Министерство науки и высшего образования Российской Федерации   
Федеральное государственное автономное образовательное  
учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

**Отчёт о научно-исследовательской работе**

**бакалавра**

**Семестр 6**

Направление подготовки: 01.03.02 Прикладная математика и информатика:   
Профиль "Компьютерные науки"

Студент  *Фаюстова Алёна Петровна*

группы  *6303 – 010302D*

Тема научно-исследовательской работы: *Исследование методов обнаружения препятствий на основе машинного зрения для наземного робота*

Научный руководитель: *Минаев Евгений Юрьевич, кандидат технических наук, доцент кафедры суперкомпьютеров и общей информатики*

Научный руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись)*

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.

Студент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись)*

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.

**Самара 2025**

РЕФЕРАТ

**Отчет по научно-исследовательской работе:** 21 c., 8 рисунков, 2 таблицы, 5 источников, 1 приложение.

обнаружение препятствий, нейронные сети, бинарная сегментация, алгоритмы компьютерного зрения, DeepLab, ResNet, Расширенная свертка, остаточное обучение

Объектом исследования являются алгоритмы машинного зрения и их применение для автоматического обнаружения препятствий в реальном времени.

Цель работы заключается в исследовании методов бинарной сегментации на основе глубоких нейронных сетей для повышения точности обнаружения препятствий автономными наземными роботами.

Задачи, которые были решены в ходе исследования:

1. Подробный разбор архитектуры и принципов работы моделей DeepLabV3+ и ResNet-50, а также их совместной работы.
2. Проверка скорости и качества работы модели на видео и изображениях произвольного размера.
3. Сравнение результатов экспериментов и составление выводов.

СОДЕРЖАНИЕ

[Введение 4](#_Toc199431935)

[1 Теоретическая часть 5](#_Toc199431936)

[1.1 Бинарная сегментация с использованием DeeplabV3+ с основой ResNet-50 5](#_Toc199431937)

[1.2 Архитектура ResNet50: остаточные блоки и извлечение признаков 6](#_Toc199431938)

[1.3 DeepLabV3+: энкодер-декодер с расширенной сверткой 7](#_Toc199431939)

[2 Практическая часть 11](#_Toc199431940)

[2.1 Исследование производительности модели 11](#_Toc199431941)

[2.2 Работа с изображениями 128×128 12](#_Toc199431942)

[2.3 Исследование качества сегментации 13](#_Toc199431943)

[Заключение 16](#_Toc199431944)

[Список использованных источников 17](#_Toc199431945)

[Приложение А Фрагменты кода 18](#_Toc199431946)

[А.1 Обработка изображений 18](#_Toc199431947)

[А.2 Обработка видео 19](#_Toc199431948)

Введение

В последние годы наблюдается значительное развитие в области робототехники, особенно в контексте наземных мобильных роботов. Они находят широкое применение в различных сферах, таких как промышленность, сельское хозяйство, логистика и бытовое обслуживание. Проблема обнаружения препятствий является ключевым аспектом, влияющим на безопасность и эффективность их работы.

Актуальность данной проблемы обусловлена необходимостью обеспечения высокой точности и надежности систем обнаружения препятствий в условиях динамической и непредсказуемой среды. Современные роботы должны быть способны самостоятельно перемещаться, избегая столкновений с различными по типу и размеру препятствиями.

Машинное зрение, как одна из самых перспективных технологий для решения задачи обнаружения препятствий, предлагает уникальные преимущества по сравнению с традиционными методами, такими как инфракрасные и ультразвуковые датчики. Системы, основанные на данном методе, предоставляют богатую информацию об окружающей среде, имеют низкую стоимость, малый вес и короткое время обработки, что делает их особенно привлекательными для использования в автономных наземных роботах [1].

1. Теоретическая часть
   1. Бинарная сегментация с использованием DeeplabV3+ с основой ResNet-50

В прошлом семестре было начато исследование метода бинарной сегментации с обучением модели DeepLabV3+ с энкодером ResNet-50 с использованием фреймворка PyTorch. На данном этапе для исследования была использована модель из гитхаб-репозитория Mukund Kumar [2].

Архитектура модели (рисунок 1) включает следующие компоненты:

1. ResNet как основа (Backbone): ResNet — это семейство нейронных сетей, созданное для решения проблемы затухающих градиентов. Основное новшество — остаточные блоки, которые обеспечивают прямые соединения, стабилизируют обучение и позволяют строить более глубокие и сложные сети.
2. Энкодер: на основе начальных слоев ResNet извлекаются признаки изображения от низкоуровневых (края, текстуры) до сложных. Эти признаки передаются в ASPP для дальнейшей обработки.
3. ASPP (Atrous Spatial Pyramid Pooling): выполняет сверточные операции с разными коэффициентами расширения, собирая контекстную информацию на разных масштабах. Это помогает обрабатывать объекты разного размера и распределения.
4. Декодер: объединяет низкоуровневые признаки из энкодера с высокоуровневыми из ASPP через пропускные соединения. Это улучшает детализацию сегментации и восстанавливает пространственную информацию.
5. Squeeze & Excitation (SE)[3]: увеличивает репрезентативную силу сети, выделяя важные каналы и подавляя менее значимые. Среднее значение по каналам вычисляется глобально, а затем каналы адаптивно перекалибруются с помощью двух полносвязных слоев и функции активации ReLU. Результат — набор весов, который определяет, какие каналы усилить или ослабить.

Изображение выглядит как дизайн, искусство

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1 – Архитектура модели DeeplabV3+ используемой в репозитории [2]

* 1. Архитектура ResNet50: остаточные блоки и извлечение признаков

ResNet50 представляет собой глубокую нейронную сеть, построенную на концепции остаточного обучения (residual learning), разработанную командой исследователей Microsoft Research под руководством Kaiming He [4]. Ключевая особенность данной архитектуры заключается в переформулировке задачи обучения слоев: вместо попытки аппроксимировать желаемое базовое отображение напрямую, архитектура явно заставляет слои аппроксимировать остаточную функцию , при этом исходное отображение изменяется кая . Это изменение мотивировано явлениями проблемы деградации, когда более глубокие сети демонстрируют более высокую ошибку обучения по сравнению с их мелкими аналогами несмотря на то, что пространство решений более глубокой модели является надмножеством пространства решений мелкой модели. Формально, остаточный блок определяется как , где и представляют входные и выходные векторы признаков, означает совокупность весов слоев, а функция представляет остаточное отображение, которое подлежит обучению (рисунок 2). Когда размерности входа и выхода не совпадают, применяется линейная проекция для согласования размерностей: .

Изображение выглядит как текст, Шрифт, диаграмма, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2 – Блок остаточного обучения [4]

ResNet50 использует архитектуру "узкого места" для повышения вычислительной эффективности, где каждый остаточный блок состоит из трех сверточных слоев: 1×1, 3×3 и 1×1, при этом слои 1×1 отвечают за уменьшение и последующее восстановление размерности, оставляя слой 3×3 в качестве узкого места с меньшими входными и выходными размерностями (рисунок 3).

Изображение выглядит как текст, диаграмма, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3 – Блок "узкое место" для ResNet-50/101/152 [4]

* 1. DeepLabV3+: энкодер-декодер с расширенной сверткой

Данная модель объединяет преимущества двух основных подходов: модулей пространственного пирамидального пулинга и структур энкодер-декодер. Архитектура DeepLabV3+ расширяет предыдущую модель DeepLabv3, добавляя модуль декодера для восстановления границ объектов. Как показано на рисунке 4, энкодер кодирует многомасштабную контекстуальную информацию, применяя расширенную свертку в нескольких масштабах, в то время как простой, но эффективный модуль декодирования уточняет результаты сегментации по границам объекта.

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 4 – Архитектура модели DeepLabV3+, представляющая собой усовершенствованную версию DeepLabV3 с модулем объединения пространственных пирамид (a) и дополненной структурой энкодер-декодер (b) [5]

Расширенная свёртка представляет собой инструмент, который позволяет явно контролировать разрешение признаков, вычисляемых глубокими свёрточными нейронными сетями, и регулировать поле зрения фильтра для захвата многомасштабной информации, обобщая стандартную операцию свёртки. В случае двумерных сигналов для каждого местоположения на выходной карте признаков и свёрточного фильтра расширенная свёртка применяется к входной карте признаков следующим образом: , где коэффициент расширения определяет шаг, с которым производится выборка входного сигнала.

Помимо использования расширенной свёртки, важную роль в архитектуре играет поканальная разделимая свёртка (depthwise separable convolution), которая разлагает стандартную свёртку на поканальную свертку (depthwise convolution), за которой следует поточечная свёртка (pointwise convolution). Конкретно, поканальная свёртка выполняет пространственную свёртку независимо для каждого входного канала, в то время как поточечная используется для объединения выходов от поканальной свёртки. В данной работе авторы назвали получившуюся свёртку расширенной поканальной свёрткой (atrous depthwise convolution) и обнаружили, что она значительно снижает вычислительную сложность предлагаемой модели при сохранении аналогичной или лучшей производительности. Принцип её работы представлен на рисунке 5.

Изображение выглядит как снимок экрана, диаграмма, линия, Прямоугольник

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 5 – Поканальная разделимая свёртка 3×3 разлагает стандартную свертку на (a) поканальную свертку и (b) поточечную свертку. В работе [] исследуется расширенная разделимая свертка, где расширенная свертка применяется в поканальной свертке, как показано в (c) с коэффициентом расширения = 2 [5]

В качестве кодировщика DeepLabv3 использует расширенные свёртки для извлечения признаков с произвольным разрешением. Выходной шаг определяется как отношение пространственного разрешения входного изображения к финальному выходному разрешению. Для семантической сегментации можно использовать выходной шаг 16 или 8 для более плотного извлечения признаков путём удаления шага в последнем одном или двух блоках и соответствующего применения расширенной свёртки. DeepLabv3 дополняет модуль расширенного объединения пространственных пирамид (ASPP) признаками уровня изображения. DeepLabv3+ использует выходные признаки DeepLabv3 (256 каналов) как результат работы энкодера, который затем передаётся в декодер для восстановления детальных границ объектов.

Авторы предлагают следующий модуль декодера: признаки энкодера сначала билинейно увеличиваются в 4 раза, затем соединяются с соответствующими низкоуровневыми признаками из основной сети (backbone), имеющими то же пространственное разрешение. Применяется свёртка 1×1 к низкоуровневым признакам для уменьшения количества каналов, поскольку они обычно содержат большое количество каналов (256 или 512), что может перевесить важность богатых признаков кодировщика (только 256 каналов). После соединения применяется несколько свёрток 3×3 для уточнения признаков, за которыми следует ещё одно билинейное увеличение в 4 раза. Использование выходного шага 16 для модуля энкодера обеспечивает лучший компромисс между скоростью и точностью [5].

1. Практическая часть
   1. Исследование производительности модели

В рамках исследования метода бинарной сегментации была проведена проверка работы обученной модели на изображениях произвольного размера и видео (таблица 1). Во всех экспериментах с видео использовалось одинаковое разрешение 1920×1080 пикселей. Характеристики устройства, на котором запускалась модель: AMD Ryzen 5 5600H, RAM 8 Gb.

При инференсе на изображениях считалась только скорость обработки. Для видео скорость считалась вместе с предобработкой и постобработкой.

Таблица 1 – Скорость работы модели

|  |  |  |  |
| --- | --- | --- | --- |
| Размер изображений при обучениии | Размер изображений при инференсе | Скорость обработки изображения, с | Скорость видео с обработкой, fps |
| 512 | 512 | 0,7257 | 1,5 |
| 512 | 256 | 0,2307 | 5,6 |
| 256 | 512 | 0,6933 | 1,5 |
| 256 | 256 | 0,2171 | 5.7 |
| 128 | 128 | 0,1505 | 14,5 |

В ходе замеров скорости работы модели было замечено, что время предобработки различается в зависимости от размера исходных изображений, что отражено в таблице 2. Предобработка включает в себя чтение и преобразование изображения, нормализацию значений пикселей, изменение размера до входного размера модели, применение дополнительных преобразований и конвертацию в тензор, готовый для модели.

Таблица 2 – Зависимость времени предобработки от размеров входного изображения (изменение размера происходит до 256×256)

|  |  |
| --- | --- |
| Размер исходного изображения, пикс. | Время предобработки, с. |
| 320×320 | 0,0234 |
| 640×640 | 0,0367 |
| 1920×1080 | 0,0563 |
| 4000×3000 | 0,2536 |

Для повышения производительности модели при обработке видеопотока была реализована оптимизация, заключающаяся в сегментации каждого второго кадра. Данный подход позволил увеличить fps**,** однако его использование может привести к уменьшению частоты обновления информации о состоянии окружающей среды во времени и потенциальному пропуску быстро изменяющихся препятствий.

* 1. Работа с изображениями 128×128

В ходе исследования возникла необходимость обучить модель на изображениях малого размера (128×128 пикселей), поскольку такие данные позволяют существенно повысить скорость обработки и сделать модель более пригодной для применения на устройствах с ограниченными вычислительными ресурсами. Изначально архитектура Deeplabv3+ не позволяла обучать модель на изображениях малого размера, поэтому в неё были внесены следующие изменения:

Динамический расчёт размера карты признаков. Был введён автоматический расчёт размера входной карты признаков для модуля ASPP (aspp\_input\_feature\_map\_h = input\_size // 16), что позволило модели корректно работать с любым размером входного изображения без ручной настройки.

Адаптивный выбор коэффициентов дилатации. Для малых изображений большие коэффициенты дилатации выходили за границы карты признаков, что приводило к потере информации. Поэтому коэффициенты дилатации стали автоматически подбираться в зависимости от размера карты признаков (например, для ≤ 8 пикселей использовались dilations = 1).

Адаптивный размер ядра свёртки. Размер ядра свёртки стал определяться минимальным коэффициентом дилатации: для малых значений использовались ядра 1×1 (что снижало вычислительные затраты), для больших — сохранялись ядра 3×3 для достаточного охвата области анализа.

Замена BatchNorm на GroupNorm. Все слои BatchNorm2d были заменены на GroupNorm (32 группы), что обеспечило стабильную нормализацию при работе с малыми батчами и картами признаков, характерными для изображений 128×128.

Адаптивный глобальный пулинг. Фиксированный глобальный пулинг был заменён на адаптивный (nn.AdaptiveAvgPool2d((2, 2))), что позволило корректно обрабатывать карты признаков любого размера, включая малые.

Улучшенная интерполяция. Для восстановления пространственного разрешения стала использоваться адаптивная билинейная интерполяция, что обеспечило точное масштабирование независимо от исходного размера карты признаков.

Все эти изменения в совокупности обеспечивают эффективную работу модели DeepLabV3Plus с изображениями размером 128×128, сохраняя при этом совместимость с изображениями других размеров и улучшая общую производительность модели.

* 1. Исследование качества сегментации

Для демонстрации зависимости качества сегментации от разрешения обучающих данных было проведено тестирование версий модели, обученных на изображениях разного размера. На рисунках [6, 7, 8] показаны результаты обработки одного тестового изображения различными версиями модели, что позволяет наглядно оценить влияние разрешения обучающих данных на качество сегментации.

Изображение выглядит как в помещении, Напольное покрытие, мебель, дизайн интерьера

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6 – Результаты сегментации модели, обученной на изображениях 512×512

Изображение выглядит как Напольное покрытие, в помещении, Деревянный пол, Ламинат

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 7 – Результаты сегментации модели, обученной на изображениях 256×256

Изображение выглядит как пол, в помещении, Напольное покрытие, красный

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 8 – Результаты сегментации модели, обученной на изображениях 128×128

Анализ представленных результатов показывает существенное влияние разрешения обучающих данных на качество бинарной сегментации. Модель, обученная на изображениях 512×512 пикселей, демонстрирует наиболее четкие границы объектов и высокую детализацию сегментированных областей. Контуры препятствий воспроизводятся с максимальной точностью, что особенно заметно на мелких деталях и сложных формах объектов.

Версия модели, обученная на разрешении 256×256, показывает умеренное качество сегментации с некоторой потерей детализации, однако сохраняет достаточную точность для большинства практических задач обнаружения препятствий.

Версия модели, адаптированная для работы с изображениями 128×128, демонстрирует заметное снижение качества сегментации. Границы объектов становятся менее четкими, и общая детализация значительно уступает моделям, обученным на больших разрешениях.

Заключение

В результате проведённого исследования была подробно изучена и экспериментально оценена модель бинарной сегментации на основе DeepLabV3+ с энкодером ResNet-50 для обнаружения препятствий на изображениях и видео. Был проведён сравнительный анализ производительности и качества моделей, обученных на различных разрешениях входных данных, и показано, что уменьшение разрешения позволяет существенно повысить скорость обработки, что важно для задач реального времени, однако сопровождается снижением детализации сегментации.

Для повышения производительности при обработке видео была реализована оптимизация, заключающаяся в сегментации каждого второго кадра. Кроме того, для корректной работы модели на изображениях малого размера были внесены изменения в архитектуру: реализован динамический расчёт размеров карт признаков, адаптивный выбор коэффициентов дилатации, замена BatchNorm на GroupNorm и адаптация глобального пулинга. Эти доработки позволяют использовать модель на устройствах с ограниченными вычислительными ресурсами, расширяя возможности её практического применения.

В ходе экспериментов также выявлено, что значительное влияние на общую производительность оказывает скорость этапов предобработки и ресайза изображений. При увеличении разрешения входных данных время предобработки существенно возрастает, что может замедлять работу системы. Это подчёркивает необходимость дальнейшей оптимизации предобработки и ресайза для повышения эффективности применения моделей машинного зрения в мобильной робототехнике

Список использованных источников

1. Lee, T. A Monocular Vision Sensor-Based Obstacle Detection Algorithm for Autonomous Robots / T. Lee, D. Yi, D. Cho. – DOI: 10.3390/s16030311 // Sensors. – 2016. – Vol. 16, Issue 3. – P. 311. – URL: https://www.mdpi.com/1424-8220/16/3/311 (дата обращения: 26.12.2024).
2. Surehli, M.K. A DeepLab V3+ Model with ResNet 50 Encoder to perform Binary Segmentation Tasks [Электронный ресурс] / M.K. Surehli, N. Aggarwal, G. Joshi. – GitHub Repository. – URL: https://github.com/mukund-ks/DeepLabV3Plus-PyTorch (дата обращения: 26.12.2024).
3. Squeeze-and-Excitation Networks / Jie Hu [et al.] // ArXiv / Cornell University. – 2019. – URL: https://arxiv.org/abs/1709.01507 (дата обращения: 27.05.2025).
4. Deep Residual Learning for Image Recognition / K. He, [et al.] // ArXiv / Cornell University. - 2015. - URL: https://arxiv.org/abs/1512.03385 (дата обращения: 27.05.2025).
5. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation / L.-C. Chen, [et al.] // ArXiv / Cornell University. - 2018. - URL: https://arxiv.org/abs/1802.02611 (дата обращения: 27.05.2025).

Приложение А  
Фрагменты кода

А.1 Обработка изображений

import torch

import cv2

import numpy as np

import albumentations as A

from albumentations.pytorch import ToTensorV2

from model import DeepLabV3Plus

from config import Config

from PIL import Image

import time

def predict\_mask(image\_path, weights\_path, input\_size=1024, return\_overlay=False):

    Config.update\_input\_size(input\_size)

    device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

    model = DeepLabV3Plus(num\_classes=Config.CLASSES)

    model.load\_state\_dict(torch.load(weights\_path, map\_location=torch.device('cpu')))

    model.to(device)

    model.eval()

    start\_time = time.time()

    pil\_image = Image.open(image\_path).convert("RGB")

    print(f"Оригинальные размеры изображения: {pil\_image.width}x{pil\_image.height}")

    original\_image = np.array(pil\_image, dtype=np.float32)

    original\_image = original\_image / 255.0

    pil\_resized = pil\_image.resize((Config.INPUT\_SIZE[0], Config.INPUT\_SIZE[1]))

    resized\_image = np.array(pil\_resized)

    transform = A.Compose([

        A.Resize(Config.INPUT\_SIZE[0], Config.INPUT\_SIZE[1]),

        A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),

        ToTensorV2(),

    ])

    transformed = transform(image=original\_image)

    image\_tensor = transformed["image"].unsqueeze(0).to(device)

    mid\_time = time.time()

    with torch.no\_grad():

        output = model(image\_tensor)

        prediction = output.cpu().numpy()[0, 0]

    end\_time = time.time()

    preprocessing\_duration = mid\_time - start\_time

    processing\_duration = end\_time - mid\_time

    print(f"Время предобработки: {preprocessing\_duration:.4f} секунд")

    print(f"Время обработки моделью: {processing\_duration:.4f} секунд")

    mask = (prediction > 0.5).astype(np.uint8) \* 255

    original\_image\_display = np.array(pil\_image)

    if return\_overlay:

        h, w = original\_image\_display.shape[:2]

        mask\_resized = cv2.resize(mask, (w, h), interpolation=cv2.INTER\_NEAREST)

        overlay = original\_image\_display.copy()

        overlay[mask\_resized > 0] = [255, 0, 0]

        blended = cv2.addWeighted(original\_image\_display, 0.7, overlay, 0.5, 0)

        return resized\_image, mask, blended

    return mask

if \_\_name\_\_ == "\_\_main\_\_":

    img, mask, overlay = predict\_mask(

        image\_path="C:\\Users\\lealo\\Documents\\SRW\\test\\photos\\obstacles-5.jpg",

        weights\_path="../output/256-41.pth",

        return\_overlay=True,

        input\_size=256

    )

    cv2.imwrite("overlay.png", cv2.cvtColor(overlay, cv2.COLOR\_RGB2BGR))

А.2 Обработка видео

import torch

import cv2

import numpy as np

import albumentations as A

from albumentations.pytorch import ToTensorV2

from model import DeepLabV3Plus

from config import Config

import time

def process\_video(video\_path, weights\_path, input\_size=256, output\_path=None, skip\_frames=0, scale\_factor=0.5):

    Config.update\_input\_size(input\_size)

    device = torch.device("cpu")

    model = DeepLabV3Plus(num\_classes=Config.CLASSES)

    model.load\_state\_dict(torch.load(weights\_path, map\_location=device))

    model.to(device)

    model.eval()

    cap = cv2.VideoCapture(0 if video\_path == 0 or video\_path.lower() == "camera" else video\_path)

    if not cap.isOpened():

        print(f"Ошибка при открытии видео {video\_path}")

        return

    display\_width = int(input\_size \* scale\_factor)

    display\_height = int(input\_size \* scale\_factor)

    out = None

    if output\_path:

        fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

        out = cv2.VideoWriter(output\_path, fourcc, cap.get(cv2.CAP\_PROP\_FPS), (display\_width, display\_height))

    transform = A.Compose([

        A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),

        ToTensorV2(),

    ])

    frame\_count = 0

    start\_total = time.time()

    last\_mask = None

    try:

        while cap.isOpened():

            ret, frame = cap.read()

            if not ret:

                break

            frame\_count += 1

            frame\_resized = cv2.resize(frame, (input\_size, input\_size))

            if skip\_frames > 0 and frame\_count % (skip\_frames + 1) != 0 and last\_mask is not None:

                mask = last\_mask

            else:

                image = cv2.cvtColor(frame\_resized, cv2.COLOR\_BGR2RGB).astype(np.float32) / 255.0

                transformed = transform(image=image)

                image\_tensor = transformed["image"].unsqueeze(0)

                with torch.no\_grad():

                    output = model(image\_tensor)

                    prediction = output.numpy()[0, 0]

                mask = (prediction > 0.5).astype(np.uint8) \* 255

                last\_mask = mask.copy()

            overlay = frame\_resized.copy()

            overlay[mask > 0] = [0, 0, 255]

            blended = cv2.addWeighted(frame\_resized, 0.7, overlay, 0.5, 0)

            if scale\_factor != 1.0:

                blended = cv2.resize(blended, (display\_width, display\_height))

            cv2.imshow("Segmentation", blended)

            if output\_path and out:

                out.write(blended)

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

                break

    except Exception as e:

        print(f"Произошла ошибка: {e}")

    finally:

        cap.release()

        if out:

            out.release()

        cv2.destroyAllWindows()

        end\_total = time.time()

        if frame\_count > 0:

            average\_effective\_fps = frame\_count / max(0.001, end\_total - start\_total)

            print(f"\nСредний эффективный FPS: {average\_effective\_fps:.1f}")

if \_\_name\_\_ == "\_\_main\_\_":

    process\_video(

        video\_path=r"C:\Users\lealo\Documents\SRW\test\video-1.mp4",

        weights\_path="../output/models/best\_model(0.4282).pth",

        input\_size=512,

        output\_path=None,

        skip\_frames=0,

        scale\_factor=1

    )