Детектирование машин при помощи YOLO

Антон Зайцев

Задача

- размер входных данных не ниже 320x320 px
- метрика mAP@50 >=0,6 на валидационной выборке
- скорость работы не ниже 15 FPS (CPU Ryzen 5 3600)
- экспорт в формат ONNX
- трекинг объектов*

Датасет

- Распаковка из zip архивов
- Доработка YAML описание (прописать пути и классы)
- сконвертировать метки из формата visdrone в yolo
- удалить из текстовых файлов меток ненужные классы
- удалить ненужные файлы изображений (на которых нет автомобилей)

Была реализована функция подготовки датасета с использованием

многопоточности. Время преобразования меток снизился с 70 сек до 4,5

Выбор модели YOLO

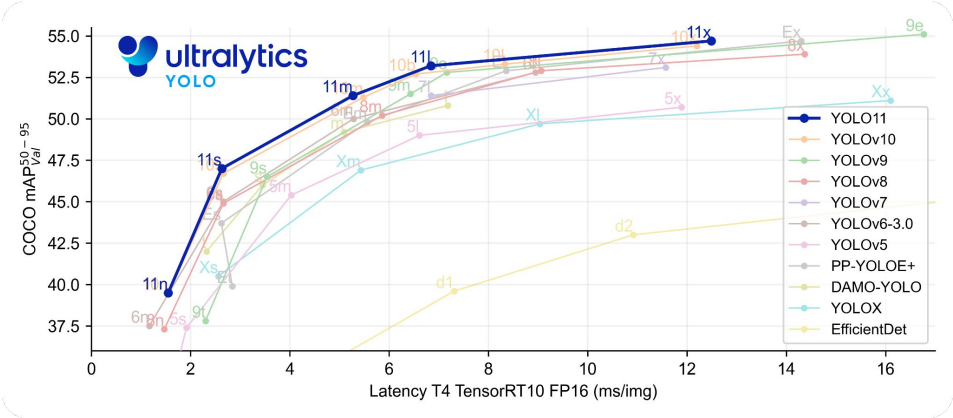
-Выбираем YOLO11, не все фреймворки поддерживают последнюю версию*.

11 версия имеет выше точность и

высокий FPS

-Нужен высокий FPS (S и N вариант)

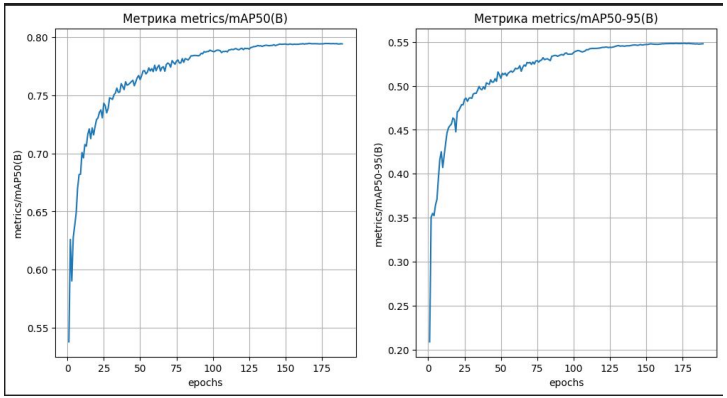
*opencv только 10 версию, RKNN - 11

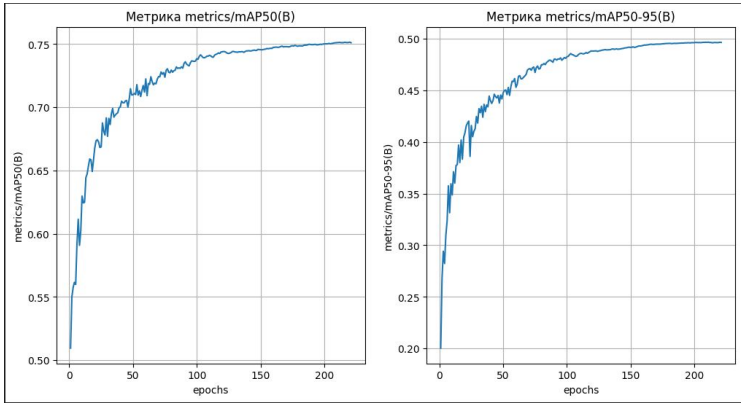


Тренировка модели

Обучено 2 модели Nano и Small

```yaml
epochs: 10
imgsz: 640
batch: 16
degrees: 45
patience: 10
save_period: 1
exist_ok: true
deterministic: false
plots: true
data: *yolo_cfg
single_cls: true
model: 'yolo11s.pt'
```



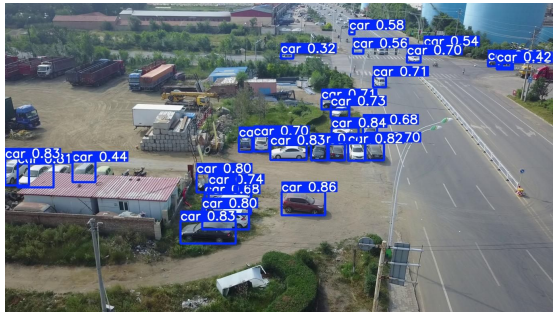


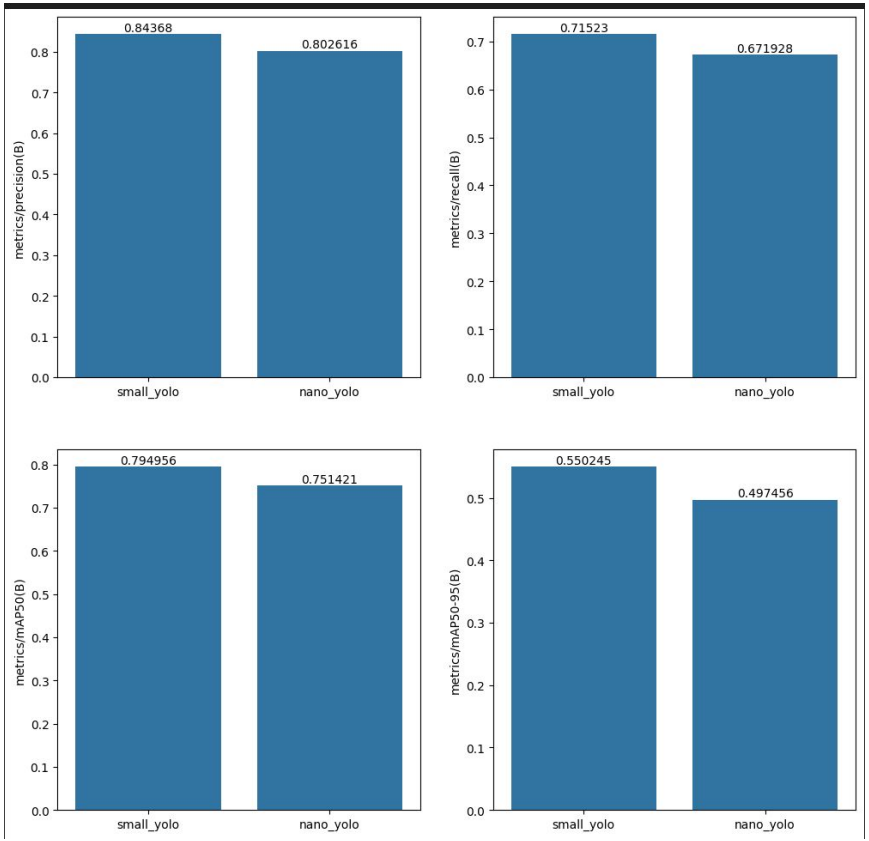
Валидация моделей

-Обе модели Y11S и Y11N превысили порог mAP50=0.6

-Модель S показала лучшие метрики

-Библиотека может сравнивать много моделей





Бенчмарк моделей

Конвертация модели Nano дала

прирост 117% на GPU и 38% на CPU

Для Small: 16% и 5% соответственно.

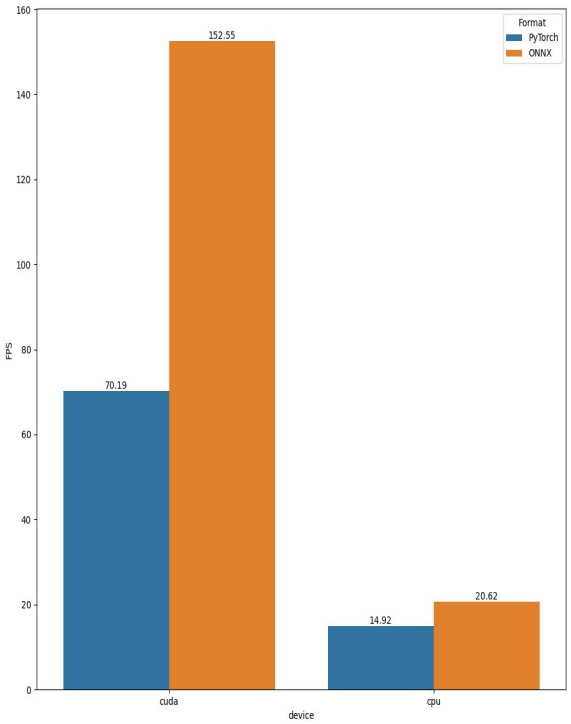
Тест проводился штатным бенчмарком

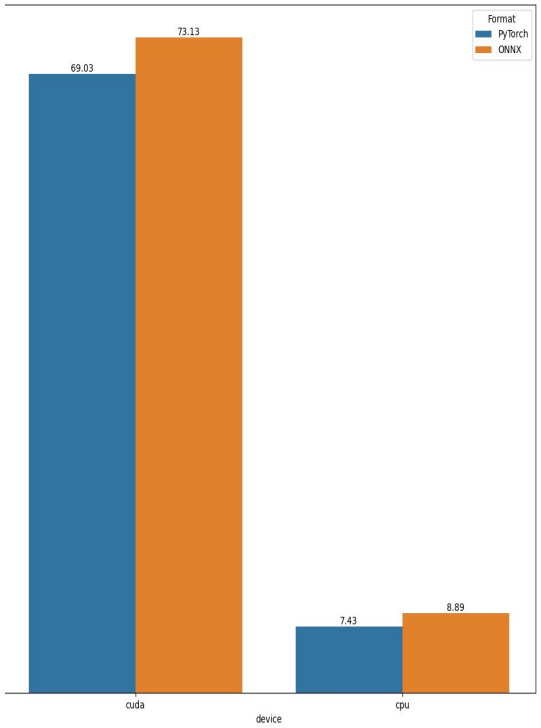
YOLO на CPU Ryzen 5 3600 32Gb RAM

RTX3070 8Gb

Только модель nano подходит

под требования T3 (15fps)





Запуск экспортированной модели ONNX

Для запуска используется ONNXRuntime

- Препроцессинг (ресайз, паддинг + преобразование в тензоры)
- Улучшенный постпроцессинг* (координаты боксов, NMS)
- Возможность инференса батчами изображений
- Тюнинг параметров сессии (количество потоков, последовательное исполнение)

*(по сравнению с примером ultralytics с 40 до 0 мс время постобработки)

Оценка производительности ONNX

- Для CPU подобрано число потоков и включена оптимизация. Так же используется ThreadPoolExecutor для полной загрузки процессора.

-Для GPU подобран оптимальный размер батча (8).

	CPU (onnx)	GPU (onnx)	CPU (OpenVino на AMD)
small	12.745 (+43%)	71.57	15.905 (+24% от ONNX)
nano	29.987 (+45%)	86.769	34.82 (+16% от ONNX)

*С пред и пост обработкой, пред и постобработка значительно снижает FPS для GPU

(перенести на карточку?)

Демо программа

- фреймворк Pyqt
- в виде exe для win10+
- opencv для чтения потока из файла
- адаптивное обновление кадров в зависимости от производительности
- выбор моделей
- трекинг объектов

Выводы и возможные улучшения

В ходе работы был разработан python модуль, позволяющий упростить процесс создания датасета, обучения, валидации, сравнения моделей YOLO, а также обучено 2 модели для детектирования машин на датасете VisDrone.

Возможные улучшения:

Интеграция работы с Docker (вместо создания вирт среды, работаем с контейнером с предустановленной средой)

Преобразование в tensorrt или другой фреймворк целевой платформы, квантизация и структурный прунинг для уменьшения модели и ускорения инференса.

Перенос предобработки на GPU

Добавить экспорт из ONNX в формат целевых платформ (RKNN, TPU MLIR …)