МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ 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		С.А. Рогачев
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЁТ ПО ПРАКТИКЕ

вид практики производственная

тип практики технологическая (проектно-технологическая)

на тему индивидуального задания Программная реализация алгоритма сегментации изображения с помощью нейронной сети

выполнен Садриевым Амиром Радимовичем
фамилия, имя, отчество обучающегося в творительном падеже

по направлению подготовки 09.03.04 (код) Программная инженерия (наименование направления)

наименование направления

направленности 02 (код) Проектирование программных систем (наименование направленности)

наименование направленности

Обучающийся группы № 4233К (номер) _____ (подпись, дата) А.Р. Садриев (инициалы, фамилия)

Санкт–Петербург 2024

ОГЛАВЛЕНИЕ

ЦЕЛЬ РАБОТЫ

Целью работы производственной практики является разработка программного обеспечения для сегментации изображений с использованием нейронной сети. В рамках данной цели предполагается выполнение следующих задач:

1. Изучение теоретических основ сегментации изображений и нейронных сетей, включая основные алгоритмы и подходы.

2. Исследование и выбор подходящей архитектуры нейронной сети для задачи сегментации изображений.

3. Разработка и обучение модели нейронной сети на основе выбранной архитектуры с использованием соответствующих наборов данных.

4. Создание программного приложения, реализующего процесс сегментации изображений с применением разработанной модели нейронной сети.

5. Проведение тестирования и оценки качества работы разработанного программного обеспечения, а также оптимизация модели для достижения наилучших результатов.

6. Оформление отчетной документации по результатам проведенной работы.

Выполнение данной работы позволит получить практические навыки в области разработки нейронных сетей и обработки изображений, а также углубить знания в области машинного обучения и искусственного интеллекта.

ИСХОДНЫЕ ДАННЫЕ И ИХ ОПИСАНИЕ

Для разработки и тестирования алгоритма сегментации изображений с использованием нейронной сети были использованы следующие исходные данные:

1. Набор данных изображений

Основным источником данных для обучения и тестирования модели сегментации является набор изображений, специально подготовленный для задачи сегментации. Данные изображения представляют собой различные сцены и объекты, разделенные на обучающую, валидационную и тестовую выборки (см. Рис. 1 - 2). Каждый набор данных содержит следующие компоненты:

Оригинальные изображения: Входные изображения в формате JPG с разрешением 1920 на 1080, которые необходимо сегментировать.

Маски сегментации: Размеченные изображения, где каждый пиксель имеет метку, указывающую на принадлежность к определенному классу. В последствии координаты масок и нумерация классов преобразуется в метаданные, формат записи которых индивидуален для каждой модели. Эти маски служат эталоном для обучения модели.

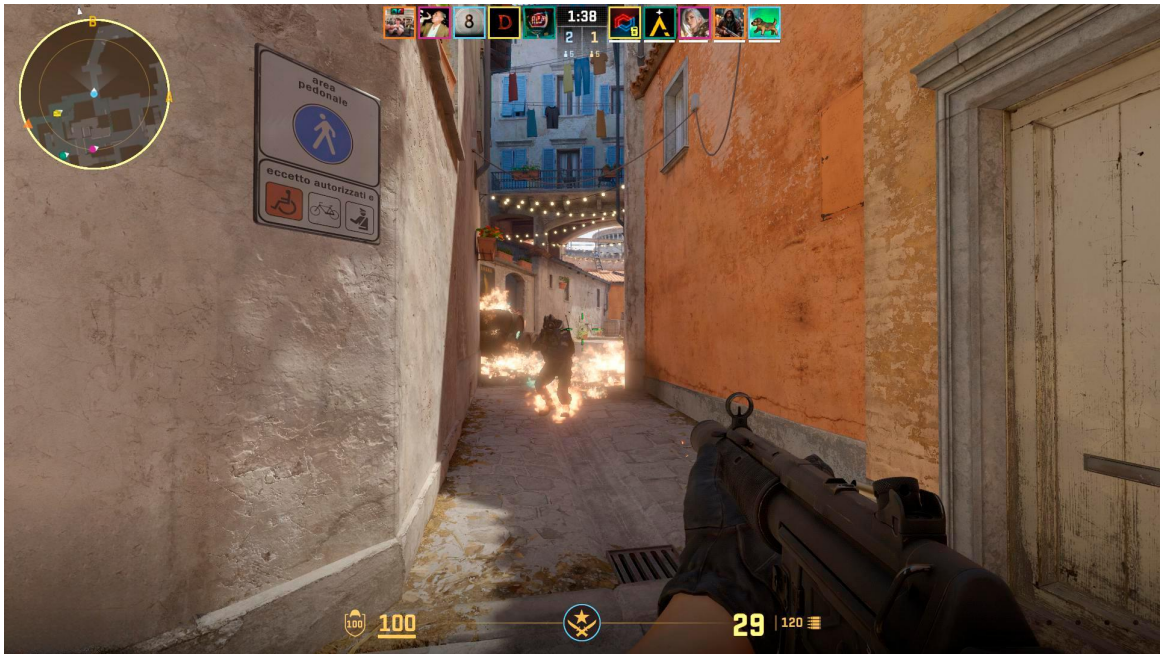

Рисунок 1 - Исходное изображение

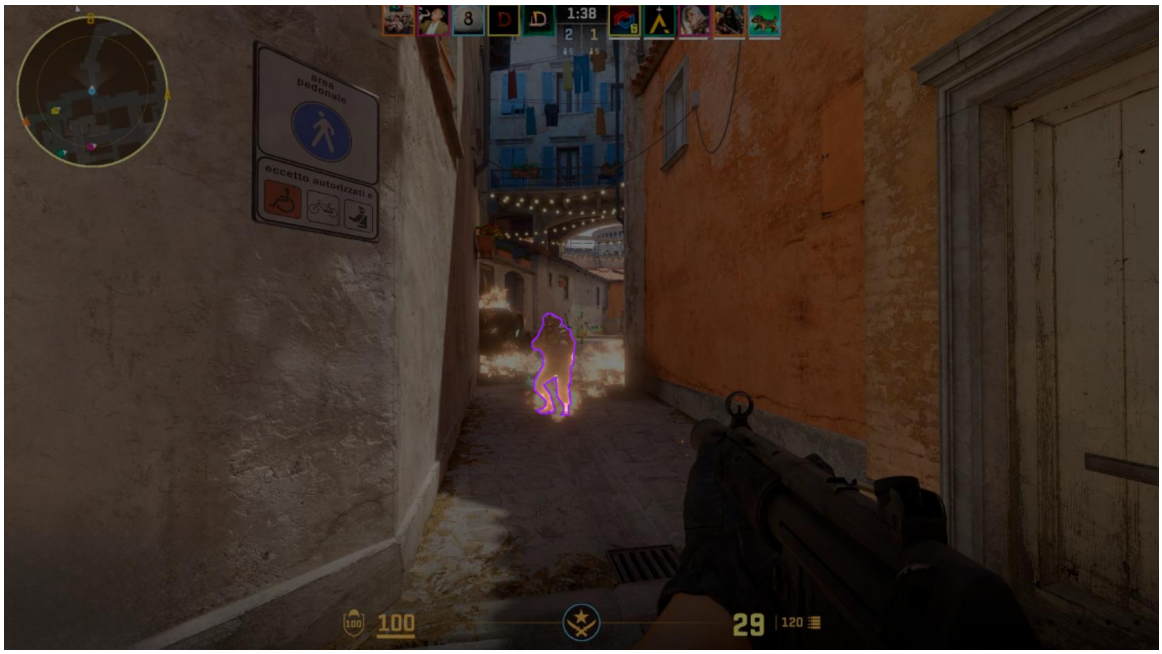

Рисунок 2 - Сегментационная маска на объекте

2. Аннотации и метаданные

Для каждого изображения из набора данных предоставлены аннотации и метаданные, включающие:

2.1 Категории объектов: Список классов, для которых производится сегментация (например, фон, объект 1, объект 2 и т.д.).

name_classes:

- alive-bandit

- alive-police

- dead-police

- died-bandit

num_classes: 4

2.2 Размеры изображений: Информация о размерах (ширина и высота) каждого изображения.

model.train(<…> imgsz=[1080, 1920] <…>)

2.3 Формат данных: Формат, в котором хранятся изображения и маски (например, PNG, JPG, TXT, JSON).

1 0.4886447594314217 0.6334907633369725 0.4886447594314217 0.6196018744480837 <…> 0.4827853844314217 0.6334907633369725

Где цифра «1» отвечает за класс, а последующие попарно отвечают за позицию пикселя по X и Y соответственно. На каждую такую маску выделяется одна строка.

3. Дополнительные данные

Для улучшения качества сегментации и устойчивости модели были использованы дополнительные данные:

3.1 Дополнительные наборы данных: Использовались дополнительные наборы данных для увеличения разнообразия обучающих данных, снижения переобучения модели и увеличения производительности.

def main(): model = YOLO('yolov8s-seg.pt')

3.2 Аугментация данных: Применение техник аугментации данных, таких как вращение, масштабирование, сдвиг, изменение яркости и контрастности, чтобы повысить устойчивость модели к различным типам искажения изображений (см. Рис. 3).

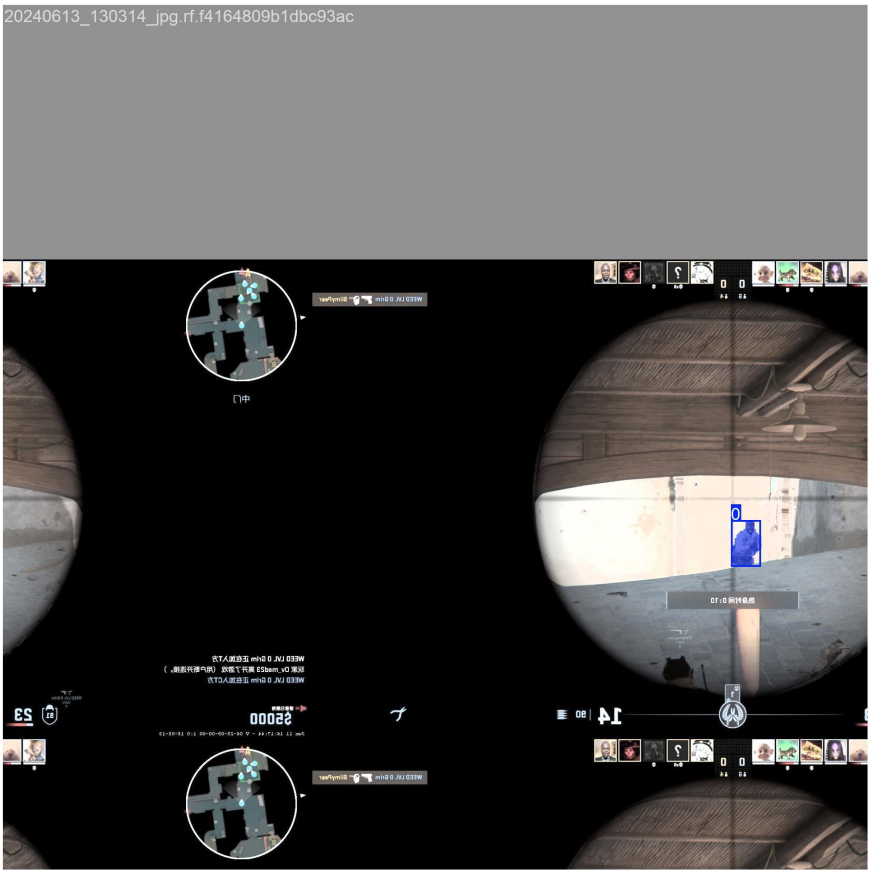

Рисунок 3 - Пример искажённого изображения

ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

В программе обучения модели для сегментации изображений использовалась предобученная модель YOLOv8s-seg. Данная модель является сверточной нейронной сетью.

Сверточная нейронная сеть (CNN) представляет собой совокупность сверточных, подвыборочных и полносвязных слоев. Эта архитектура обеспечивает наивысшую скорость и точность предсказания объектов на изображении благодаря ряду факторов, таких как текстура, освещение, детали, дефекты и неполнота данных. В отличие от других типов нейронных сетей, которые часто сталкиваются с трудностями в обработке этих факторов, сверточные нейронные сети успешно справляются с ними, что делает их идеальными для задач, связанных с анализом изображений.

Сверточный слой является фундаментом для машинного зрения. Каждое черно-белое изображение можно представить в градации серого от 0 до 255 (черный и белый соответственно). Аналогичный метод применятся для цветных изображений, но перед разбиением на градации изображение делиться на 3 канала в диапазоне RGB, где каждый пиксель имеет свою собственную градацию в зависимости от канала (см. Рис. 4).

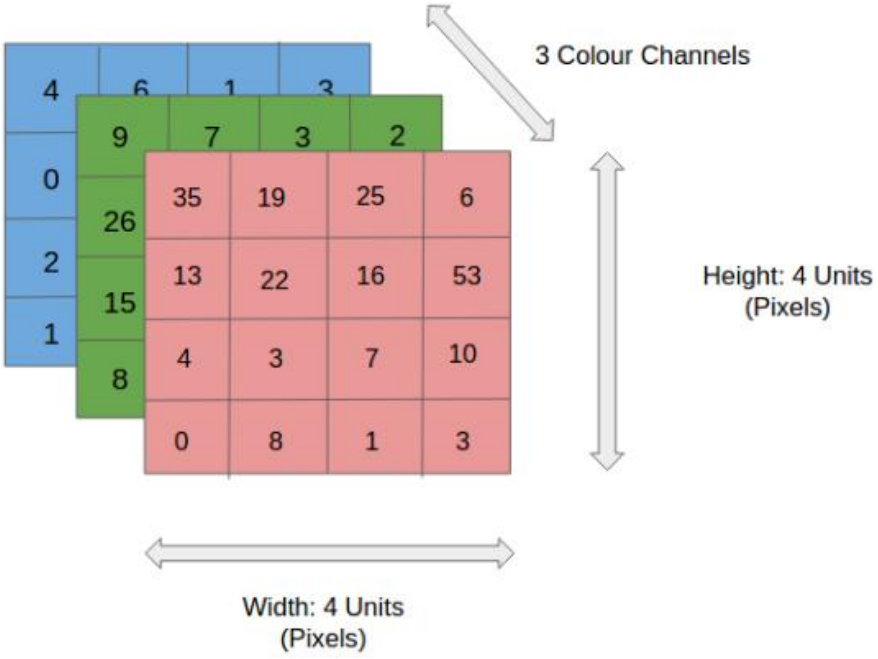

Рисунок 4 - RGB-изображение

Данные изображения представляются в виде матрицы имеющую глубину, что называется тензором.

Для прочтения тензора и получения свернутого признака используется фильтр, который перемещается по всему изображению с шагом K (см. Рис. 5). Фильтр представляет собой небольшую матрицу весов, которая умножается на соответствующие области входного изображения. Этот процесс называется сверткой. Конкретнее, каждый элемент фильтра умножается на соответствующий элемент области изображения, после чего результаты этих умножений суммируются.

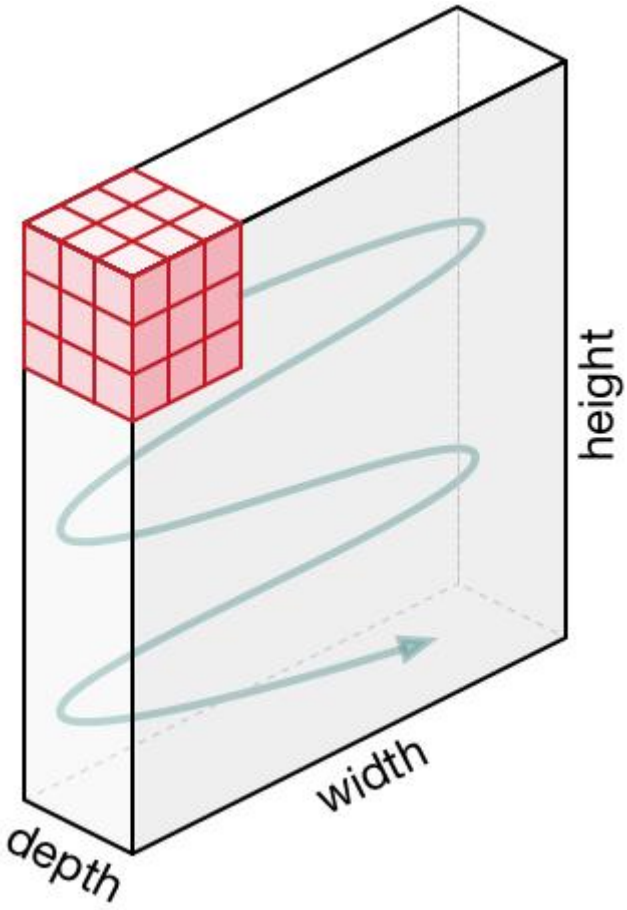

Рисунок 5 - Перемещение фильтра по тензору

Instance segmentation (сегментация экземпляров) представляет собой метод, целью которого является одновременное обнаружение объектов и семантической сегментации. В отличие от семантической сегментации, которая присваивает каждому пикселю изображения класс, instance

segmentation не только классифицирует пиксели, но и различает отдельные экземпляры объектов внутри одного класса.

Основные аспекты Instance Segmentation:

1. Первая стадия заключается в определении границ каждого объекта на изображении. Эта задача решается с помощью методов, таких как Region Proposal Networks (RPN) или Single Shot MultiBox Detector (SSD), которые предлагают потенциальные области, содержащие объекты.

2. На второй стадии каждый пиксель внутри предложенных областей классифицируется по классу. Этот процесс аналогичен семантической сегментации, однако выполняется только в пределах областей, предложенных на первом этапе.

3. Заключительная стадия объединяет результаты обнаружения объектов и семантической сегментации для создания масок, которые различают отдельные экземпляры объектов, принадлежащих к одному классу. Это позволяет сегментировать каждый объект отдельно, даже если они принадлежат к одному классу.

ПРАКТИЧЕСКИЙ РАЗДЕЛ

Программа обучения

Перед обучением модели с использованием предобученной модели YOLOv8s-seg и созданию собственных весов необходимо задать параметры обучения.

Первый параметр: файл «data.yaml» представляет собой файл с наименованием, количеством классов и пути к датасету.

Второй параметр: указываем количество эпох.

Третий параметр: указываем размер изображений датасета.

Четвертый параметр: указываем размер батчфайла.

Пятый параметр: указываем количество эффективных эпох (если в течение n эпох не улучшается точность - остановить обучение).

Шестой параметр: размещаем обработку тензоров на видеокарте.

После обучения модели необходимо проанализировать графики точности и других значений и выявить успешное обучение (см. Рис. 6).

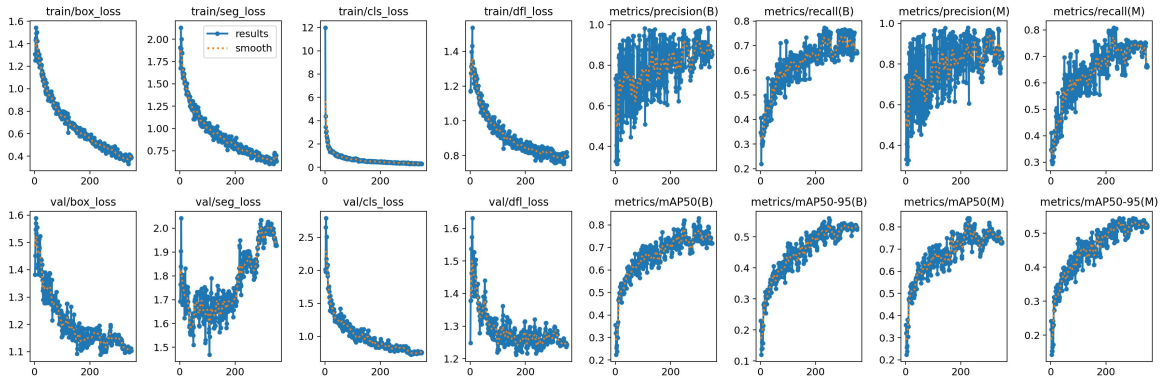

Рисунок 6 - Графики точности и других значений

Листинг программы trainRABOTAET.py представлен в приложении А.

Программа сегментации изображения с экрана

Для использования обученной модели сегментации экземпляров изображения и практическое применение в автонаведении центра экрана на ближайший box объекта, необходимо действовать пошагово:

1. Импортируем модель с использованием «cuda».

2. Определяем для каждого класса свой цвет.

3. Захватываем изображение с главного экрана с помощью «dxcam»,

используя цветовое пространство «BGR2RGB».

4. Представляем захваченное изображение в виде тензора, где каждый пиксель имеет 8-ми битовое представление.

5. Получаем результаты сегментации с пороговой уверенностью не ниже 70%.

6. Аннотируем и обрабатываем результаты

6.1 Создаем копию изображения в формате uint8 и инициализируем аннотатор для добавления меток и прямоугольников.

6.2 Инициализируем переменные для нахождения ближайшего объекта к центру экрана.

6.3 Проходим по каждому результату и проверяем наличие прямоугольников (боксов).

6.4 Проходим по каждому прямоугольнику в результатах и выполняем следующие действия:

Определяем координаты и класс прямоугольника.

Если класс объекта в ['alive-bandit', 'alive-police'], аннотируем изображение.

Вычисляем центр прямоугольника и расстояние до центра изображения.

Находим ближайший объект к центру экрана.

Если имеются маски, накладываем их на изображение.

7. Возвращаем аннотированное изображение и координаты ближайшего объекта.

8. С указанными параметрами и полученными координатами до ближайшего бокса отправляем пакет для движения мыши, пока ближайший бокс не достигнет центра экрана.

Листинг программы aimRABOTAET.py представлен в приложении Б.

РЕЗУЛЬТАТ

Представленные результаты показывают высокую эффективность и точность обученной модели (см. Рис. 7 - 16).

Модель прекрасно справляется с сегментированием и распознаванием объекта с учетом всех помех (расстояние, заграждающие объекты, множество сегментируемых объектов, разный свет и ландшафт).

Прикладное использование модели показало возможность использования сегментирования изображения для автонаводки на определённые классы и игнорирование не нужных классов.

Рисунок 7 - Исходное изображение «спецназовца»

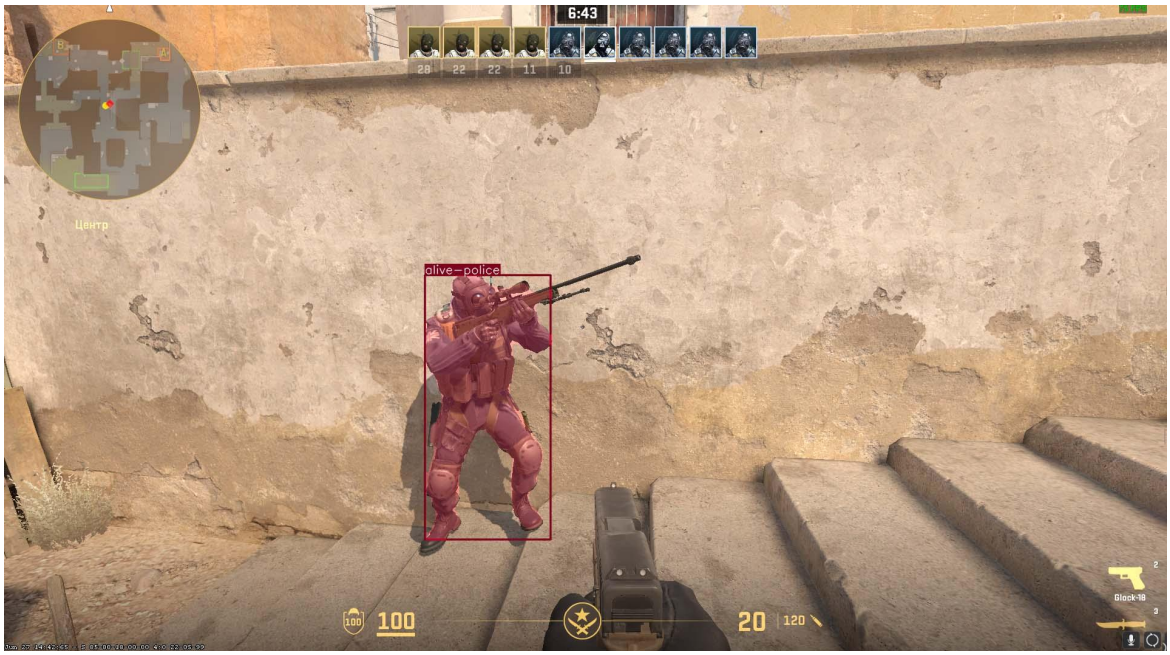

Рисунок 8 - Сегментированное изображение «спецназовца»

Рисунок 9 - Исходное изображение части «бандита»

Рисунок 10 - Сегментированное изображение части «бандита»

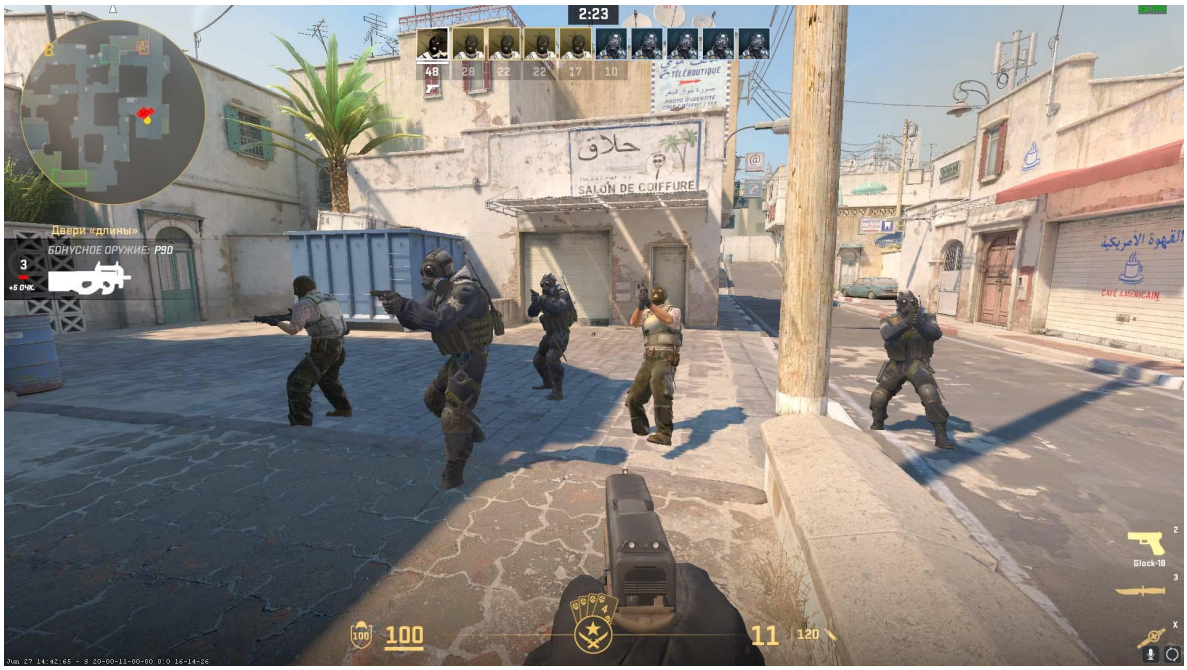

Рисунок 11 - Исходное изображение игровых NPC

Рисунок 12 - Сегментированное изображение игровых NPC

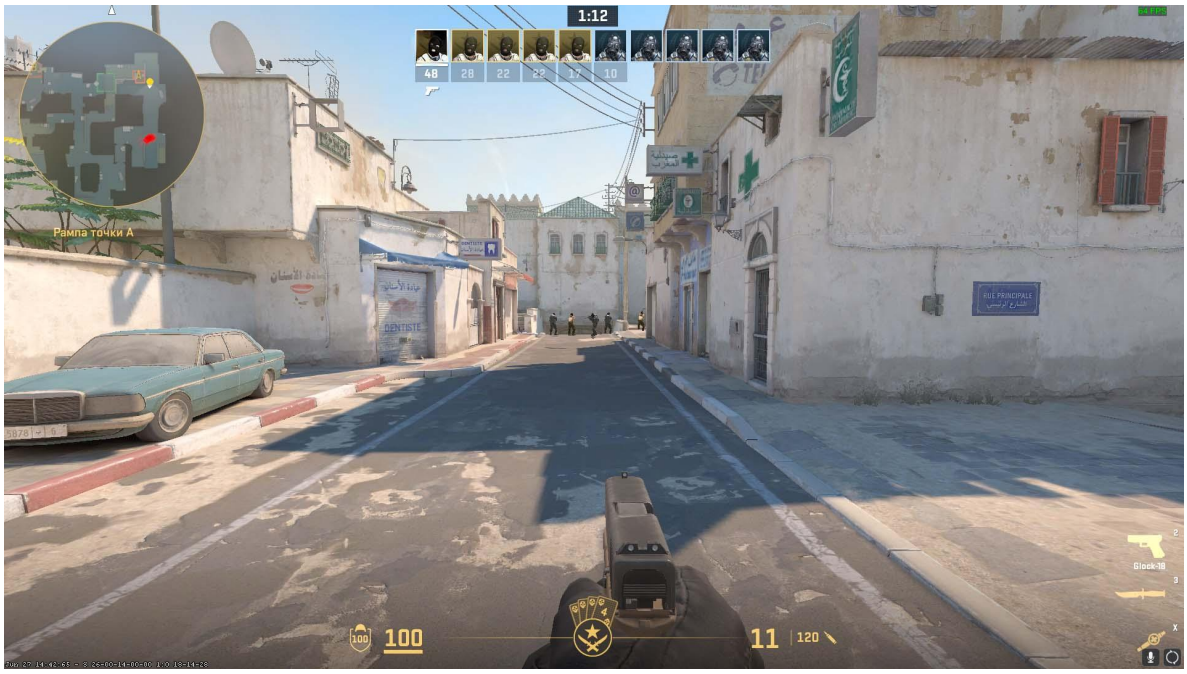

Рисунок 13 - Исходное отдалённое изображение игровых NPC

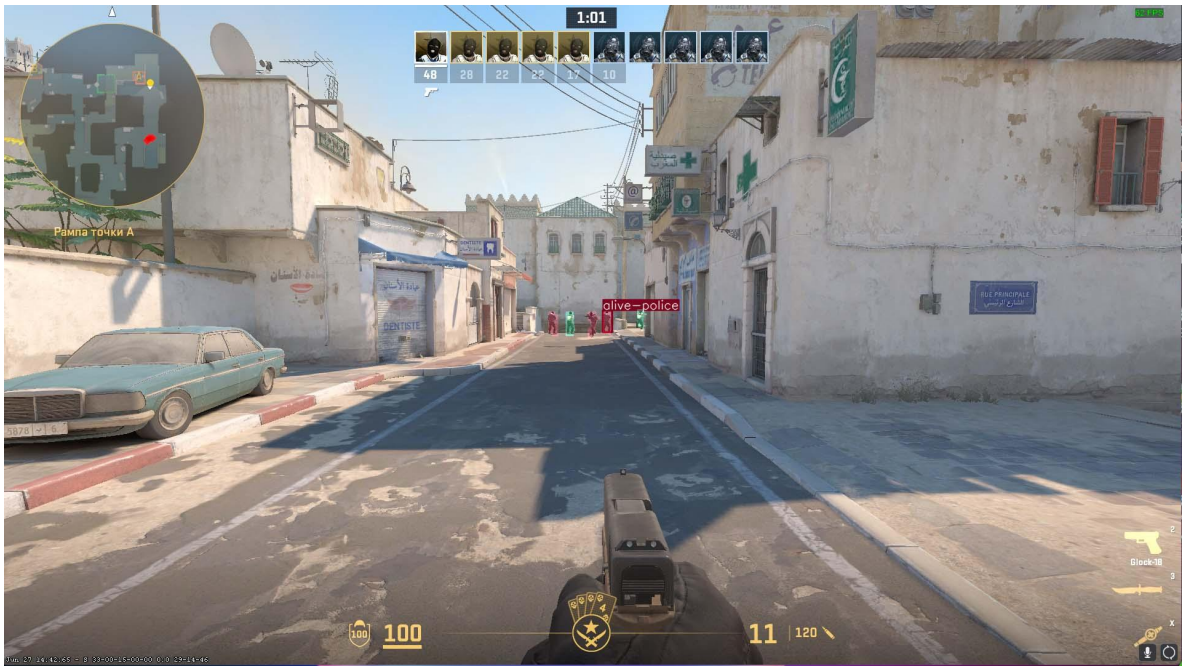

Рисунок 14 - Сегментированное отдалённое изображение игровых NPC

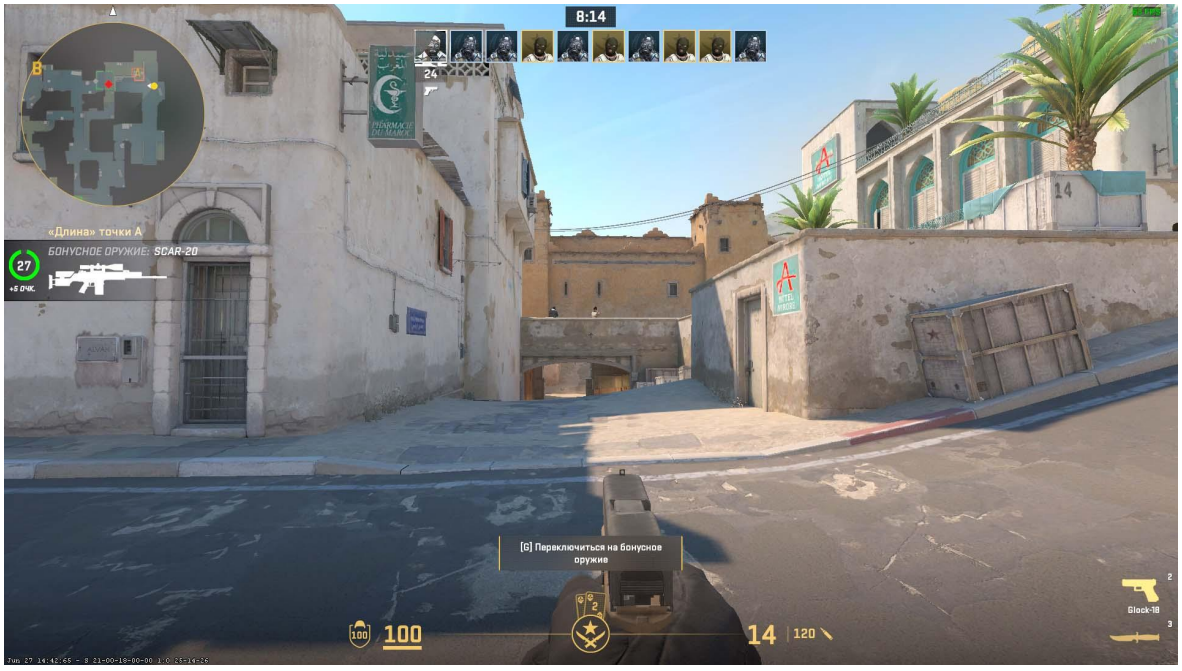

Рисунок 15 - Исходное изображение части игровых NPC

Рисунок 16 - Сегментированное изображение части игровых NPC

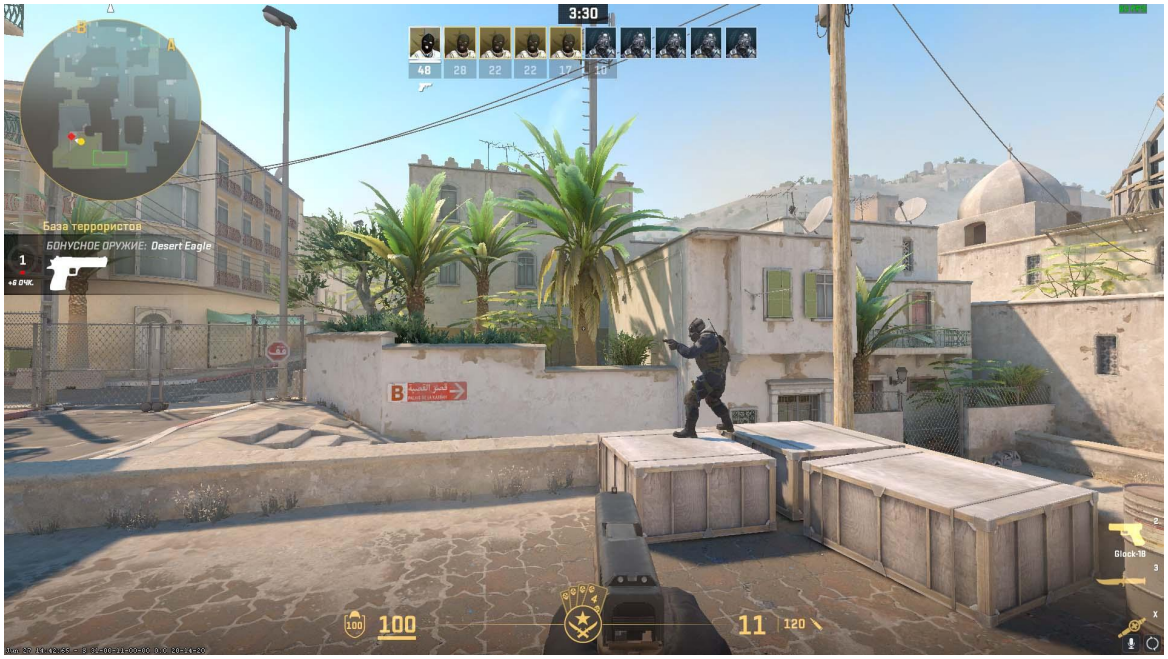

Рисунок 17 - Демонстрация автонаводки: прицел отклонён

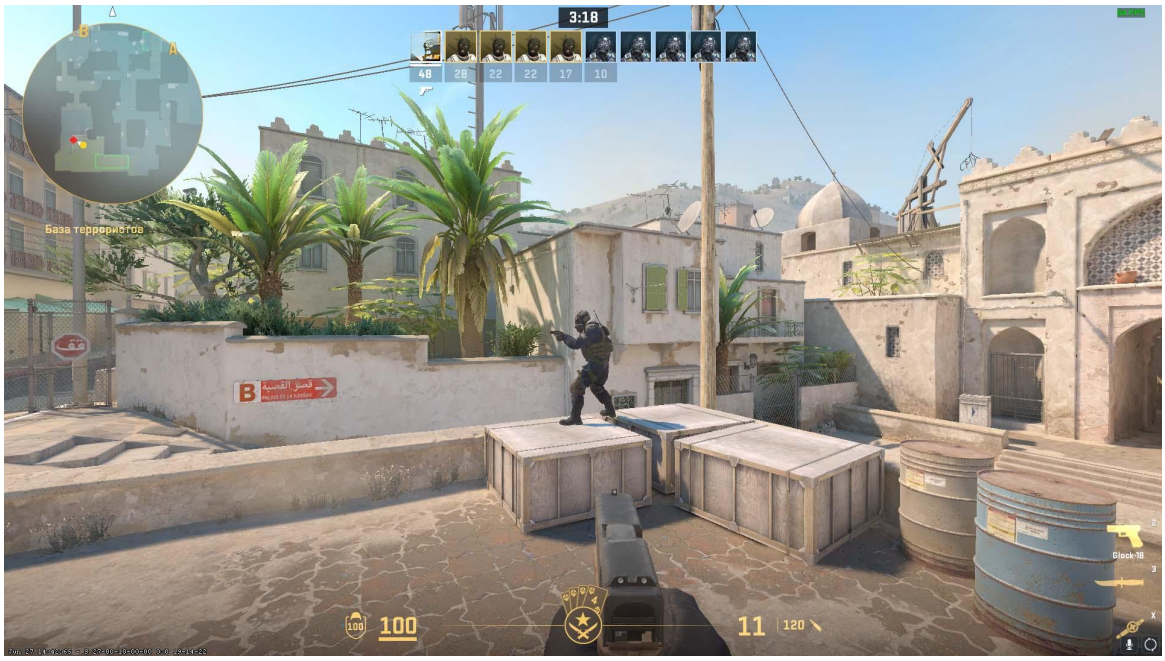

Рисунок 18 - Демонстрация автонаводки: прицел наведён

ВЫВОД

В ходе производственной практики была успешно достигнута основная цель – разработано программное обеспечение для сегментации изображений с использованием нейронной сети. В рамках выполнения данной работы были решены следующие задачи:

1. Изучены теоретические основы сегментации изображений и применения нейронных сетей для этой задачи, что позволило выбрать оптимальную архитектуру модели.

2. Разработана и обучена модель нейронной сети на соответствующем наборе данных, продемонстрировавшая высокое качество сегментации.

3. Создано программное приложение, реализующее процесс сегментации изображений с использованием обученной модели.

4. Проведены тестирование и оценка качества работы программного обеспечения, что позволило выявить и устранить возможные недостатки, а также оптимизировать модель.

5. Подготовлена и оформлена отчётная документация по результатам проделанной работы.

Практическая работа позволила приобрести ценные навыки в области разработки и применения нейронных сетей для обработки изображений.

Результаты сегментирования и автонаводки в реальном времени на примере Counter-Strike 2 показало высокую отзывчивость и точность модели, но использование данного метода в игре защищённым авторским правом и правилами сообщества наказуемо.

Машинное обучение способно автоматизировать и ускорить рутинные процессы, таких как: пилотирование летательных аппаратов, наземных транспортов, обнаружение техногенных и природных угроз с помощью спутниковых снимков.

СПИСОК ЛИТЕРАТУРЫ

1. YouTube: Обучение YOLOv8 для задачи инстанс сегментации (YOLOv8-seg) – URL: https://www.youtube.com/watch?v=FF3mIWF0vFs (дата обращения: 25.06.2024).

2. YOLOv8 – URL: https://docs.ultralytics.com/models/yolov8/ (дата обращения: 26.06.2024).

3. Configuration YOLOv8 – URL: https://docs.ultralytics.com/usage/cfg/ (дата обращения: 26.06.2024)

4. NovaUm.Ru: ИСПОЛЬЗОВАНИЕ НЕЙРОННОЙ СЕТИ YOLOV8 ДЛЯ ДЕТЕКТИРОВАНИЯ НА ИЗОБРАЖЕНИИ ЛАБОРАТОРНОГО ГРЫЗУНА (КРЫСЫ ИЛИ МЫШИ) В УСТАНОВКЕ «КВАДРАТНОЕ ОТКРЫТОЕ ПОЛЕ» – URL: http://novaum.ru/public/p2690?ysclid=ly3napbwln628855210 (дата обращения: 28.06.2024)

5. Хабр: Наглядно о том, как работает свёрточная нейронная сеть – URL: https://habr.com/ru/companies/skillfactory/articles/565232/ (дата обращения: 27.06.2024)

ПРИЛОЖЕНИЕ А

```
from ultralytics import YOLO

def main():
    model = YOLO('yolov8s-seg.pt')
    results = model.train(data='data.yaml', epochs=350, imgsz=[1080, 1920], batch=-1, patience=200, device=0)

if __name__ == '__main__':
    main()
```

ПРИЛОЖЕНИЕ Б

```
from ultralytics import YOLO
import cv2
import numpy as np
import dxcam
from pynput import keyboard
import ctypes
import time
from ultralytics.utils.plotting import Annotator
import threading


model = YOLO('best.pt')
model.to('cuda')

num_classes = len(model.names)
colors = [tuple(np.random.randint(0, 255, 3).tolist()) for _ in range(num_classes)]

cam = dxcam.create()
target_fps = 60
frame_time = 1.0 / target_fps

SendInput = ctypes.windll.user32.SendInput
PUL = ctypes.POINTER(ctypes.c_ulong)
class MouseInput(ctypes.Structure):
    _fields_ = [("dx", ctypes.c_long),
                ("dy", ctypes.c_long),
                ("mouseData", ctypes.c_ulong),
                ("dwFlags", ctypes.c_ulong),
```

```
            ("time", ctypes.c_ulong),
            ("dwExtraInfo", PUL)]

class Input_I(ctypes.Union):
    _fields_ = [("mi", MouseInput)]

class Input(ctypes.Structure):
    _fields_ = [("type", ctypes.c_ulong),
            ("ii", Input_I)]

def move_mouse(x, y, speed=1, steps=5):
    x = int(x * speed)
    y = int(y * speed)
    dx = x // steps
    dy = y // steps
    extra = ctypes.c_ulong(0)

    for i in range(steps):
        ii_ = Input_I()
        ii_.mi = MouseInput(dx, dy, 0, 0x0001, 0, ctypes.pointer(extra))
        command = Input(ctypes.c_ulong(0), ii_)
        SendInput(1, ctypes.pointer(command), ctypes.sizeof(command))
        time.sleep(0.005)

auto_aim_enabled = False

def toggle_auto_aim():
    global auto_aim_enabled
    auto_aim_enabled = not auto_aim_enabled
    status = "включена" if auto_aim_enabled else "выключена"
```

```
    print(f"Автонаводка {status}.")

def on_press(key):
    if key == keyboard.Key.alt_l or key == keyboard.Key.alt_gr:
        toggle_auto_aim()

listener = keyboard.Listener(on_press=on_press)
listener.start()

def capture_screen():
    frame = cam.get_latest_frame()
    if frame is not None:
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        return frame

def segment_image(frame):
    results = model.predict(frame, conf=0.70, verbose=False)
    frame_uint8 = np.ascontiguousarray(frame, dtype=np.uint8)
    annotator = Annotator(frame_uint8, line_width=2)

    closest_box_center = None
    min_distance = float('inf')

    for result in results:
        if hasattr(result, 'boxes') and len(result.boxes.xyxy) > 0:
            boxes = result.boxes
            masks = result.masks if hasattr(result, 'masks') else None

            for i, box in enumerate(boxes):
                b = box.xyxy[0]
```

```
                c = box.cls
                if model.names[int(c)] in ['alive-bandit', 'alive-police']:
                    color = colors[int(c)]
                    annotator.box_label(b, model.names[int(c)], color=color)



                    center_x = int((b[0] + b[2]) / 2)
                    center_y = int(b[1] + 0.1 * (b[3] - b[1]))
                    distance = np.sqrt((center_x - frame.shape[1] // 2) ** 2 +
(center_y - frame.shape[0] // 2) ** 2)
                    if distance < min_distance:
                        min_distance = distance
                        closest_box_center = (center_x, center_y)

                    if masks is not None:
                        mask = masks.data[i].cpu().numpy()
                        mask = cv2.resize(mask, (frame.shape[1], frame.shape[0]))
                        mask = mask > 0.5
                        colored_mask = np.zeros_like(frame, dtype=np.uint8)
                        colored_mask[mask] = color
                        frame_uint8 = cv2.addWeighted(frame_uint8, 1.0,
colored_mask, 0.5, 0)

            return frame_uint8, closest_box_center

        def main():
            cam.start(target_fps=60)
            cv2.namedWindow('Segmented Image', cv2.WINDOW_NORMAL)

            try:
```

```
        while True:
            start_time = time.time()

            screenshot = capture_screen()
            if screenshot is None:
                continue

            processed_image, closest_box_center = segment_image(screenshot)
            cv2.imshow('Segmented Image', processed_image)

            if auto_aim_enabled and closest_box_center:
                move_x = int(closest_box_center[0] - processed_image.shape[1] //
2)
                move_y = int(closest_box_center[1] - processed_image.shape[0] //
2)
                #move_mouse(move_x, move_y)
                t = threading.Thread(target=move_mouse, args=(move_x,
move_y))
                t.start()

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

            #  частота кадров
            elapsed_time = time.time() - start_time
            if elapsed_time < frame_time:
                time.sleep(frame_time - elapsed_time)

    finally:
        cam.stop()
```

```
        cv2.destroyAllWindows()

if __name__ == '__main__':
    main()
```