«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)

Направление	01.03.02 Прикладная математика и информатика
Профиль	Без профиля
Факультет	КТИ
Кафедра	МО ЭВМ

К защите допустить
Зав. кафедрой Кринкин К.В.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: ИСПОЛЬЗОВАНИЕ OPENVINO TOOLKIT ДЛЯ РАСПРЕДЕЛЕННОЙ ОБРАБОТКИ ИНФОРМАЦИИ

Студент		_____ *подпись*	Киреев К.А.
Руководитель	к.т.н., доцент	_____ *подпись*	Черниченко Д.А.
Консультанты	ст. преподаватель	_____ *подпись*	Житенева М.И.
	к.т.н., доцент	_____ *подпись*	Заславский М.М.

Санкт-Петербург
2022

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Утверждаю
Зав. кафедрой МО ЭВМ
_____ Кринкин К.В.
«___»_____2022 г.

Студент Киреев К.А. Группа 8383

Тема работы: Использование OpenVINO Toolkit для распределенной обработки информации

Место выполнения ВКР: Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)

Исходные данные (технические требования):

Разработка метода оценки оптимизации инференса с помощью средств для распределенной обработки информации инструмента OpenVINO Toolkit

Содержание ВКР:

Обзор предметной области, Intel OpenVINO Toolkit, Применение OpenVINO Toolkit, Анализ эффективности, Экономическое обоснование ВКР.

Перечень отчетных материалов: пояснительная записка, иллюстративный материал.

Дополнительные разделы: Экономическое обоснование ВКР.

Дата выдачи задания
«20» апреля 2022 г.

Дата представления ВКР к защите
«6» июня 2022 г.

Студент _____ Киреев К.А.

Руководитель к.т.н., доцент _____ Черниченко Д.А.

КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Утверждаю
Зав. кафедрой МО ЭВМ
_____ Кринкин К.В.
«___»_____2022 г.

Студент Киреев К.А. Группа 8383

Тема работы: Использование OpenVINO Toolkit для распределенной обработки информации

№ п/п	Наименование работ	Срок выполнения
1	Обзор литературы по теме работы	20.04 – 25.04
2	Аналитическая часть	26.04 – 06.05
3	Разработка метода оценки оптимизации инференса	07.05 – 10.05
4	Создание демо-приложения	11.05 – 14.05
4	Анализ эффективности оптимизации	15.05 – 18.05
5	Результаты исследования	19.05 – 20.05
6	Экономическое описание ВКР	21.05 – 22.05
7	Оформление пояснительной записки	23.05 – 24.05
8	Оформление иллюстративного материала	25.06 – 26.06
9	Предзащита	27.06

Студент _____ Киреев К.А.

Руководитель к.т.н., доцент _____ Черниченко Д.А.

РЕФЕРАТ

Пояснительная записка 87 стр., 52 рис., 12 табл., 18 ист.

ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ, ОПТИМИЗАЦИЯ ИНФЕРЕНСА, INTEL OPENVINO TOOLKIT, РАСПРЕДЕЛЕННАЯ ОБРАБОТКА ИНФОРМАЦИИ.

Объектом исследования является функционал OpenVINO Toolkit для распределенной обработки данных.

Предметом исследования является процесс оптимизации инференса нейросетей с помощью плагинов для распределенной обработки.

Цель работы: исследовать функционал OpenVINO Toolkit и проанализировать эффективность оптимизации инференса нейронных сетей при использовании данного инструмента в случае распределенной обработки информации.

В настоящей выпускной квалификационной работе был изучен функционал OpenVINO Toolkit для распределенной обработки информации, разработаны методы для оценки оптимизации инференса с помощью данного инструмента, разработано демо-приложение для демонстрации работы и оценки оптимизации, проанализирована эффективность оптимизации инференса OpenVINO Toolkit с использованием гетерогенного и мультидевайсного плагинов.

ABSTRACT

Explanatory note 87 p, 52 fig, 12 tables, 18 sources.

ARTIFICIAL NEURAL NETWORKS, INFERENCE OPTIMIZATION, INTEL OPENVINO TOOLKIT, DISTRIBUTED DATA PROCESSING.

The object of this research is functionality of OpenVINO Toolkit for distributed data processing.

The subject of the research work is a process of neural network inference optimization with the help of plugins for distributed processing.

The goal of the work is to explore the functionality of OpenVINO Toolkit and analyze the efficiency of neural network inference optimization when using this tool in the case of distributed processing of information.

As part of this work, the functionality of OpenVINO Toolkit for distributed information processing was studied, methods were developed for evaluating inference optimization using this tool, a demo application was developed to demonstrate the work and evaluate optimization, and the effectiveness of OpenVINO Toolkit inference optimization using heterogeneous and multi-device plug-ins was analyzed.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей выпускной квалификационной работе применены следующие сокращения и обозначения:

API (Application Programming Interface или интерфейс программирования приложений) – это совокупность инструментов и функций в виде интерфейса для создания новых приложений, благодаря которому одна программа будет взаимодействовать с другой.

Батч – количество изображений для анализа во время одного вызова метода infer.

Искусственная нейронная сеть – математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей.

Операция – абстрактное понятие математической функции, выбранной для определенной цели, например, свертка.

Свертка – операция, при которой ядро проходит над двумерным изображением, поэлементно выполняя операцию умножения с той частью входных данных, над которой оно сейчас находится, и затем суммирует все полученные значения в один выходной пиксель.

GAN (Generative Adversarial Network) – алгоритм машинного обучения без учителя, построенный на комбинации из двух нейронных сетей, одна из которых генерирует образцы, а другая старается отличить правильные образцы от неправильных.

Фреймворк – программная среда специального назначения, используемая для того, чтобы значительно облегчить процесс объединения определенных компонентов при создании программ.

Инференс – запуск готовой натренированной нейронной сети как готовой программы.

Model Zoo – набор бесплатных, предварительно обученных моделей глубокого обучения и демо-приложений, которые предоставляют шаблоны приложений, предназначенных для специфичной реализации глубокого обучения на Python и C++.

Model Optimizer – кроссплатформенный инструментарий командной строки, который упрощает переход между средой обучения и средой развертывания, осуществляет статический анализ модели и настраивает модели глубокого обучения для оптимального исполнения на целевых устройствах.

IR (Intermediate representation или промежуточное представление) – специальный формат представления нейронной сети, состоящий из xml и bin-файлов, разработанный Intel, с которым работает Inference Engine и все инструменты, которые есть в OpenVINO Toolkit.

Inference Engine – набор библиотек C++ с привязкой к C и Python, предоставляющих общий API для создания решений для инференса на определенной платформе. API Inference Engine используется для чтения промежуточного представления и выполнения модели на целевых устройствах. API Inference Engine позволяет: прочитать модель из файлов; загрузить модель в плагин, работающий с конкретным устройством; отправить данные для обработки; получить результаты обработки.

clDNN (Compute Library for Deep Neural Networks) – вычислительная библиотека для глубоких нейронных сетей.

CPU (Central Processing Unit) – центральный процессор.

GPU (Graphics Processing Unit) – графический процессор.

VPU (Vision Processing Unit) – процессор машинного зрения.

FPGA (Field-Programmable Gate Array) – программируемая логическая интегральная схема.

GNA (Gaussian & Neural Accelerator) – процессор для обработки звука.

Model Downloader – скрипт, загружающий файлы моделей из онлайн-источников и, при необходимости, исправляющий их, чтобы сделать более пригодными для использования в Model Optimizer

Model Quantizer – скрипт, который квантизует модели полной точности в формате IR в версии с низкой точностью.

Квантизация – метод для уменьшения размера нейронной сети в оперативной памяти.

FP (Floating Point) – плавающая точка.

nGraph – внутреннее представление графов в инструментарии OpenVINO Toolkit, которое используется для построения модели из исходного кода.

Пропускная способность – количество данных, обработанных за определенный период времени.

FPS (Frames Per Second) – средняя скорость обработки видеокадров (кадров в секунду).

Задержка инференса – среднее время, необходимое для обработки одного кадра (время от начала инференса входных данных до получения результатов).

OpenCV – библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом.

Multi-device плагин – плагин, который позволяет автоматически распределить запросы на инференс по доступным вычислительным устройствам для параллельного выполнения запросов.

Гетерогенный плагин – плагин, который позволяет производить инференс нейронной сети на нескольких устройствах, то есть отправлять различные слои на другой девайс, например, слои, не поддерживаемые определенным устройством; в отличие от Multi-device плагина, гетерогенный не позволяет проводить инференс параллельно на нескольких устройствах.

Benchmark App – утилита OpenVINO Toolkit, позволяющая с помощью специфичных методологий, разработанных Intel, измерить скорость работы сети.

ВВЕДЕНИЕ

Компьютерное зрение – одна из тех технологий, которые в последние несколько лет развиваются скачками. Если оглянуться на 10 лет назад, то это было не так, поскольку данная область была скорее темой академического интереса. Однако сейчас компьютерное зрение явно является движущей силой и помощником искусственного интеллекта. Основной проблемой данного подхода является большое количество требуемых ресурсов, что вызывает некоторые сложности при использовании его на практике, такие как необходимость больших финансовых вложений. Они необходимы не только при использовании самого алгоритма, но и для его обучения и сбора данных для этого же обучения, для чего требуется еще и непомерное количество времени.

Решая эту проблему компания Intel создала программное обеспечение OpenVINO Toolkit, позволяющее оптимизировать уже созданные ранее модели и выполнять их на вычислительных устройствах компании Intel с гораздо большой эффективностью. Данный инструмент предоставляет возможность использования нескольких устройств для инференса нейронных сетей. Использование Intel OpenVINO Toolkit для распределенной обработки данных позволяет:

- Использовать мощность ускорителей для обработки наиболее тяжелых частей сети и выполнять неподдерживаемые уровни на резервных устройствах, таких как CPU.
- Эффективнее использовать все доступные аппаратные средства в течение одного инференса, что дает более стабильную производительность, так как устройства разделяют нагрузку на выводы
- Повышение пропускной способности за счет использования нескольких устройств (по сравнению с выполнением на одном устройстве).

Цель работы: проанализировать эффективность оптимизации инференса инструмента OpenVINO Toolkit в случае распределенной обработки информации.

Для достижения цели необходимо решить следующий **задачи** – изучить функционал OpenVINO Toolkit для распределенной обработки данных; разработать метод оценки оптимизации инференса; проанализировать эффективность оптимизации инференса данного инструмента в случае использования гетерогенного и мультидевайсного плагинов.

Объектом исследования является функционал OpenVINO Toolkit для распределенной обработки данных.

Предметом исследования является процесс оптимизации инференса нейросетей с помощью плагинов для распределенной обработки.

Практическая значимость исследования заключается в сокращении требуемых для использования глубокого обучения ресурсов, как денежных, так и вычислительных и увеличении производительности моделей глубокого обучения. Результаты исследования показывают, что использование оптимизированной модели обеспечивает хорошую точность при значительном увеличении производительности на этапе инференса, используя CPU+GPU по сравнению со стандартным использованием фреймворка TensorFlow.

ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обработка изображений

Изображения определяют мир, каждое изображение имеет свою собственную историю, оно содержит много важной информации, которая может быть полезна во многих отношениях. Эта информация может быть получена с помощью техники, известной как обработка изображений.

Это основная часть компьютерного зрения, которая играет решающую роль во многих примерах реального мира, таких как робототехника, самоуправляемые автомобили и обнаружение объектов. Обработка изображений позволяет нам преобразовывать и манипулировать тысячами изображений одновременно и извлекать из них полезные сведения. Она имеет широкий спектр применения практически во всех областях.

Как видно из названия, обработка изображений означает обработку изображения, которая может включать в себя множество различных методов, пока мы не достигнем нашей цели.

Конечный результат может быть либо в виде изображения, либо в виде соответствующей характеристики этого изображения. Это может быть использовано для дальнейшего анализа и принятия решений.

Изображение можно представить в виде двумерной функции $F(x, y)$, где x и y – пространственные координаты. Амплитуда F при определенном значении x, y называется интенсивностью изображения в этой точке. Если x, y и значение амплитуды конечны, то мы называем это цифровым изображением. Оно представляет собой массив пикселей, расположенных в столбцах и строках. Пиксели – это элементы изображения, которые содержат информацию об интенсивности и цвете. Изображение также может быть представлено в 3D, где x, y и z становятся пространственными координатами. Пиксели располагаются в виде матрицы. Такое изображение известно, как RGB-изображение [1].

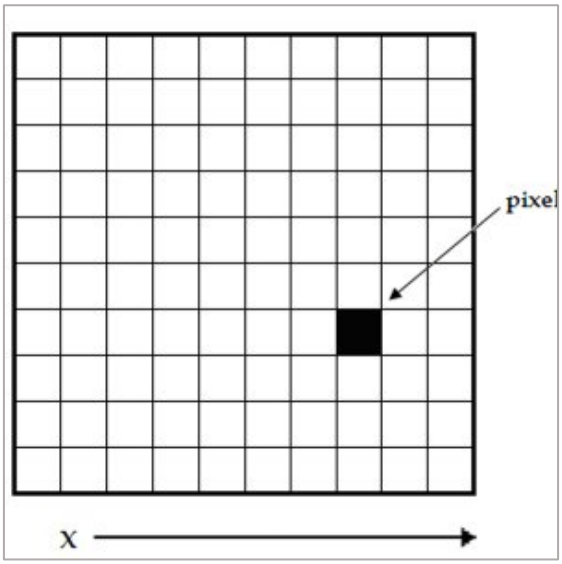

Рисунок 1.1 – Двумерный массив пикселей

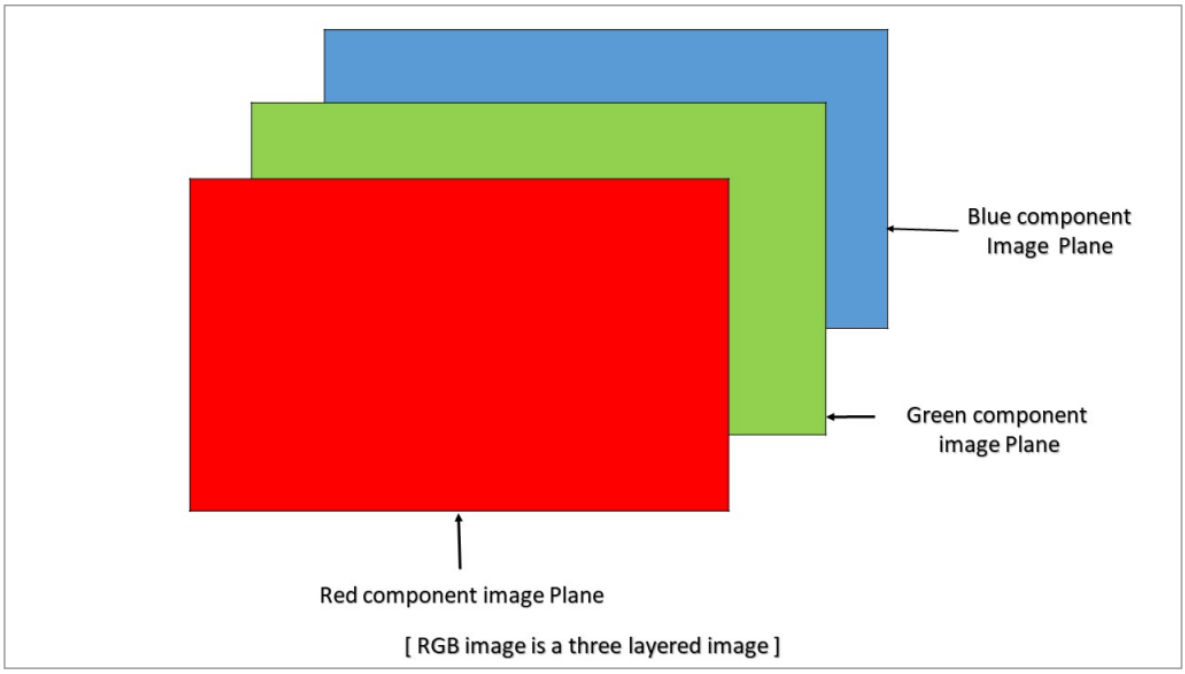

Рисунок 1.2 – RGB-изображение

Существуют различные типы изображений:

- RGB-изображение: содержит три слоя двухмерного изображения – красный, зеленый и синий каналы.
- Изображение в оттенках серого: эти изображения содержат оттенки черного и белого цветов и содержат только один канал.

1.2 Классические алгоритмы обработки изображений

1.2.1 Морфологическая обработка изображений

Морфологическая обработка изображений пытается удалить недостатки из бинарных изображений, поскольку бинарные области, полученные простым пороговым выделением, могут быть искажены шумом. Она также помогает сгладить изображение с помощью операций открытия и закрытия.

Морфологические операции могут быть распространены на полутоновые изображения. Они состоят из нелинейных операций, связанных со структурой особенностей изображения. Они зависят не от связанного упорядочивания пикселей, а от их числовых значений. Эта техника анализирует изображение с помощью небольшого шаблона, известного как структурирующий элемент, который размещается в различных возможных местах изображения и сравнивается с соответствующими соседними пикселями. Структурирующий элемент – это небольшая матрица со значениями 0 и 1. [2]

Рассмотрим две фундаментальные операции морфологической обработки изображений – дилатацию и эрозию:

- Операция *дилатации* добавляет пиксели к границам объекта на изображении
- Операция *эрозии* удаляет пиксели от границ объекта.

Количество пикселей, удаленных или добавленных к исходному изображению, зависит от размера структурирующего элемента.

Структурирующий элемент – это матрица, состоящая только из 0 и 1, которая может иметь произвольную форму и размер. Она размещается во всех возможных местах изображения и сравнивается с соответствующей окрестностью пикселей.

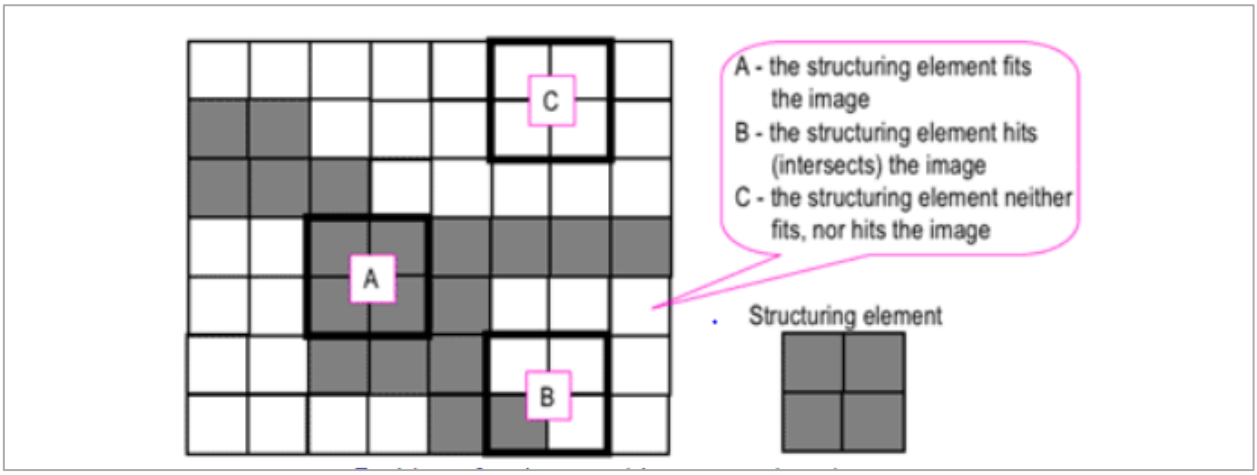

Рисунок 1.3 – Квадратное структурирование

Квадратный структурирующий элемент 'A' помещается в объект, который мы хотим выделить, 'B' пересекает объект, а 'C' находится вне объекта.

Шаблон "ноль-один" определяет конфигурацию структурирующего элемента. Она соответствует форме объекта, который мы хотим выбрать. Центр структурирующего элемента определяет обрабатываемый пиксель.

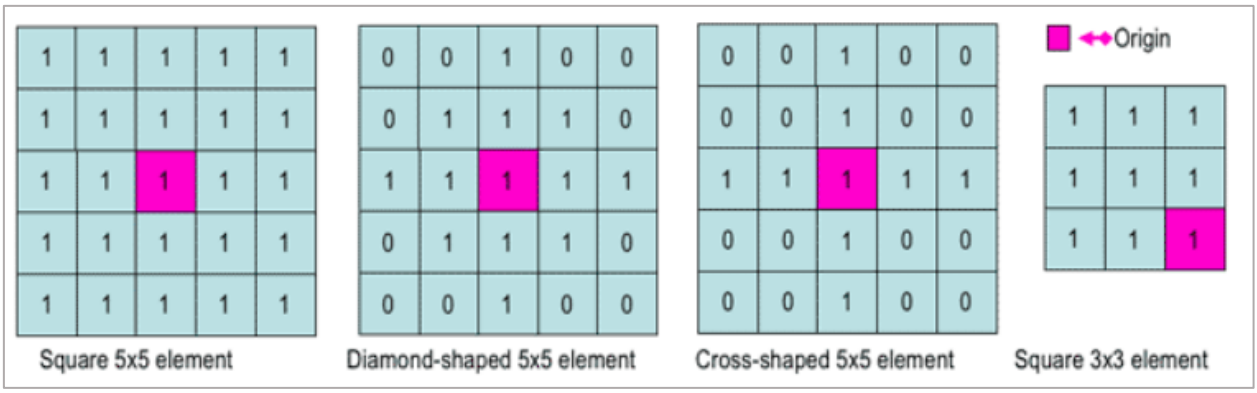

Рисунок 1.4 – Квадратное структурирование

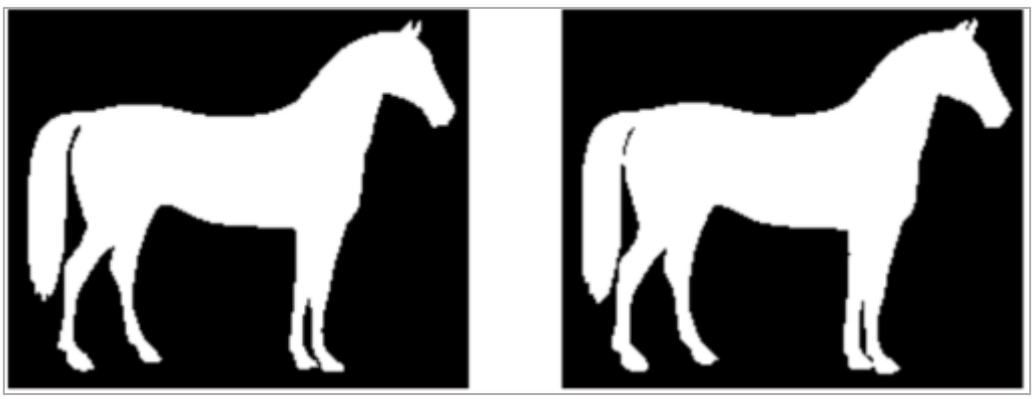

Рисунок 1.5 – Дилатация

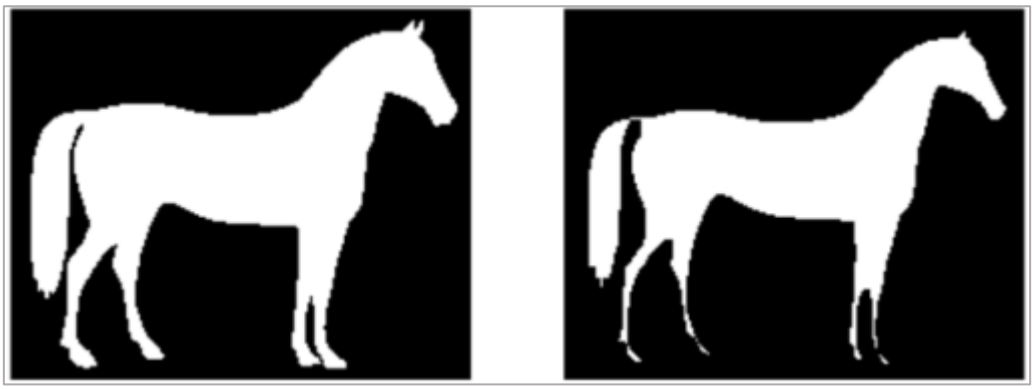

Рисунок 1.6 – Эрозия

1.2.2 Гауссова обработка изображений

Размытие по Гауссу, которое также известно, как гауссово сглаживание, является результатом размытия изображения с помощью гауссовой функции.

Оно используется для уменьшения шума изображения и уменьшения деталей. Визуальный эффект этой техники размытия похож на взгляд на изображение через полупрозрачный экран. Иногда она используется в компьютерном зрении для улучшения изображений в различных масштабах или как техника преобразования данных в глубоком обучении.

Базовая гауссова функция имеет вид:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

Рисунок 1.7 – Гауссова функция

На практике лучше всего воспользоваться таким свойством гауссова размытия как сепарабельность, разделив процесс на два прохода. В первом проходе используется одномерное ядро для размытия изображения только в горизонтальном или вертикальном направлении. Во время второго прохода то же самое одномерное ядро используется для размытия в остальных направлениях. В результате получается тот же эффект, что и при свертке с

двумерным ядром за один проход. Чтобы понять, что именно гауссовы фильтры делают с изображением, был рассмотрен следующий пример.

Если есть нормально распределенный фильтр, то при его применении к изображению результаты выглядят следующим образом:

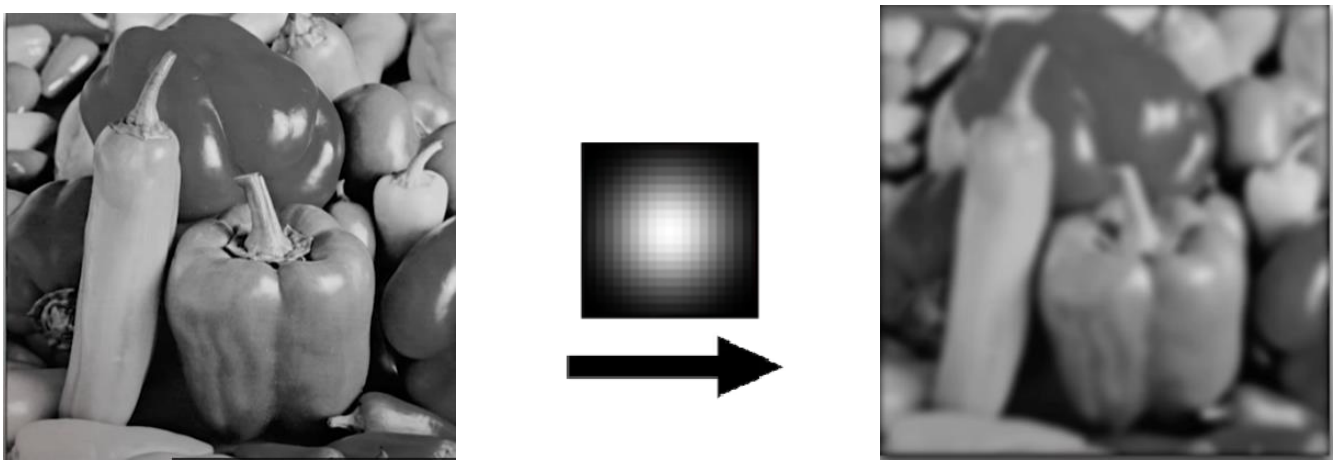

Рисунок 1.8 – Слева направо: оригинал, гауссовый фильтр, результат

Видно, что некоторые края имеют немного меньше деталей. Фильтр придает больший вес пикселям в центре, чем пикселям вдали от центра. Гауссовы фильтры являются фильтрами низких частот, то есть ослабляют высокие частоты. Он широко используется при обнаружении границ.

1.2.3 Преобразование Фурье в обработке изображений

Преобразование Фурье разбивает изображение на синусоидальную и косинусоидальную составляющие.

Оно имеет множество применений, таких как реконструкция изображений, сжатие изображений или фильтрация изображений.

Поскольку рассматриваются изображения, будет рассматриваться дискретное преобразование Фурье.

Рассмотрим синусоиду, она состоит из трех составляющих:

- Величина – связана с контрастом
- Пространственная частота – связана с яркостью

- Фаза – связана с информацией о цвете

Изображение в частотной области выглядит следующим образом:

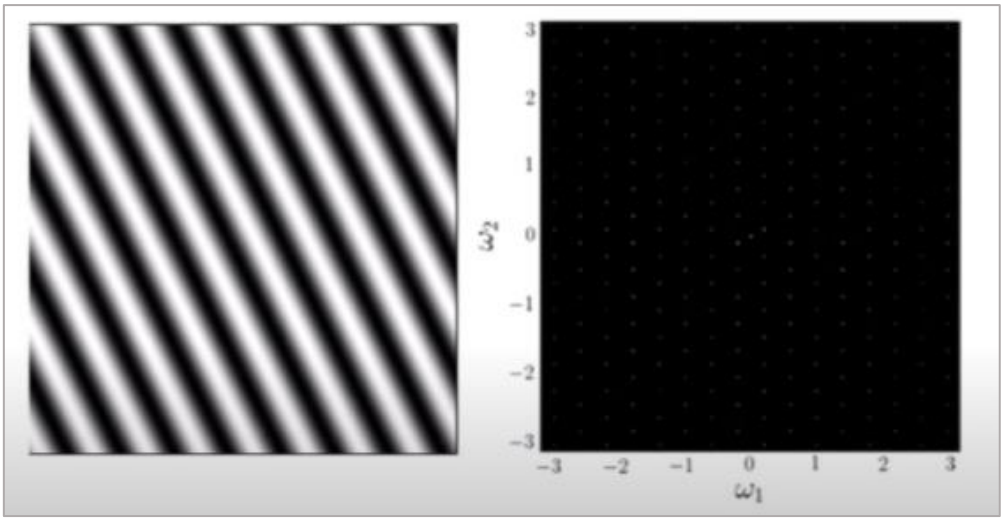

Рисунок 1.9 – Изображение в частотной области

Формула для двумерного дискретного преобразования Фурье имеет вид:

$$F(u, v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)\, e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

В приведенной выше формуле $f(x, y)$ обозначает изображение.

Обратное преобразование Фурье преобразует преобразование обратно в изображение. Формула двумерного обратного дискретного преобразования Фурье имеет вид:

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v)\, e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

1.2.4 Обнаружение краев в обработке изображений

Определение краев – это метод обработки изображений для нахождения границ объектов на изображениях. Он работает путем обнаружения перепадов яркости.

Это может быть очень полезно для извлечения полезной информации из изображения, поскольку большая часть информации о форме заключена в краях. Классические методы определения краев работают путем обнаружения разрывов в яркости.

Они могут быстро отреагировать, если на изображении обнаружен шум, при обнаружении изменений уровней серого. Края определяются как локальные максимумы градиента.

Наиболее распространенным алгоритмом обнаружения краев является алгоритм обнаружения краев Собеля. Оператор обнаружения Собеля использует ядра свертки 3 x 3. Простое ядро Gx и повернутое на 90 градусов ядро Gy. Отдельные измерения производятся путем применения обоих ядер по отдельности к изображению.

$$Gx = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * Image\ matrix$$

Рисунок 1.10 – Изображение Gx

$$Gy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * Image\ matrix$$

Рисунок 1.11 – Изображение Gy

, где $*$ обозначает двумерную операцию свертки

Результирующий градиент может быть рассчитан как:

$$G = sqrt(Gx^2 + Gy^2)$$

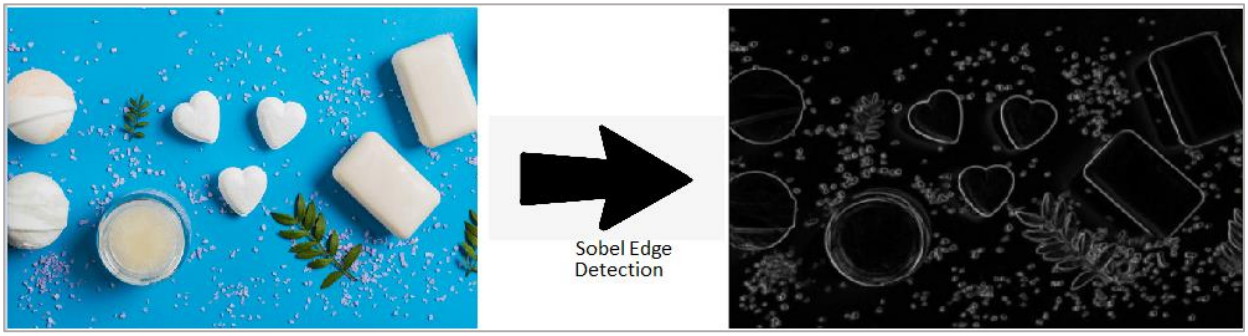

Рисунок 1.12 – Определение краев

1.3 Обработка изображений с помощью нейронных сетей

1.3.1 Нейронные сети

Нейронные сети – это многослойные сети, состоящие из нейронов или узлов. Эти нейроны являются основными вычислительными единицами нейронной сети. Они разработаны таким образом, чтобы действовать подобно человеческому мозгу. Они принимают данные, обучаются распознавать закономерности в данных и затем предсказывают результат.

Стандартная нейронная сеть состоит из трех слоев:

- Входной слой
- Скрытый слой
- Выходной слой

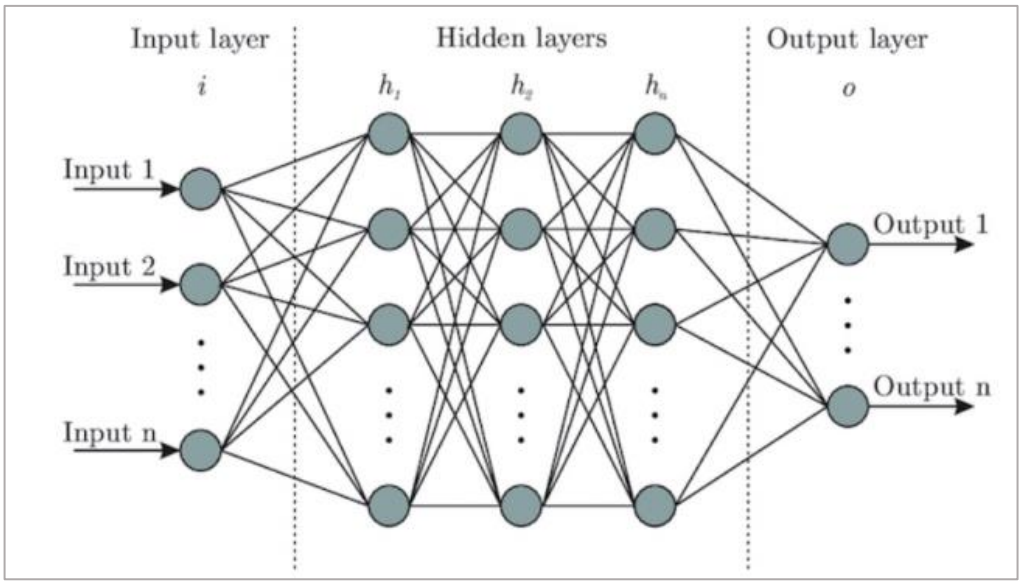

Рисунок 1.13 – Стандартная нейронная сеть

Входные слои получают входные данные, выходной слой прогнозирует выходной сигнал, а скрытые слои выполняют большую часть вычислений. Количество скрытых слоев может быть изменено в соответствии с требованиями. В нейронной сети должен быть как минимум один скрытый слой.

Принцип работы нейронной сети:

1. Рассмотрим изображение – каждый пиксель подается на вход каждого нейрона первого слоя, нейроны одного слоя соединены с нейронами следующего слоя через каналы.
2. Каждому из этих каналов присваивается числовое значение, известное как вес.
3. Входы умножаются на соответствующие веса, и эта взвешенная сумма подается на вход скрытых слоев.
4. Выходные данные скрытых слоев проходят через функцию активации, которая определяет, будет ли конкретный нейрон активирован или нет.
5. Активированные нейроны передают данные на следующие скрытые слои. Таким образом, данные распространяются по сети, что известно, как прямое распространение.
6. В выходном слое нейрон с наибольшим значением предсказывает выход. Эти выходы являются значениями вероятности.
7. Предсказанный выход сравнивается с фактическим выходом для получения ошибки. Эта информация затем передается обратно через сеть, этот процесс известен как обратное распространение ошибки.
8. На основе этой информации корректируются веса. Этот цикл прямого и обратного распространения выполняется несколько раз на нескольких входах, пока сеть не предскажет выход правильно в большинстве случаев.
9. На этом процесс обучения нейронной сети заканчивается. В некоторых случаях время обучения нейронной сети может оказаться существенным.

На рисунке ниже a_i – набор входных значений, w_i – веса, z – выходное значение, а g – функция активации.

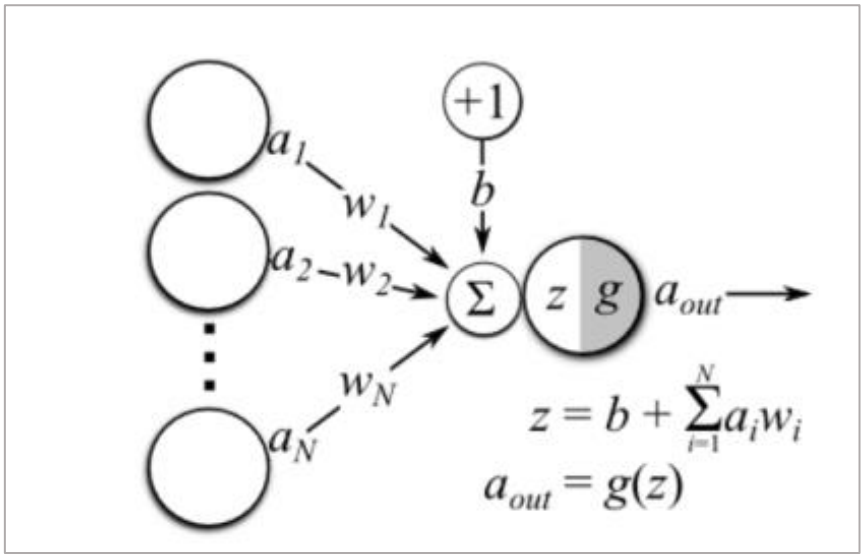

Рисунок 1.14 – Операции в одном нейроне

Можно выделить несколько рекомендаций по подготовке данных для обработки изображений.

- Для получения лучших результатов в модель необходимо подавать больше данных.
- Набор данных изображений должен быть высокого качества, чтобы получить более четкую информацию, но для их обработки могут потребоваться более глубокие нейронные сети.
- Во многих случаях RGB-изображения преобразуются в градации серого перед подачей их в нейронную сеть.

1.3.2 Типы нейронных сетей в обработке изображений

Свёрточная нейронная сеть

Свёрточная нейронная сеть имеет три типа слоёв:

- Слой свёртки (Conv) – это основной строительный блок CNN, он отвечает за выполнение операции свертки. Элемент, участвующий в выполнении операции свертки в этом слое, называется ядро/фильтр (матрица). Ядро выполняет горизонтальные и вертикальные сдвиги, пока не будет пройдено все изображение.

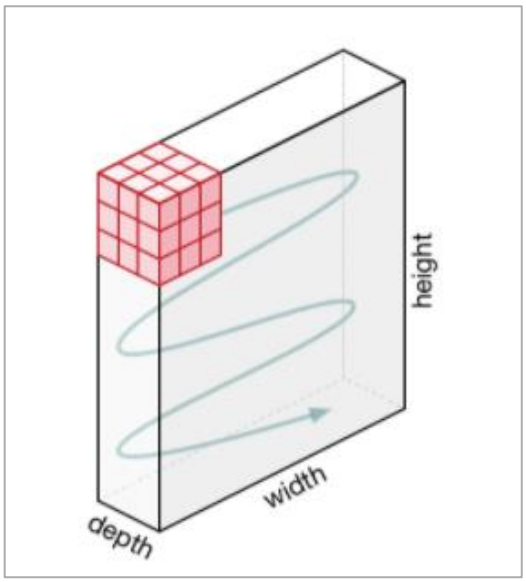

Рисунок 1.15 – Движение ядра

o Слой субдискретизации или пулинг (Pool) – этот слой отвечает за снижение размерности. Он помогает уменьшить вычислительную мощность, необходимую для обработки данных. Существует два типа пулинга: MaxPooling и AveragePooling. Первый возвращает максимальное значение из области, покрытой ядром на изображении. Второй же возвращает среднее значение всех значений в части изображения, покрытой ядром.

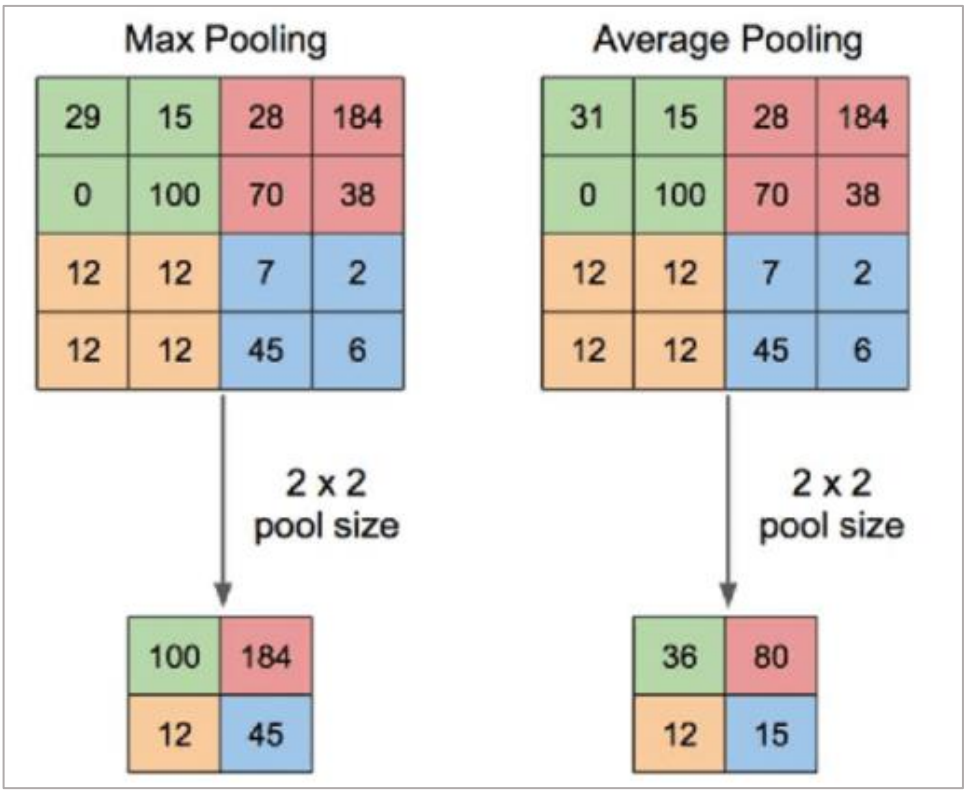

Рисунок 1.16 – Операция пулинга

- Полносвязная нейронная сеть (FC) – полносвязная нейронная сеть работает через входной слой Flatten, где каждый вход подключен ко всем нейронам. При наличии в сети, данные слои обычно находятся в конце архитектуры CNN.

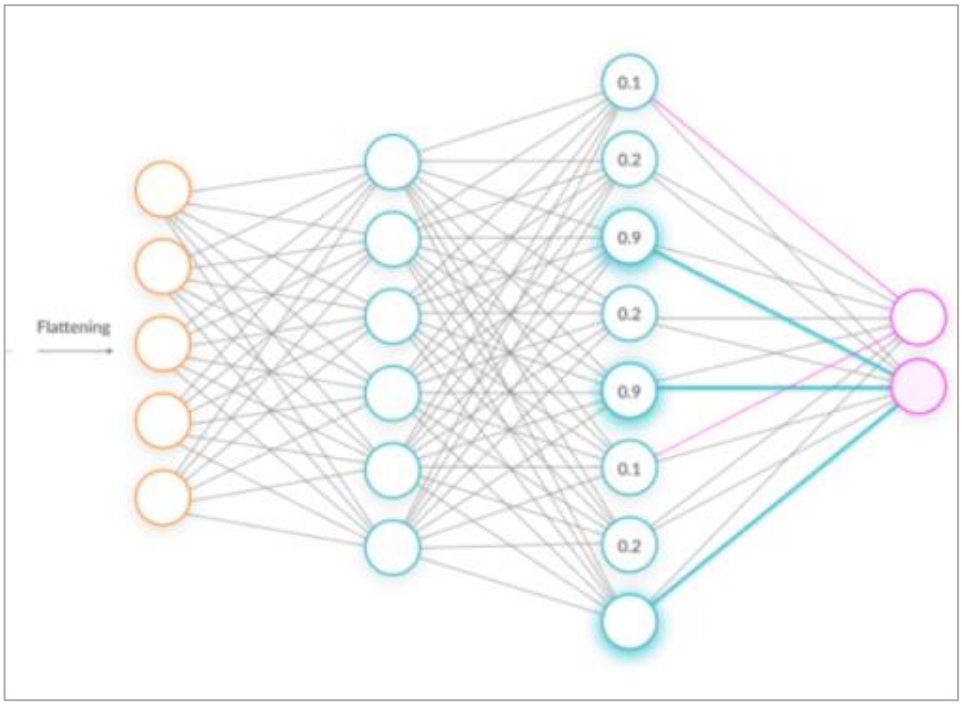

Рисунок 1.17 – Полносвязная нейронная сеть

CNN в основном используется для извлечения особенностей из изображения с помощью своих слоев. CNN широко используются для классификации изображений, где каждое входное изображение пропускается через ряд слоев для получения вероятностного значения между 0 и 1.

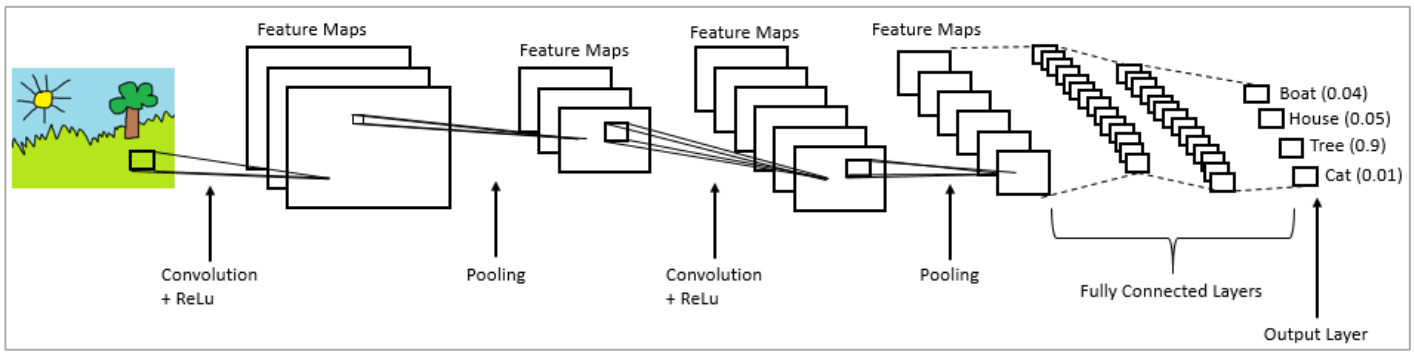

Рисунок 1.18 – CNN-обработка изображений

Генеративно-состязательная сеть (GAN)

Генеративные модели используют подход обучения без учителя (есть изображения, но нет меток).

GAN состоят из двух моделей Generator (Генеративная модель) и Discriminator (Дискриминативная модель). Генератор учится создавать поддельные изображения, которые выглядят реалистично, чтобы обмануть дискриминатор, а дискриминатор учится отличать поддельные изображения от настоящих (старается не быть обманутым).

Генератору не разрешается видеть реальные изображения, поэтому на начальном этапе он может давать плохие результаты, в то время как дискриминатору разрешается смотреть на реальные изображения, но они перемешаны с поддельными, созданными генератором, которые он должен классифицировать как настоящие или поддельные.

На вход генератора подается некоторый шум, чтобы он мог каждый раз выдавать разные примеры, а не однотипные изображения. На основе оценок, предсказанных дискриминатором, генератор пытается улучшить свои результаты, через определенное время генератор сможет производить изображения, которые будет сложнее отличить. Дискриминатор также совершенствуется, так как получает от генератора все более реалистичные изображения на каждом раунде.

Популярными типами GAN являются:

- Глубокие сверточные GAN (DCGAN)
- Условные GAN (cGAN)
- StyleGAN
- CycleGAN
- DiscoGAN
- GauGAN

GAN отлично подходят для генерации и обработки изображений. Некоторые применения GAN включают в себя: состаривание лица, смешивание фотографий, Super Resolution, восстановление изображения.

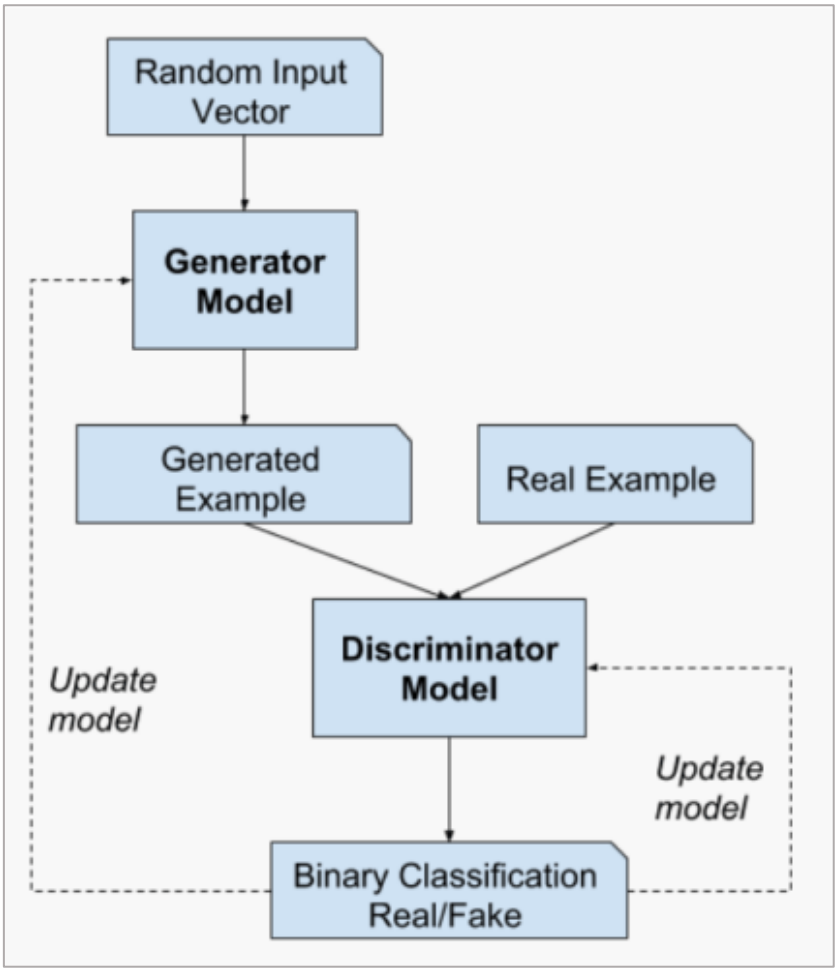

Рисунок 1.19 – Принцип работы GAN

ГЛАВА 2. INTEL OPENVINO TOOLKIT

Общее определение Intel OpenVINO Toolkit можно обозначить, как набор полезных инструментов, которые объединяют обработку видео, компьютерное зрение, обработку естественного языка, машинное обучение и оптимизацию нейронных сетей в единый пакет. [3-6]

Но, выделив самую главную мысль и говоря простым языком можно сказать, что *OpenVINO* – это набор инструментов, который призван максимально эффективно запустить нейронную сеть на вашем железе, то есть выполнить какие-то оптимизации, чтобы все работало максимально быстро на предоставляемых девайсах.

2.1 Процесс разработки

2.1.1 Процесс разработки нейронной сети

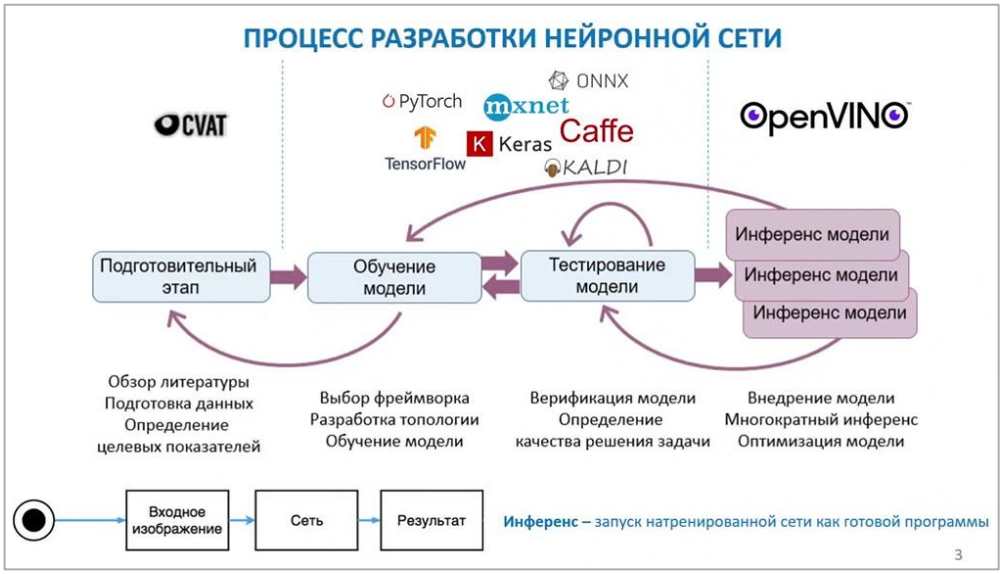

Рисунок 2.1 – Процесс разработки нейронной сети

Процесс разработки нейронной сети можно разделить на три(четыре) этапа:

1. Подготовительный
   - Чтение литературы на предмет поиска близких для решения задачи архитектур;

- Определение целевых показателей, которые необходимо достичь по точности
- Подготовка данных является самым важным этапом, так как не всегда существует датасет, который подходит под конкретную задачу, иногда его приходится создавать самостоятельно – для создания датасета, относящегося к области компьютерного зрения у Intel существует инструмент CVAT для разметки данных [7]

2. Обучение и тестирование модели
   - Выбор фреймворка
   - Разработка топологии сети / выбор готовой
   - Обучение модели
   - Определить качества решения задачи (этим занимаются фреймворки)
3. Исполнение – инференс модели

Инференс – запуск готовой натренированной сети как готовой программы.

Важно заметить, что OpenVINO toolkit не занимается ни обучением, ни тестированием, а направлен только на инференс.

2.1.2 Фреймворки и OpenVINO

Фреймворки можно использовать для инференса, но делать этого не стоит по нескольким причинам [8]:

1. Фреймворки обычно тяжеловесны – содержат большое количество библиотек, большое количество кода, которое направлено как раз на обучение и тестирование моделей
2. Не имеют никаких оптимизаций
3. Фреймворки часто не задействуют весь потенциал аппаратного обеспечения – например во фреймворке TensorFlow, чтобы задействовать GPU надо устанавливать дополнительные драйвера, то есть во фреймворк они не встроены

В свою очередь OpenVINO toolkit имеет следующие плюсы:

1. Использует облегченную среду исполнения, которая отвечает только за инференс модели, то есть исполнение нейронной сети и не привносит за собой какие-либо модули, отвечающие за обучение и тестирование модели
2. Software оптимизации – например оптимизация топологии сети
3. Оптимизации под железо Intel
4. Один API для всех целей
5. Model Zoo – можно найти модели обученные Intel и сторонними людьми, но оптимизированные для OpenVINO Toolkit [9].

2.1.3 Архитектура OpenVINO

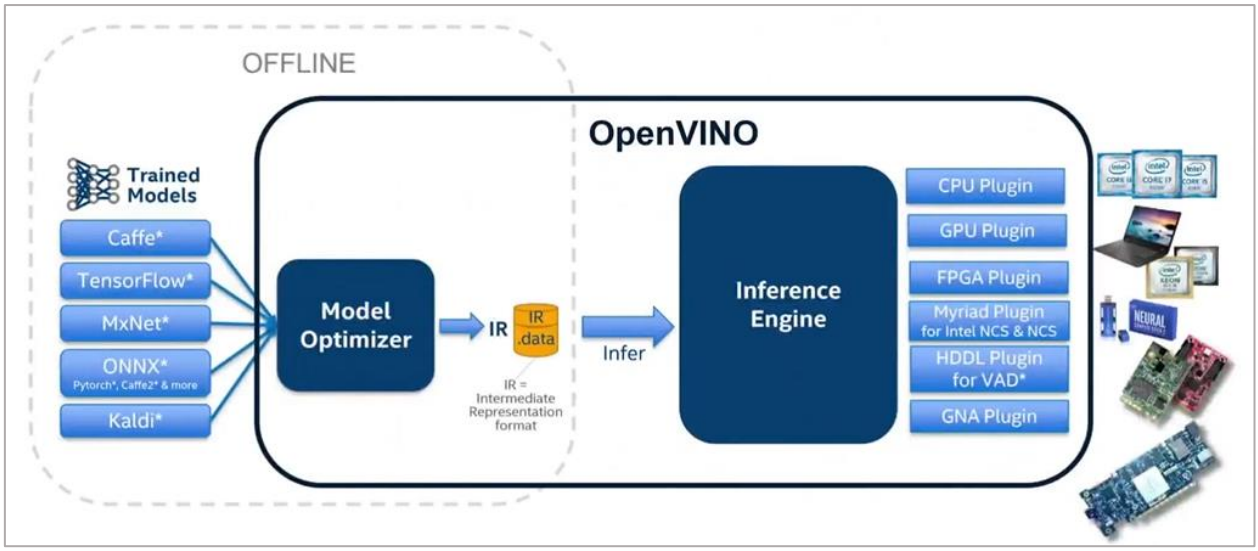

Рисунок 2.2 – Архитектура OpenVINO

Архитектуру OpenVINO глобально можно разделить на две части:

- Model Optimizer – набор скриптов, написанных на Python, которые позволяют перевести сеть, обученную в каком-то фреймворке в IR-формат.

  IR (Intermediate representation) – специальный формат, разработанный Intel (промежуточное представление), с которым дальше работает Inference Engine и все инструменты, которые есть в OpenVINO Toolkit.
- Inference engine (дословно механизм инференса) – отвечает за как раз таки исполнение сетей на каком-либо железе (подгружаются плагины, которые отвечают за каждый девайс и затем сеть исполняется)

2.2 Model Optimizer

2.2.1 Инструментарий

Model Optimizer – это кроссплатформенный инструментарий командной строки, который упрощает переход между средой обучения и средой развертывания, осуществляет статический анализ модели и настраивает модели глубокого обучения для оптимального исполнения на целевых устройствах [10].

Процесс исполнения Model Optimizer предполагает, что у вас есть сетевая модель, обученная с применением поддерживаемых фреймворков глубокого обучения: Caffe, TensorFlow, Kaldi, MXNet или преобразованная в формат ONNX. Model Optimizer создает промежуточное представление (IR) сети, которое может быть выведено с помощью Inference Engine. То есть MO получает на вход модель, а на выходе выдает файл в формате IR, что представлено на рисунке 2.2.1.1.

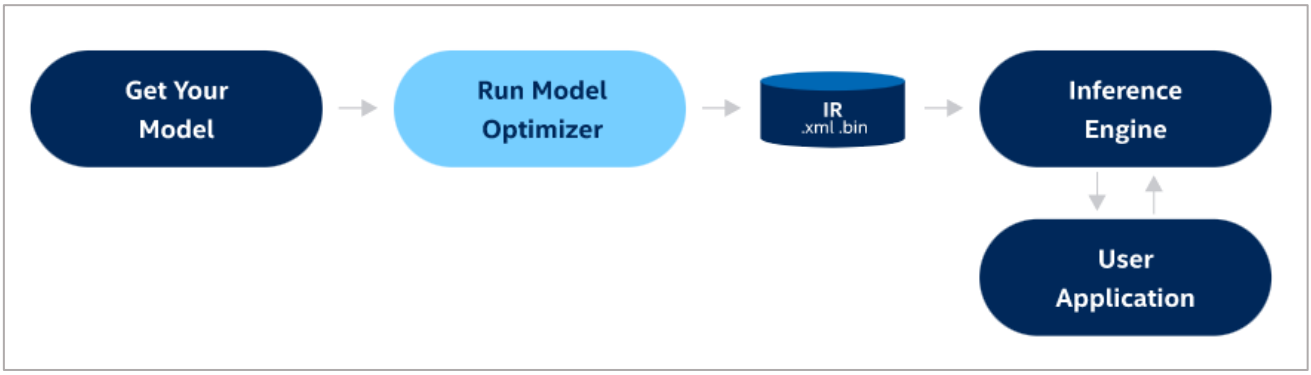

Рисунок 2.3 – Развертывание модели глубокого обучения

IR состоит из двух файлов:

- Xml, в котором находится описание топологии сети (какие слои, какого типа, как они соединены)
- Bin файл, где в бинарном формате описаны веса модели

Однако есть еще один способ загрузки модели в Inference Engine – минуя Model Optimizer – это сделать с помощью API, то есть каким-то программным способом. Если говорить в общем, то именно поэтому сейчас в Model Optimizer становится все меньше оптимизаций, потому что если модель будет

подгружаться с помощью API, то эти оптимизации потеряются, так как Model Optimizer не задействуется, поэтому все оптимизации сейчас переходят в Inference Engine и другие сторонние инструменты и сейчас Model Optimizer, не смотря на свое название становится больше конвертером, чем оптимизатором.

Model Optimizer выполняет предварительную обработку модели. Можно оптимизировать этот шаг и улучшить время первого вывода, для чего приведены две основные рекомендации:

1. Больший размер батча

   Такие устройства, как GPU, работают лучше при большем размере пакета. Хотя есть возможность установить размер батча и во время выполнения с помощью Inference Engine *ShapeInference feature*.

2. Результирующая точность IR

   Результирующая точность IR, например, FP16 или FP32, напрямую влияет на производительность. Поскольку CPU теперь поддерживает FP16 (при этом внутреннее масштабирование все равно происходит до FP32) и поскольку это наилучшая точность для GPU, вы можете захотеть всегда преобразовывать модели в FP16.

Модель в памяти можно представить в виде направленного графа, где узлы – это операции, а ребра соответствуют передаче данных от операции производителя (узел) к операции потребителя (узел).

Конвейер преобразования модели можно представить в виде следующей диаграммы:

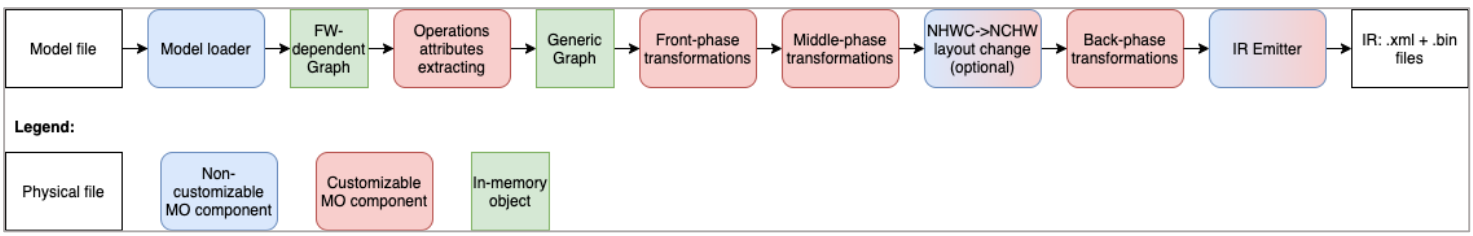

Рисунок 2.4 – Конвейер преобразования модели

2.2.2 Оптимизации

Model Optimizer не только преобразует модель в формат IR, но и выполняет ряд оптимизаций. Например, отдельные примитивы, такие как линейные операции (BatchNorm и ScaleShift), автоматически соединяются в свертки [11]. Как правило, эти слои не должны выражаться в результирующем IR:

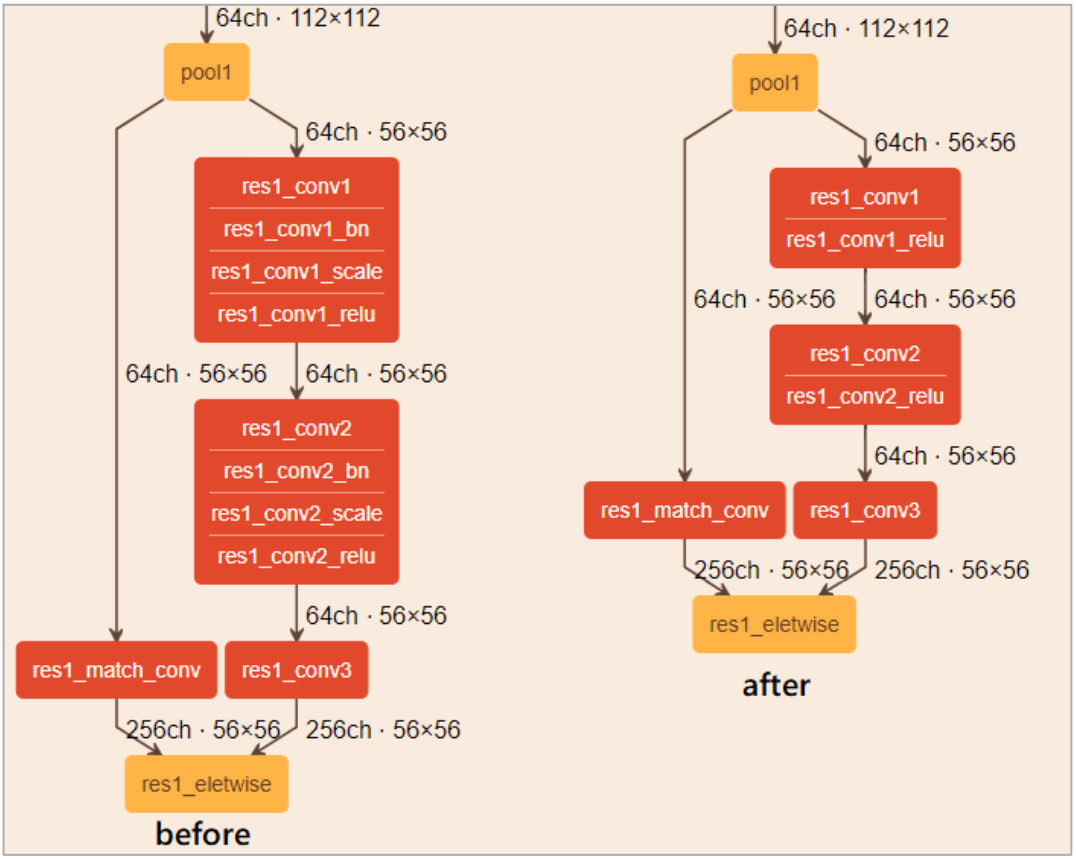

Рисунок 2.5 – Оптимизации

На рисунке выше изображена топология Resnet269. Слева – первоначальная модель, а справа – модель после преобразования, которую делает Model Optimizer, притом слои BatchNorm и ScaleShift слиты с весами свертки, а не представляют собой единичные слои.

Но Model Optimizer иногда не может выполнить слияние. Например, нелинейные операции (активации) между свертками и линейными операциями могут помешать данной оптимизации. Когда производительность является приоритетом, можно видоизменить (и, может быть, переобучить) топологию.

Важно обратить внимание на то, что активация (слой relu) не затрагивается оптимизатором модели, и, хотя она также может быть объединена в свертку, это скорее специфическая для устройства оптимизация, которая покрывается Inference Engine во время загрузки модели.

Загрузчик моделей

Данный каталог содержит скрипты, автоматизирующие некоторые задачи, связанные с моделями, на основе конфигурационных файлов в каталогах моделей [12]:

- Model Downloader: downloader.py загружает файлы моделей из онлайн-источников и, при необходимости, исправляет их, чтобы сделать более пригодными для использования в Model Optimizer
- Конвертер моделей: converter.py конвертирует модели, которые не находятся в формате Inference Engine IR, в этот формат с помощью Model Optimizer.
- Model Quantizer: quantizer.py квантизует модели полной точности в формате IR в версии с низкой точностью с помощью Post-Training Optimization Toolkit.
- Model Information Dumper: info_dumper.py печатает информацию о моделях в стабильном машиночитаемом формате.

2.3 Intermediate Representation OpenVINO

2.3.1 Описание формата

Инструментарий OpenVINO представляет свой собственный формат представления графов и свой набор операций. Граф представляется в виде двух файлов: XML-файла и бинарного файла. Данное представление принято называть Intermediate Representation или IR (промежуточное представление).

XML-файл описывает топологию сети, используя тег *layer* для узла операции и тег *edge* для соединения потока данных. Любая операция имеет

фиксированное количество атрибутов, которые определяют логику операции, используемой для текущего узла.

В XML-файле нет больших константных значений, таких как веса свертки. Взамен этого он ссылается на часть сопроводительного бинарного файла, который хранит такие значения в формате bin.

2.3.2 Набор операций

Операции в наборе операций OpenVINO избираются на основе возможностей поддерживаемых фреймворков глубокого обучения и аппаратных возможностей устройства вывода. Категории операций представлены ниже:

- Стандартные слои глубокого обучения, такие как Convolution, MaxPool, MatMul (FullyConnected)
- Разные функции активации, например, ReLU, Tanh, PReLU.
- Общие арифметические тензорные операции, такие как сложение, вычитание, умножение.
- Операции сравнения, которые сравнивают два числовых тензора и создают булевы тензоры, например, Less, Equeal, Greater.
- Логические операции, которые работают с булевыми тензорами, например, And, Xor, Not.
- Операции перемещения данных, которые работают с частями тензоров: Concat, Split, StridedSlice, Select.
- Специализированные операции для моделей определенного типа: DetectionOutput, RegionYolo, PriorBox.

2.4 Inference Engine

2.4.1 API Inference Engine

Inference Engine - это набор библиотек C++ с привязкой к C и Python, предоставляющих общий API для создания решений для инференса на вашей

платформе. API Inference Engine используется для чтения промежуточного представления (IR), ONNX и выполнения модели на целевых девайсах. API Inference Engine позволяет:

- Прочитать модель из файлов (IR)
- Загрузить модель в плагин, работающий с конкретным устройством
- Отправить данные для обработки
- Получить результаты обработки

Inference Engine использует архитектуру плагинов. Плагин Inference Engine – это программный компонент, который содержит полную реализацию для выполнения инференса на определенном аппаратном устройстве Intel: CPU, GPU, VPU и т.д. Главная идея – это единый API для разных устройств, выпускаемых Intel, позволяющий запускать нейронные сети, оставляя при этом возможность на каждом устройстве выполнить какую-либо тонкую настройку, то есть передавать какие-то специфические параметры при запуске сети, если это необходимо.

Приведенная далее схема иллюстрирует стандартный рабочий процесс развертывания обученной модели:

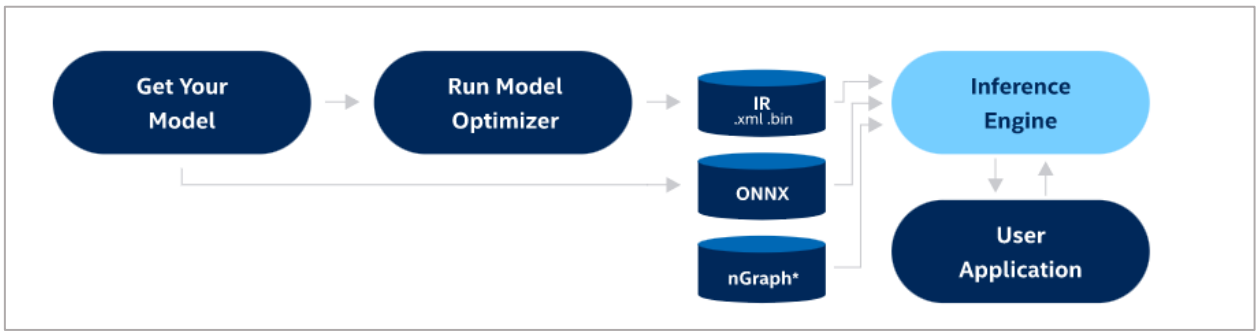

Рисунок 2.6 – Процесс развертывания обученной модели

Использование API Inference Engine

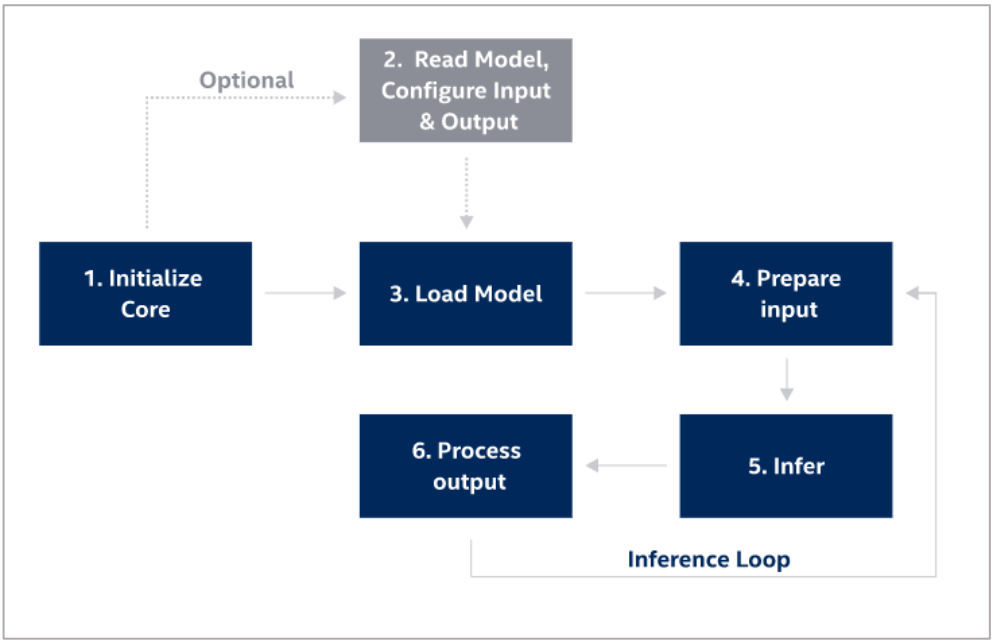

Рисунок 2.7 – Конвейер инференса

В этом разделе приведены пошаговые инструкции по реализации типичного конвейера выводов с помощью Inference Engine API:

1. Создание Inference Engine Core для управления доступными устройствами и чтения сетевых объектов

```
ie = IECore()
```

2. (Необязательно). Считывание модели. Настройка входов и выходов модели
3. Загрузка модели в устройство

```
exec_net = ie.load_network(network= "model.xml", device_name="CPU")
```

4. Подготовка входного слоя

   Например, конвертация изображения в формат NCWH с типом FP32.

```
input_data = np.expand_dims(np.transpose(image, (2, 0, 1)),
0).astype(np.float32)
```

5. Начало инференса

```
result = exec_net.infer({input_name: input_data})
```

6. Обработка результатов инференса

```
output = result[output_name]
```

2.4.2 Программный стек Inference Engine

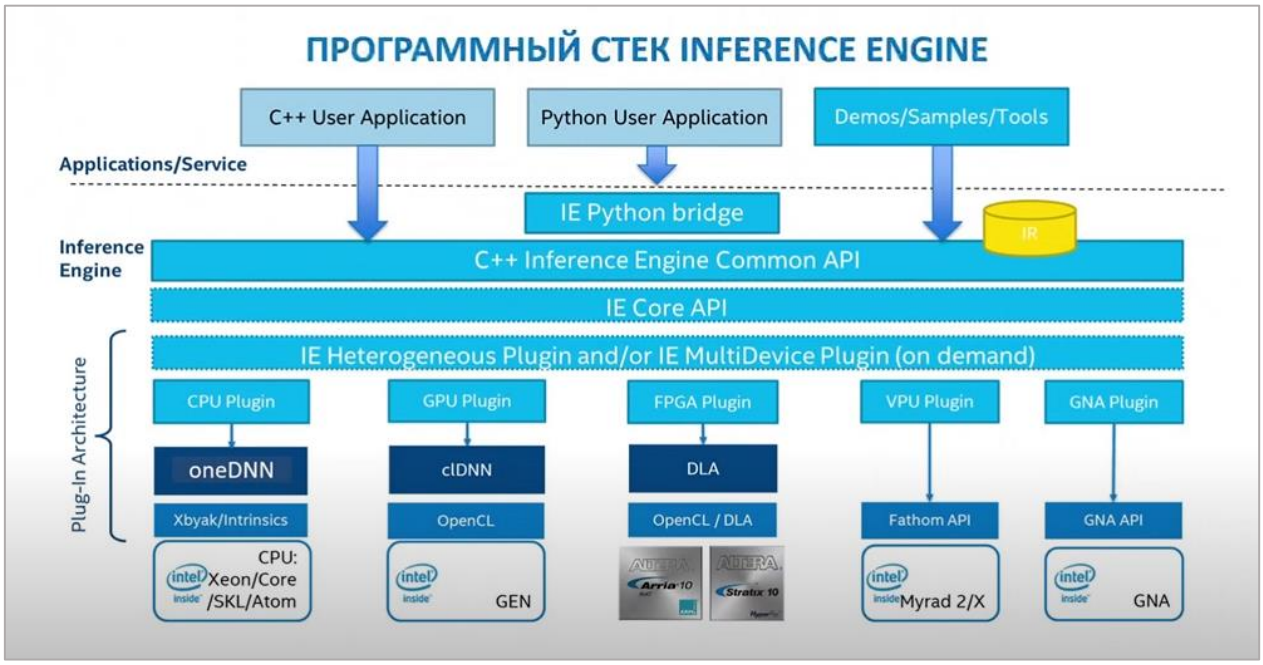

Рисунок 2.8 – Программный стек IE

Основная идея в том, что приложение пользователя, написанное на Python/C++ через некий API, позволяет включить в работу один из плагинов устройства, на котором запускается сеть.

С одной стороны, в Inference Engine есть интерфейс, которым пользуются пользователи для загрузки своего кода, с другой стороны есть интерфейс, который позволяет добавить поддержку любого устройства.

Каждый из плагинов, который относится к каждому конкретному устройству может иметь свои низкоуровневые библиотеки. В общем Inference Engine имеет множество разноуровневых задач: есть какие-то общие задачи, общие оптимизации, которые не зависят от устройства, есть низкоуровневые оптимизации под конкретное устройство и есть совсем низкоуровневые оптимизации, уже перед включением железа в работу (например, примитивы из OpenCL или средства парализации, относящиеся к конкретному процессору). За счет всех оптимизаций Inference Engine как раз и достигает высокой производительности.

2.4.5 Оптимизация развертывания

Для оптимизации результатов производительности на этапе выполнения можно использовать:

- Предварительную обработку
- Режим пропускной способности
- Асинхронное API
- Снижение точности инференса
- Оптимизацию устройств
- Комбинацию устройств

2.4.6 Режим пропускной способности

Одним из методов увеличения эффективности вычислений является обработка батчей, которая объединяет множество входных изображений для достижения оптимальной пропускной способности. Внутренние ресурсы выполнения разделяются/распределяются по потокам выполнения. Применение данной функции позволяет получить значительно более высокую производительность для сетей, которые изначально плохо масштабируются с количеством потоков (например, легкие топологии).

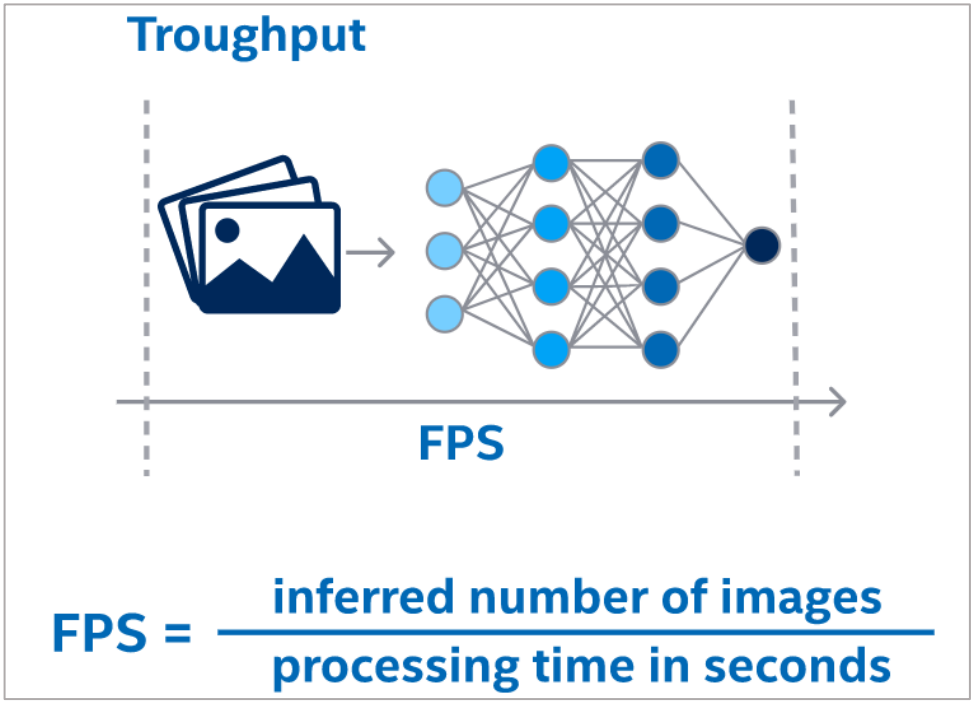

Рисунок 2.9 – Пропускная способность

Режим пропускной способности ослабляет требование насыщения процессора за счет использования большого пакета: параллельный запуск нескольких независимых запросов на инференс зачастую дает значительно лучшую производительность, чем использование только батча. Это позволяет облегчить логику приложения, так как нет необходимости объединять несколько входов в батч для достижения хорошей производительности процессора. Вместо этого можно держать отдельный запрос на инференс для каждой камеры или другого источника входных данных и обрабатывать запросы параллельно, используя асинхронное API.

2.4.7 Асинхронное API Inference Engine

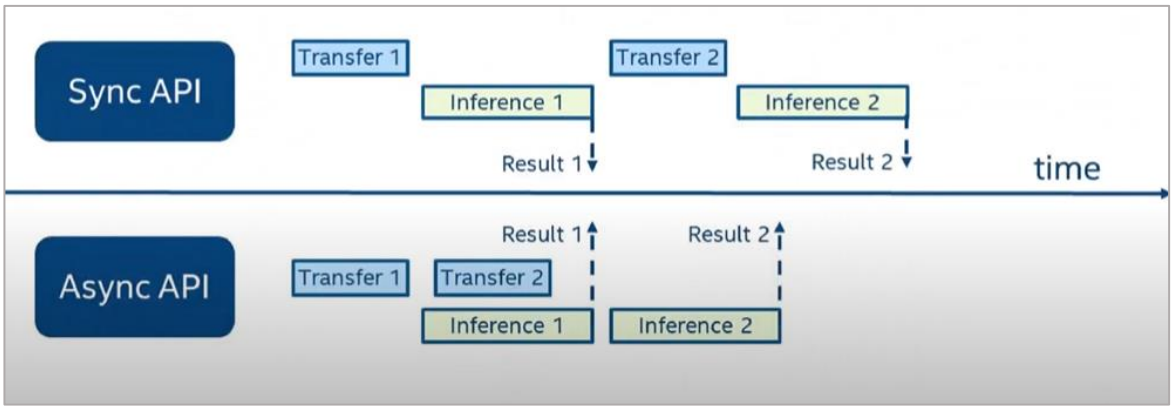

Рисунок 2.10 – Синхронный и асинхронный режимы

Асинхронное API Inference Engine позволяет увеличить общую частоту кадров приложения. Пока ускоритель занимается инференсом, приложение может продолжать работу на хосте, а не ждать завершения инференса.

В приведенном примере инференс применяется к результатам декодирования видео. Таким образом, можно вести два параллельных запроса выполнения, и пока обрабатывается текущий, происходит захват входного кадра для следующего.

- В синхронном режиме обычно, кадр захватывается с помощью OpenCV, а затем сразу же обрабатывается, то есть выполнение блокируется до исполнения предыдущего запроса:

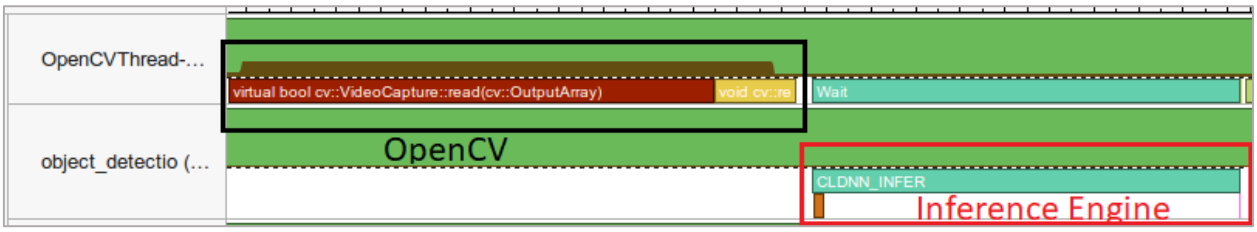

Рисунок 2.11 – Синхронный режим

- В асинхронном режиме следующий запрос заполняется в основном потоке, в то время как текущий запрос обрабатывается, то есть накидываются запросы на обработку и не дожидаясь выполнения этого запроса вызывается следующий, таким образом они и накапливаются. Окончание отслеживается с помощью механизма callback:

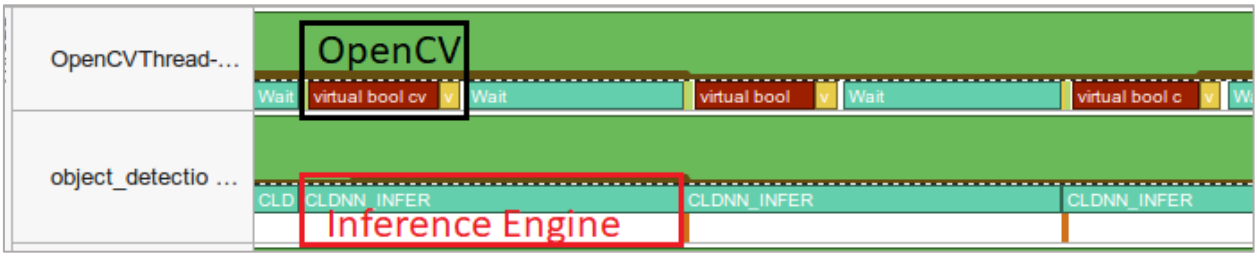

Рисунок 2.12 – Асинхронный режим

2.4.8 Автоматическое снижение точности инференса

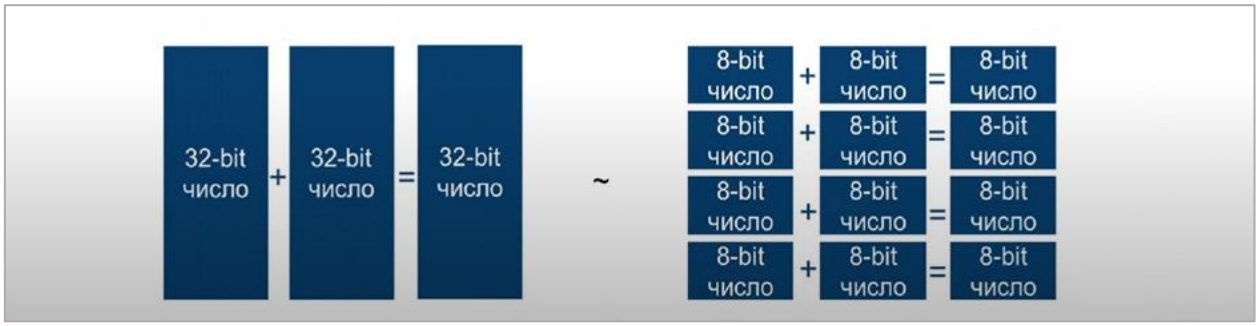

Рисунок 2.13 – Использование вычислений с меньшей разрядностью

Точность инференса прямо влияет на производительность. Использование вычислений с меньшей разрядностью – позволяет, например, при небольшой потере точности выполнить гораздо больше операций за такт.

Model Optimizer может создавать IR с различной точностью. Например, IR FP16 изначально предназначен для устройств VPU и GPU, тогда как, например, для CPU IR FP16 обычно масштабируется до обычного FP32

автоматически при загрузке. Но доступны дополнительные опции точности инференса для определенного устройства, например, 8-битное целое или bfloat16. Для мультидевайсного плагина, который поддерживает автоматический инференс на нескольких устройствах параллельно, можно использовать FP16 IR (FP32 не требуется).

2.5 Поддержка плагинов для устройств

Inference Engine использует архитектуру плагинов. Плагин Inference Engine – это программный компонент, который содержит полную реализацию для вывода на определенном аппаратном устройстве Intel: CPU, GPU, VPU, GNA и т.д. Каждый плагин реализует унифицированный API и предоставляет дополнительные API, специфичные для конкретного оборудования.

Inference Engine предоставляет возможности для инференса моделей глубокого обучения на представленных далее типах устройств с соответствующими плагинами.

2.5.1 Плагины CPU и GPU

Плагин для CPU был разработан для достижения высокой производительности нейронных сетей на CPU с использованием библиотеки Intel Math Kernel Library for Deep Neural Networks. В настоящее время плагин CPU использует Intel Threading Building Blocks для распараллеливания вычислений.

Плагин для GPU использует библиотеку Intel Compute Library for Deep Neural Networks для вывода глубоких нейронных сетей. clDNN – это библиотека производительности с открытым исходным кодом для приложений Deep Learning, предназначенная для ускорения вывода глубокого обучения на графических процессорах Intel.

Плагины CPU/GPU поддерживают несколько алгоритмов оптимизации графа, таких как слияние или удаление слоев:

- Снижение точности выводов

Плагин CPU использует стандартный подход к оптимизации. Этот подход означает, что выводы делаются с меньшей точностью, если на данной платформе возможно достичь лучшей производительности с приемлемым диапазоном точности.

- Слияние слоев свертки и простых слоев

  Слияние слоя свертки и любого из простых слоев, перечисленных ниже:

  1. Активации: ReLU, ELU, Sigmoid, Clamp.
  2. Глубокие: ScaleShift, PReLU
  3. FakeQuantize
- Объединение слоев Pooling и FakeQuantize
- Объединение слоев FullyConnected и Activation
- Объединение слоев свертки и глубокой свертки, сгруппированных с простыми слоями
- Объединение слоев свертки и суммирования
- Объединение группы сверток

2.5.2 Multi-device плагин

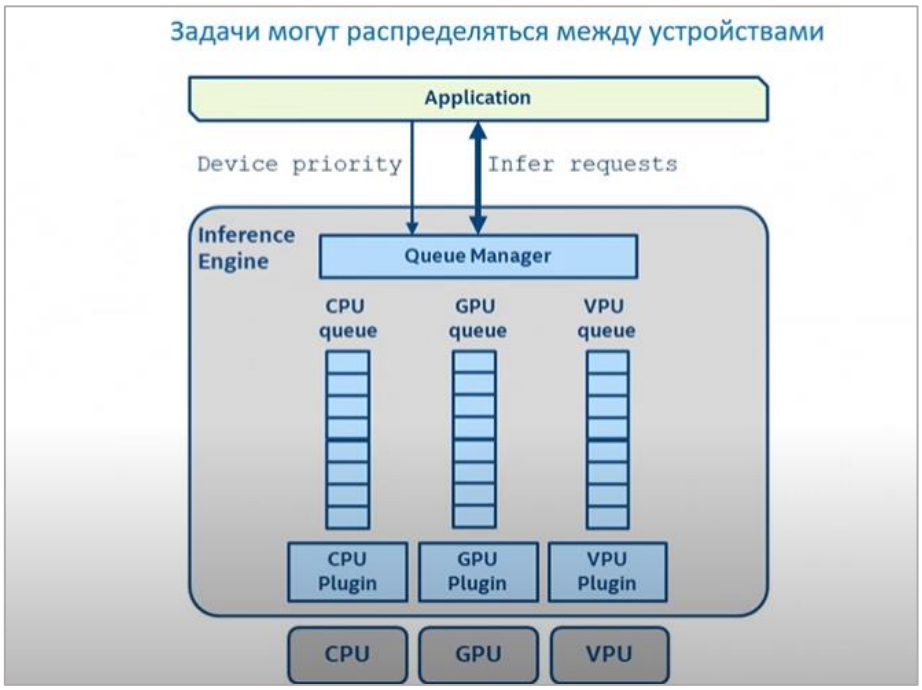

Рисунок 2.14 – Multi-Device режим работы

Плагин Multi-Device автоматически распределяет запросы на инференс по доступным вычислительным устройствам для параллельного выполнения запросов. В отличие от него, гетерогенный плагин, речь о котором пойдет далее, может запускать разные слои на различных устройствах, но не параллельно [13].

Например, может быть желательно выполнять инференс модели глубокого обучения на интегрированном GPU. Хотя такие GPU гораздо менее мощные, чем дискретные, они способны выполнять больше операций с плавающей точкой в секунду, чем CPU, расположенный на том же кристалле.

Потенциальными преимуществами плагина Multi-Device являются:

1. Увеличение пропускной способности за счет использования нескольких устройств (по сравнению с выполнением на одном устройстве).

2. Более стабильная производительность, так как устройства разделяют нагрузку на выводы (если одно устройство слишком занято, другое может взять на себя больше нагрузки).

При применении плагина Multi-Device логика приложения остается неизменной, поэтому не нужно явно загружать сеть на каждое устройство, создавать и балансировать запросы на инференс и т.д. С точки зрения приложения, это просто еще одно устройство, которое обрабатывает фактический механизм.

Настройка плагина Multi-Device может быть описана тремя основными шагами:

1. Настройка каждого устройства как обычно

2. Загрузка сети в плагин Multi-Device, созданный поверх (приоритетного) списка настроенных устройств. Это единственное изменение, необходимое в приложении.

3. Как и при любом другом вызове ExecutableNetwork (в результате load_network), вы создаете столько запросов, сколько необходимо для насыщения устройств.

2.5.3 Гетерогенный плагин

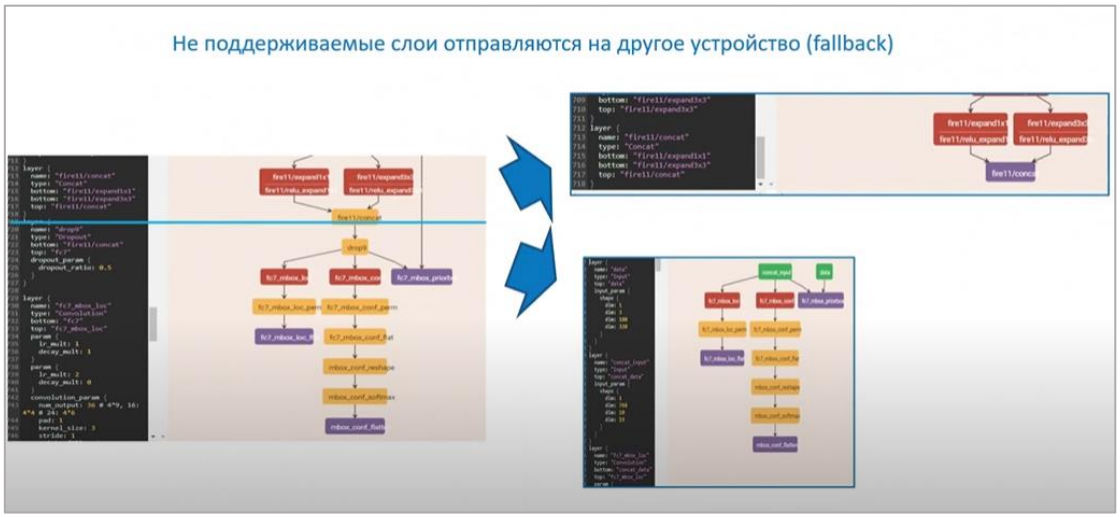

Рисунок 2.15 – Гетерогенный режим работы

Гетерогенный плагин позволяет производить инференс одной сети на многих устройствах, то есть отправлять не поддерживаемые слои на другое устройство (fallback). Целями инференса сетей в гетерогенном режиме являются [14]:

- Использование мощности ускорителей для обработки особо тяжелых частей сети и выполнять неподдерживаемые уровни на резервных устройствах, таких как CPU.
- Эффективнее утилизировать все доступные аппаратные средства в течение одного инференса.

Инференс через гетерогенный плагин можно разделить на два независимых этапа:

1. Установка аппаратного сродства к слоям
2. Загрузка сети в гетерогенный плагин, разделение сети на составляющие и выполнение их через плагин.

Автоматическая политика инициирует "жадное" поведение и назначает все слои, которые могут быть выполнены на определенном устройстве, в соответствии с установленными приоритетами. Автоматическая политика не учитывает особенности плагина, такие как неосуществимость инференса некоторых слоев без других специфических слоев, размещенных до

или после этого слоя. Если плагин устройства не поддерживает топологию подграфа, построенную гетерогенным плагином, то необходимо установить сродство вручную.

Отдельные топологии плохо поддерживаются для гетерогенного выполнения на определенных устройствах или вообще не могут быть выполнены в этом режиме.

2.6 Дополнительные утилиты

- Post-training Optimization – позволяет оптимизировать нейронную сеть с использованием целочисленной арифметики, то есть например перейти из FP16 в INT8. Проблема в том, что теряется точность при переходе из чисел с плавающей запятой в целые числа. Возникает проблема нахождения scale-фактора, коэффициента преобразования из вещественных чисел в целые. Если взять слишком маленькое, то получится слишком маленький входной диапазон, а если слишком большое, то число может банально не влезть в INT и будет переполнение, что плохо скажется на работе сети. Данная утилита запускается на реальных данных и для каждого слоя сети выясняет диапазон входных данных и таким образом получается подобрать коэффициент преобразования scale.
- Model Analyzer – позволяет узнать теоретический предел производительности, который можно достичь с вашим железом и сравнить с реальным результатом.
- Benchmark App – позволяет с помощью специфичных методологий, разработанных Intel, измерить скорость работы сети.
- Deployment Manager – позволяет минимизировать набор компонентов, который требуется для разработки, то есть позволяет устанавливать не весь пакет OpenVINO, а какие-то его определенные инструменты.
- Accuracy Checker – позволяет измерить точность модели
- Model Downloader – позволяет загружать модели из открытого зоопарка OpenVINO

ГЛАВА 3. ПРИМЕНЕНИЕ OPENVINO TOOLKIT

В данном разделе представлено описание всех шагов по оптимизации и развертыванию моделей на устройствах Intel, обученных с помощью различных фреймворков, используя инструменты Intel OpenVINO Toolkit:

1. Выбор обученной модели и настройка оптимизатора моделей для конкретного фреймворка.
2. Преобразование модели фреймворка для создания оптимизированного промежуточного представления (IR) модели на основе обученной топологии сети, и значений весов и смещений.
3. Тестирование модели в формате промежуточного представления с помощью Inference Engine в целевой среде.
4. Интеграция Inference Engine в приложение, для развертывания модели на целевом устройстве.

3.1 Выбор модели

Первым этапом использования OpenVINO Toolkit является выбор модели. На данном этапе предполагается использование уже обученной каким-либо фреймворком модели, которая и будет использоваться для оптимизации. У пользователей существует два варианта выбора модели – сделать свое решение или выбрать готовое. Естественно, первый вариант предполагает в первую очередь изучение литературы для нахождения архитектур для решения поставленной задачи, определение целевых показателей, подготовка обучающей выборки и само обучение модели с помощью фреймворка. Все эти этапы занимают большое количество времени. Для большинства задач удобнее будет использовать уже обученную модель, находящуюся в открытом доступе. OpenVINO поддерживает только определенные фреймворки:

- Caffe
- TensorFlow

- MXNet
- ONNX
- Kaldi

Но также стоит обратить внимание на то, что в представленных фреймворках могут не поддерживаться определенные топологии сетей, и всегда стоит сверяться с документацией перед использованием следующего инструмента.

OpenVINO предоставляет возможность использования набора предварительно обученных моделей – Open Model Zoo. В зоопарке открытых моделей можно найти модели, обученные Intel и сторонними людьми, но оптимизированные для использования с OpenVINO, то есть модели с поддерживающимися топологиями и предварительно сконвертированные в формат промежуточного представления (IR). Для получения модели из открытого зоопарка нужно воспользоваться скриптом “загрузчик моделей”: downloader.py. Он загружает файлы моделей из онлайн-источников и, при необходимости, исправляет их, чтобы сделать более пригодными для использования в Model Optimizer:

```
python <директория OpenVINO>/deployment_tools/tools/downloader/
downloader.py --name <название модели>
```

Для использования в demo-приложении была выбрана задача оценки позы нескольких людей, так как данная задача является актуальной на данный момент и использование которой требует использования большого количества вычислительных ресурсов. Была использована модель MobileNet v1 [https://github.com/shicai/MobileNet-Caffe], обученная на фреймворке Caffe, в качестве экстрактора признаков на кадре. Модель показывает отличную точность.

Для каждого человека на изображении сеть определяет позу человека: скелет тела, состоящий из ключевых точек и связей между ними. Поза может

содержать до 18 ключевых точек: уши, глаза, нос, шея, плечи, локти, запястья, бедра, колени и лодыжки.

3.2 Преобразование модели

Следующий пункт – это преобразование модели фреймворка для создания оптимизированного промежуточного представления. Первое в чем нужно удостовериться – топология сети поддерживается OpenVINO. Это можно проверить в официальной документации OpenVINO Toolkit. Выбранная топология MobileNet поддерживается.

Далее перед запуском Model Optimizer необходимо установить предварительные зависимости Model Optimizer для фреймворка, который использовался для обучения модели. В данном случае были установлены настройки для фреймворка Caffe:

```
cd <директория OpenVINO>\deployment_tools\model_optimizer\install_
prerequisites\
install_prerequisites_caffe.bat
```

Данный скрипт, поставляемый с OpenVINO, устанавливает необходимые зависимости и позволяет использовать самый простой способ настройки Model Optimizer.

Чтобы преобразовать модель Caffe, требуется запустить скрипт Model Optimizer mo.py, указав путь к входному файлу модели .caffemodel и путь к выходному каталогу с правами на запись. Результатом преобразования будет модель в формате IR:

```
cd <директория OpenVINO>/deployment_tools/model_optimizer/
python mo.py --input_model <модель>.caffemodel --output_dir <конечная
директория>
```

Для преобразования модели доступны две группы параметров:

- Параметры, не зависящие от фреймворка, используются для преобразования модели, обученной с помощью любого поддерживаемого фреймворка.

В данном случае использовались параметры:

```
--input_model <модель>.caffemodel
                                    Файл с весами тренированной модели

--model_name <название модели>
                                    Название модели для сгенерированного IR и
                                    и выходных файлов .xml/.bin

--output_dir <конечная директория>
                                    Директория, в которую сохраняется IR

--data_type {FP16, FP32, half, float}
                                     Тип данных для всех
                                     промежуточных тензоров и весов
```

- Параметры, специфичные для Caffe, используются для преобразования только моделей Caffe.

Использовались параметры:

```
--input_proto <название файла>.prototxt
                                     Файл с описанием топологии сети
```

Конечная команда преобразования модели Caffe в формат IR с помощью Model Optimizer:

```
python mo.py --input_model pose_iter_160000.caffemodel --input_proto
pose_deploy_linevec_faster_4_stages.prototxt --output_dir model –
model_name human-pose-estimation –data_type FP16
```

В результате будет сгенерировано два файла:

- human-pose-estimation.bin – веса модели
- human-pose-estimation.xml – топология сети

Точность оригинальной модели – FP16. Но для последующих экспериментов также будет использоваться точность FP32. Поэтому было принято решение использовать модель из открытого зоопарка OpenVINO human-pose-estimation-0001, также обученную с использованием фреймворка Caffe и датасета COCO. Сравнение метрик инференса первоначальной модели и модели из открытого зоопарка показало, что различие допустимое, чтобы считать модели одинаковыми.

Для того, чтобы проверить насколько сильно отличаются модели, был использован инструмент из OpenVINO benchmark_app. Он позволяет с помощью специфичных методологий, разработанных Intel, измерить скорость работы модели. Оригинальная модель в формате IR и модели из открытого зоопарка были протестированы с точностью FP16 на CPU:

```
python .\benchmark_app.py -m <путь до модели> -d CPU
```

Результаты сравнения приведены в таблице 3.1.

Таблица 3.1 – Сравнение моделей

	Первоначальная модель	**Модель OpenVINO**
Задержка, мс	250.63	236.43
FPS	15.96	16.45

При дальнейших экспериментах в качестве оригинальной модели без каких-либо оптимизаций будет использоваться первоначальная модель, а при использовании инструментария OpenVINO Toolkit будет использоваться модель из открытого зоопарка.

3.3 Приложение

Последней стадией является интеграция Inference Engine в приложение, для развертывания модели на целевом устройстве. Выбранная задача и модель описаны в предыдущих пунктах. Данное приложение позволяет оценить оптимизацию инференса в случае распределенной обработки данных.

Распределенная обработка представлена в OpenVINO Toolkit двумя плагинами для целевых устройств: гетерогенным и мультидевайсным.

Плагин Multi-Device автоматически распределяет запросы на инференс по доступным вычислительным устройствам для параллельного выполнения запросов. Гетерогенный плагин, в свою очередь, может запускать разные слои

на различных устройствах, но не параллельно. Преимуществами использования нескольких устройств являются:

- Использование мощностей ускорителей для обработки наиболее тяжелых частей сети.
- Выполнение неподдерживаемых слоев на резервных устройствах.
- Эффективное использование всех доступных аппаратных средств в течение одного инференса, что дает более стабильную производительность, так как устройства разделяют нагрузку на выводы.
- Повышение пропускной способности за счет использования нескольких устройств.

Следует обратить внимание на то, что при использовании приложения и последующих экспериментах загрузка сети на GPU будет медленнее, чем загрузка сети на CPU, но выводы будут быстрее.

3.3.1 Процесс интеграции и инференса

Полученные с помощью Model Optimizer файлы IR-представления далее интегрируются в Inference Engine. Интеграция модели происходит в несколько этапов, которые представлены на рис. 3.1.

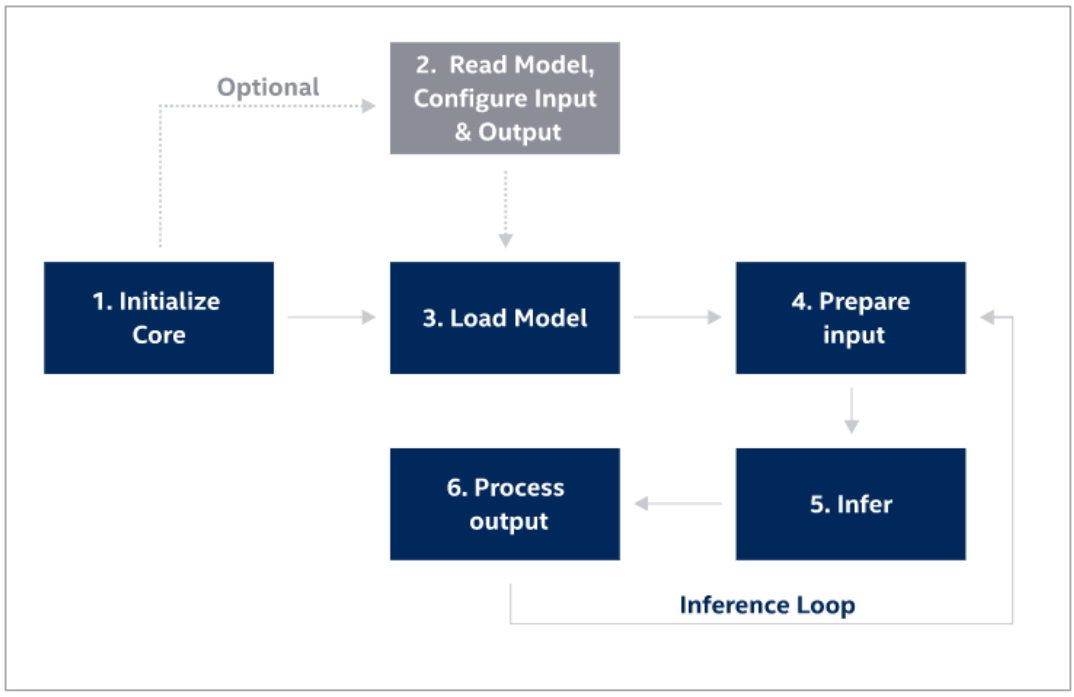

Рисунок 3.1 – Конвейер инференса

Полный код программы приведен в приложении 1. Сама же реализация конвейера инференса приведена далее:

- Инициализация ядра Inference Engine для управления доступными устройствами и чтения сетевых объектов.

```
from openvino import inference_engine as ie
ie_core = ie.IECore()
```

- Считывание модели.

```
base_model_dir = f'{path}\model'
model_name = 'human-pose-estimation-0001'
precision = 'FP16'

model_path = f"{base_model_dir}\{precision}\{model_name}.xml"
model_weights_path =
f"{base_model_dir}\{precision}\{model_name}.bin"

net = ie_core.read_network(model=model_path,
weights=model_weights_path)
```

- Настройка входов и выходов модели

```
input_key = list(net.input_info)[0]
output_keys = list(net.outputs.keys())

height, width = net.input_info[input_key].tensor_desc.dims[2:]
```

Экземпляр OpenVINO IENetwork хранит информацию о модели. Информация о входах и выходах модели находится в net.input_info и net.outputs. Они также являются свойствами экземпляра ExecutableNetwork. Там, где мы используется net.input_info и net.outputs, можно также использовать exec_net.input_info и exec_net.outputs.

- Загрузка модели в устройство

```
exec_net = ie_core.load_network(net, device_name="CPU")
```

- Подготовка входного слоя

```
input_img = cv2.resize(frame, (width, height),
interpolation=cv2.INTER_AREA)

input_img = input_img.transpose(2, 0, 1)[np.newaxis, ...]
```

Параметрами входного слоя для данной модели являются:

- Форма – (1, 3, 256, 456)
- Макет ввода – NCHW
- Цветовой формат BGR

Для изменения размеров кадра видеопотока используется метод библиотеки OpenCV resize со значениями, которые были получены в пункте 3. Для изменения макета ввода используется метод transpose, а также newaxis из библиотеки numpy для добавления новой оси, которая отвечает за размер батча.

- Инференс

Для исполнения инференса, вызывается метод infer() исполняемой сети, exec_net, которая была загружена в пункте 4. Метод infer() ожидает один аргумент: inputs. Это словарь, отображающий имена входных слоев на входные данные.

```
results = exec_net.infer(inputs={input_key: input_img})
```

- Обработка результатов инференса

```
poses, scores = process_results(frame, results)
```

3.3.2 Доступные устройства

Inference Engine загружает сеть на устройство. Устройство в данном контексте означает CPU, Intel GPU, Neural Compute Stick 2 и т.д. Для того, чтобы определить список устройств в текущей системе, которые поддерживают использование OpenVINO можно воспользоваться свойством

available_devices ядра Inference Engeine. Опция "FULL_DEVICE_NAME" в методе ie.get_metric() показывает имя устройства.

```
devices = ie.available_devices
for device in devices:
    device_name = ie.get_metric(device_name=device,
metric_name="FULL_DEVICE_NAME")
    print(f"{device}: {device_name}")
```

Доступные устройства представлены в таблице 3.2.

Таблица 3.2 – Доступные устройства

Тип устройства	**Название**
CPU	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
GPU	Intel(R) UHD Graphics 630 (iGPU)

3.3.3 Использование приложения

Требования для использования приложения (рекомендуется использовать виртуальную среду): python 3.6 и выше, дистрибутив OpenVINO Toolkit, а также библиотеки, описанные в файле requirements.

1. Необходимо выбрать точность модели. В исследовании использовались точности моделей – FP32, FP16, FP16-INT8. FP16 используется по умолчанию.
2. Выбрать обработку видеофайла или же обработку веб-камеры компьютера. Исследовалась обработка заданного файла, расположенного на github [https://github.com/GandhiKK/OpenVINO_hetero/blob/master/smert/diploma/data/gym-detection4x.mp4].
3. Функции:

- Нажатие клавиши TAB переключает устройство для инференса модели. По умолчанию инференс выполняется на CPU. Также доступны режимы выполнения GPU, Multi-device, при котором происходит параллельное выполнение запросов на инференс на CPU и GPU и гетерогенный, при

котором только определенные слои выполняются на другом, выбранном устройстве.

В течение инференса заданные метрики сохраняются для последующего анализа.

- Нажатие клавиши ESC завершает приложение.

Пример использования приложения представлен на рис. 3.2.

Рисунок 3.2 – Демо-приложение

3.3.4 Результат использования приложения

После выбора модели, ее обработки и интеграции Inference Engine в приложение можно сравнить производительность при использовании OpenVINO Toolkit. Исходная модель и модель OpenVINO были запущены только на CPU, так как дальнейшее исследование предполагает сравнение производительности с другими устройствами. В качестве входных данных было использовано видео пожилых людей, занимающихся фитнесом.

Кадр из данного видео представлен на рис. 3.3.

Рисунок 3.3 – Кадр из видео для определения поз

Результаты определения поз людей с помощью исходной модели и модели OpenVINO представлены в таблице 3.3.

Таблица 3.3 – Сравнение метрик

Метрики	**Исходная модель**	**Модель OpenVINO**
Задержка, мс	1485.76	218.23
FPS	2.69	17.79

Как видно из таблицы 3.3 исходной неоптимизированной модели понадобилось примерно полторы секунд для определения поз людей, присутствующих на кадре, в то же время модель, оптимизированная с помощью OpenVINO Toolkit смога решить данную задачу за 0.2 сек. Разница чуть менее, чем в 7 раз. Дальнейшие исследования включают в себя оценку эффективности оптимизации инференса при использовании нескольких устройств.

ГЛАВА 4. АНАЛИЗ ЭФФЕКТИВНОСТИ

В данной главе представлен анализ эффективности оптимизации инференса в случае его выполнения на нескольких устройствах. Необходимо протестировать насколько улучшаются целевые метрики для моделей с разной точностью, используя гетерогенный режим и режим нескольких устройств. Сравниваются модели с точностью FP32, FP16 и INT8. То есть 32-битный формат IEEE 754 с плавающей запятой одинарной точности FP32, 16-битный FP16 и квантизованная модель для 8-битных целочисленных вычислений. Запись чисел с плавающей запятой представлена на рис. 4.1.

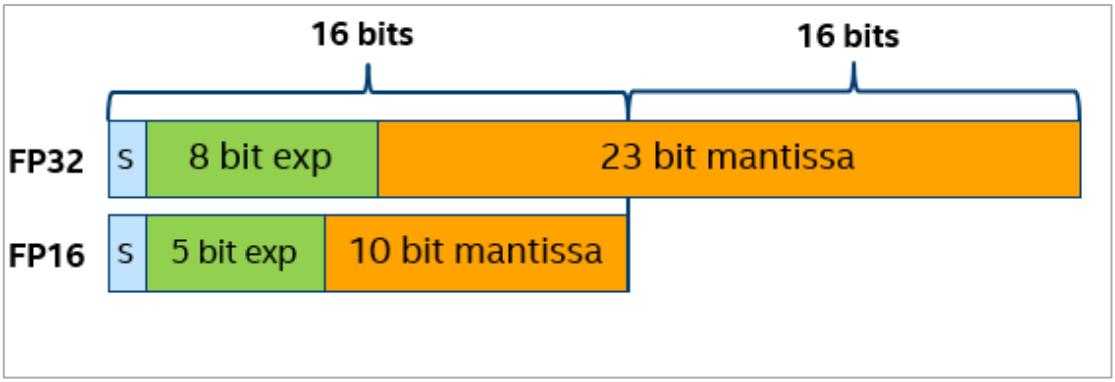

Рисунок 4.1 – Запись чисел с плавающей запятой

Исследование проводилось на компьютере с характеристиками, представленными в таблице 4.1.

Таблица 4.1 – Характеристики компьютера

ОС	Microsoft Windows 10 Pro
CPU	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz
Модель основной платы	LNVNB161216
RAM	8 Гб
GPU	Intel(R) UHD Graphics 630
GPU	NVIDIA GeForce GTX 1650

Поддерживаемые OpenVINO Toolkit устройства были представлены в таблице 3.2. Дискретный GPU NVIDIA GeForce GTX 1650 не используется из-за того, что не принадлежит продуктам компании Intel.

Для сравнения были выбраны следующие целевые метрики [15], которые смогут объективно оценить качество инференса модели на целевых устройствах, но стоит заметить, что инструментарий OpenVINO Toolkit почти не влияет на веса нейронной сети или ее топологию, поэтому точность моделей не сравнивалась:

- Пропускная способность – количество данных, обработанных за определенный период времени. В данном случае в качестве пропускной способности был выбран FPS. FPS – это средняя скорость обработки видеокадров (кадров в секунду).
- Задержка инференса – среднее время, необходимое для обработки одного кадра (время от начала инференса входных данных до получения результатов) в миллисекундах.

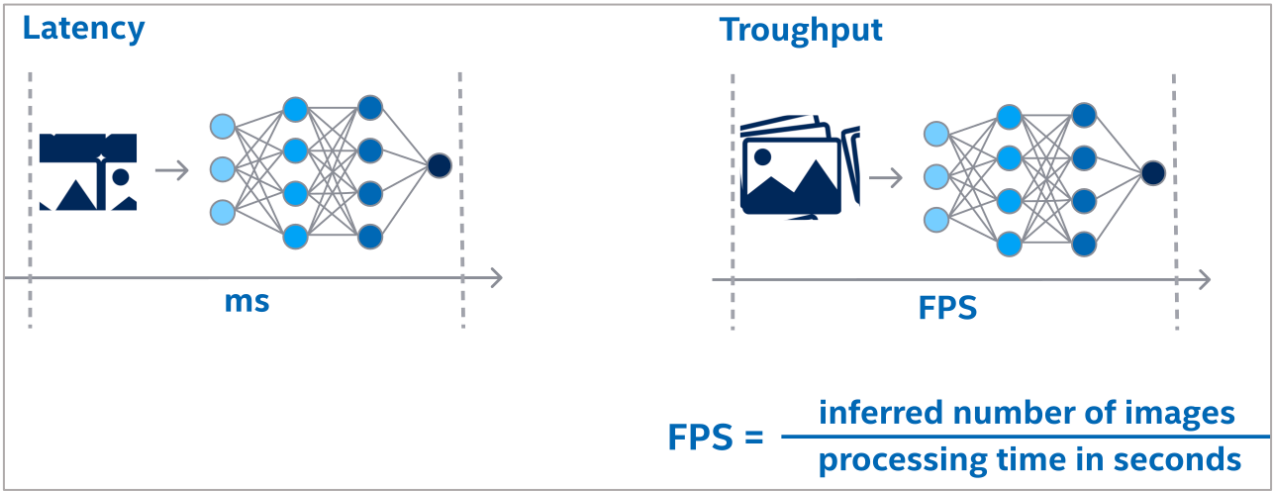

Рисунок 4.2 – Метрики

Приложение сохраняет метрики в файл для дальнейшего анализа. Задержка представлена как минимальная и средняя, FPS же представлен как средний и максимальный. Пример сохраненных метрик представлен на рис. 4.3.

```
    MEAN                    MIN                 MAX
Latency:
  CPU: 65.991 ms          61.417 ms
  GPU: 39.589 ms          39.257 ms
  MUL: 32.314 ms          31.889 ms
  HET: 34.701 ms          34.653 ms
FPS:
  CPU: 15.154                                 16.282
  GPU: 25.26                                  25.473
  MUL: 30.95                                  32.021
  HET: 28.82                                  29.192
```

Рисунок 4.3 – Пример метрик

4.1 Multi-device режим

В данном случае использовался плагин Multi-device с заданными моделями. Представлено сравнение метрик при инференсе на CPU, GPU, Multi-device и оригинальной модели. Далее построены столбчатые диаграммы для метрик и их пиковых значений. На рисунках 4.4, 4.5 и 4.6 представлены графики для модели с точностью FP32 со сравнением с оригинальной моделью.

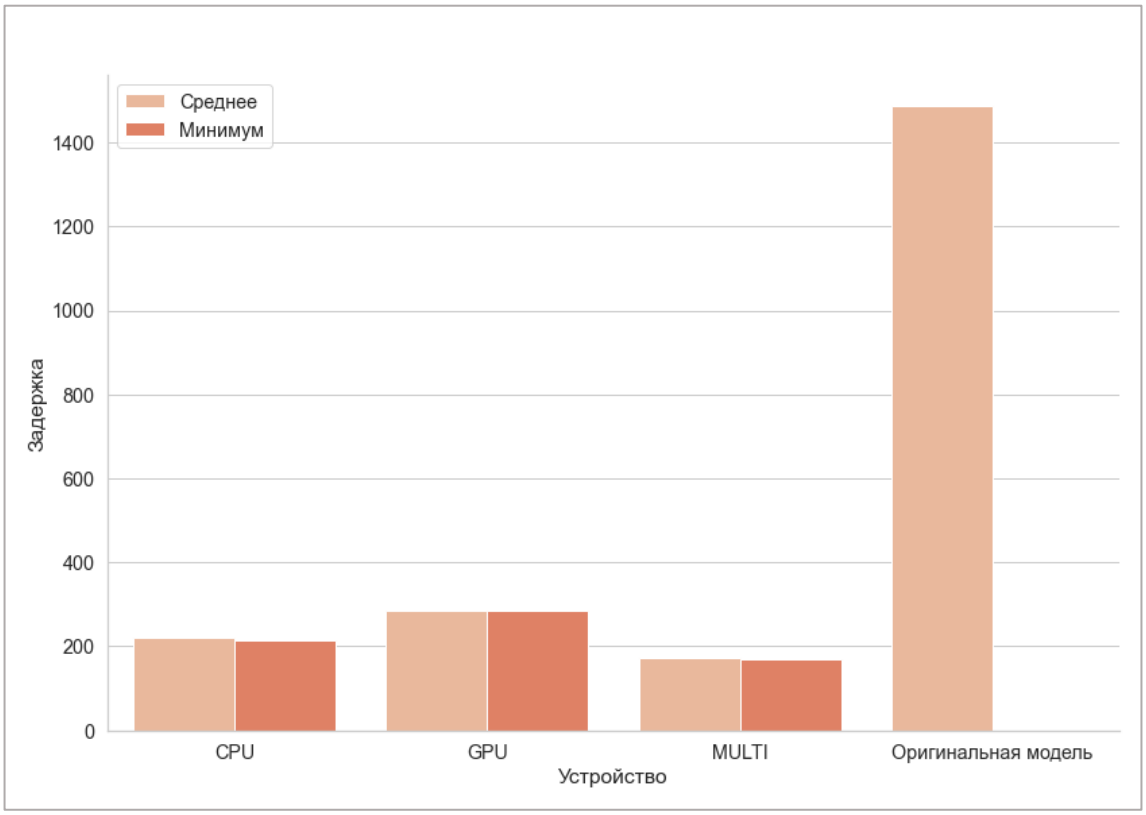

Рисунок 4.4 – График задержки инференса модели точности FP32 и сравнение с оригинальной моделью

Из графика 4.4 видно, что по сравнению с оригинальной моделью, использование OpenVINO Toolkit дает значительный прирост производительности вне зависимости от устройства исполнения. Можно увидеть прирост производительности в 7 раз. Но так как значения оригинальной модели и модели OpenVINO слишком сильно отличаются и графики получаются не информативными, в последующих графиках будет представлено сравнение инференса только на разных устройствах.

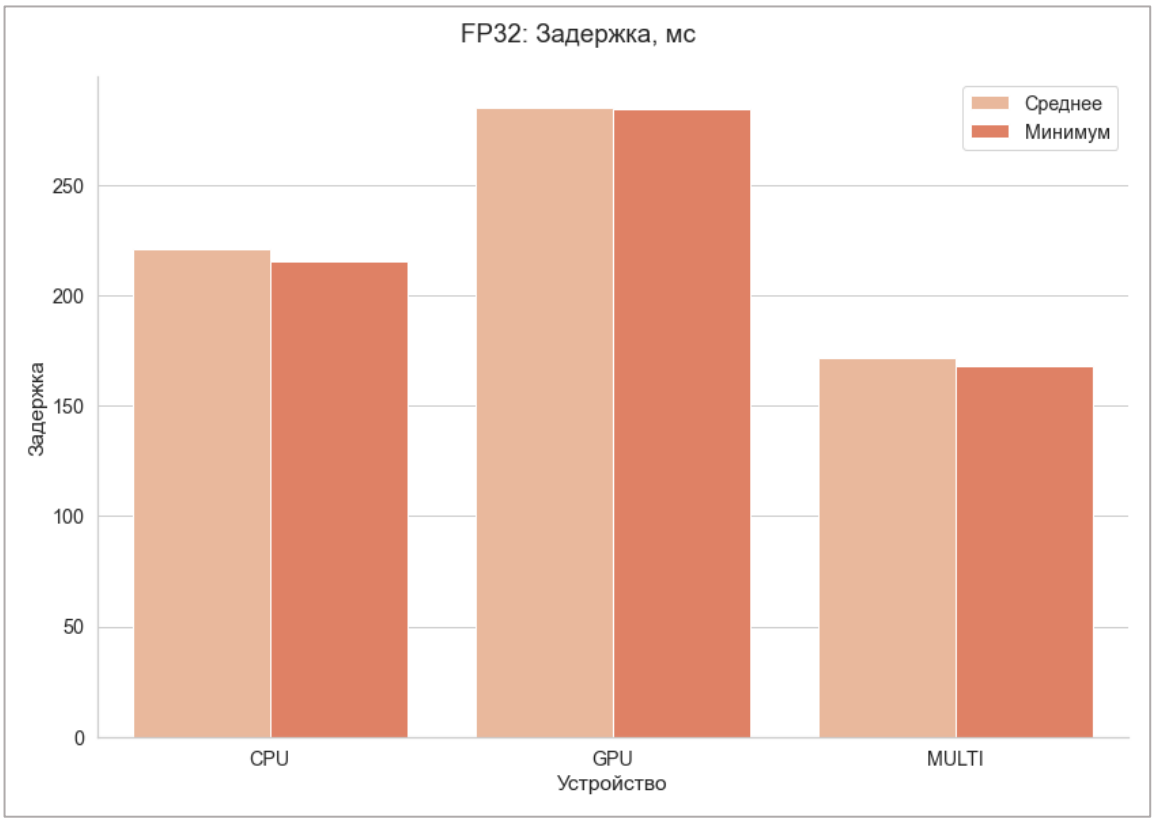

Рисунок 4.5 – График задержки инференса модели точности FP32

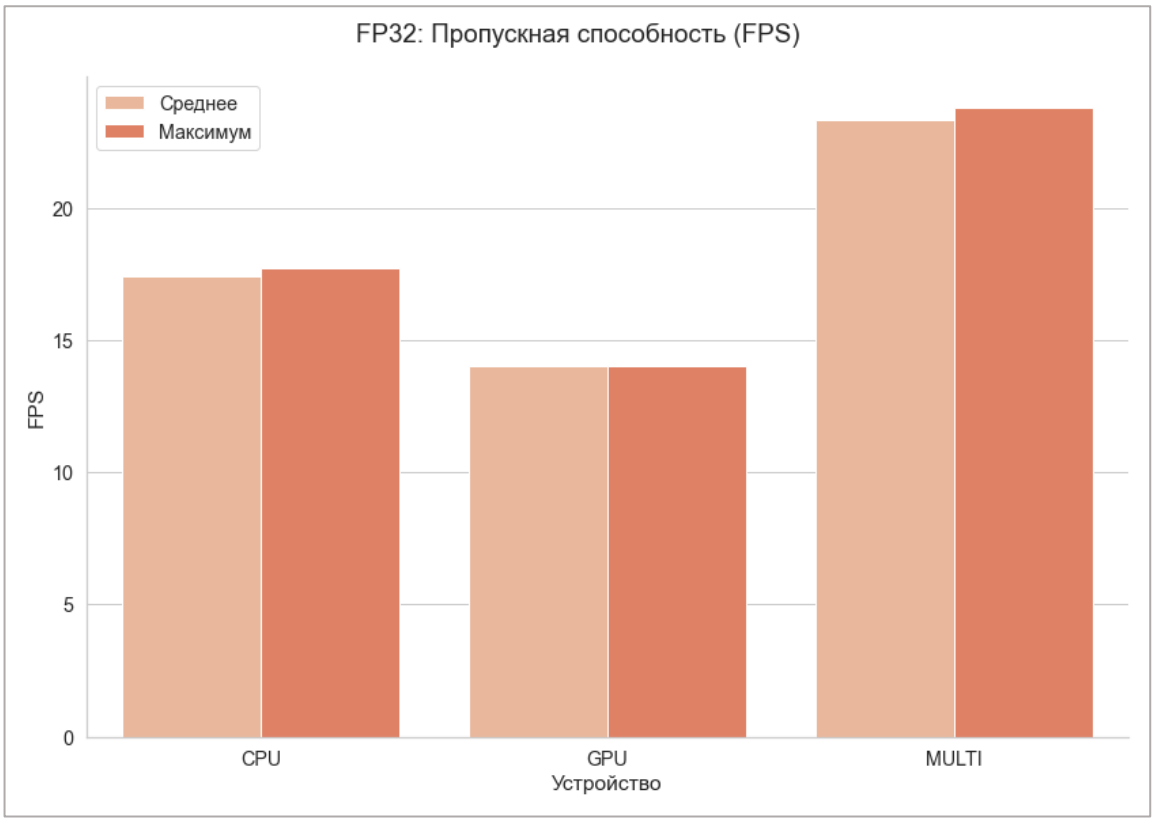

Рисунок 4.6 – График пропускной способности модели точности FP32

Видно, что использование мультидевайсного плагина, действительно повышает производительность по сравнению с выполнением только на CPU или GPU. Прирост для данной точности по сравнению с CPU составляет 33%, а с GPU – 66%.

На рис. 4.7 и 4.8 представлены графики задержки и FPS для модели с точностью FP16.

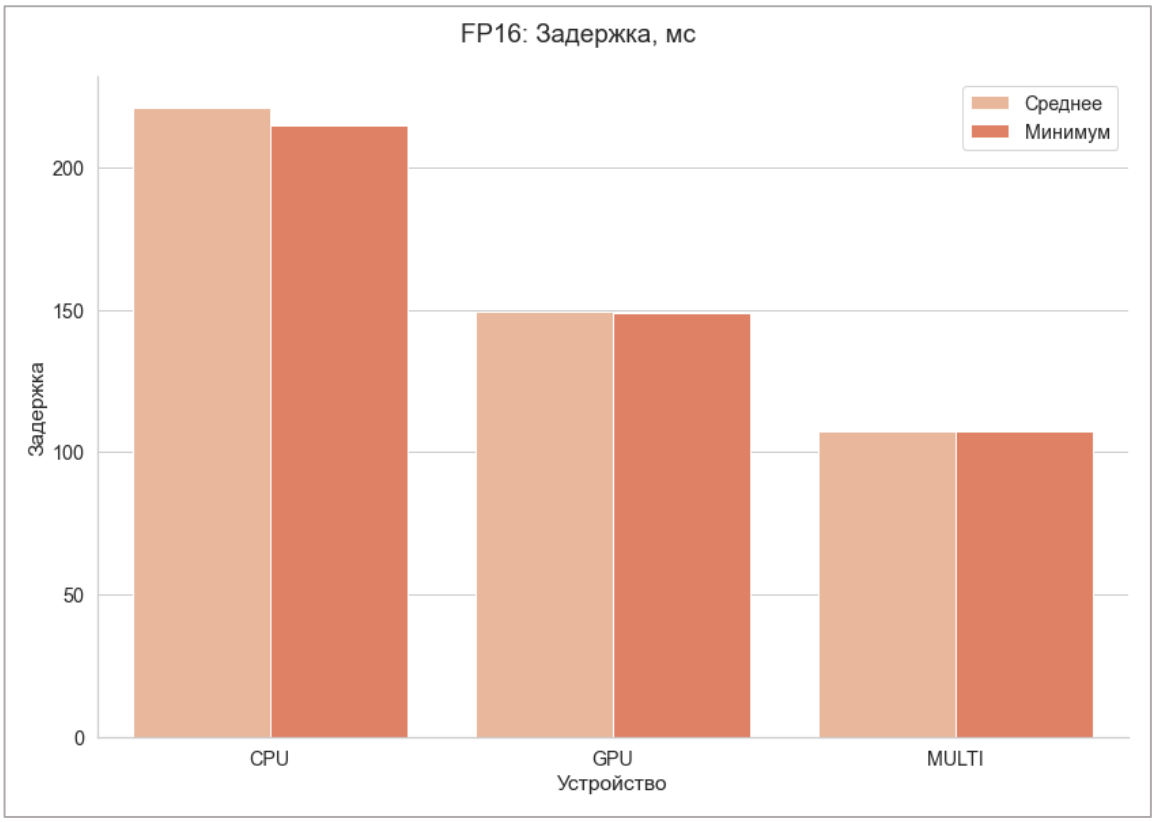

Рисунок 4.7 – График задержки инференса модели точности FP16

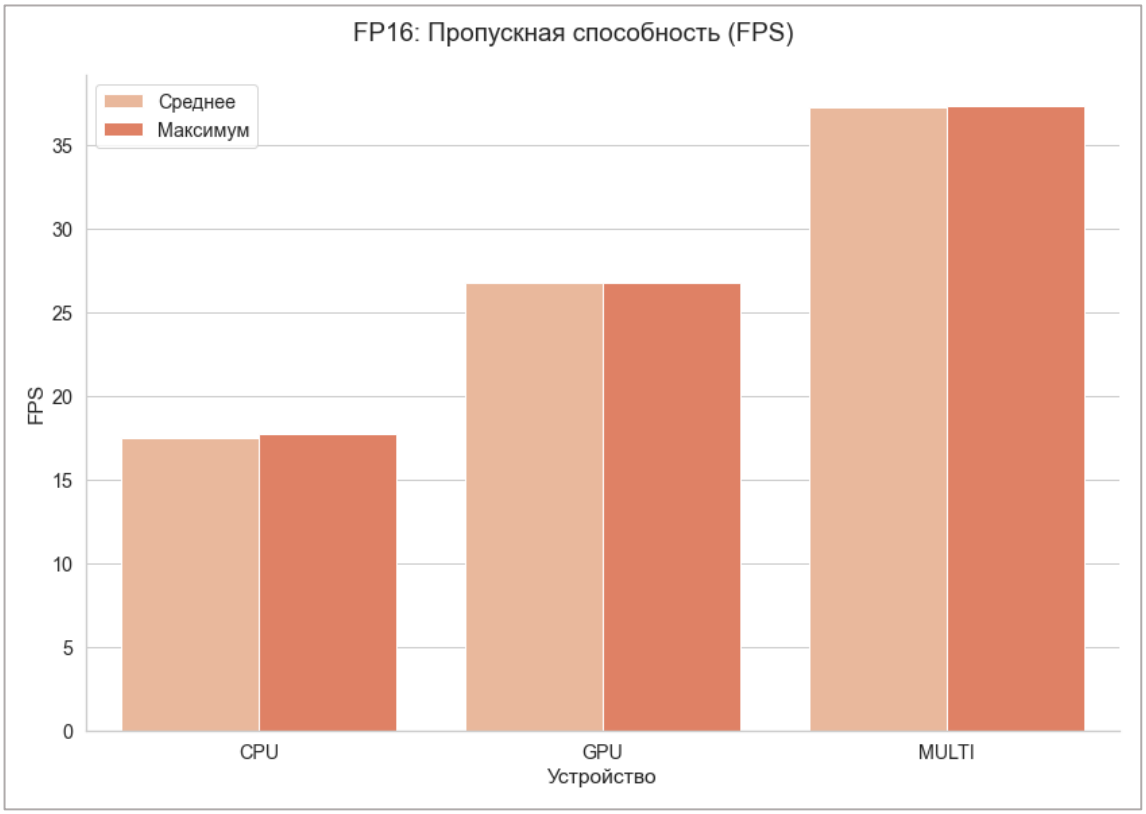

Рисунок 4.8 – График пропускной способности модели точности FP16

Из графиков, можно заметить, что при точности FP32 CPU показывает лучшую производительность, а при FP16 GPU. Это связано с ограничениями поддерживаемых форматов моделей и устройств [16], которые представлены на рис. 4.9. Например, IR FP16 изначально предназначен для устройств GPU, тогда как для CPU IR FP16 обычно масштабируется до FP32 автоматически при загрузке. Прирост для данной точности по сравнению с CPU составляет 113%, а с GPU – 39%.

Plugin	FP32	FP16	I8
CPU plugin	Supported and preferred	Supported	Supported
GPU plugin	Supported	Supported and preferred	Supported*
VPU plugins	Not supported	Supported	Not supported
GNA plugin	Supported	Supported	Not supported

Рисунок 4.9 – Поддерживаемые форматы моделей

На рис. 4.10 и 4.11 представлены графики для точности модели INT8.

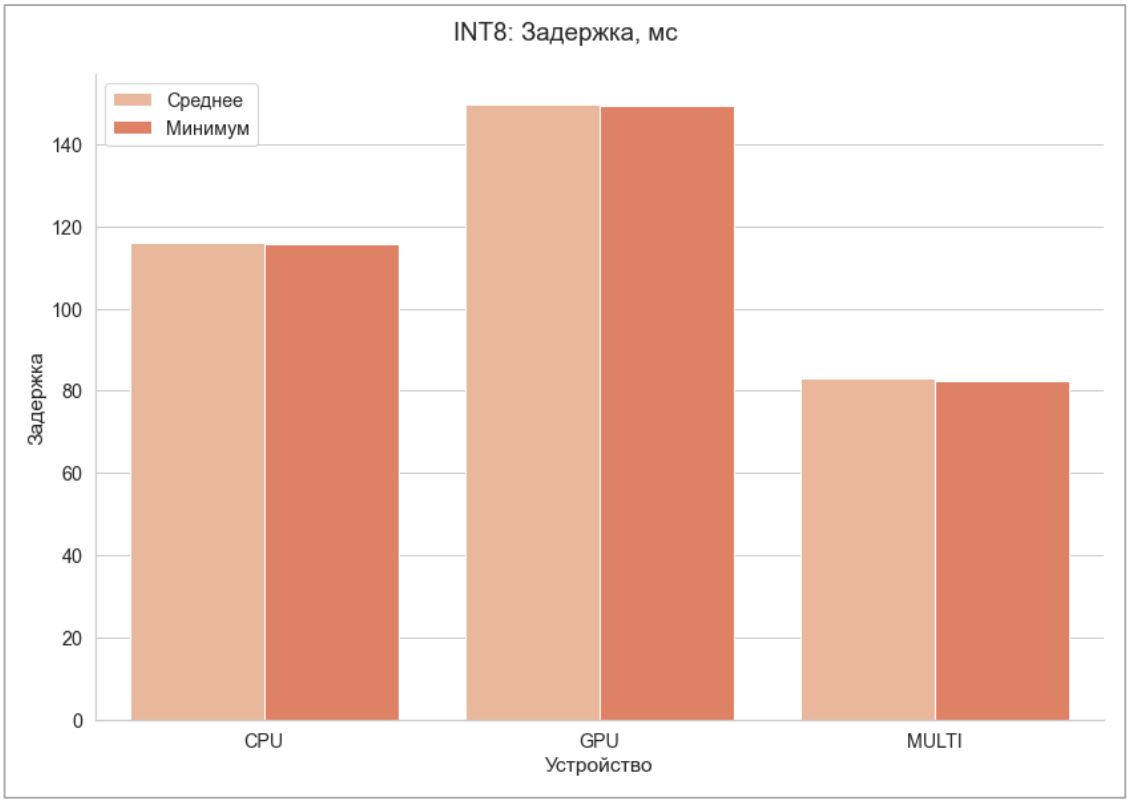

Рисунок 4.10 – График задержки инференса модели точности INT8

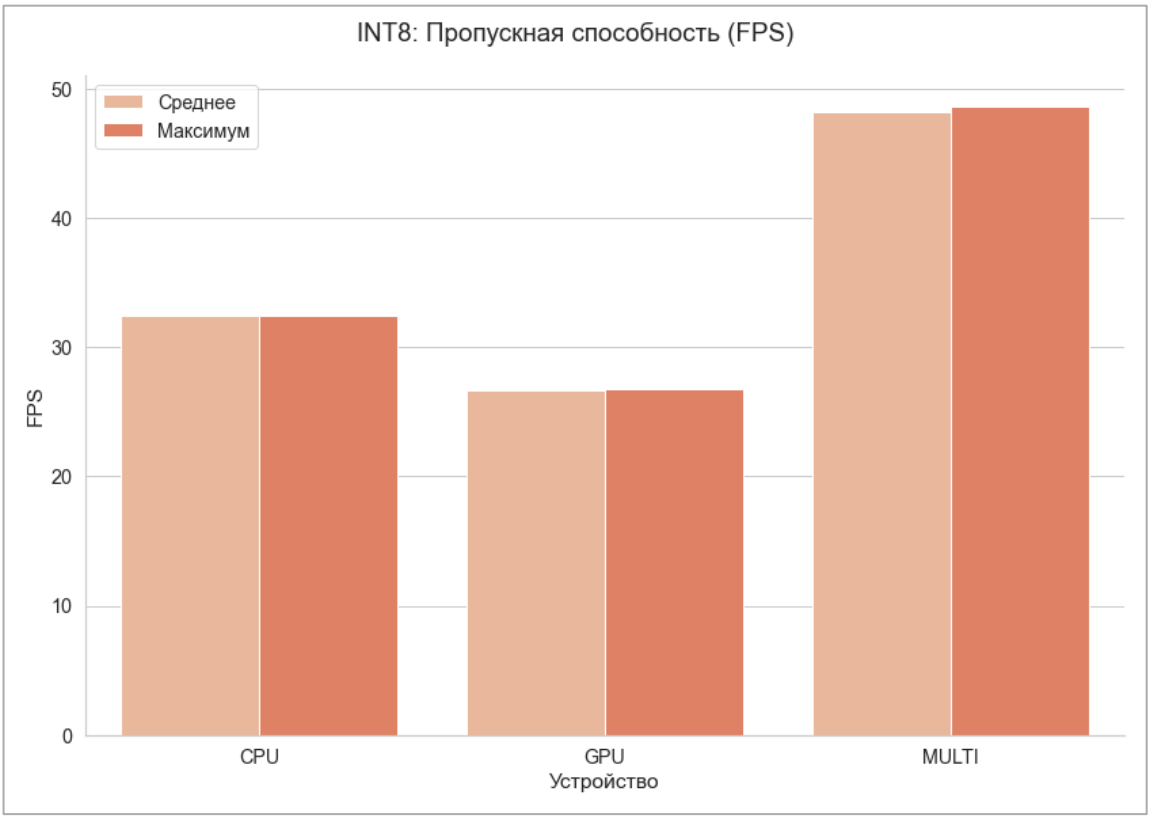

Рисунок 4.11 – График пропускной способности модели точности INT8

Прирост для данной точности по сравнению с CPU составляет 48%, а с GPU – 80%.

На данных графиках видно, что CPU снова показал лучшую производительность, чем GPU. Это объясняется тем, что в настоящее время только ограниченный набор топологий может выиграть от включения модели INT8 на GPU, и в данном случае так и происходит.

Для Multi-device и гетерогенного исполнения поддерживаемые форматы моделей зависят от фактических базовых устройств. Как правило, предпочтительным является FP16, так как он наиболее распространен и производителен (FP32 не требуется). Но в данном эксперименте формат INT8 показал наилучшую производительность.

Из всех представленных графиков можно заметить, что лучшую производительность действительно показывает Multi-device режим.

Сравнительный график работы данного режима для моделей разной точности представлен на рис. 4.12.

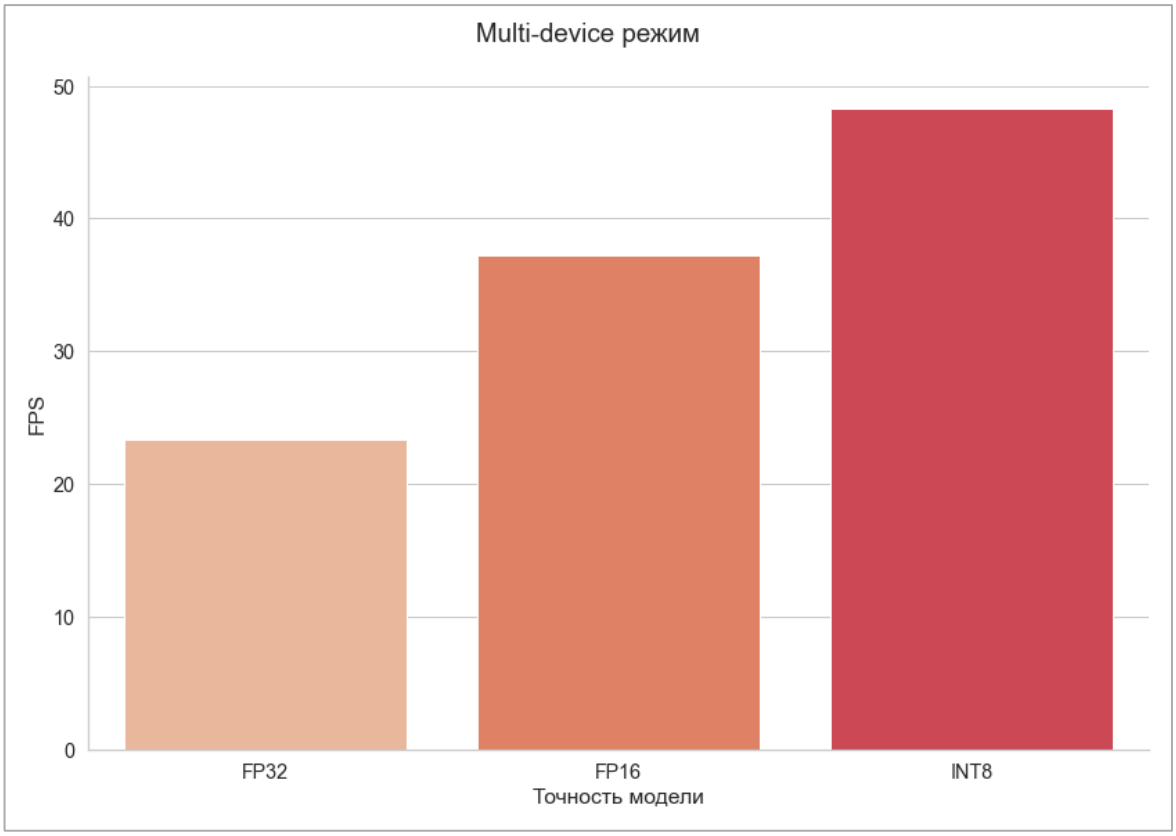

Рисунок 4.12 – Сравнение производительности Multi-device режима для моделей с разной точностью

Как видно точностью напрямую влияет на производительность. В общем можно подытожить, что использование данного режима значительно ускоряет инференс нейронной сети по сравнению с оригинальной моделью, так как прирост производительности составил 9, 13 и 17 раз для точностей FP32, FP16 и INT8 соответственно.

4.2 Гетерогенный режим

В данном случае использовался гетерогенный плагин с заданными моделями. Представлено сравнение метрик при инференсе на CPU, GPU и гетерогенном режиме. Построены столбчатые диаграммы для средних значений метрик.

Выполнение через гетерогенный плагин можно разделить на два независимых этапа:

- Установка аппаратного сродства к слоям
- Загрузка сети в плагин Heterogeneous, разбиение сети на части и выполнение их через плагин

Следует отметить, что инференс через гетерогенный плагин в отличии от Multi-device выполняется не параллельно, и, хотя, он используется для ускорения инференса, не следует ожидать какого-либо скачка производительности.

Некоторые топологии плохо поддерживаются для гетерогенного выполнения на некоторых устройствах или вообще не могут быть выполнены в этом режиме. Если передача данных из одной части сети в другую в гетерогенном режиме занимает больше времени, чем в обычном режиме, возможно, не имеет смысла выполнять их в гетерогенном режиме.

В данном случае попробуем выполнить наиболее требовательную к вычислениям часть сети на GPU, то есть слои свертки. Для этого определим какие слои относятся к данной части и установим сродство вручную, так как по умолчанию только неподдерживаемые слои выполняются на резервном устройстве:

1. Сначала получим карту связывания для конфигурации по умолчанию, где все слои выполняются на CPU.

```
layers_map = ie_core.query_network(network=net,
device_name="HETERO:CPU,GPU")
cur_affinity = open(f'{path}\metrics\affinity.txt', 'w')
for layer in layers_map:
    cur_affinity.write(f'{layer}: {layers_map[layer]}\n')
cur_affinity.close()
```

2. Получим сеть в виде nGraph.

```
ng_func = ng.function_from_cnn(net)
```

3. Найдем все сверточные слои в сети.

```
conv_layers = []
for layer in layers_map:
    index = layer.find('conv')
    if index != -1:
        conv_layers.append(layer)
```

4. Присваиваем каждому сверточному слою выполнение на GPU.

```
for i in conv_layers:
    layers_map[i] = 'GPU'
```

5. Создаем новое связывание

```
for node in ng_func.get_ordered_ops():
    affinity = layers_map[node.get_friendly_name()]
    node.get_rt_info()["affinity"] = affinity
ie_core.set_config(config={'HETERO_DUMP_GRAPH_DOT':'YES'},
device_name='HETERO')
exec_net = ie_core.load_network(net,
device_name="HETERO:CPU,GPU")
```

На рисунках 4.13-4.15 представлены графики для модели с точностью FP16 со сравнением с оригинальной моделью. Из предыдущего раздела можно понять, что FP16 – предпочтительная для использования точность.

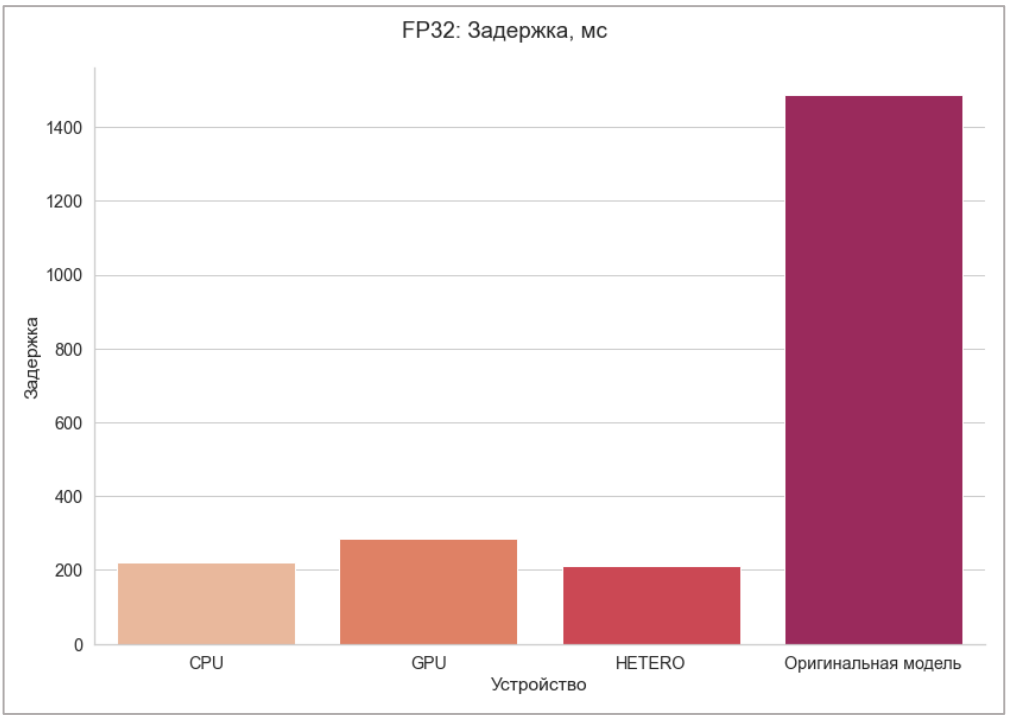

Рисунок 4.13 – График задержки инференса модели точности FP32 и сравнение с оригинальной моделью

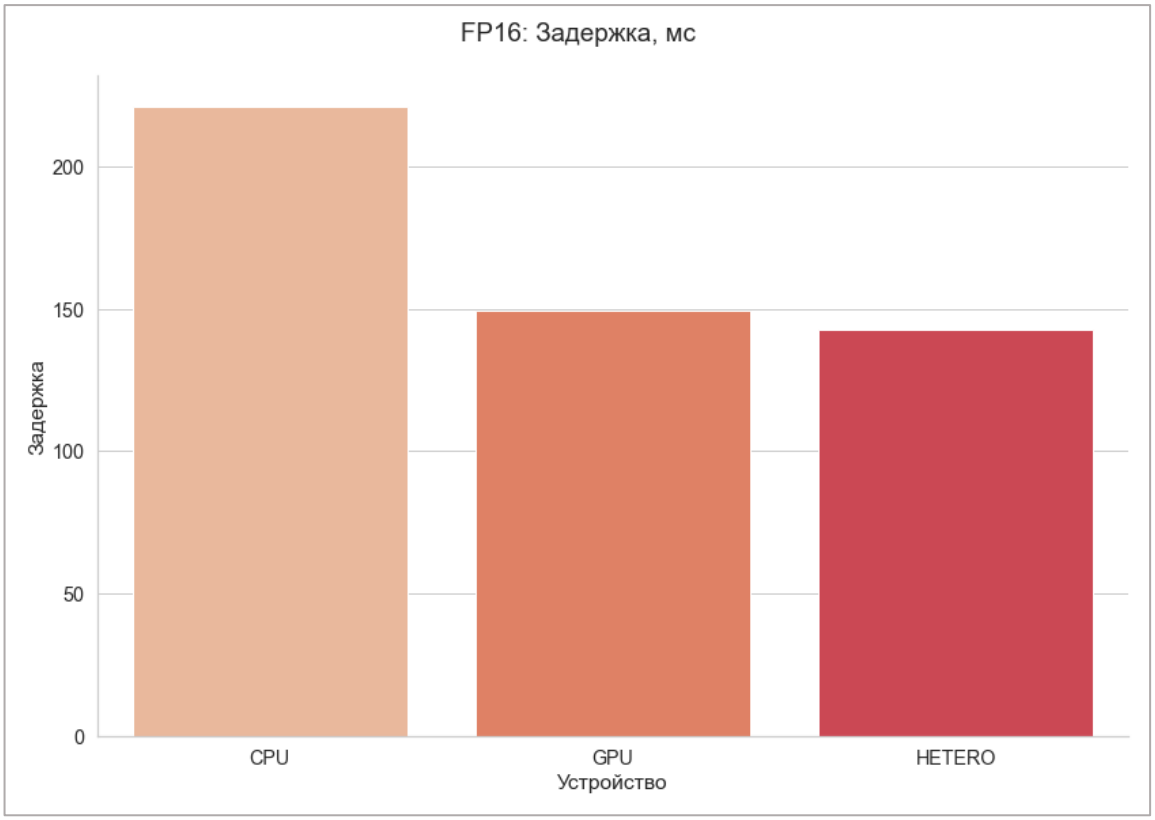

Рисунок 4.14 – График задержки инференса модели точности FP16

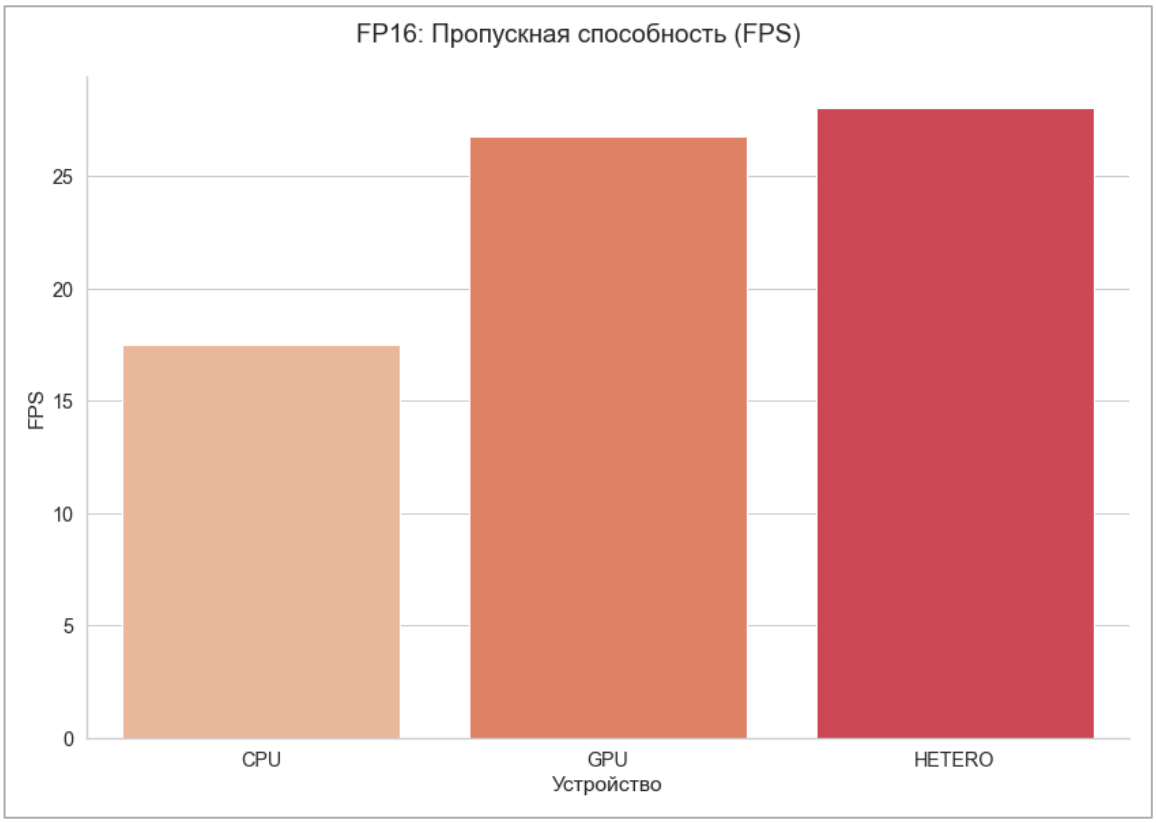

Рисунок 4.15 – График пропускной способности модели точности FP16

На представленных графиках видно, что гетерогенный режим дает совсем незначительный прирост производительности по сравнению с выполнением на CPU или GPU. Скорее всего это связано со сложной топологией сети и передача данных из одной части сети в другую в гетерогенном режиме занимает больше времени. Прирост производительности в случае выполнения в гетерогенном режиме по сравнению с CPU/GPU в среднем составляет всего 8%.

Можно сделать вывод, что применение Multi-device режима предпочтительнее, из-за намного большего прироста производительности.

ГЛАВА 5. ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ВКР

Данная выпускная квалификационная работа посвящена исследованию эффективности инструмента Intel OpenVINO Toolkit, который предназначен для оптимизации инференса нейронных сетей на целевых устройствах, в частности исследуются оптимизации в случае распределенной обработки информации.

Актуальность исследования обосновывается потребностью в оптимизации количества потребляемых ресурсов при использовании моделей глубокого обучения, как денежных, так и вычислительных и увеличением производительности моделей глубокого обучения.

В разделе экономического обоснования рассчитываются:

1. Трудоемкость выполнения ВКР.
2. Заработная плата исполнителя и руководителя.
3. Отчисления на социальные нужды
4. Затраты на сырье и приобретение расходных материалов
5. Затраты на эксплуатацию и содержание оборудования
6. Затраты на услуги сторонних организаций
7. Амортизационные отчисления
8. Накладные расходы

5.1 Календарный план выполнения работ

Исследованием занимались два человека: научный руководитель и студент. Для расчета расходов на оплату труда была определена длительность всех этапов исследования эффективности оптимизации Intel OpenVINO Toolkit – количество рабочих дней, затраченных исполнителем при проведении исследования.

В таблице 5.1 представлены данные о затраченных трудовых ресурсах. В таблице указаны этапы проведения исследования и соответствующие трудоемкости исполнителя и научного руководителя.

Таблица 5.1 – Трудоемкость работ

№	Наименование работы	Длительность работы, чел.дн.	
		Руководитель	Исполнитель
1	Разработка технического задания	1	2
2	Обзор предметной области	1	3
3	Сбор информации по используемому инструменту	1	6
4	Разработка метода оценки оптимизации инференса	—	6
5	Создание демо-приложения	—	8
6	Анализ эффективности оптимизации	—	13
7	Оформление пояснительной записки	4	10
8	Защита проекта	3	3
ИТОГО		10	51

5.2 Расходы на оплату труда

На статью “Расходы на оплату труда” относят заработную плату научных сотрудников, инженеров и прочего инженерно-технического персонала, непосредственно занятых выполнением работы [17].

Для каждого из участников исследования была рассчитана дневная ставка, которую можно посчитать как отношение месячной заработной платы к количеству дней в одном месяце. В Приказе ректора № ОД/0539 от 27.09.2019 «Об увеличении уровня оплаты труда работникам университета и. об изменении размеров минимальных должностных окладов и должностных окладов по профессионально-квалификационным группам» (Приказ 15.10.2020 № ОД/0432), должностной оклад руководителя, как кандидата технических наук и доцента составляет 40600 руб., исполнителя – 14500 руб.

Количество рабочих дней равно 21. Следовательно, ставка заработной платы за единицу времени (день) равна:

- для руководителя – 1933.3 руб.

- для исполнителя – 690.48 руб.

Расходы на выплату основной и дополнительной заработной платы каждого их исполнителей были вычислены с использованием данных о дневной ставке и длительности работ из таблицы 5.1. Для вычислений использовалась формула (5.1):

$$З_{\text{осн.з/пл}} = \sum_{i=1}^{k} T_i * C_i \qquad (5.1)$$

где $З_{\text{осн.з.пл}}$ – расходы на основную заработную плату исполнителей, руб.; k – количество исполнителей; T_i – время, затраченное i-м исполнителем на проведение исследования, дни или часы; C_i – ставка i-го исполнителя, руб./день или руб./час.

Результаты приведены в таблице 5.2.

Таблица 5.2 – Основная заработная плата исполнителей

Исполнитель	**Дневная ставка, руб./день**	**Длительность работы, чел.дн.**	**Основная заработная плата, руб.**
Студент	690.48	51	35214.48
Научный руководитель	1933.3	10	19333
Итого			54547.48

Согласно «Положению об оплате труда работников университета» от 15.10.2020, норматив дополнительной заработной платы составляет 8,3%. Для расчета дополнительной заработной платы была использована формула (5.2).

$$З_{\text{доп.з/пл}} = З_{\text{осн.з/пл}} * \frac{Н_{\text{доп}}}{100} \qquad (5.2)$$

где $З_{доп.з/пл}$ – расходы на дополнительную заработную плату исполнителей, руб.; $З_{осн.з/пл}$– расходы на основную заработную плату исполнителей, руб.; $Н_{доп}$ – норматив дополнительной заработной платы, %.

Результаты приведены в таблице 5.3.

Таблица 5.3 – Дополнительная и общая заработная плата исполнителей

Исполнитель	**Основная заработная плата, руб.**	**Дополнительная заработная плата, руб.**	**Заработная плата, руб.**
Студент	35214.48	2922.8	38137.28
Научный руководитель	19333	1604.64	20937.64
Итого			59074.92

5.3 Отчисления на социальные нужды

Далее, помимо основной и дополнительной заработной платы были посчитаны затраты на отчисления на социальные нужды. На статью “Отчисления на социальные нужды” относят затраты, связанные с выплатой социальных отчислений с заработной платы.

Страховые взносы отчисляются в следующие государственные внебюджетные фонды социального назначения:

- Пенсионный фонд России (ПФР)
- Федеральный фонд обязательного медицинского страхования (ФФОМС)
- Фонд социального страхования (ФСС)

Отчисления на страховые взносы составляют 30.2% от основной и дополнительной заработной платы. Данные отчисления на страховые взносы рассчитываются по формуле (5.3)

$$З_{соц} = (З_{доп.з/пл} + З_{осн.з/пл}) * \frac{Н_{соц}}{100} \quad (5.3)$$

где $З_{соц}$ – отчисления на социальные нужды с заработной платы, руб.; $З_{осн.з/пл}$ – расходы на основную заработную плату исполнителей, руб.; $З_{доп.з/пл}$– расходы на дополнительную заработную плату исполнителей, руб.; $Н_{соц}$ – норматив отчислений страховых взносов на обязательное социальное, пенсионное и медицинское страхование, %.

$$З_{соц} = (54547.48 + 4527.44) * 0{,}302 = 17840.63 \text{ руб.}$$

5.4 Затраты на сырье и приобретение расходных материалов

Были посчитаны затраты на материальные расходы, к которым относят затраты на сырье, основные и вспомогательные материалы, покупные полуфабрикаты и комплектующие изделия, необходимые для выполнения работы с учетом транспортно-заготовительных расходов. В данном исследовании были использованы бумага и ручка для описания и обсуждения системы для оценки инференса и флеш-накопитель для получения и передачи необходимых файлов. Расчет материальных расходов был произведен по формуле (5.4).

$$З_{м} = \sum_{l=1}^{L} G_l * Ц_l * (1 + \frac{Н_{т.з.}}{100}) \quad (5.4)$$

где $З_{м}$ – затраты на сырье и материалы, руб.; l – индекс вида сырья или материала; G_l – норма расхода l-го материала на единицу продукции, ед.; $Ц_l$ – цена приобретения единицы l-го материала, руб./ед.; $Н_{т.з.}$ – норма транспортно-заготовительных расходов, %.

Результаты представлены в таблице 5.4.

Таблица 5.4 – Расходные материалы

Материалы	**Норма расхода на единицу продукции, шт.**	**Цена, руб.**	**Сумма, руб.**
Бумага офисная “Снегурочка”	1	470	470
Ручка шариковая “Deli Think”	2	159	318
USB Флеш-накопитель “FOLK”	1	799	799
Итого			1587
Транспортные расходы (10%)			158.7
Всего			1745.7

5.5 Затраты на услуги сторонних организаций

На статью “Затраты на услуги сторонних организаций” относят затраты по оплате всех видов работ, выполняемых сторонними организациями.

Для проведения исследования требуется свободно-распространяемая утилита Intel OpenVINO Toolkit. Для установки утилиты использовался доступ в сеть Интернет. Стоимость услуги без НДС была рассчитана по формуле (5.5).

$$З_{\text{без НДС}} = З_{\text{с НДС}} * \left(1 - \frac{20\%}{120\%}\right) \quad (5.5)$$

Расчет затрат для доступа в Интернет представлен в таблице 5.5.

Таблица 5.5 – Затраты на услуги сторонних организаций

Наименование	**Цена, руб.**	**Цена без НДС, руб.**	**Время использования, мес.**	**Общая стоимость, руб.**
Услуги Интернет-провайдера “Ростелеком”	810	675	2	1350
Итого				1350

5.6 Затраты на эксплуатацию и содержание оборудования

На статью “Затраты на эксплуатацию и содержание оборудования” относят затраты на содержание и эксплуатацию всех видов оборудования, используемого в работе.

В данном исследовании эксплуатировался только персональный компьютер. Для расчета затрат на эксплуатацию оборудования была использована формула (5.6).

$$З_{эо} = \sum_{i=1}^{m} C_i^{мч} * t_i^{м} \qquad (5.6)$$

где $З_{эо}$ – затраты на содержание и эксплуатацию оборудования, руб.; $C_i^{мч}$ – расчетная себестоимость одного машино-часа работы оборудования на i-й технологической операции, руб./м-ч; $t_i^{м}$ – количество машино-часов, затрачиваемых на выполнение i-й технологической операции, м-ч.

В данном исследовании длительность работы с компьютером составила 368 ч. Стоимость 1 кВт/ч по тарифу – 5.71 руб [18]. Компьютер потребляет 0.6 кВт/ч. Расчет затрат на эксплуатацию оборудования представлен в таблице 5.6.

Таблица 5.6 – Затраты на эксплуатацию оборудования

Оборудование	**Стоимость одного машино-часа, руб.**	**Длительность работы, ч.**	**Общая стоимость, руб.**
Компьютер	3.43	368	1262.24
Итого			1262.24

5.7 Амортизационные отчисления

В статье “Амортизационные отчисления” учитываются амортизационные отчисления по всем видам основных средств, которые используются при выполнении ВКР. При исследовании был использован

персональный компьютер, стоимостью 72000 руб. Амортизационные отчисления по основному средству за год можно рассчитать по формуле (5.7).

$$\text{А}_i = \text{Ц}_{\text{п.н.}i} * \frac{H_{ai}}{100} \quad (5.7)$$

Согласно Постановлению Правительства РФ от 01.01.2002 №1 (ред. от 07.07.2016) «О классификации основных средств, включаемых в амортизационные группы», нормативный срок полезного использования используемого оборудования 2 – 3 года. Было выбрано 3 года.

Далее была определена годовая норма амортизации как обратная величина от срока полезного использования, умноженную на 100%:

$$\text{Н}_{ai} = 100\% * \frac{1}{3} = 33.3\%$$

Для определения величины амортизационных отчислений по основным средствам, используемым в процессе выполнения ВКР, было определено время, в течение которого использует основное средство с помощью формулы (5.8).

$$\text{А}_{i\text{ВКР}} = \text{А}_i * \frac{T_{i\text{ВКР}}}{12} \quad (5.8)$$

где $\text{А}_{i\text{ВКР}}$ – амортизационные отчисления по i-му основному средству, используемому в работе над ВКР в руб.; А_i – амортизационные отчисления за год по i-му основному средству в руб.; $T_{i\text{ВКР}}$ – время, в течение которого использовалось i-ое основное средство, мес.

Основные средства использовались в процессе выполнения ВКР в течение 2-х месяцев.

Результаты расчетов приведены в таблице 5.7.

Таблица 5.7 – Амортизационные отчисления

Оборудование	**Годовая норма амортизации, %**	**Амортизационные отчисления за год, руб.**	**Амортизационные отчисления по средству, используемому в работе над ВКР, руб.**
Компьютер	33.3	23976	3996
Итого			3996

5.8 Накладные расходы

Накладные расходы – это расходы, связанные с расходами на управление и обслуживание, в том числе коммунальные платежи и затраты на продвижение продукта.

Норматив накладных расходов был принят равным 20%. Накладные расходы были рассчитаны по формуле (5.9).

$$З_н = 20\% * (З_{осн.з/пл} + З_{доп.з/пл}) \quad (5.9)$$

$$З_н = 0.2 * 59074.92 = 11814.98 \text{ руб.}$$

5.9 Спецоборудование

На статью “Спецоборудование” относятся затраты на приобретение (или изготовление) специальных приборов, стендов, другого специального оборудования, необходимого для выполнения работы.

В данном исследовании расходы на спецоборудование не предусмотрены.

5.10 Себестоимость проекта

Расчет себестоимости представлен в таблице 5.8.

Таблица 5.8 – Калькуляция затрат на ВКР

№ п/п	Наименование статьи	Сумма, руб.
1	Расходы на оплату труда	59074.92
2	Отчисления на социальные нужды	17840.63
3	Сырье и расходные материалы	1745.7
4	Затраты на услуги сторонних организаций	1350
5	Расходы на эксплуатацию и содержание оборудования	1262.24
6	Амортизационные отчисления	3996
7	Накладные расходы	11814.98
8	Спецоборудование	—
ИТОГО		97084.47

5.11 Выводы

В данном разделе работы рассчитана себестоимость исследования эффективности инструмента Intel OpenVINO Toolkit, который предназначен для оптимизации инференса нейронных сетей на целевых устройствах в случае распределенной обработки информации.

Были рассчитаны следующие статьи калькуляции:

1. Расходы на оплату труда
2. Отчисления на социальные нужды
3. Сырье и расходные материалы
4. Затраты на услуги сторонних организаций
5. Расходы на эксплуатацию и содержание оборудования
6. Амортизационные отчисления
7. Накладные расходы

Определена себестоимость исследования равная 97084.47.

Компьютерное зрение – одна из тех технологий, которые в последние несколько лет развиваются скачками. Сейчас компьютерное зрение является движущей силой и помощником искусственного интеллекта. Крупнейшие технические корпорации мира инвестируют в исследование компьютерного зрения огромные средства, особенно интересен вопрос оптимизации потребляемых ресурсов при использовании моделей глубокого обучения, из чего можно сделать вывод о целесообразности данной работы.

ЗАКЛЮЧЕНИЕ

В ходе данной исследовательской работы было выяснено, что использование Intel OpenVINO Toolkit для распределенной обработки информации позволило сильно улучшить производительность работы модели. В течении данного исследования были решены следующие задачи:

- Изучен функционал OpenVINO Toolkit. Данный инструмент глобально состоит из двух частей: Model Optimizer и Inference Engine. Model Optimizer позволяет преобразовать модель, обученную с помощью какого-либо фреймворка для создания оптимизированного промежуточного представления (IR) модели на основе обученной топологии сети, и значений весов и смещений. В свою очередь Inference Engine позволяет достичь максимально эффективного использования всех ресурсов доступных устройств исполнения. Также именно Inference Engine позволяет запускать модель сразу на нескольких устройствах, используя Multi-device и гетерогенный плагины.
- Разработана система оценки оптимизации инференса. Было разработано демо-приложение, которое позволяет оценить эффективность оптимизации инференса модели для распределённой обработки данных. Приложение позволяет использовать различные устройства для инференса и собирать статистику о производительности. Сохраняются такие метрики как задержка и пропускная способность, которые могут наиболее объективно оценить производительность.
- Проанализирована эффективность оптимизации для Multi-device плагина. Исследование показало, что использование данного плагина улучшает производительность в 9, 13 и 17 раз для точностей FP32, FP16 и INT8 соответственно. Но стоит отметить, что использование данного режима не всегда способно предоставить нужные улучшения, так все зависит от топологии нейронной сети и используемых устройств, а также точности модели.

- Проанализирована эффективность оптимизации для гетерогенного режима. Анализ использования данного плагина показал, что по сравнению с использованием Multi-device режима данный режим не показывает большого прироста производительности. Прирост составил всего 8%. Скорее всего это связано с топологией выбранной сети.

В общем можно с уверенностью сказать об эффективности данного инструмента, так как он изначально предполагает использование уже существующих моделей, прост в интеграции в приложение и предоставляет множество оптимизаций производительности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Горячкин Б.С., Китов М.А. КОМПЬЮТЕРНОЕ ЗРЕНИЕ // E-Scio. 2020. №9 (48).

2. Обработка изображений [Электронный ресурс]. URL: https://neptune.ai/blog/image-processing-python (дата обращения: 02.05.2022)

3. Официальная документация Intel OpenVINO Toolkit [Электронный ресурс]. URL: https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/ overview.html (дата обращения: 26.04.2022)

4. Инструкция по установке OpenVINO [Электронный ресурс]. URL: https://docs.openvino.ai/latest/openvino_docs_install_guides_installing_openvino_windows.html (дата обращения: 26.04.2022)

5. OpenVINO Toolkit Hello Object Detection [Электронный ресурс]. URL: https://docs.openvino.ai/2021.4/notebooks/004-hello-detection-with-output.html (дата обращения: 01.05.2022).

6. Детектирование лиц с помощью OpenVINO Toolkit [Электронный ресурс]. URL: https://gist.github.com/dkurt/bc567159e5173a6e86c5781badff9ac6 (дата обращения: 29.04.2022)

7. Computer Vision Annotation Tool [Электронный ресурс]. URL: https://habr.com/ru/company/intel/blog/433772/ (дата обращения: 02.05.2022)

8. Курс обучающих видео-лекций по OpenVINO Toolkit [Электронный ресурс]. URL: https://www.youtube.com/watch?v=kY9nZbX1DWM&list=PLDKCjIU5YH6jMzc TV5_cxX9aPHsborbXQ (дата обращения: 30.04.2022)

9. Open Model Zoo [Электронный ресурс]. URL: https://github.com/opencv/open_model_zoo (дата обращения: 30.04.2022)

10. Обзор инструментария OpenVINO Toolkit [Электронный ресурс]. URL: https://delta-course.org/docs/IntelCVSchool/IntelCVSchoolOpenVINO.pdf (дата обращения: 29.04.2022)

11. Приемы повышения производительности инференса глубоких моделей с DL Workbench [Электронный ресурс]. URL: https://habr.com/ru/company/ intel/blog/549634/ (дата обращения: 01.05.2022)

12. OpenVINO Toolkit Model Downloader [Электронный ресурс]. URL: https://docs.openvino.ai/2021.4/openvino_docs_IE_DG_Tools_Model_Downloader.html (дата обращения: 10.05.2022).

13. Multi-device плагин Intel OpenVINO Toolkit [Электронный ресурс]. URL: https://docs.openvino.ai/2021.4/openvino_docs_IE_DG_supported_plugins_MULTI.html?sw_type=switcher-python (дата обращения: 28.04.2022)

14. Гетерогенный плагин Intel OpenVINO Toolkit [Электронный ресурс]. URL: https://docs.openvino.ai/2021.4/openvino_docs_IE_DG_supported_plugins_HETERO.html?sw_type=switcher-python (дата обращения: 29.04.2022)

15. OpenVINO Toolkit Runtime Optimization Guide [Электронный ресурс]. URL: https://docs.openvino.ai/2021.4/openvino_docs_IE_DG_Intro_to_Performance.html?highlight=runtime (дата обращения: 23.05.2022).

16. OpenVINO Toolkit Supported Devices [Электронный ресурс]. URL: https://docs.openvino.ai/2021.4/openvino_docs_IE_DG_supported_plugins_Supported_Devices.html#doxid-openvino-docs-i-e-d-g-supported-plugins-supported-devices (дата обращения: 23.05.2022).

17. Алексеева О.Г. Методические указания по экономическому обоснованию выпускных квалификационных работ бакалавров: Метод. указания, СПб.: Изд-во СПбГЭТУ “ЛЭТИ”, 2013. 17 с.

18. Петербургская сбытовая компания “Петроэлектросбыт”: тарифы на электроэнергию по Санкт-Петербургу [Электронный ресурс]. URL: https://www.pes.spb.ru/for_customers/electricity_tariffs/electricity_tariffs_for_st_petersburg/ (дата обращения: 22.05.2022).