Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики

Отчёт

Тема: Сравнительный анализ производительности форматов модели YOLOv11m-obb

Команда: XYZ Labs

Москва, 2024

Оглавление

1. Введение

1.1. Актуальность задачи

В современном мире задачи детекции объектов приобретают все большую актуальность в различных областях. Разработка и применение моделей глубокого обучения для детекции объектов, таких как модели семейства YOLO, требует значительных вычислительных ресурсов. Оптимизация этих моделей для эффективной работы на различных платформах является критически важной, особенно в условиях ограниченных ресурсов. Существует множество различных форматов моделей, и выбор наиболее подходящего формата для конкретной задачи требует тщательного анализа и сравнения. Поэтому, оценка производительности различных форматов моделей YOLO является актуальной и важной задачей.

1.2. Цель и задачи анализа

Целью данного анализа является сравнительная оценка производительности различных форматов моделей YOLO (ONNX, OpenVINO, PyTorch, TensorFlow, TorchScript) на платформах CPU и GPU для определения оптимального формата с точки зрения времени инференса и потребления памяти. Для достижения этой цели были поставлены следующие задачи: преобразование модели YOLO в различные форматы; проведение серии тестов для измерения времени инференса и потребления памяти каждого формата на CPU и GPU; анализ полученных результатов и сравнение производительности различных форматов; выявление наиболее быстрого и эффективного по памяти формата для каждой платформы; и формулирование рекомендаций по выбору оптимального формата модели YOLO для различных сценариев использования.

1.3. Обзор форматов моделей

В данном анализе рассматриваются следующие форматы моделей, которые представляют различные подходы к оптимизации, развертыванию и совместимости моделей глубокого обучения:

- **PyTorch:** Будучи популярным фреймворком для машинного обучения, PyTorch изначально используется для загрузки и обучения моделей, таких как YOLO. Модели PyTorch, созданные на основе предобученных весов, могут быть использованы непосредственно для инференса или экспортированы в другие форматы для оптимизации и развертывания в различных средах.
- **TorchScript:** TorchScript – это промежуточное представление моделей PyTorch, предназначенное для обеспечения переносимости и возможности запуска моделей в средах, где полный фреймворк Python недоступен. Он ориентирован на развертывание моделей компьютерного зрения в различных средах, включая встроенные системы, веб-браузеры или платформы с ограниченной поддержкой Python.
- **ONNX (Open Neural Network Exchange):** ONNX – это открытый стандарт для представления моделей машинного обучения, разработанный для обеспечения совместимости между различными платформами. Экспорт моделей YOLO в формат ONNX упрощает развертывание и обеспечивает оптимальную производительность в различных средах, предоставляя гибкий и совместимый формат, особенно полезный при работе с несколькими платформами.
- **OpenVINO (Open Visual Inference and Neural Network Optimization toolkit):** OpenVINO – это комплексный набор инструментов от Intel для оптимизации и развертывания моделей искусственного интеллекта, включая модели компьютерного зрения. OpenVINO позволяет экспортировать модели YOLO в свой формат, что обеспечивает ускорение до 3x на CPU, а также ускорение вывода на Intel GPU и NPU.

- **TensorRT:** TensorRT – это расширенный набор средств разработки программного обеспечения (SDK) от NVIDIA, предназначенный для высокоскоростного глубокого обучения. Этот инструментарий оптимизирует модели глубокого обучения для графических процессоров NVIDIA и обеспечивает более быструю и эффективную работу. TensorRT известен своей совместимостью с различными форматами моделей, включая TensorFlow, PyTorch и ONNX, предоставляя разработчикам гибкое решение для интеграции и оптимизации моделей из различных фреймворков.
- **TF SavedModel:** TF SavedModel – это формат, используемый TensorFlow для последовательной загрузки моделей машинного обучения. Он обеспечивает возможность развертывания моделей на различных платформах и в разных средах, упрощая процесс проведения выводов с помощью моделей на различных устройствах. TF SavedModel представляет собой эффективный и гибкий формат для развертывания моделей машинного обучения.

2. Методология

2.1. Инструменты и модель

Тестирование проводилось в среде Google Colaboratory. Для теста на CPU использовался предоставляемый Google Colab центральный процессор, а для теста на GPU - NVIDIA Tesla T4.

Для тестирования использовалась модель YOLO11m-obb, представляющая собой модель среднего размера из семейства YOLO11, предназначенную для детекции объектов, ориентированных произвольным образом (Oriented Bounding Boxes, OBB). Модели YOLO11 OBB используют суффикс '-obb', например, yolo11m-obb.pt, и предварительно обучены на датасете DOTAv1. Файл с предобученными весами был загружен из Ultralytics. Согласно данным, представленным Ultralytics, модель YOLOv11m-obb имеет следующие характеристики:

Таблица 1– Характеристики модели YOLO11m-obb

Model	Size (pixels)	mAPtest 50	Speed CPU ONNX (ms)	Speed T4 TensorRT10 (ms)	Params (M)	FLOPs (B)
YOLO11m-obb	1024	80.9	562.8 ± 2.9	10.1 ± 0.4	20.9	183.5

2.2. Набор данных

Для тестирования использовались изображения размера 512x512 пикселей из тестовой выборки датасета DOTA. Для тестирования на CPU использовалось 150 изображений. Для тестирования на GPU использовалась вся тестовая выборка из DOTA, в которой 6854 изображения.

2.3. Процедура тестирования

Для перевода модели из PyTorch в нужные форматы использовался метод: .export(format=…).

Для тестирования на CPU для каждой модели для каждого изображения подсчитывалось время, затраченное на предобработку,

инференс и постобработку (эти характеристики предоставляет метод .speed() у объекта результат инференса), затем высчитывается общее время инференса, максимальный объём памяти этого процесса для конкретной модели на всех изображениях и среднее время предобработки, инференса и постобработки.

Тестирование на GPU выполняется практически по такому же сценарию, за исключением того, что кол-во изображений для теста существенно больше.

3. Результаты тестирования

3.1. Результаты тестирования на CPU

При тестировании на CPU использовались форматы ONNX, OpenVINO, PyTorch, TensorFlow, TorchScript:

Таблица 2-Результаты тестирования на CPU

Format	Total Inference Time (s)	Peak Memory Usage (MB)	Avg. Preprocess Time (ms)	Avg. Inference Time (ms)	Avg. Postprocess Time (ms)
ONNX	606.610289	4359.355469	11.220058	4029.934858	2.913679
OpenVINO	559.547208	4152.652344	12.136841	3715.425536	2.752344
PyTorch	617.424494	4359.347656	10.707085	4102.751284	2.704921
TensorFlow	660.460672	4157.035156	12.572005	4387.646454	2.852689
TorchScript	615.055860	4359.359375	11.560323	4086.181971	2.630108

Визуализация полученных результатов:

Рисунок 1-Визуализация времени инференса

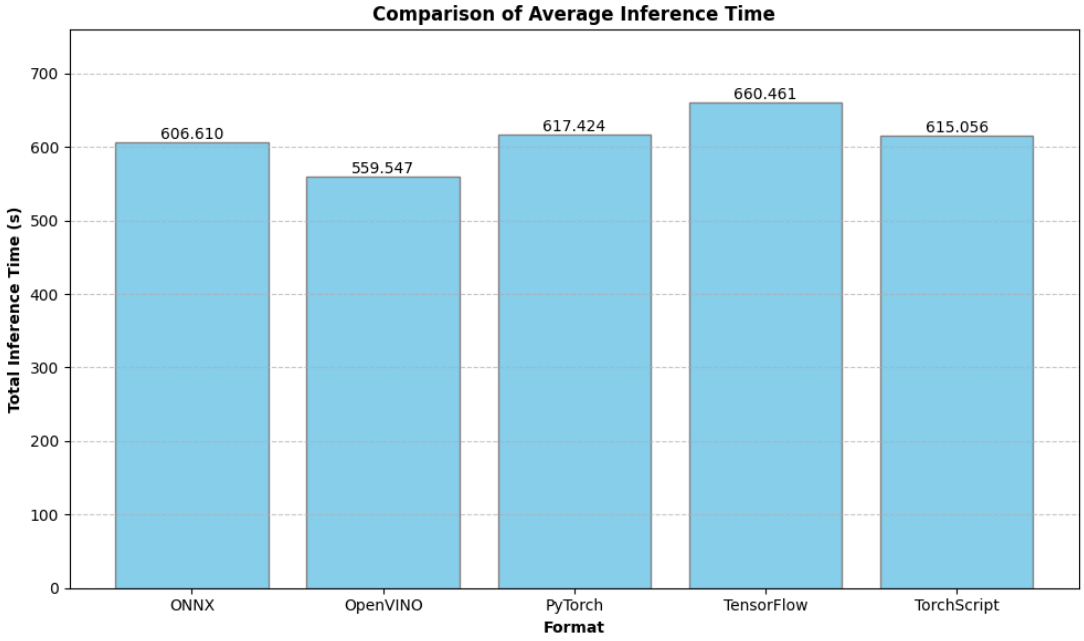
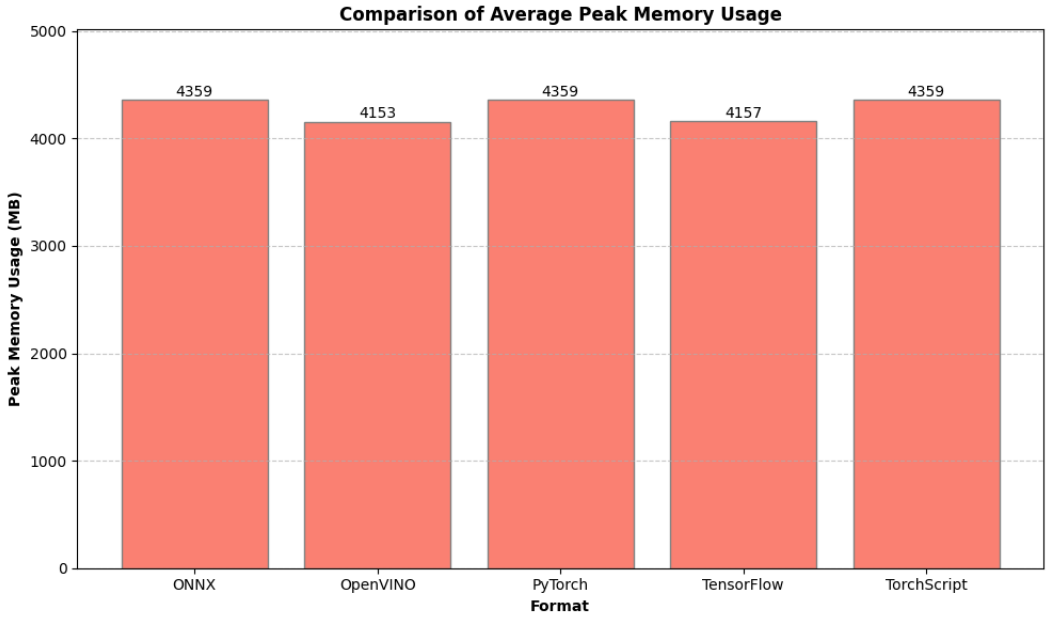


Рисунок 2-Визуализация пиковой памяти, затраченной на инференс

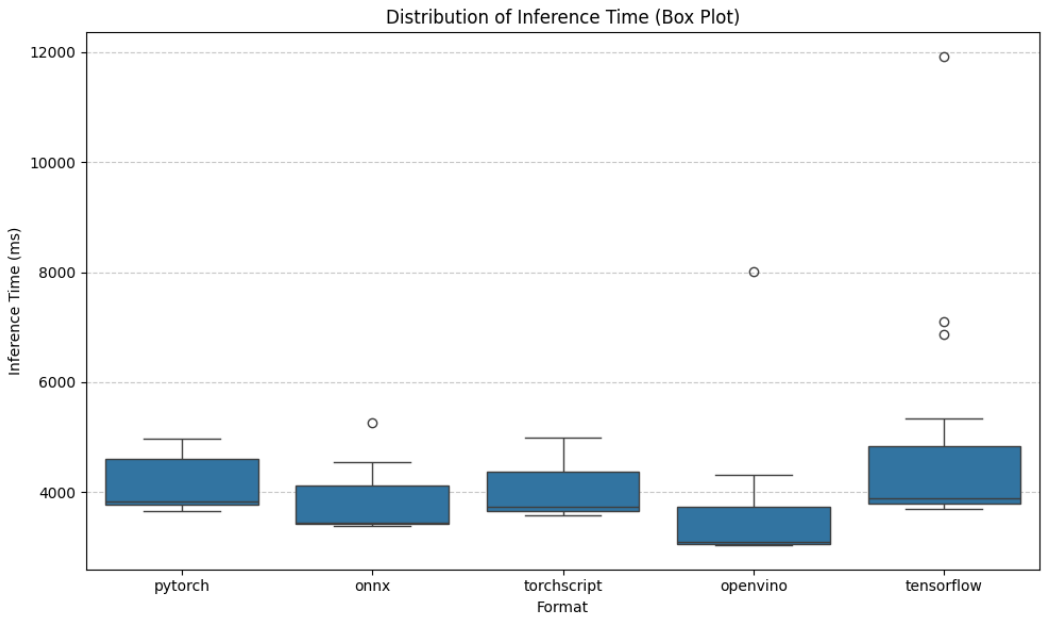


Результаты показывают, что OpenVINO продемонстрировал наиболее высокую производительность, обеспечив наименьшее общее время инференса. Это указывает на эффективность оптимизаций, реализованных в данном формате для CPU. TensorFlow, напротив, показал самое длительное время инференса среди рассматриваемых форматов. При этом, стоит отметить, что пиковое потребление памяти у всех форматов примерно одинаковое, хотя OpenVINO и TensorFlow показали несколько лучшие результаты в этом аспекте. PyTorch и TorchScript продемонстрировали практически идентичную производительность. ONNX показал результат немного лучше, чем PyTorch и TorchScript, но хуже, чем OpenVINO.

Рисунок 3-Время инференса на каждом изображении

График времени инференса для каждого изображения демонстрирует значительные различия в стабильности работы различных форматов YOLO на CPU. OpenVINO выделяется наименьшим и наиболее стабильным временем инференса, что подтверждает его оптимизацию. TensorFlow, напротив, показывает наибольшую изменчивость и подверженность выбросам, что может негативно сказаться на приложениях, требующих предсказуемой производительности. PyTorch, ONNX, TorchScript занимают промежуточную позицию и довольно хаотичны, хотя меньше, чем TensorFlow. Выбор формата должен учитывать не только среднее время инференса, но и его стабильность, особенно в задачах реального времени.

Рисунок 4-Распределение времени инференса для разных форматов



Box Plot наглядно демонстрирует распределение времени инференса для различных форматов. OpenVINO выделяется как наиболее быстрый и стабильный формат для инференса на CPU. Он имеет самую низкую медиану и наименьший разброс. ONNX, TorchScript, PyTorch также показывают хорошие результаты. TensorFlow демонстрирует наихудшую производительность, характеризующуюся относительно высокой медианой, большим разбросом и наличием выбросов. Выбросы в TensorFlow могут быть связаны с оптимизациями, которые не всегда эффективны, или с другими процессами, влияющими на производительность фреймворка.

3.2. Результаты тестирования на GPU

При тестировании на GPU первоначально использовались форматы ONNX, OpenVINO, PyTorch, TorchScript. Формат TensorFlow был отброшен, так как при тесте на CPU показал нестабильную и медленную работу, что может указывать на потенциальные проблемы с оптимизацией TensorFlow для данной модели.

Таблица 3-Тестовый инференс на GPU

Library	Total Inference Time (s)	Peak Memory Usage (MB)	Avg. Preprocess Time (ms)	Avg. Inference Time (ms)	Avg. Postprocess Time (ms)
OpenVINO	66.056186	2579.900781	9.248135	2623.812985	9.186336
PyTorch	1.645345	2336.203906	7.290769	51.705342	6.817673
TensorRT	1.263236	2427.173438	6.857078	41.043166	2.629199
TorchScript	1.462817	2356.853906	6.926625	48.999002	2.587037

Однако, после первого тестового запуска (на 25 изображениях из датасета DOTA) оказалось, что модель OpenVINO, показавшая хорошие результаты на CPU, демонстрирует крайне низкую производительность на GPU, в 60 раз хуже остальных моделей. Это связано с тем, что OpenVINO в полной мере оптимизирован для работы на Intel-архитектуре (например, интегрированные GPU Intel или процессоры Intel с VPU), а Tesla T4, используемая в Google Colab, имеет архитектуру NVIDIA. Таким образом, OpenVINO не смогло эффективно использовать ресурсы GPU NVIDIA, что привело к столь значительному падению производительности. В связи с этим, для дальнейших тестов OpenVINO был исключён из рассмотрения как неоптимальный вариант для данной инфраструктуры.

На замену OpenVINO был добавлен формат TensorRT. TensorRT, разработанный NVIDIA, представляет собой SDK для высокоскоростного инференса глубокого обучения, особенно эффективный на GPU NVIDIA. Он оптимизирует модели за счет слияния слоев и динамического управления памятью tensor. Его основная задача - раскрыть весь потенциал NVIDIA GPU, обеспечивая максимальную скорость и эффективность, что критически важно для развертывания моделей компьютерного зрения в высокопроизводительных средах, таких как обнаружение объектов в реальном времени. Автонастройка ядра в TensorRT выбирает наиболее подходящие ядра GPU для каждого слоя модели, максимизируя использование вычислительной мощности.

Протестировав модель в форматах ONNX, PyTorch, TensorRT, TorchScript, получили следующие результаты:

Таблица 4-Результаты инференса на GPU

Format	Total Inference Time (s)	Peak Memory Usage (MB)	Avg. Preprocess Time (ms)	Avg. Inference Time (ms)	Avg. Postprocess Time (ms)
ONNX	585.570583	3295.515625	9.603325	73.038790	2.792753
PyTorch	435.282531	2918.468750	9.071319	51.707667	2.728824
TensorRT	402.753944	3223.148438	9.275291	46.872377	2.614215
TorchScript	440.548046	3298.011719	9.351965	52.136482	2.787603

Визуализация полученных результатов:

Рисунок 5-Визуализация времени инференса

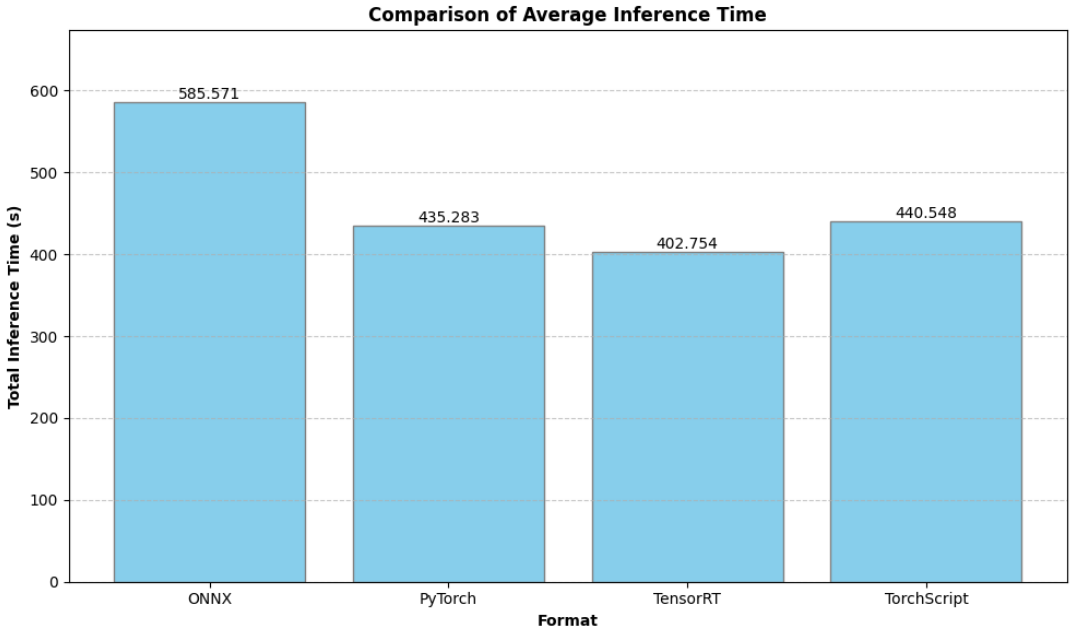
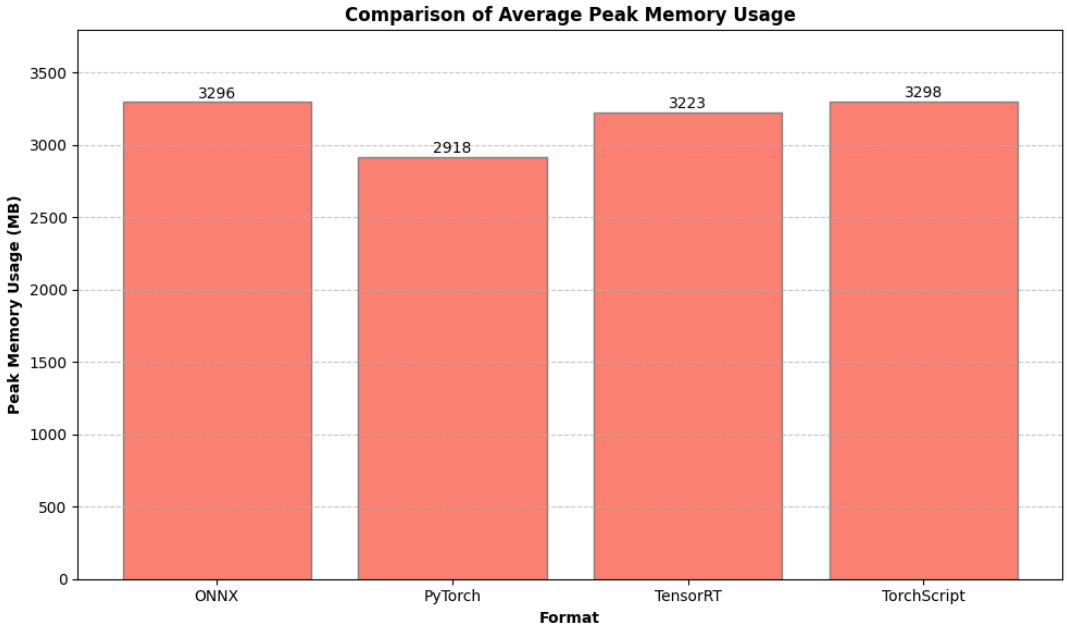
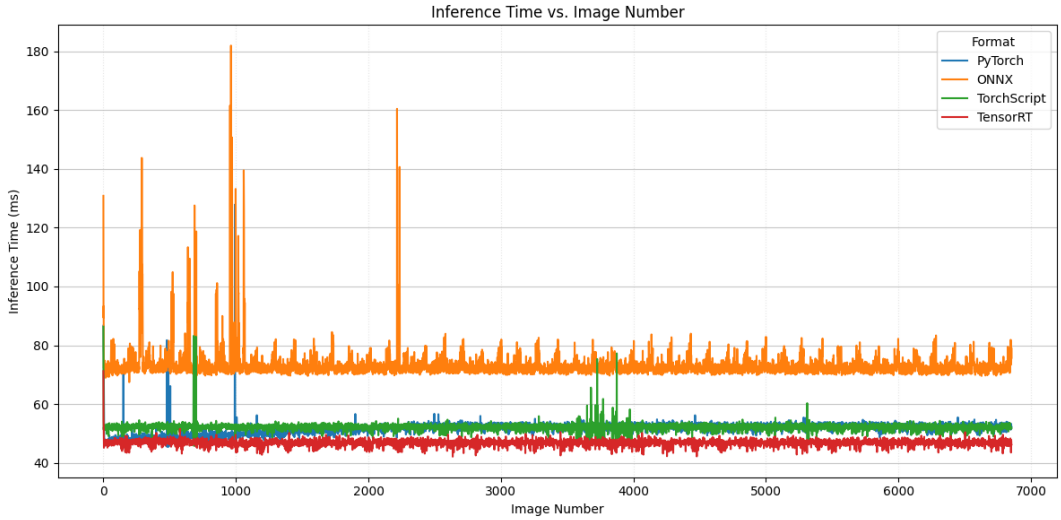


Рисунок 6-Визуализация пиковой памяти, затраченной на инференс



Результаты тестирования на GPU показали, что TensorRT обеспечивает наименьшее общее время инференса, демонстрируя лучшую производительность по сравнению с остальными форматами (ONNX, PyTorch и TorchScript). PyTorch и TorchScript показывают сопоставимые результаты, в то время как ONNX значительно уступает им по скорости. Что касается пикового потребления памяти, PyTorch демонстрирует наименьшее значение, тогда как ONNX и TorchScript показывают примерно одинаковый, немного более высокий уровень. TensorRT занимает промежуточное положение по потреблению памяти.

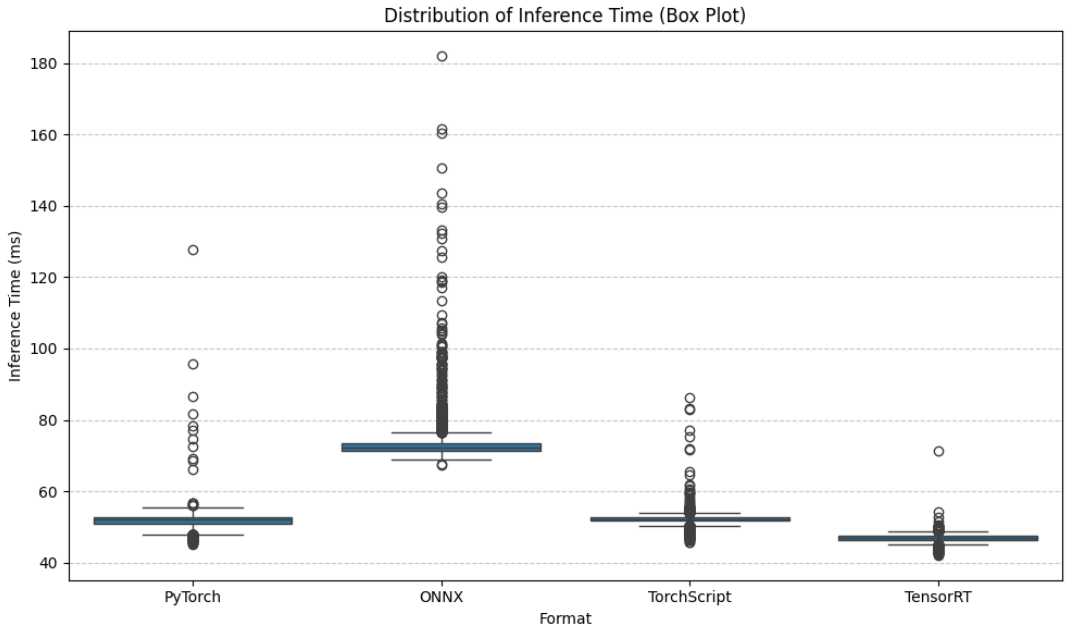
Рисунок 7-Время инференса на каждом изображении



Анализируя этот график, можно отметить, что TensorRT демонстрирует наименьшее и наиболее стабильное время инференса на протяжении всего набора данных, располагаясь в самой нижней части графика. PyTorch показывает в начале схожую производительность, но затем становится больше похож на TorchScript, этот формат демонстрирует большую вариативность. TorchScript показывает немного более высокое время инференса, но при этом сохраняет относительную стабильность. ONNX демонстрирует значительно более высокое время инференса и характеризуется наличием множественных выбросов, особенно в начале процесса, что указывает на нестабильность. Вероятно, это связано с неэффективным распределением вычислительной нагрузки

между блоками GPU. В целом, TensorRT и PyTorch обеспечивают наиболее стабильную и быструю инференцию.

Рисунок 8-Распределение времени инференса для разных форматов



Box Plot наглядно демонстрирует распределение времени инференса для различных форматов. TensorRT характеризуется наиболее компактным box и наименьшим количеством выбросов, что говорит о его стабильности и предсказуемости. PyTorch также показывает хорошую стабильность, но с немного большим количеством выбросов. TorchScript имеет сопоставимый с PyTorch медианный уровень, но с несколько большим разбросом значений. ONNX демонстрирует значительно более широкое распределение и обилие выбросов, подтверждая его нестабильность и непригодность для эффективного инференса в данном сценарии. Таким образом, TensorRT выглядит наиболее предпочтительным вариантом с точки зрения стабильности и скорости.

4. Выводы по анализу инференса

4.1 Общие сведения

В данной главе представлены ключевые выводы по результатам тестирования инференса модели YOLO11M-OBB в различных форматах на CPU и GPU. Особое внимание уделено выбору оптимального решения для веб-приложения по детекции техники на изображениях. Рассмотрены особенности работы моделей на различных вычислительных платформах и возможности дальнейшего улучшения производительности.

4.2 Выводы по работе на CPU

Наиболее релевантным форматом для нашей задачи оказался OpenVINO, потому что он показал лучшее время инференса на CPU среди всех протестированных вариантов. Основные преимущества OpenVINO:

1) Высокая производительность на CPU: OpenVINO обеспечивает значительное ускорение инференса на процессорах Intel, позволяя достичь до трехкратного увеличения скорости по сравнению с неоптимизированными моделями.

2) Оптимизация и квантизация моделей: Инструментарий OpenVINO поддерживает квантизацию моделей, что позволяет уменьшить их размер и повысить скорость инференса без существенной потери точности.

3) Режим Throughput Mode: Этот режим в OpenVINO позволяет эффективно использовать ресурсы CPU, обрабатывая несколько изображений одновременно и увеличивая общую пропускную способность системы.

4.3 Выводы по работе на GPU

На GPU наиболее эффективным оказался формат TensorRT, который обеспечил самое низкое время инференса среди всех протестированных моделей. Основные причины высокой производительности TensorRT:

1) Оптимизация под GPU от NVIDIA: TensorRT разработан специально для видеокарт NVIDIA и использует низкоуровневые оптимизации, такие как слияние слоев и высокоэффективные вычисления с плавающей запятой.

2) Поддержка динамического выделения ресурсов: TensorRT позволяет эффективно распределять память и вычислительные ресурсы, что значительно ускоряет обработку данных.

3) Оптимизация под конкретное железо: В ходе тестирования использовалась видеокарта Tesla T4 (Google Colab), которая хорошо поддерживает TensorRT. Это может объяснять значительное преимущество в скорости по сравнению с другими форматами.

При этом OpenVINO на GPU показал неудовлетворительные результаты, поэтому его пришлось исключить из анализа. Вероятно, это связано с тем, что OpenVINO ориентирован в первую очередь на графические процессоры Intel, а в Google Colab используется оборудование NVIDIA.

4.4 Возможности дальнейшего развития

В качестве направления для дальнейшего улучшения производительности можно рассмотреть реализацию гибридного подхода к инференсу, используя преимущества как CPU, так и GPU:

- Параллельная обработка: Распределение задач между CPU и GPU в зависимости от их сложности и требуемого времени обработки позволит оптимально использовать доступные ресурсы и повысить общую производительность системы.

- Автоматический выбор оптимального бэкенда: Разработка механизма, который будет автоматически определять наиболее подходящий аппаратный ресурс для выполнения конкретной задачи инференса, исходя из текущей нагрузки и доступности ресурсов.

Реализация таких подходов позволит еще более эффективно использовать вычислительные ресурсы и повысить скорость детекции техники на фотографиях в нашем веб-приложении.

5. Источники

1. Ultralytics. YOLO: Real-Time Object Detection. URL: https://ultralytics.com

2. DOTA: HichTala/dota URL: https://huggingface.co/datasets/HichTala/dota

3. Intel. OpenVINO Toolkit Documentation. URL: https://docs.openvino.ai

4. NVIDIA. TensorRT Documentation. URL: https://docs.nvidia.com/deeplearning/tensorrt

5. PyTorch Documentation. URL: https://pytorch.org/docs/stable/index.html

6. ONNX Documentation. URL: https://onnx.ai