МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по проектной работе

по дисциплине «Пространственный искусственный интеллект»

Тема: Сегментация объектов на изображении

Студент гр. 0304 _____ Сергеев Д.А.

Студентка гр. 0304 _____ Чегодаева Е.А.

Студент гр. 0304 _____ Карпий И.Ю.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2025

Цель работы

Сгенерировать датасет (минимум 5 классов), реализовать модель для сегментации и детекции объектов на изображении из данного датасета, сделать модель доступной из любой среды.

Распределение ролей в проекте:

- Создание датасета: Карпий Игорь
- Выбор, обучение модели: Чегодаева Елизавета
- Тестирование, контейнеризация модели: Сергеев Дмитрий

Ход работы

1. Генерация синтетических данных

1.1 Инструментарий и входные условия

В соответствии с техническим заданием, для формирования обучающей выборки использовался фреймворк Kubric (Google Research). Работа осуществлялась в контейнеризированной среде (Docker kubruntu), обеспечивающей доступ к движкам Blender (рендеринг) и PyBullet (физическая симуляция).

В качестве источника 3D-объектов использовался набор данных ShapeNetCore.v2, подключенный через Google Cloud Storage (`kubric-unlisted`), с фильтрацией по целевым классам: Chair, Table, Sofa, Car, Airplane.

1.2 Реализация алгоритма генерации сцен

Разработан программный модуль (generate_single_scene.py), реализующий процедурную генерацию сцен по следующему алгоритму:

1) Инициализация окружения: Создание сцены с разрешением 512x512. Установка прозрачного пола с функционалом *Shadow Catcher* для сохранения реалистичных теней без привязки к фону.
2) Настройка освещения: Интеграция системы освещения типа CLEVR (трёхточечный свет с рандомизацией) для обеспечения равномерной подсветки объектов сложной геометрии.

3) Сэмплирование объектов: Случайная выборка 3–7 объектов из ShapeNet. Для каждого объекта применяются аффинные преобразования (случайное масштабирование в диапазоне 0.8–1.5, поворот на 90° по оси X для корректной ориентации).
4) Физическая стабилизация: Запуск симуляции физики твердых тел (Rigid Body Simulation) на 40 кадров. Данный этап необходим для устранения коллизий (взаимных пересечений) и обеспечения естественного расположения объектов на плоскости под действием гравитации.

1.3 Проблематика реализации и технические решения

В ходе адаптации фреймворка Kubric под задачи Instance Segmentation был выявлен ряд архитектурных ограничений и ошибок, потребовавших доработки стандартного пайплайна.

1.3.1. Проблема управления ресурсами физического движка

Описание проблемы: При попытке циклической генерации множества сцен внутри одного Python-процесса происходила утечка памяти в модуле PyBullet, приводящая к критической ошибке `Not connected to physics server` после 2–3 итераций.

Решение: Разработан внешний механизм оркестрации на языке Bash (`run_batch.sh`). Архитектура генерации изменена на изолированную: каждая сцена генерируется в отдельном эфемерном Docker-контейнере, который принудительно завершается после рендеринга. Это гарантирует полную очистку ресурсов ОЗУ и дескрипторов PyBullet между итерациями.

1.3.2. Ошибка синхронизации метаданных сегментации

Описание проблемы: В текущей версии Kubric наблюдалась рассинхронизация между движком рендеринга и Python-объектами. Автоматически назначаемые `segmentation_id` не возвращались в метаданные объектов, в результате чего файл разметки классов (`classes.json`) формировался пустым, делая невозможным сопоставление масок с классами.

Решение: Внедрен механизм принудительной индексации. Перед вызовом рендеринга скрипт вручную назначает уникальные целочисленные идентификаторы всем объектам сцены и формирует карту соответствия "ID — Класс YOLO". Это обеспечило детерминированность разметки.

1.3.3. Несовместимость форматов данных

Описание проблемы: Штатный вывод Kubric (многослойные PNG с индексированной палитрой и альфа-каналом) не поддерживается архитектурой YOLOv8, требующей плоские RGB-изображения и векторную разметку полигонами.

Решение: Разработан модуль пост-процессинга (`convert_to_yolo.py`), выполняющий:

- Замену альфа-канала на белый фон (композитинг).
- Чтение масок через библиотеку Pillow для извлечения "сырых" индексов палитры (Raw Palette Indices), минуя автоматическое преобразование в RGB.
- Векторизацию растровых масок (алгоритм `findContours`) с нормализацией координат под формат YOLO-TXT.

1.4. Результаты работы подсистемы

В результате проведенных работ сформирован синтетический датасет, полностью готовый для дообучения модели. На рисунках 1 и 2 приведен пример сцены из сформированного набора данных.

Рисунок 1 — Пример сцены из сгенерированного набора данных

```
4 0.302734 0.531250 0.300781 0.533203 0.300781 0.535156 0.298828 0.537109 0.298828 0.556641 0.296875 0.558594 0.296875 0.562500 0.287109 0.572266 0.
812 0.318359 0.632812 0.318359 0.630859 0.316406 0.628906 0.314453 0.628906 0.310547 0.625000 0.308594 0.625000 0.302734 0.619141 0.302734 0.613281
1 0.705078 0.621094 0.703125 0.623047 0.701172 0.623047 0.701172 0.632812 0.699219 0.634766 0.699219 0.654297 0.697266 0.656250 0.697266 0.671875 0.
391 0.851562 0.648438 0.849609 0.648438 0.849609 0.652344 0.847656 0.654297 0.841797 0.654297 0.839844 0.652344 0.830078 0.652344 0.828125 0.650391
1 0.818359 0.503906 0.816406 0.505859 0.802734 0.505859 0.800781 0.507812 0.785156 0.507812 0.783203 0.509766 0.769531 0.509766 0.767578 0.511719 0.
750 0.843750 0.591797 0.843750 0.589844 0.845703 0.587891 0.845703 0.585938 0.847656 0.583984 0.847656 0.582031 0.849609 0.580078 0.849609 0.576172
0 0.662109 0.380859 0.660156 0.382812 0.656250 0.382812 0.654297 0.384766 0.650391 0.384766 0.648438 0.386719 0.644531 0.386719 0.642578 0.388672 0.
750 0.630859 0.466797 0.634766 0.466797 0.636719 0.464844 0.642578 0.464844 0.644531 0.462891 0.650391 0.462891 0.652344 0.460938 0.660156 0.460938
1 0.181641 0.267578 0.179688 0.269531 0.169922 0.269531 0.167969 0.271484 0.160156 0.271484 0.154297 0.277344 0.154297 0.279297 0.144531 0.289062 0.
688 0.183594 0.423828 0.185547 0.423828 0.189453 0.427734 0.191406 0.427734 0.195312 0.431641 0.197266 0.431641 0.199219 0.433594 0.201172 0.431641
92969 0.218750 0.289062 0.216797 0.289062 0.214844 0.287109 0.212891 0.287109 0.208984 0.283203 0.207031 0.283203 0.203125 0.279297 0.201172 0.27929
1 0.474609 0.544922 0.472656 0.546875 0.466797 0.546875 0.464844 0.548828 0.457031 0.548828 0.455078 0.550781 0.449219 0.550781 0.447266 0.552734 0.
594 0.484375 0.556641 0.486328 0.556641 0.486328 0.554688 0.482422 0.554688 0.480469 0.552734 0.478516 0.552734 0.476562 0.550781 0.476562 0.544922
```

Рисунок 2 — Разметка, соответствующая сцене на рисунке 1

2. Обучение модели

Для решения задачи сегментации и детекции выбрана архитектура ***yolov8n-seg*** (Ultralytics). Обучение проводилось при следующих параметрах:

- Число эпох: 50
- Ранняя остановка: 30 эпох
- Размер батча: 16
- Оптимизатор: AdamW

По завершению обучения построены графики, отражающие динамику метрик при обучении и валидации модели. Графики приведены на рисунке 3.

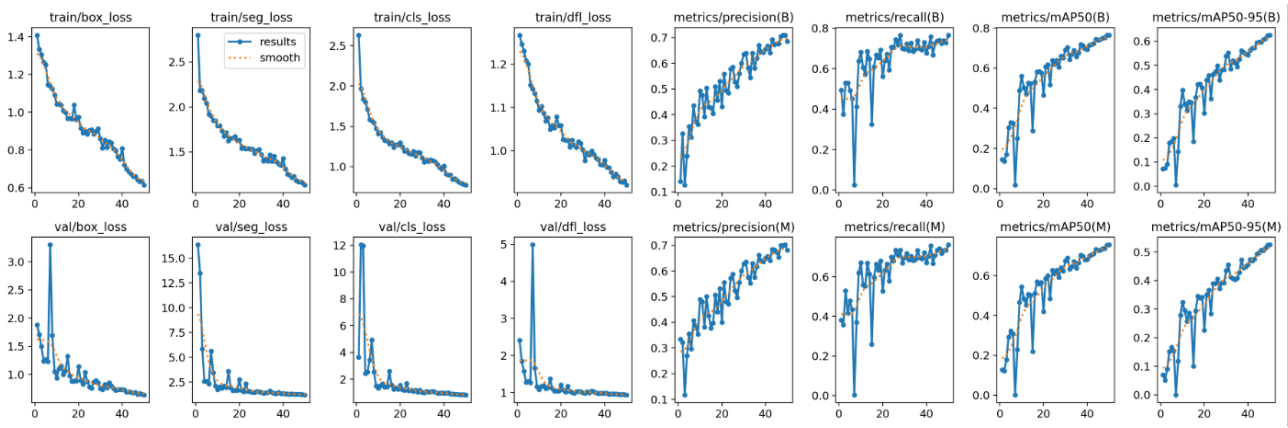

Рисунок 3 — Метрики обучения и валидации модели

Согласно графикам, получены следующие выводы: значение потерь при обучении постепенно и стабильно уменьшаются на протяжении 50 эпох. Валидационные потери также постепенно снижаются, хотя и с более сильными колебаниями на первых эпохах. Метрики детекции (B) растут по мере обучения, динамика метрик сегментации (M) аналогична детекции, но значения немного ниже.

Помимо этого, была построена матрица ошибок (матрица путаницы), приведенная на рисунке 4.

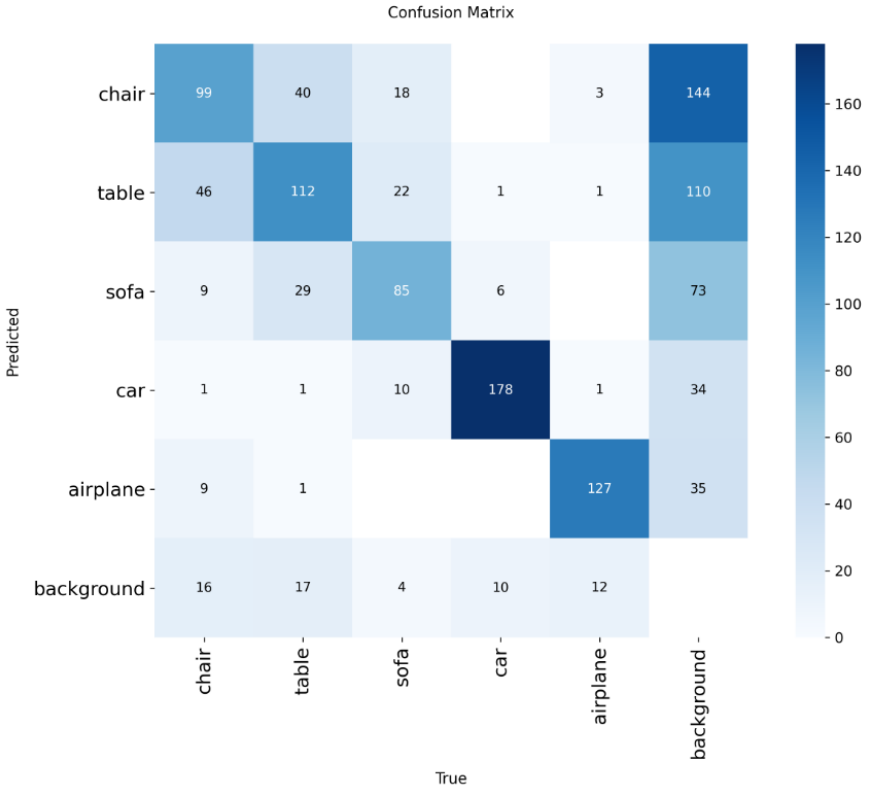

Рисунок 4 — Матрица ошибок модели

Таким образом, car и airplane имеют наиболее высокую частоту верных предсказаний, chair, table и sofa — чуть ниже. Наиболее частая ошибка — слияние объектов с категорией background. Общая структура матрицы показывает, что модель обладает хорошей способностью к обобщению и довольно успешно различает большинство классов => валидна для дальнейшего использования.

На рисунках 5, 6 приведен пример работы модели для фрагмента данных.

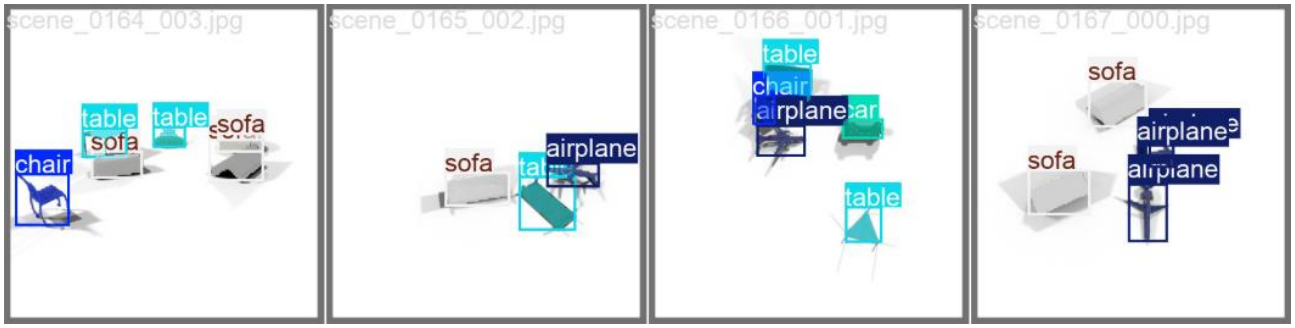

Рисунок 5 — Фрагмент исходных данных с истинными метками

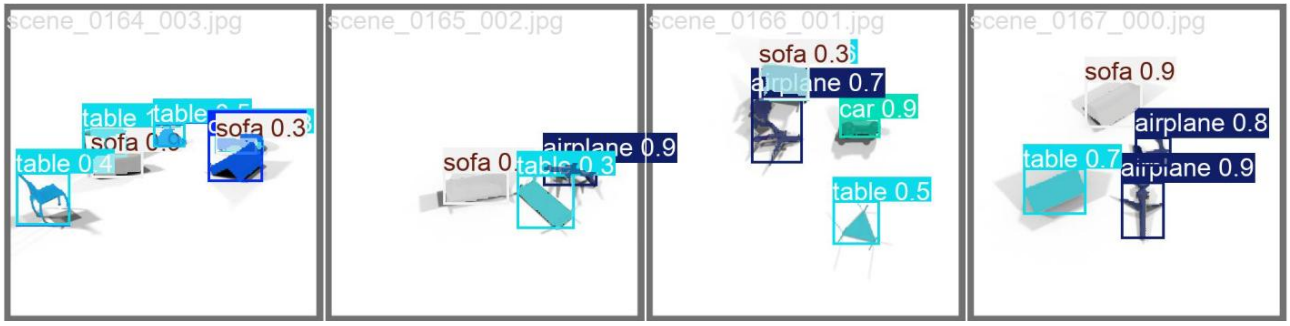

Рисунок 6 — Фрагмент обработанных данных с сегментацией и уверенностью принадлежности к конкретному классу

Для обеспечения переносимости модели и возможности запуска в различных средах дообученная модель YOLOv8n-seg была экспортирована в формат ONNX. После экспорта была выполнена проверка корректности ONNX-модели. По результату тестирования получены значения метрик, указанные на рисунке 7.

```
Метрики для сегментации
mAP@50-95 (seg): 0.5252
mAP@50 (seg): 0.7541
mAP@75 (seg): 0.5664
mAP@50-95 по классам (seg): [    0.35009     0.40369     0.61958     0.75855     0.49408]

Метрики для детекции (box)
mAP@50-95 (box): 0.6282
Precision: 0.6824
Recall: 0.7633
F1: 0.7196
```

Рисунок 7 — Метрики при тестировании ONNX-модели.

3. Тестирование сети, контейнеризация модели

Для использования модели в любом окружении был составлен Docker-образ на основе nvcr.io/nvidia/pytorch. Дополнительно были установлены следующие пакеты:

- ultralytics – содержит реализацию модели YOLO;
- onnxruntime – используется для применения моделей, сохраненных в формате ONNX;
- onnxruntime-gpu – применяется для переноса вычислений на графический процессор.

Далее были составлены два скрипта: test.py и inference.py. Первый скрипт загружает модель в ONNX формате, а затем применяет её к тестовой выборке, на основе полученного результата в консоль выводятся вычисленные метрики.

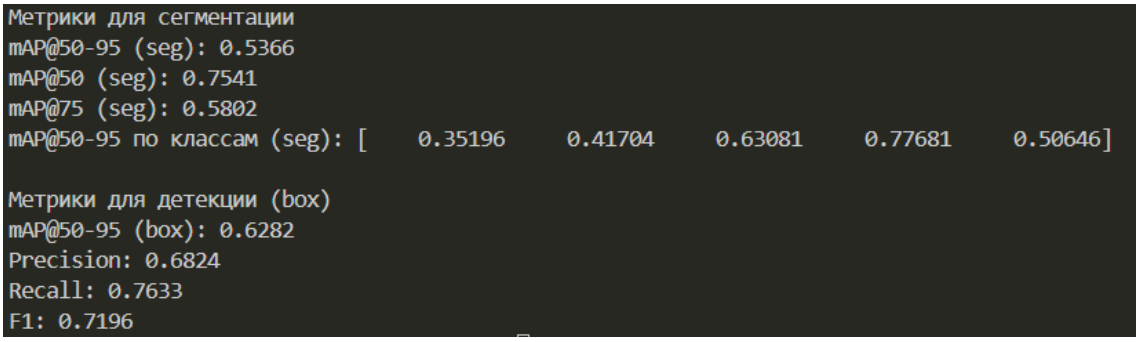

Рисунок 8 — Значения метрик после применения модели к тестовой выборке

Метрики, приведенные на рисунке 8, близки к метрикам, полученным при обучении модели на другом устройстве (рисунок 7), что подтверждает факт корректного переноса модели с обучающей среды на реальную.

Второй скрипт загружает модель в ONNX формате, а затем применяет её к одному или нескольким изображениям. Результаты применения модели сохраняются в текущую директорию в формате jpg.

На рисунке 9 представлена обработанная моделью сцена, приведенная на рисунке 1, на которой прямоугольниками обозначены примерные границы объектов, а также пиксели, которые по мнению модели относятся к объекту, выделены цветом.

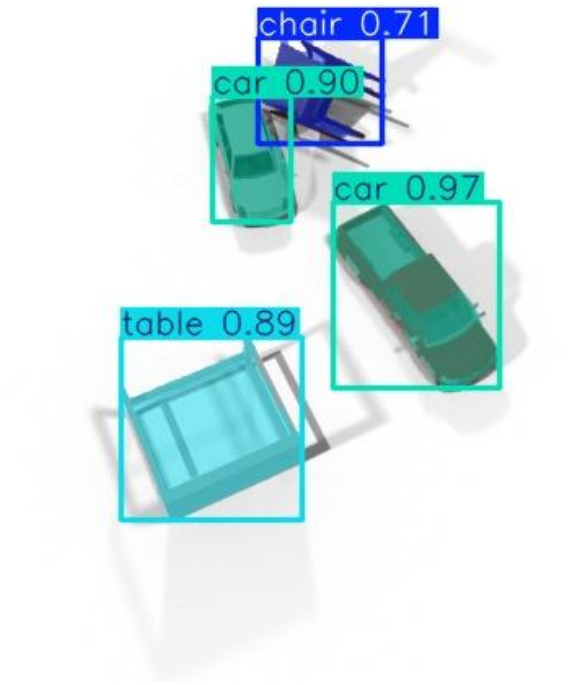

Рисунок 9 — Результат применения модели к сцене, представленной на рисунке 1.

Вывод

В ходе выполнения проектной работы была достигнута поставленная цель, а именно:

1) Посредством фреймворка Kubric сгенерирован датасет, состоящий из 5 классов: Chair, Table, Sofa, Car, Airplane. Таким образом, был сформирован набор данных, разделенный на обучающую (790 экземпляров) и тестовую (200 экземпляров) выборки.

2) На основе кастомного датасета реализовано дообучение модели yolov8n-seg для задачи сегментации и детекции. Модель была экспортирована в ONNX. При тестировании модели были получены следующие значения метрик:

- Сегментация:
  mAP@50-95 – 0.5252; mAP@50 – 0.7541; mAP@75 – 0.5664.
- Детекция:
  mAP@50-95 – 0.6282; Precision – 0.6824; Recall – 0.7633; F1 – 0.7196.

3) Дообученная модель была перенесена в Docker-контейнер для возможности использования в иной среде. По результатам запуска в новой среде можно сделать вывод о том, что отклонение метрик от исходной среды незначительно.