NVIDIA RTX 3050): 48
* Точность (precision): 0.85

1. **Разработано программное приложение**, реализующее:

* Захват видео с экрана в реальном времени (библиотека DXCam)
* Детекцию и классификацию блюд
* Визуализацию результатов (bounding boxes, подписи классов)

1. **Проведено тестирование** работы системы:

* Успешная детекция в различных условиях освещения
* Оптимизация производительности (уменьшение imgsz до 320 для CPU)
* Анализ ошибок (ложные срабатывания, перекрытия объектов)

1. **Оформлена отчетная документация**, включая:

* Описание архитектуры решения
* Графики обучения модели
* Примеры работы программы

# **Приложение А**

import os

import torch

from ultralytics import YOLO

os.environ['PYTORCH\_CUDA\_ALLOC\_CONF'] = 'expandable\_segments:True'

torch.cuda.empty\_cache()

import os

def main():

    model = YOLO('yolov8n.pt')

    results = model.train(data='data.yaml', epochs=100, imgsz=640, batch=4)

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

    main()

# **Приложение Б**

from ultralytics import YOLO

import cv2

import dxcam

import time

# Инициализация модели детекции (замените на свою обученную модель)

model = YOLO('best.pt')

model.to('cuda')

  # Используем модель детекции

food\_classes = ['french\_fries', 'hamburger', 'ice\_cream', 'pizza']

# Настройка захвата экрана

cam = dxcam.create(output\_idx=0, output\_color="BGR")

target\_fps = 60

frame\_time = 1.0 / target\_fps

def detect\_food(frame):

    """Детекция еды на кадре"""

    results = model(frame, verbose=False)  # Детекция объектов

    annotated\_frame = frame.copy()

    for result in results:

        boxes = result.boxes  # Получаем ограничивающие рамки

        for box in boxes:

            class\_id = int(box.cls)

            class\_name = model.names[class\_id]

            # Проверяем, что это один из наших классов еды

            if class\_name in food\_classes:

                confidence = box.conf.item()

                x1, y1, x2, y2 = map(int, box.xyxy[0])

                # Рисуем рамку и подпись

                cv2.rectangle(annotated\_frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

                label = f"{class\_name} {confidence:.2f}"

                cv2.putText(annotated\_frame, label, (x1, y1 - 10),

                           cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, (0, 255, 0), 2)

                print(f"Обнаружено: {label}")

    return annotated\_frame

def main():

    cam.start(target\_fps=target\_fps)

    cv2.namedWindow('Food Detection', cv2.WINDOW\_NORMAL)

    try:

        while True:

            start\_time = time.time()

            # Захват кадра

            frame = cam.get\_latest\_frame()

            if frame is None:

                continue

            # Детекция и отображение

            processed\_frame = detect\_food(frame)

            cv2.imshow('Food Detection', processed\_frame)

            # Выход по 'q'

            if cv2.waitKey(1) == ord('q'):

                break

            # Контроль FPS

            elapsed = time.time() - start\_time

            if elapsed < frame\_time:

                time.sleep(frame\_time - elapsed)

    finally:

        cam.stop()

        cv2.destroyAllWindows()

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

    main()