СОДЕРЖАНИЕ

ТЕРМИНЫ И ПОНЯТИЯ

Операция – абстрактное понятие математической функции, выбранной для определенной цели. Примеры операций: ReLU, Свертка, Add и др.

Расширение механизма вывода – специфичный для устройства модуль, реализующий пользовательские операции (набор ядер).

Ядро – реализация операционной функции в плагине OpenVINO, в этом случае математика запрограммирована (на C++ и OpenCL) для выполнения операции на целевом оборудовании (CPU или GPU).

API (Application Programming Interface) – Интерфейс прикладного программирования.

AVX (Advanced Vector Extensions) – расширение системы команд x86 для микропроцессоров Intel и Amd.

Batch – количество изображений для анализа во время одного вызова infer. Максимальный размер пакета является свойством сети и устанавливается перед загрузкой сети в плагин. В представлении макета данных изображений NHWC, NCHW и NCDHW N относится к количеству изображений в пакете.

Blob – контейнер памяти используется для хранения входов, выходов сети, весов и смещений слоев.

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

CNN - Свёрточная нейронная сеть.

CPU (Central Processing Unit) – центральный процессор.

CV (Computer Vision) – компьютерное зрение.

DL (Deep learning) – Глубокое обучение.

DLL (Dynamic Link Library) – библиотека динамических ссылок.

DNN (Deep Neural Networks) – глубокие нейронные сети.

ELU (Exponential Linear rectification Unit) – Экспоненциальная линейная единица.

FP (Floating Point) – Плавающая точка.

FPGA (Field-Programmable Gate Array) – Программируемая пользователем вентильная матрица.

GPU (Graphics Processing Unit) – Графический процессор.

ICNNNetwork – интерфейс свёрточной нейронной сети, который Механизм вывода считывает из ИК. Состоит из топологии, весов и смещений

IExecutableNetwork - Экземпляр загруженной сети, который позволяет механизму вывода запрашивать несколько запросов вывода и выполнять вывод синхронно или асинхронно.

IInferRequest - интерфейс, представляющий конечную точку вывода на модели, загруженной в плагин и представленной исполняемой сетью.

InferenceEngineProfileInfo - представляет базовую информацию профилирования вывода для каждого слоя.

Inference Engine – библиотека C++ с набором классов, которые можно использовать в приложении для вывода входных данных (изображений) и получения результата.

Inference Engine API – базовый API по умолчанию для всех поддерживаемых устройств, который позволяет загружать модель из промежуточного представления, устанавливать форматы ввода и вывода и выполнять модель на различных устройствах.

Inference Engine Core – это программный компонент, который управляет выводом на определенных аппаратных устройствах Intel(R): CPU, GPU, MYRIAD, GNA и т.д.

IR (Intermediate Representation) – Промежуточное представление– нейронная сеть, используемая только Inference Engine в OpenVINO, абстрагирующая различные фреймворки и описывающая топологию модели, параметры операций и веса.

Layout – макет данных изображения относится к представлению пакета изображений. Макет показывает последовательность 4D или 5D тензорных данных в памяти. Типичный формат NCHW представляет пиксель в

горизонтальном направлении, строки по вертикальному измерению, плоскости по каналу и изображения в пакетном режиме.

MO (Model Optimizer) – кроссплатформенный инструмент командной строки, который облегчает переход между средой обучения и развертывания, выполняет статический анализ моделей и настраивает модели глубокого обучения для оптимального выполнения на конечных целевых устройствах.

NCDHW (Number of images, Channels, Depth, Height, Width) – Количество изображений, каналов, глубина, высота, ширина.

NCHW (Number of images, Channels, Height, Width) – Количество изображений, каналов, высота, ширина.

NN (Neural Network) – Нейронная сеть.

OS (Operating System) – Операционная система.

ReLU (Rectified Linear Unit) – Линейная функция активации.

ВВВЕДЕНИЕ

В современном мире большую популярность приобрел искусственный интеллект. Его применение позволяет решить множество рутинных и сложных задач, но самое главное его можно применить на большом объёме данных, который человек не в состоянии обработать.

По этой причине бурное развитие ИИ, в частности и компьютерного зрения, привело к появлению больших и сложных алгоритмов обработки и хранения данных. Эти алгоритмы даже превзошли человека в определенных областях, к примеру, в распознавании образов на изображении.

По мнению некоторых аналитических изданий, ожидается, что рынок компьютерного зрения вырастет в среднем на 8,38% в прогнозируемом периоде с 2020 по 2027 год.

Факторами, стимулирующими рост рынка компьютерного зрения, являются спрос на приложения для обработки изображений и видео из различных отраслевых сегментов, внедрение искусственного интеллекта в бизнес-сегменты и увеличение технологических разработок в области передовых систем безопасности. Факторами, препятствующими развитию рынка компьютерного зрения, являются высокие затраты и нехватка профессиональной рабочей силы.

На данный момент самой популярной эффективной технологией для компьютерного зрения являются алгоритмы машинного обучения Deep Learning. Именно этот подход и позволил обогнать человека в эффективности по распознаванию объектов на изображении.

Но проблема данного подхода в том, что он требуют много ресурсов. По этой причине его сложно применять на практике, либо внедрение подхода может обойтись в круглую сумму. И это касается не только работы самого алгоритма, но и обязательных подготовительных мероприятий, таких как обучение, которое может занимать кучу времени и ресурсов. А если брать в учёт ещё и сбор обучающей выборки, то счёт может пойти на месяца.

Производительность Deep Learning, как и любого другого алгоритма, напрямую зависит от платформы, на которой его запускают, и нас сколько эффективно эта платформа работает с вычислительными устройствами (CPU, GPU и др.). Под платформой понимается программа или интерпретатор (к примеру, python), или операционная систем. Ускорение любого из данных звеньев ускоряет и сам алгоритм. Улучшить производительность можно за счёт сокращения количества звеньев и запуска оптимизированного алгоритма непосредственно на вычислительном устройстве. Или можно использовать ресурсы вычислительного устройства эффективнее, чем это возможно сейчас.

Решая эту проблему компания Intel создало программное обеспечение OpenVINO Toolkit, позволяющее оптимизировать уже созданные ранее модели и алгоритмы и выполнять их на вычислительных устройствах компании Intel с гораздо большой эффективностью в сравнении со стандартным использованием. По заявлениям самой компании, модели будут не только эффективно выполняться на вычислительных устройствах, но и получат широкий набор функций для легкого масштабирования и интегрирования в существующие системы.

Целью данной работы является исследование предоставленного функционала и анализ эффективности оптимизации данного инструмента.

ГЛАВА 1. КОМПЬЮТЕРНОЕ ЗРЕНИЕ

Люди воспринимают трёхмерную структуру окружающего мира с очевидной лёгкостью. Стоит подумать, насколько ярким является трёхмерное восприятие, когда человек смотрит на вазу с цветами, стоящую рядом с ним. Можно определить форму и прозрачность каждого лепестка через тонкие узоры света и тени, которые играют на его поверхности. Психологи-перцептологи потратили десятилетия на то, чтобы понять, как работает система зрения, и, хотя они уже могут создавать оптические иллюзии, решение этой головоломки остаётся неуловимым.

Компьютерное зрение появилось в 1970-е годы. Тогда оно рассматривалось как компонент визуального восприятия амбициозной программы имитации человеческого интеллекта и наделения робота разумным поведением.

Исследователи компьютерного зрения со стремительным прогрессом за последние два десятилетия разрабатывают математические методы восстановления трёхмерной формы и внешнего вида объектов на изображениях.

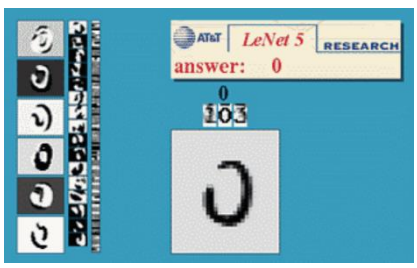
В компьютерном зрении учёные пытаются описать мир, который люди видят на одном или нескольких изображениях, и восстановить его свойства, такие как форма, освещение и цветовое распределение. Люди и животные делают это так легко, в то время как алгоритмы компьютерного зрения подвержены множеству ошибок.

В настоящее время компьютерное зрение используется в широком спектре реальных приложений, которые включают в себя:

- Оптическое распознавание символов (OCR): чтение рукописных почтовых индексов на письмах (Рисунок 1.а) и автоматическое распознавание номерных знаков (ANPR);
- Инспекция оборудования: быстрая проверка деталей для обеспечения качества с использованием стереовидения со спец подсветкой для

измерения допусков на крыльях самолётов или частях кузова автомобилей (рис. 1.б), поиск дефектов в стальных отливах с помощью рентгеновского зрения;

- Розничная торговля: распознавание объектов для автоматических касс и полностью автоматизированных магазинов;
- Складская логистика: автономная доставка посылок и сбор деталей роботами-манипуляторами (рис. 1.в);
- Медицинская визуализация: регистрация дооперационных и интраоперационных снимков (рис. 1.г);
- Беспилотные автомобили: способны перемещаться из пункта А в пункт Б (рис. 1.д), а также автономный полёт;
- Построение 3D-моделей (фотограмметрия): полностью автоматизированное построение 3D-моделей по аэрофотоснимкам и фотографиям с дронов (рис. 1.е);
- Match Move: объединение компьютерных изображений (CGI) с видеозаписями путём отслеживания характерных точек в исходном видео для оценки трёхмерного движения камеры и формы окружения. Такой метод широко используется в голливудских фильмах;
- Захват движения (motion capture): использование световозвращающих маркеров при просмотре с нескольких камер или других методов, основанных на зрении, для захвата актёров для дальнейшей компьютерной анимации;
- Наблюдение: наблюдение за нарушителями, анализ дорожного движения и контроль бассейнов для поиска жертв утопления;
- Распознавание отпечатков пальцев и биометрия: для автоматической аутентификации доступа, а также для криминалистических приложений.



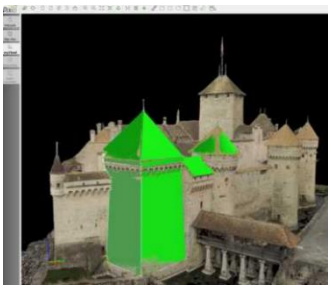
(а)

(б)

(в)

(г)

(д)



(е)

Рисунок 1 – Промышленное применение компьютерного зрения: (а) OCR, (б) механический осмотр, (в) сбор деталей роботом, (г) медицинское изображение, (д) беспилотные автомобили, (е) фотограмметрия с дронов.

1.1 Обработка изображения

Изображения формируются посредством воздействия элементов 3D-сцены, освещения, оптики и датчиков камеры. Но первый этап большинства алгоритмов компьютерного зрения – предварительная обработка изображения и преобразование его в форму, пригодную для дальнейшего анализа. Обработка включает в себя коррекцию экспозиции, балансировку цвета, уменьшение шумов, увеличение резкости или выпрямление изображения путем его поворота. Дополнительно может использоваться искажение изображения и смешивание изображений, часто используемые для визуальных эффектов.

Хотя некоторые могут считать, что обработка изображений не входит в компетенцию компьютерного зрения, большинство приложений компьютерного зрения, таких как вычислительная фотография и даже распознавание, требуют тщательной разработки этапов обработки

изображений для достижения приемлемых результатов.

В этой главе рассмотрены стандартные операторы обработки изображений, которые отображают значения пикселей из одного изображения в другое.

1.1.1 Пиксельные операторы

Самыми простыми видами преобразований обработки изображений являются пиксельные (или точечные) операторы, в которых каждое выходное значение пикселя зависит только от соответствующего значения входного пикселя. Примерами таких операторов являются регулировка яркости и контрастность, а также цветокоррекция и преобразования. В литературе по обработке изображений такие операции также известны как точечные процессы.

Пиксельные преобразования

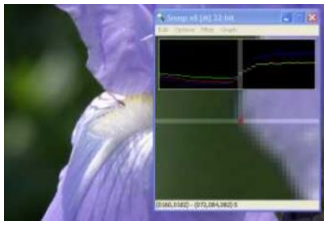
Общий оператор обработки изображений – это функция, которая принимает одно или несколько входных изображений и производит выходное изображение. В непрерывной области это можно обозначить как

$$g(x) = h(f(x)) \text{ или } g(x) = h(f_0(x), \ldots, f_n(x)),$$

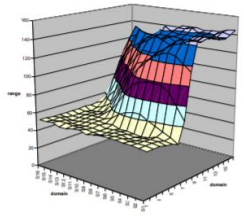
где x находится в D-мерной области функций (обычно $D=2$ для изображений), а функции f и g работают в некотором диапазоне, который может быть скалярным или векторным, например, для цветных изображений или 2D движения. Для дискретных (дискретизированных) изображений область состоит из конечного количество местоположений пикселей, $x = (i, j)$, и мы можем написать

$$g(i, j) = h(f(i, j)).$$

На рисунке 2 показано, как изображение может быть представлено либо по цвету (внешнему виду), либо в виде сетки чисел, или как двумерная функция (поверхностный график).

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120



(а) (б) (в) (г)

Рисунок 2 – Визуализация данных изображения: (а) исходное изображение; (б) обрезанный фрагмент и сканлайн-график с использованием инструмента для проверки изображений; (в) числовая сетка; (г) поверхностный участок.

Композитинг и матирование

Во многих приложениях для редактирования фотографий и визуальных эффектов часто требуется вырезать объект переднего плана из одной сцены и поместить его поверх другого фона (рис. 3).




(а) (б) (в) (г)

Рисунок 3 – Матирование и композитинг изображений: (а) исходное изображение; (б) извлеченный объект переднего плана F; (в) альфа-матирование α, показанное в градациях серого; (г) новый композит C.

Процесс извлечения объекта из исходного изображения часто называют матированием, а процесс его вставки в другое изображение (без видимых артефактов) – композитингом.

1.1.2 Линейная фильтрация

Локально адаптивное выравнивание гистограммы является примером оператора соседства или локального оператора, который использует набор значений пикселей в окрестностях данного пикселя для определения конечного выходного значения (рис. 6). Помимо выполнения локальной корректировки тона, операторы соседства могут использоваться для

фильтрации изображений с целью добавления мягкого размытия, повышения резкости деталей, подчеркивания краев или удаления шума.

Операторы линейной фильтрации включают фиксированные взвешенные комбинации пикселей в небольших окрестностях.

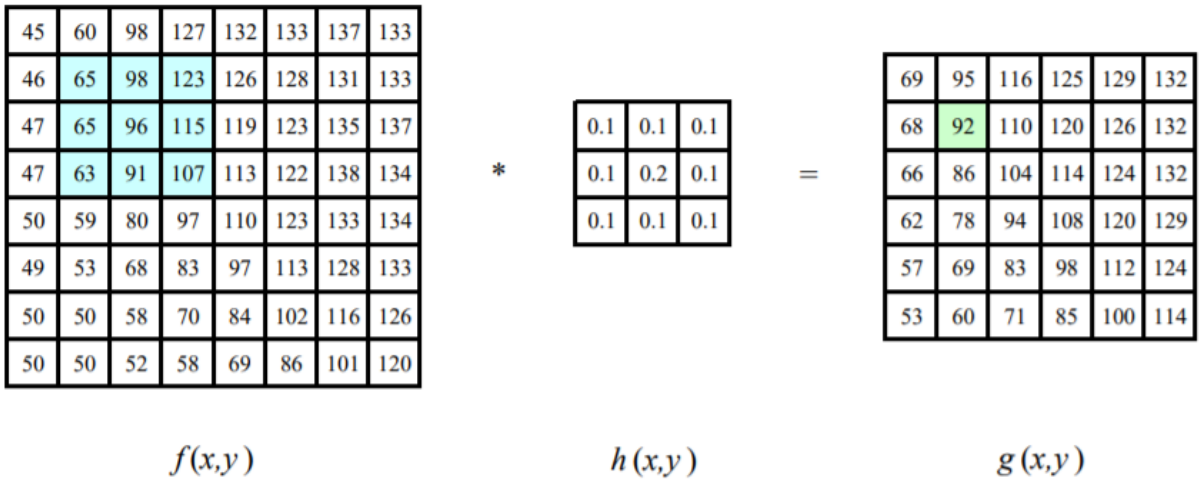


Рисунок 6 – Фильтрация соседства (свертка)

На рисунке изображение слева свернуто с фильтром посередине, чтобы получить изображение справа. Голубые пиксели указывают на источник соседства для светло-зеленого целевого пикселя (результат справа).

1.1.3 Дополнительные операторы соседства

Линейные фильтры могут выполнять самые разнообразные преобразования изображений. Однако нелинейные фильтры, такие как сохраняющие края медианные или билатеральные (двусторонние) фильтры, иногда могут работать лучше линейных. Другие примеры операторов соседства включают морфологические операторы, которые работают с бинарными изображениями, а также полуглобальные операторы, которые вычисляют преобразования расстояний и находят связанные компоненты в бинарных изображениях.

Нелинейная фильтрация

Все фильтры, которые были рассмотрены до сих пор, были линейными, т.е. их реакция на сумму двух сигналов совпадает с суммой отдельных откликов. Это эквивалентно тому, что каждый выходной пиксель является

взвешенным суммированием некоторого количества входных пикселей. Линейные фильтры легче составлять, и они поддаются анализу частотных характеристик.

Однако во многих случаях лучшие характеристики можно получить, используя нелинейную комбинацию соседних пикселей. Рассмотрим, например, изображение на рисунке 7 (д), где шум, вместо того чтобы быть гауссовским, является дробовым шумом (shot noise), т.е. иногда имеет очень большие значения. В этом случае обычное размытие с помощью гауссовского фильтра не удаляет зашумленные пиксели и вместо этого превращает их в более мягкие (но все еще видимые) пятна (рис. 7 е).

1.1.4 Преобразования Фурье

Преобразование Фурье может быть использовано для анализа частотных характеристик различных фильтров. В данной главе будет объяснено, как анализ Фурье позволяет определить эти характеристики (т.е. частотное содержание изображения) и как использование быстрого преобразования Фурье (БПФ) позволяет выполнять свертки больших ядер за время, не зависящее от размера ядра.

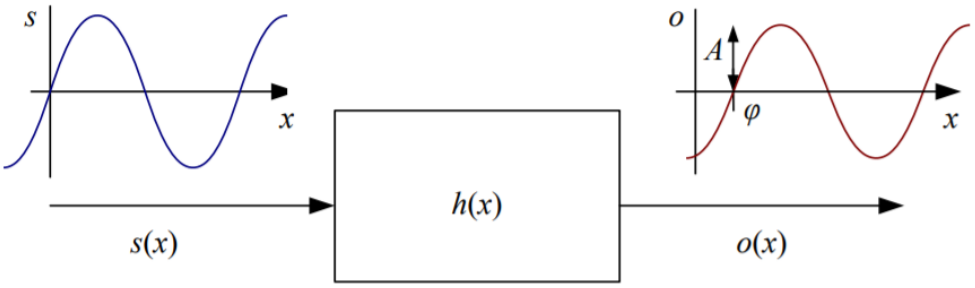


Рисунок 8 – Преобразование Фурье как реакция фильтра h(x) на входную синусоиду $s(x) = e^{j\omega x}$, и выходная синусоида $o(x) = h(x) * s(x) = Ae^{j\omega x+\phi}$.

1.1.5 Пирамида и вейвлет

До этого раздела все преобразования изображений давали на выходе изображения того же размера, что и на входе. Однако часто появляется

потребность в изменении разрешения изображения, прежде чем продолжить работу. Например, может понадобиться интерполировать маленькое изображение, чтобы его разрешение соответствовало разрешению выходного принтера или экрана компьютера. Или же нужно уменьшить размер изображения, чтобы ускорить выполнение алгоритма или сэкономить место в памяти или время передачи.

Иногда нет возможности узнать, каким должно быть разрешение изображения. быть. Рассмотрим, например, задачу поиска лица на изображении. Поскольку неизвестно, в каком масштабе появится лицо, нужно сгенерировать целую пирамиду изображений разного размера и просканировать каждое из них на предмет возможных лиц. Такая пирамида также может быть очень полезной в ускорении поиска объекта, сначала найдя меньший экземпляр этого объекта на более грубом уровне пирамиды, а затем искать объект с полным разрешением только в окрестностях грубого уровня. Наконец, пирамиды изображений чрезвычайно полезны для выполнения многомасштабных операций редактирования, таких как смешивание изображений с сохранением деталей.

Существуют фильтры для изменения разрешения изображения, т.е. апсамплинг (интерполяция) и даунсамплинг (децимация). Также существует концепция пирамид многократного разрешения, которые могут быть использованы для создания полной иерархии изображений разного размера и для различных приложений. С этим понятием тесно связана концепция вейвлетов, которые представляют собой особый вид пирамид с более высокой частотной избирательностью и другими полезными свойствами.

Пирамиды

Пирамиду (рисунок 9) можно использовать для ускорения алгоритмов грубого и тонкого поиска, для поиска объектов или деталей в разных масштабах, а также для выполнения операций смешивания в разных разрешениях. Пирамиды также широко используются в аппаратном и

программном обеспечении компьютерной графики для выполнения децимации дробного уровня с помощью MIP-карты.

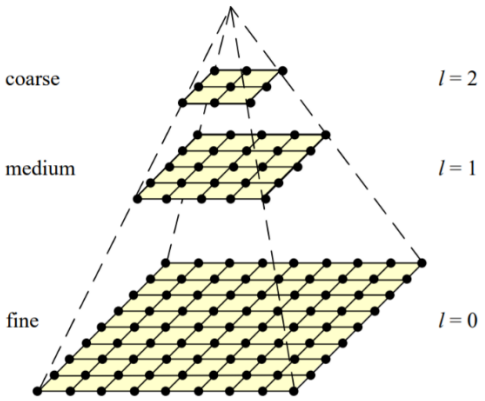


Рисунок 9 – Традиционная пирамида изображений.

На рисунке каждый уровень от грубого к наилучшему имеет половину разрешения (ширину и высоту) и, следовательно, четверть пикселей его родительского уровня.

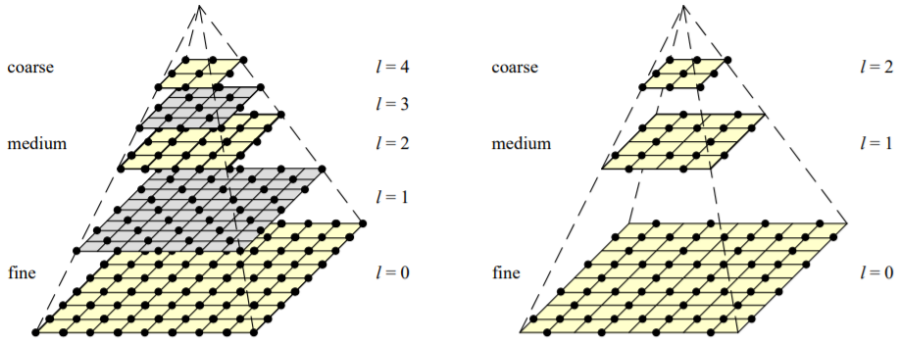
Наиболее известной (и наиболее широко используемой) пирамидой в компьютерном зрении является пирамида Лапласа. Чтобы построить такую пирамиду, мы сначала нужно размыть и продискретизировать исходное изображение в два раза и сохранить его на следующем уровне пирамиды (рисунок 10). Поскольку соседние уровни в пирамиде связаны между собой частотой дискретизации r = 2, такая пирамида называется октавной.

Вейвлеты

Хотя пирамиды широко используются в приложениях компьютерного зрения, некоторые люди используют вейвлет-разложение в качестве альтернативы. Вейвлеты – это фильтры, которые локализуют сигнал в пространстве и частоте и определяются по иерархии масштабов. Вейвлеты обеспечивают плавный способ разложения сигнала на частотные компоненты без блокировки и тесно связаны с пирамидами.

И пирамиды изображений, и вейвлеты раскладывают изображение на многоразрешающие описания, локализованные как в пространстве, так и в частоте. Но традиционные пирамиды являются чрезмерно полными, т.е. они используют больше пикселей, чем исходное изображение для представления

разложения, в то время как вейвлеты обеспечивают плотную рамку, т.е. они сохраняют размер разложения таким же, как у изображения (рис. 10 б). Однако некоторые семейства вейвлетов, по сути, являются переполненными, чтобы обеспечить лучшую сдвигаемость или управление ориентации. Поэтому более точным отличием может быть то, что вейвлеты более избирательны по ориентации, чем обычные полосовые пирамиды.



(а) (б)

Рисунок 10 – Пирамиды с несколькими разрешениями: (а) пирамида с полуоктавной дискретизацией. (б) пирамида вейвлетов - каждый уровень вейвлетов хранит 3/4исходных пикселей, так что общее количество вейвлет-коэффициентов и исходных пикселей одинаковое.

1.2 Глубокое обучение

Методы машинного обучения всегда играли важную, а зачастую и центральную роль в разработке алгоритмов компьютерного зрения. Компьютерное зрение в 1970-х годах выросло из областей искусственного интеллекта, цифровой обработки изображений и распознавания образов (сейчас это называется Машинное обучение).

Обработка изображений, интерполяция рассеянных данных, вариационная минимизация энергии и методы графических моделей являются важнейшими инструментами в компьютерном зрении на протяжении последних пяти десятилетий. Хотя элементы машинного обучения и распознавания образов также широко использовались, например, для точной настройки параметров алгоритмов, они по-настоящему проявили себя с

появлением крупномасштабных наборов данных маркированных изображений, таких как ImageNet, Microsoft COCO и LVIS. В настоящее время глубокие нейронные сети являются наиболее популярными и широко используемыми моделями машинного обучения в компьютерном зрении, не только для семантической классификации и сегментации, но даже для задач более низкого уровня, таких как улучшение изображений, оценка движения и восстановление глубины.

На рисунке 11 показаны основные различия между традиционными методами компьютерного зрения, в которых все этапы обработки разрабатывались вручную, алгоритмами машинного обучения, в которых созданные вручную характеристики передавались на этап машинного обучения, и глубокими сетями, в которых все компоненты алгоритма, включая представления среднего уровня, изучаются непосредственно на основе обучающих данных.

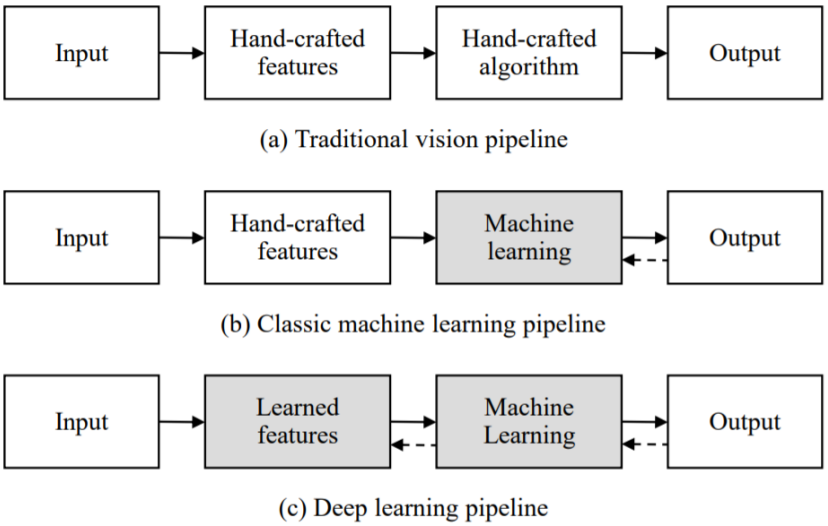


Рисунок 11 – Традиционные конвейеры, конвейеры машинного обучения и глубокого обучения.

В классическом конвейере, таком как структура из движения, и признаки, и алгоритм традиционно разрабатываются вручную (хотя методы обучения могут использоваться, например, для разработки более повторяемых признаков). Классические подходы машинного обучения берут извлеченные признаки и используют машинное обучение для построения классификатора.

Конвейеры глубокого обучения изучают весь конвейер, начиная с пикселей и заканчивая выходными данными, используя сквозное обучение (обозначено пунктирными стрелками назад) для точной настройки параметров модели.

Раздел начинается с обзора классических подходов машинного обучения, таких как ближайшие соседи, логистическая регрессия, машины опорных векторов и леса решений.

Далее рассказывается о глубоких нейронных сетях, которые за последнее десятилетие стали методом выбора для большинства задач распознавания компьютерного зрения и задач зрения нижнего уровня. Принято начинать с элементов, из которых состоят глубокие нейронные сети, включая веса и активации, условия регуляризации и обучение с помощью обратного распространения и стохастического градиентного спуска. Далее будет рассказано про свёрточные слои, рассмотрены некоторые классические архитектуры, как проводить предварительное обучение сетей и визуализировать их работу.

Поскольку машинное обучение и глубокое обучение - такие богатые и глубокие темы, в этой главе лишь кратко изложены некоторые из основных концепций и методов.

1.2.1 Обучение с учителем

Алгоритмы машинного обучения обычно делятся на контролируемые, когда алгоритму обучения предоставляются парные входы и выходы (рисунок 12), и неконтролируемые, когда статистические образцы предоставляются без соответствующих маркированных выходов.

Как показано на рисунке 12, контролируемое обучение включает в себя подачу пар входных данных $\{x_i\}$ и соответствующие им целевые выходные значения $\{t_i\}$ в алгоритм обучения, который настраивает параметры модели таким образом, чтобы максимизировать согласие между предсказаниями модели и целевыми выходами. Во время выполнения параметры модели замораживаются, и модель применяется к новым входам для получения

желаемых выходов. Выходы могут быть либо дискретными метками, которые поступают из набора классов $\{C_k\}$, либо это может быть набор непрерывных, потенциально векторнозначных значений, которые мы обозначим через y_i чтобы сделать различие между этими двумя случаями более четким. Первая задача называется классификацией, поскольку мы пытаемся предсказать принадлежность к классу, в то время как вторая задача называется регрессией, поскольку исторически подгонка тенденции к данным называлась именно так.

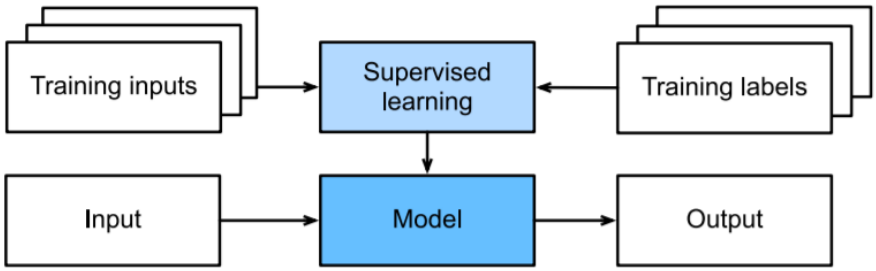


Рисунок 12 – Контролируемое обучение.

После этапа обучения, во время которого все обучающие данные (помеченные пары вход-выход) были обработаны (часто путем многократных итераций), обученная модель может быть использована для предсказания новых выходных значений для ранее невидимых входов. Этот этап часто называют этапом тестирования, хотя это иногда вводит людей в заблуждение, заставляя их уделять чрезмерное внимание производительности на заданном тестовом наборе вместо того, чтобы создавать систему, которая будет устойчиво работать для любых правдоподобных входных данных, которые могут возникнуть.

1.2.2 Обучение без учителя

Выше был представлен метод обучения с учителем, когда нам предоставляются обучающие данные, состоящие из парных входных и целевых примеров. Однако в некоторых приложениях дается только набор данных, которые нужно охарактеризовать, например, посмотреть, есть ли в них есть ли какие-либо закономерности или типичные распределения. Это, как правило, относится к области классической статистики. В сообществе машинного обучения этот сценарий обычно называют несамостоятельным,

так как данные выборки поступают без меток. Примеры применения в компьютерном зрении включают сегментацию изображений, распознавание и реконструкцию лиц и тел.

Из наиболее широко используемых методов компьютерного зрения можно выделить кластеризацию, смешанное моделирование и анализ главных компонент (для моделирования внешности и формы).

1.2.3 Глубокие нейронные сети

Конвейеры глубокого обучения используют сквозной подход к машинному обучению, оптимизируя каждый этап обработки путем поиска параметров, которые минимизируют потери при обучении. Для того чтобы такой поиск был осуществим, необходимо, чтобы потери были дифференцируемой функцией всех этих параметров. Глубокие нейронные сети обеспечивают единую, дифференцируемую архитектуру вычислений, одновременно автоматически обнаруживая полезные внутренние представления.

Наиболее популярные на сегодня глубокие нейронные сети – это детерминированные дискриминационные фидфорвардные сети с реальными значениями активаций, обучаемые с помощью градиентного спуска, т.е. правила обучения обратного распространения. В сочетании с идеями конволюционных сетей, глубокие многослойные нейронные сети привели к прорыву в распознавании речи и визуальном распознавании, который наблюдался в начале 2010-х годов.

По сравнению с другими методами машинного обучения, которые обычно опираются на несколько этапов предварительной обработки для извлечения признаков, на основе которых можно построить классификаторы, подходы глубокого обучения обычно обучаются из конца в конец, переходя непосредственно от необработанных пикселей к конечным желаемым результатам (будь то классификация или другие изображения).

Веса и слои

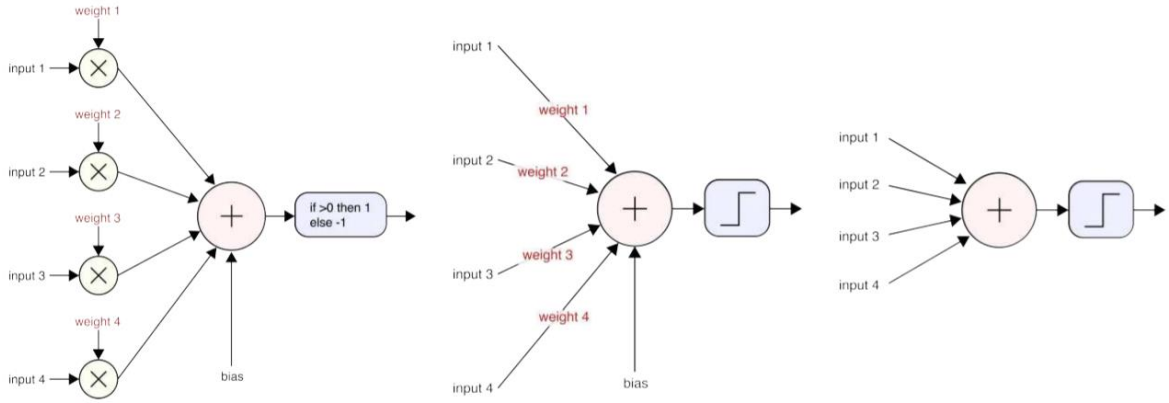
Глубокие нейронные сети (DNNs) – это вычислительные графы с прямой связью, состоящие из тысяч простых взаимосвязанных "нейронов" (блоков), которые, подобно логистической регрессии, выполняют взвешенные суммы своих входов

$$s_i = \mathbf{w}_i^T \mathbf{x}_i + b_i,$$

с последующим преобразованием нелинейной функции активации

$$y_i = h(s_i),$$

как показано на рисунке 13, x_i - это входы для i-го блока, w_i и b_i – его обучаемые веса и смещения, s_i – это результат взвешенной линейной суммы, а y_i – конечный результат, после s_i подаётся через функцию активации h. Выходы каждого этапа, которые часто называются активациями, подаются в блоки на более поздних стадиях (рис. 14).



(а) (б) (в)

Рисунок 13 – Блок персептрона.

На рисунке 13 показан блок персептрона (а) с явным отображением весов, умноженных на входы, (б) с весами, записанными на входных связях, и (в) наиболее распространенная форма, в которой веса и смещение опущены. Нелинейная функция активации следует за взвешенным суммированием. Стоит обратить внимание, что на первой диаграмме веса, которые оптимизируются на этапе обучения, показаны в явном виде вместе с поэлементным умножением. На рисунке 13 (б) показана форма, в которой веса записаны поверх связей (стрелки между блоками, хотя наконечники стрелок

часто опускаются. Еще более распространена схема сетей, как на рисунке 13 (в), в которой веса (и смещения) полностью опущены и предполагается, что они присутствуют.

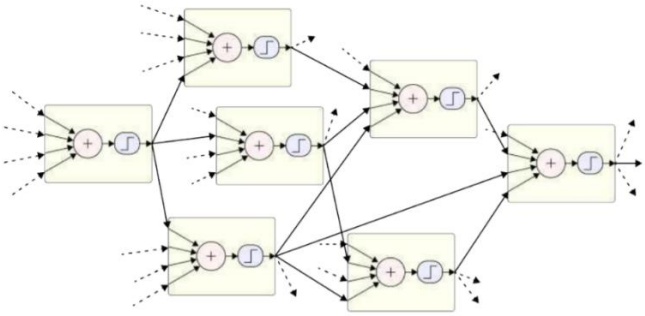


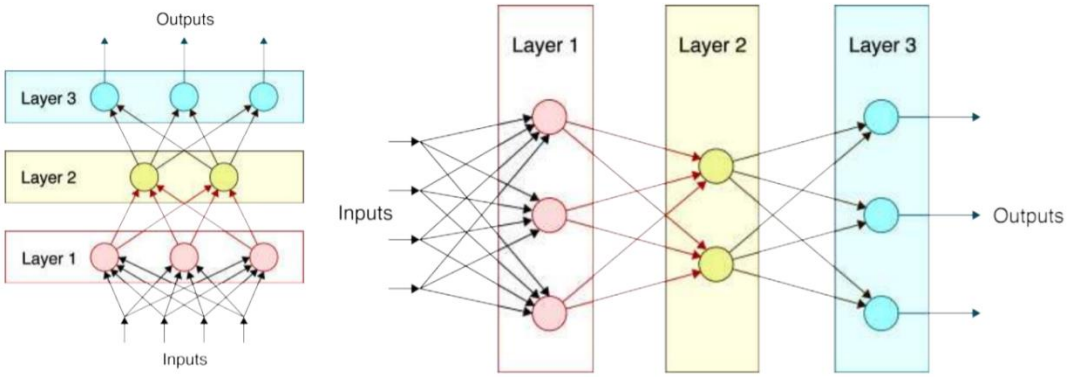
Рисунок 14 – Многослойная сеть, показывающая, как выходы одного блока подаются на дополнительные блоки.

Вместо того, чтобы всё соединять в нерегулярный вычислительный граф, как на рисунке 14, нейронные сети обычно организовывают в последовательные слои, как показано на рисунке 15. Теперь можно представить все единицы внутри слоя как вектор, а соответствующие линейные комбинации записаны как

$$\mathbf{s}_l = \mathbf{W}_l \mathbf{x}_l,$$

где x_l - входы в слой l, W_l - весовая матрица, а s_l - взвешенная сумма, к которой применяется поэлементная нелинейность с использованием набора функций активации,

$$\mathbf{x}_{l+1} = \mathbf{y}_l = \mathbf{h}(\mathbf{s}_l).$$



(а) (б)

Рисунок 15 – Два разных способа рисования нейронных сетей: (а) входы внизу, выходы вверху, (б) входы слева, выходы справа.

Слой, в котором для линейной комбинации используется полная (плотная) весовая матрица, называется полносвязным слоем, поскольку все входы одного слоя подключены ко всем его выходами. При обработке пикселей (или других сигналов), на ранних этапах обработки используются свёртки вместо плотных соединений, что обеспечивает пространственную инвариантность и большую эффективность. Сеть, состоящая только из полносвязных (и безсвёрточных) слоёв, теперь часто называется многослойным персептроном (MLP).

Функции активации

Большинство ранних нейронных сетей использовали сигмоидальные функции, похожие на те, что используются в логистической регрессии. Более новые сети, начиная с 2010 года, используют выпрямленную линейную функцию активации (ReLU) или ее варианты. Функция активации ReLU определяется как

$$h(y) = \max(0, y)$$

и показана на рисунке 16 наряду с другими популярными функциями. ReLU в настоящее время является самой популярной функцией активации.

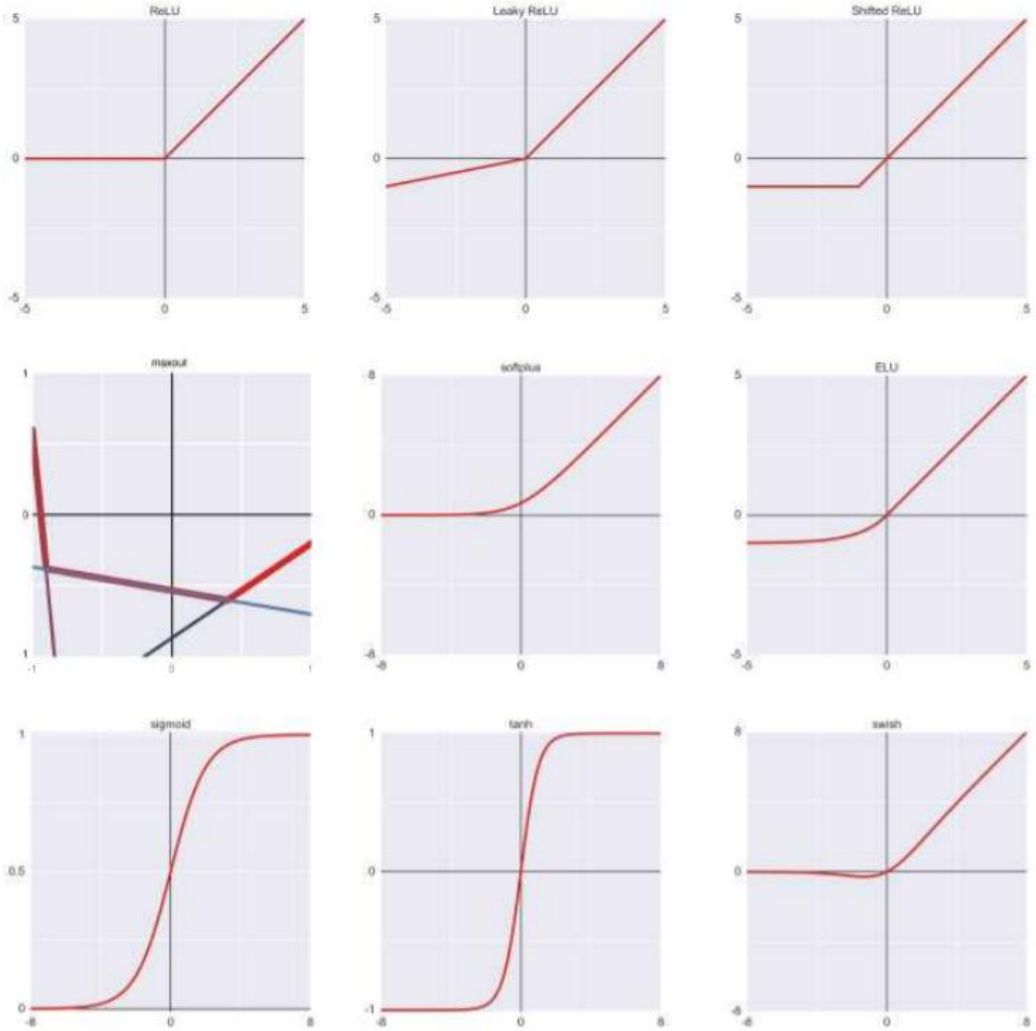


Рисунок 16 – Нелинейные функции активации: слева направо ReLU, leaky ReLU, shifted ReLU, maxout, softplus, ELU, sigmoid, tanh, swish.

Регуляризация и нормализация

Как и в других формах машинного обучения, регуляризация и другие методы могут быть использованы для предотвращения чрезмерной подгонки нейронных сетей, чтобы лучше обобщать невидимые данные. Существуют традиционные методы, такие как регуляризация и увеличение данных, которые можно применять к большинству систем машинного обучения, а также такие методы, как отсев и пакетная нормализация, которые специфичны для нейронных сетей.

Регуляризация и снижение весов

Квадратичные или p-нормальные штрафы за ошибку на весах могут быть использованы для улучшения обусловленности системы и уменьшения перебора. Задание $p = 2$ приводит к обычной регуляризации *L2* и делает большие веса меньше, в то время как использование $p = 1$ называется лассо (оператор наименьшего абсолютного сокращения и выбора) и может привести к тому, что некоторые веса будут полностью равны нулю. Поскольку веса оптимизируются внутри нейронной сети, эти условия делают веса меньше, поэтому такой вид регуляризации также известен как уменьшение веса.

Заметим, что для более сложных алгоритмов оптимизации, таких как Adam, *L2* регуляризация и весовой распад не эквивалентны, но желательные свойства весового распада могут быть восстановлены с помощью модифицированного алгоритма.

Увеличение набора данных

Еще один мощный метод уменьшения чрезмерной подгонки – это добавление дополнительных обучающих выборок путем пертурбации входных и/или выходных данных уже собранных образцов. Эта техника называется увеличением набора данных и может быть чрезвычайно эффективной для задач классификации изображений, поскольку получение маркированных примеров требует больших затрат, а также поскольку классы изображений не должны меняться при небольших локальных возмущениях.

Ранним примером такой работы, примененной к задаче нейросетевой классификации, является метод упругого искажения. В этом подходе случайные низкочастотные поля смещения (искривления) синтетически генерируются для каждого учебного примера и применяются к входам во время обучения. Искажения перемещают пиксели, и поэтому вносят гораздо большие изменения в пространство входных векторов, сохраняя при этом семантическое значение примеров.

Отсев нейронов или связей между ними

Отсев – это метод регуляризации, при котором в каждой мини-партии во время обучения некоторый процент p (скажем, 50%) единиц в каждом слое зажимается до нуля. Случайная установка единиц на ноль вносит шум в процесс обучения, а также не позволяет сети чрезмерно специализировать единицы к определенным образцам или задачам, что может помочь уменьшить переподгонку и улучшить обобщение.

Поскольку отбрасывание (обнуление) p единиц уменьшает ожидаемое значение любой суммы на долю *(1 - p),* взвешенные суммы s_i в каждом слое умножаются (во время обучения) на *(1 - p) - 1*. Во время тестирования сеть запускается без отсева и без компенсации сумм.

Пакетная нормализация

Оптимизация весов в глубокой нейронной сети – сложный процесс, который может медленно сходиться или застревать в локальных минимумах. Одной из классических проблем итеративных методов оптимизации является плохая обусловленность, когда компоненты градиента сильно различаются по величине. Хотя иногда можно уменьшить этот эффект с помощью методов предварительного кондиционирования, которые масштабируют отдельные элементы в градиенте перед выполнением шага, обычно предпочтительнее контролировать число условий системы во время постановки задачи.

В глубоких сетях плохое кондиционирование может проявиться в том случае, если размеры весов или активаций в последовательных слоях становятся несбалансированными. Например, если взять данную сеть и

масштабировать все веса в одном слое на $100\times$ и уменьшить веса в следующем слое на ту же величину. Поскольку функция активации ReLU является линейной в обеих своих областях, выходы второго слоя будут одинаковыми, хотя активации на выходе первых слоев будут в 100 раз больше. Во время шага градиентного спуска производные по отношению к весам будут значительно отличаться после этого изменения масштаба, и фактически будут противоположны по величине самим весам, что потребует крошечных шагов градиентного спуска чтобы предотвратить перебор.

Функция потерь

Для того чтобы оптимизировать веса в нейронной сети, необходимо сначала определить функцию потерь, которую минимизируют на обучающих примерах.

Для классификации большинство нейронных сетей используют конечный слой softmax. Поскольку выходы должны быть вероятностями классов, которые в сумме равны 1, естественно использовать потери от перекрестной энтропии в качестве функции, которую нужно минимизировать во время обучения.

Таким образом, кросс-энтропийную потерю нескольких классов можно переписать так:

$$E(\mathbf{w}) = \sum_n E_n(\mathbf{w}) = -\sum_n \log p_{nt_n},$$

где w – вектор всех весов, смещений и других параметров модели, а p_{nk} – текущая оценка сети вероятности класса k для выборки n, а t_n - целое число, обозначающее правильный класс.

Инициализация весов

Ранние нейронные сети использовали небольшие случайные веса, чтобы нарушить симметрию, т.е. чтобы убедиться, что все градиенты не были равны нулю. Однако в более глубоких слоях активации становились все меньше.

Для того чтобы сохранить сопоставимую дисперсию активаций последовательных слоев, нужно учитывать веерность каждого слоя, т.е.

количество входящих связей, где активации умножаются на веса. Также нужно установить дисперсию случайного начального веса как обратную дисперсию веера.

Если инициализировать веса с нулевым средним и дисперсией V_l для слоя l и установить исходные базисы равными нулю, линейное слагаемое будет иметь дисперсию

$$Var[s_l] = n_l V_l E[x_l^2],$$

где n_l – количество входящих активаций/весов, а $E[x_l^2]$ – ожидание квадрата входящих активаций. Когда слагаемые s_l, имеющие нулевое среднее, проходят через ReLU, отрицательные слагаемые зажимаются до нуля, поэтому ожидание квадратичного выхода $E[y_l^2]$ равно половине дисперсии s_l, $Var[s_l]$.

Обратное распространение

После того, как нейронная сеть настроена, определено количество слоев, их ширина и глубина, добавлены некоторые регуляризационные условия, определена функция потерь и инициализированы веса, всё готово к обучению сети на выборочных данных. Для этого нужно использовать градиентный спуск или один из его вариантов для итеративного изменения весов, пока сеть не сойдется с хорошим набором значений, т.е. с приемлемым уровнем производительности на обучающих и тестовых данных.

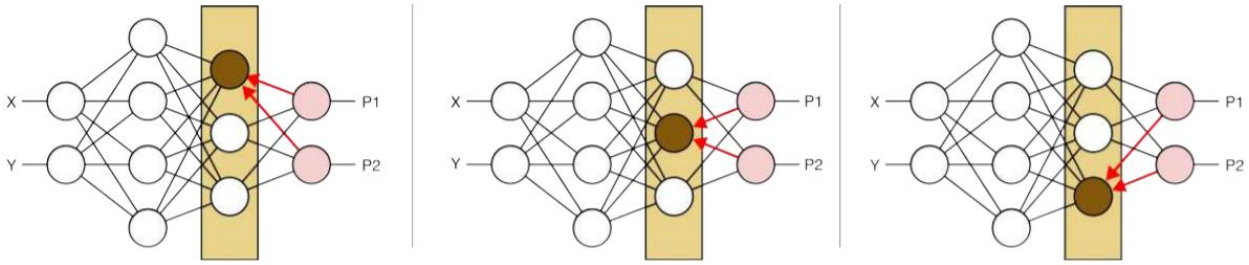


Рисунок 17 – Обратное распространение производных (ошибок) через промежуточный слой глубокой сети.

Обучение и оптимизация

На данный момент определена топология сети с точки зрения размеров и глубины каждого слоя, заданы функции активации, добавлены условия регуляризации, заданы функции потерь и инициализированы веса. Сейчас

нужен алгоритм, который превратит градиенты в обновление весов, что позволит оптимизировать функцию потерь и создать сеть, которая хорошо обобщает новые, невидимые данные.

В большинстве алгоритмов компьютерного зрения, таких как оптический поток, трехмерная реконструкция с использованием настройки пучков, и даже в менее масштабных задачах машинного обучения, таких как логистическая регрессия, методом выбора является линеаризованный метод наименьших квадратов. Оптимизация выполняется с помощью метода второго порядка, такого как метод Гаусса-Ньютона, в котором оцениваем все члены нашей функции потерь, а затем берем оптимального размера шаг вниз, используя направление, полученное из градиентов и гессиана функции энергии.

К сожалению, задачи глубокого обучения слишком велики (с точки зрения количества параметров и обучающих выборок), чтобы сделать этот подход практичным. Вместо этого специалисты разработали ряд алгоритмов оптимизации, основанных на расширениях SGD (стохастического градиентного спуска). В SGD вместо того, чтобы оценивать функцию потерь путем суммирования по всем обучающим выборкам, просто оцениваем одну обучающую выборку n и вычисляем производные соответствующей потери $E_n(w)$. Затем делаем маленький шаг вниз по направлению этого градиента

На практике направления, полученные только из одной выборки, являются невероятно шумными оценками хорошего направления спуска, поэтому потери и градиенты обычно суммируются по небольшому подмножеству обучающих данных,

$$E_{\mathcal{B}}(\mathbf{w}) = \sum_{n \in \mathcal{B}} E_n(\mathbf{w}),$$

где каждое подмножество B называется минипакетом (минибатчем).

Перед началом обучения случайным образом распределяем обучающие выборки по фиксированному набору минибатчей, каждый из которых имеет фиксированный размер, обычно варьирующийся от 32 в нижнем пределе до 8k

в верхнем. Полученный алгоритм называется стохастическим градиентным спуском по минибатчам, хотя на практике большинство людей называют его просто SGD (опуская упоминание о минибатчах).

1.2.4 Свёрточные нейронные сети

В предыдущих разделах, посвященных глубокому обучению, были рассмотрены все основные элементы построения и обучения глубоких сетей. Однако было упущено, что, вероятно, является наиболее важным компонентом глубоких сетей для обработки изображений и компьютерного зрения – использование обучаемых многослойных свёрток. Идея свёрточных нейронных сетей была популярна в 1998, где была представлена сеть LeNet-5 для распознавания цифр, показанная на рисунке 18. Эта сеть использует несколько каналов на каждом уровне и чередует многоканальные свёртки с операциями понижающей дискретизации, за которыми следуют некоторые полносвязные слои, которые производят одну активацию для каждой из 10 классифицируемых цифр.

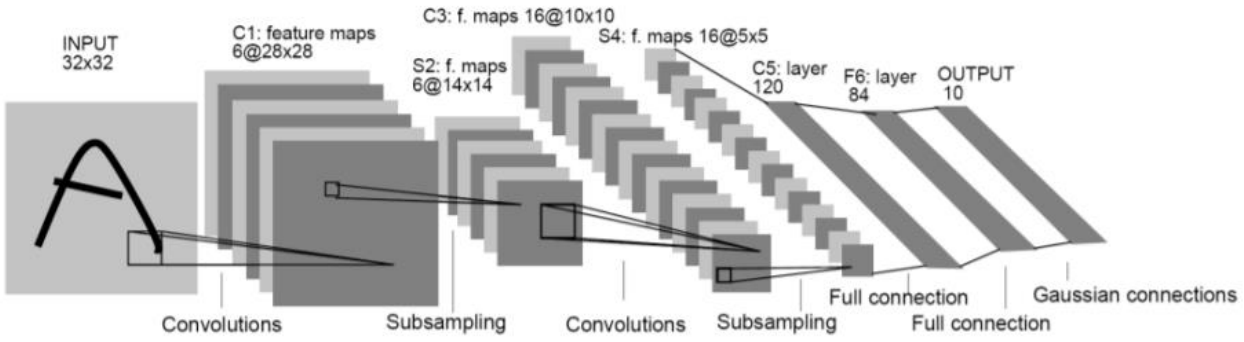


Рисунок 18 – Архитектура LeNet-5, свёрточной нейронной сети для распознавания цифр.

Вместо того чтобы соединять все единицы в слое со всеми единицами в предыдущем слое, свёрточные сети организуют каждый слой в карты признаков, которые можно представить как параллельные плоскости или каналы. В слое свертки взвешенные суммы выполняются только в пределах небольшого локального окна, а веса идентичны для всех пикселей, как и при обычной свёртке и корреляции изображений со сдвигом.

Однако в отличие от свёртки изображений, где один и тот же фильтр применяется к каждому каналу, свёртки нейронных сетей обычно линейно объединяют активации от каждого из входных каналов C_1 в предыдущем слое и используют различные ядра свертки для каждого канала. C_1 входных каналов в предыдущем слое и используют различные ядра свертки для каждого из выходных каналов C_2. Это имеет смысл, так как основная задача в слоях свёрточной нейронной сети является построение локальных признаков, а затем комбинирование их различными способами для получения более дискриминационных и семантически значимых признаков.

Исходя из этих соображений, можно записать взвешенные линейные суммы, выполняемые в свёрточном слое, как

$$s(i, j, c_2) = \sum_{c_1 \in \{C_1\}} \sum_{(k,l) \in \mathcal{N}} w(k, l, c_1, c_2) x(i + k, j + l, c_1) + b(c_2),$$

где $x(i, j, c_1)$ – это активации в предыдущем слое, N это S_2 знаковые смещения в двумерном пространственном ядре, и обозначение $c_1 \in \{C_1\}$ обозначает $c_1 \in [0, C_1)$. Поскольку смещения (k, l) прибавляются, а не вычитаются из пиксельных координат (i, j), эта операция фактически является корреляцией, но это различие обычно не замечается.

1.3 Распознавание объектов

Визуальное распознавание претерпело наибольшие изменения и самое быстрое развитие за последнее десятилетие, что отчасти объясняется наличием гораздо большего количества помеченных данных, а также прорывов в глубоком обучении.

Общий подход к поиску отличительных признаков с учетом локальной вариации внешнего вида, а затем проверка их совпадения и относительного расположения на изображении, по-прежнему широко используется для обнаружения трехмерных объектов, восстановления трехмерной структуры и распознавания местоположения.

Распознавание объектов делится на две большие категории, а именно – распознавание экземпляров и распознавание классов. Первое включает в себя повторное распознавание известного двумерного или трёхмерного жесткого объекта, потенциально рассматриваемого с новой точки зрения, на загроможденном фоне и с частичными окклюзиями. Второе распознавание также известно, как распознавание на уровне категорий или общих объектов, представляет собой гораздо более сложную задачу распознавания любого экземпляра определенного общего класса, такого как «кошка», «автомобиль», «велосипед» и т.д.

За прошедшие годы было разработано множество различных алгоритмов для распознавания экземпляров. Ранние подходы были направлены на извлечение линий, контуров или 3D-поверхностей из изображений и сопоставлении их с известными 3D-моделями объектов. Другим популярным подходом было получение изображений с большого набора точек обзора и освещения и представление их с помощью разложения в электронное пространство. Более современные подходы, как правило, используют инвариантные к точке зрения двумерные признаки. После извлечения информативных разреженных двумерных признаков как из нового изображения, так и из изображений в базе данных, признаки изображения сопоставляются с базой данных объектов, используя одну из стратегий сопоставления разреженных признаков.

Геометрическое выравнивание

Для распознавания одного или нескольких экземпляров некоторых известных объектов, система распознавания сначала извлекает набор точек интереса в каждом изображении базы данных и сохраняет связанные с ними дескрипторы (и исходные позиции) в индексирующей структуре, такой как дерево поиска. Во время распознавания признаки извлекаются из нового изображения и сравниваются с сохраненными характеристиками объекта. При достижении достаточного количества совпадающих признаков (трёх или более) для данного объекта, система запускает этап проверки соответствия,

задача которого – определить, соответствует ли пространственное расположение совпадающих признаков с признаками на изображении в базе данных.

1.3.1 Классификация изображений

В то время как методы распознавания экземпляров уже относительно развиты и используются в коммерческих приложениях, таких как распознавание дорожных знаков, распознавание общих категорий (классов) все еще является быстро развивающейся областью исследований. Например, набор фотографий на рисунке 19, на котором изображены объекты, взятые из 10 различных визуальных категорий. Как можно написать программу для отнесения каждого из этих изображений к соответствующему классу, особенно если также был предоставлен выбор "ничего из вышеперечисленного"?

Как видно из этого примера, распознавание визуальных категорий – чрезвычайно сложная задача. Однако прогресс в этой области был весьма значительным, если судить по тому, насколько лучше современные алгоритмы по сравнению с алгоритмами десятилетней давности.

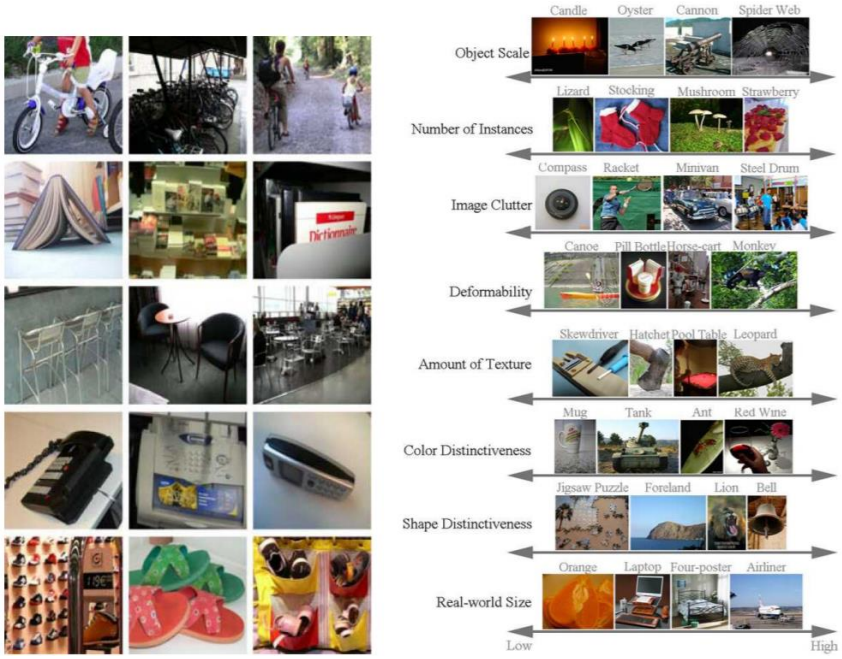


Рисунок 19 – Проблемы распознавания изображений: слева – образцы изображений, справа – оси сложности и вариаций.

Существуют основные классы алгоритмов, используемых для классификации цельных изображений, и классические подходы, основанные на признаках, которые полагаются на созданные вручную признаки и их статистике, а также на машинном обучении для окончательной классификации. Также существует поиск визуального сходства, где задача состоит в том, чтобы найти визуально и семантически похожие изображения, а не классифицировать их по фиксированному набору категорий.

Глубокие нейронные сети

Современные системы классификации изображений основаны на глубоких нейронных сетях. На сегодня точность распознавания значительно возросла, что в значительной степени было обусловлено более глубокими сетями и лучшими алгоритмами обучения. В последнее время в центре внимания исследователей оказались более эффективные сети, а также большие (немаркированные) наборы данных для обучения.

Поиск визуального сходства

Автоматическая классификация изображений по категориям и маркировка их атрибутами с помощью алгоритмов компьютерного зрения облегчает их поиск в каталогах и в Интернете. Она обычно используется в системах поиска изображений, которые находят подходящие изображения по ключевым словам. по ключевым словам, так же как обычные поисковые системы в Интернете находят нужные документы и страницы.

Однако иногда проще найти нужную информацию непосредственно по изображению, что называется визуальным поиском. В качестве примера можно привести поиск экземпляров, т.е. поиск точно такого же объекта или местоположения. Другой вариант - поиск визуально похожих изображений (часто называемый поиском визуального сходства или обратным поиском изображений), который полезен, когда цель поиска не может быть кратко выражена в словах.

Распознавание лиц

Распознавание лиц используется в различных дополнительных приложениях, включая взаимодействие человека и компьютера, подтверждение личности, вход в систему, родительский контроль и мониторинг пациентов. Распознаватели работают лучше всего, когда им предоставляются изображения лиц в разных положениях, освещении и т.д.

Некоторые из самых ранних подходов к распознаванию лиц заключались в определении местоположения отличительных черт изображения, таких как глаза, нос и рот, и измерение расстояний между этими чертами. Другие подходы основаны на сравнении изображений серого уровня, спроецированных на подпространства меньшей размерности, называемые собственными лицами, и совместном моделировании вариаций формы и внешнего вида (при этом не учитываются вариации поз) с помощью активных моделей внешнего вида.

ГЛАВА 2. INTEL OPENVINO TOOLKIT

OpenVINO toolkit – это комплексный инструментарий для быстрой разработки приложений и решений. Решает такие задачи как эмуляция человеческого зрения, автоматическое распознавание речи, обработка естественного языка, рекомендательные системы и многие другие. Инструментарий основан на последних поколениях искусственных нейронных сетей, включая сверточные нейронные сети (CNNs). Он расширяет рабочие нагрузки компьютерного зрения и схожих задач на аппаратном обеспечении Intel, и при этом максимизирует производительность. Также он ускоряет работу приложений с искусственным интеллектом и глубоким обучением, развернутых как на локальной машине, так и в облаке.

Инструмент имеет следующие преимущества:

1. Позволяет работать с моделями CNN с глубоким обучением.
2. Поддерживает гетерогенное выполнение через процессоры Intel, графические ускорители Intel, Intel Neural Compute Stick 2 и Intel Vision Accelerator Design с Vpus Intel Movidius.
3. Ускоряет время выхода на рынок с помощью простой в использовании библиотеки функций компьютерного зрения и предварительно оптимизированных ядер.
4. Ускоряет время выхода на рынок с помощью простой в использовании библиотеки функций компьютерного зрения и предварительно оптимизированных ядер

2.1 Рабочий процесс

В приложении 1 показан типичный рабочий процесс OpenVINO.

Предполагается, что нулевой и первый этапы выполнены и есть готовая обученная модель, с которой уже можно работать.

Сам процесс взаимодействия с OpenVINO начинается на втором этапе. На этом этапе основным инструментом является оптимизатор модели. Оптимизатор импортирует модели, обученные в популярных фреймворках,

таких как Caffe, TensorFlow, MXNet, Kaldi и ONNX, и выполняет несколько оптимизаций для удаления лишних слоёв и групповых операций, когда это возможно, в более простые и быстрые графы. Полученные графы называются Intermediate representation (IR).

Следующие два этапа выполняются на стороне конечного приложения и являются интеграцией IR-моделей в само приложение, а также запуск модели на конечных устройствах. За это отвечает компонент механизма вывода (Inference Engine).

Механизм вывода управляет загрузкой и компиляцией оптимизированной модели нейронной сети, выполняет запуск моделей с входными данными и выводит результаты. Механизм вывода может выполняться синхронно или асинхронно, а его архитектура плагинов управляет соответствующими компиляциями для выполнения на нескольких устройствах Intel, включая как процессоры workhorse, так и специализированные платформы обработки графики и видео.

2.2 Model Optimizer

Model Optimizer – это кроссплатформенный инструмент командной строки, который облегчает переход между средой обучения и развертывания, выполняет статический анализ моделей и настраивает модели глубокого обучения для оптимального выполнения на конечных целевых устройствах.

Процесс оптимизации модели предполагает, что у вас есть сетевая модель, обученная с использованием поддерживаемой платформы глубокого обучения. Приведенная ниже схема иллюстрирует типичный рабочий процесс для развертывания обученной модели глубокого обучения:

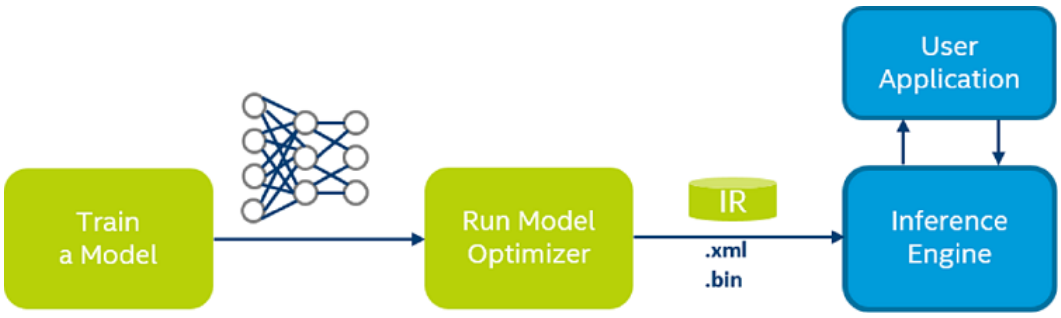


Рисунок 20 – Схема типичного рабочего процесса

Процесс оптимизации модели предполагает, что у есть сетевая модель, обученная с использованием поддерживаемой платформы глубокого обучения. Приведенная схема иллюстрирует типичный рабочий процесс для развертывания обученной модели глубокого обучения:

- .xml – описывает топологию сети
- .bin – содержит двоичные данные с весами и смещениями.

Механизм вывода позволяет развернуть сетевую модель, обученную с помощью любой из поддерживаемых платформ глубокого обучения: Caffe, TensorFlow, Kaldi, MXNet или преобразованную в формат ONNX. Механизм вывода работает не с исходной моделью, а с ее промежуточным представлением (IR), которое оптимизировано для выполнения на конечных целевых устройствах. Для генерации IR-сигнала для обученной модели используется инструмент Model Optimizer.

Оптимизатор модели загружает модель в память, считывает ее, строит внутреннее представление модели, оптимизирует его и создает промежуточное представление. Промежуточное представление – это единственный формат, который принимает механизм вывода.

Оптимизатор модели имеет две основные цели:

- Создать допустимое промежуточное представление. Если этот основной артефакт преобразования не является допустимым, механизм вывода не сможет его преобразовать. Основная ответственность оптимизатора модели заключается в создании двух файлов (.xml и .bin), которые образуют промежуточное представление.

- Создать оптимизированное промежуточное представление. Предварительно обученные модели содержат слои, которые важны для обучения, например, слой Dropout. Эти слои бесполезны во время вывода и могут увеличить время работы. Во многих случаях эти операции могут быть автоматически удалены из результирующего промежуточного представления. Или если группа операций может быть представлена как одна математическая операция и, следовательно, как один операционный узел в графе модели, Оптимизатор модели распознаёт такие шаблоны и заменяет группу операционных узлов только одной операцией. В результате получается Промежуточное представление, имеющее меньше операционных узлов, чем исходная модель. Это уменьшает время вывода.

Чтобы создать достоверное Промежуточное представление, Оптимизатор модели должен уметь считывать исходные операции модели, обрабатывать их свойства и представлять их в формате Промежуточного представления, сохраняя при этом достоверность результирующего Промежуточного представления. Полученная модель состоит из различных операций, полностью описанных в «Спецификации операций» [https://docs.openvinotoolkit.org/latest/openvino_docs_ops_opset.html].

Многие общие слои существуют в известных фреймворках и топологиях нейронных сетей. Примерами таких слоёв являются Convolution, Pooling, и Activation. Чтобы считать исходную модель и получить Промежуточное представление модели, Оптимизатор модели должен уметь работать с этими слоями.

Полный список данных слоёв зависит от фреймворка и может быть найден в разделе «Поддерживаемые Слои Фреймворка» [https://docs.openvinotoolkit.org/latest/openvino_docs_MO_DG_prepare_model_Supported_Frameworks_Layers.html]. Если топология содержит только те слои, которые содержатся в списке слоёв (как в случае с топологиями, используемыми большинством пользователей), то Оптимизатор модели легко

создаст Промежуточное представление. После этого можно приступать к работе с Механизмом вывода.

2.2.1 Оптимизация модели

Обучение сетям, как правило, проводится в высококлассных центрах обработки данных с использованием популярных обучающих фреймворков, таких как Caffe, TensorFlow и MXNet. Оптимизатор моделей преобразует обученную модель в оригинальные проприетарные форматы в IR, описывающий топологию. IR сопровождается двоичным файлом с весами. Эти файлы, в свою очередь, используются механизмом вывода.

Существует ряд устройств агностических оптимизаций, которые выполняет инструмент. Например, некоторые примитивы, такие как линейные операции (BatchNorm и ScaleShift), автоматически объединяются в свёртки. Как правило, эти слои не должны проявляться в результирующем IR-представлении:

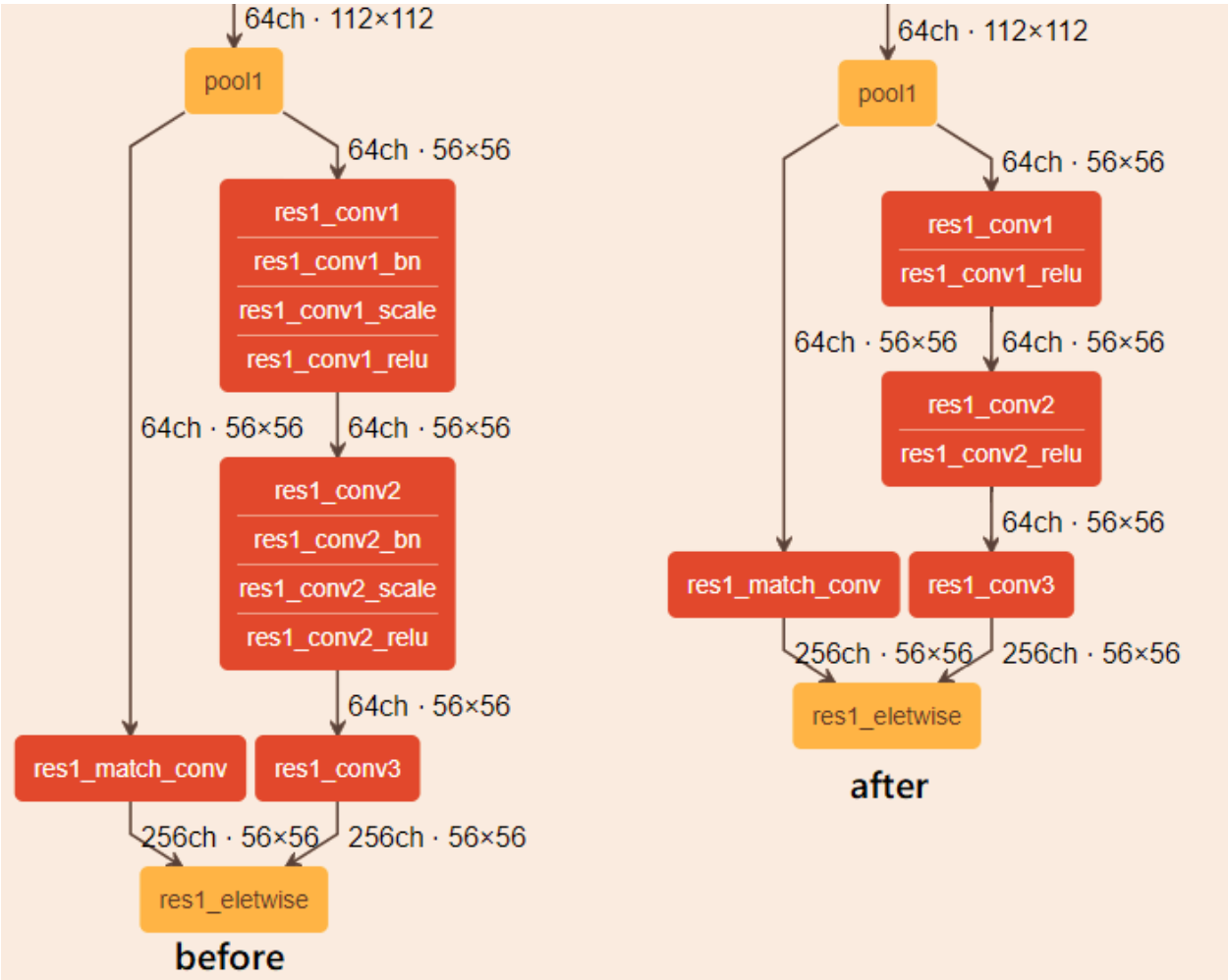


Рисунок 22 – Пример оптимизации топологии.

На рисунке выше показана топология Caffe Resnet269. Левая модель – это исходная модель, а правая (после преобразования) – результирующая модель, которую производит оптимизатор моделей, со слоями BatchNorm и ScaleShift, слитыми в веса свертки, а не образующими отдельные слои.

Но существуют моменты, когда слияние невозможно. Например, нелинейные операции (такие, как активации) между свёртками и линейными операциями могут предотвратить слияние. Если производительность вызывает беспокойство, необходимо изменить (и потенциально переобучить) топологию.

Важно, что активация (_relu) не затрагивается оптимизатором модели, и, хотя он также может быть объединен в свЯртку, это скорее специфичная для устройства оптимизация, охватываемая механизмом вывода во время загрузки

модели. Рекомендуется проверять счетчики производительности из плагинов, которые должны указывать на то, что эти конкретные слои не выполняются.

Под оптимизацию попадают следующее:

- Параметры среднего/масштабного изображения. Актуально в случае, когда используются параметры среднего/масштабного входного изображения (--scale и –mean_values) с оптимизатором модели, когда вам нужна предварительная обработка. Это позволяет инструменту генерировать предварительную обработку в IR, чтобы получить ускорение с помощью механизма вывода.
- RGB или BGR. Если, например, сеть принимает входы RGB, оптимизатор модели может поменять местами каналы в первой свертке, используя --reverse_input_channels опцию командной строки. Поэтому не нужно преобразовывать входные данные в RGB каждый раз, когда получается изображение BGR, например, из OpenCV.
- Больший размер batch. Обратите внимание, что такие устройства, как GPU, работают лучше с большим размером пакета. В то время как можно установить размер пакета во время выполнения с помощью механизма вывода Функция ShapeInference.
- Результирующая IR-точность. Например, результирующая IR-точность, FP16 или FP32, непосредственно влияет на производительность. Как процессор теперь поддерживает FP16 (при внутреннем масштабировании до FP32 в любом случае) и поскольку это лучшая точность для цели GPU, можно всегда конвертировать модели в FP16. Обратите внимание, что это единственная точность, которую поддерживают VPUS Intel Movidius Myriad 2 и Intel Myriad X.

2.2.2 Пользовательским операциям

Пользовательские операции – это операции, которые не включены в список известных операций. Если ваша модель содержит какую-либо

операцию, которой нет в списке известных операций, Оптимизатор модели не может создать промежуточное представление (IR) для этой модели.

Существует три шага для поддержки вывода модели с пользовательскими операциями):

- Добавить поддержку пользовательской операции в оптимизаторе модели, чтобы оптимизатор модели мог генерировать IR-сигнал вместе с этой операцией.
- Создать набор операций и реализовать в нем пользовательскую операцию nGraph.
- Реализовать операцию клиента в одном из плагинов Inference Engine для поддержки вывода этой операции с использованием определенного целевого оборудования (CPU, GPU или VPU).

Если устройство не поддерживает определенную операцию, альтернативой созданию новой операции является нацеливание дополнительного устройства с помощью плагина HETERO. Гетерогенный плагин может использоваться для запуска модели вывода на нескольких устройствах, позволяя неподдерживаемым операциям на одном устройстве «отступать» на другое устройство (например, процессор), которое поддерживает эти операции.

Два типа расширений оптимизатора модели должны быть реализованы для поддержки пользовательской операции, как минимум:

- Класс для новой операции. Этот класс хранит информацию об операции, её атрибутах, функции вывода формы, атрибутах, которые должны быть сохранены в IR, и некоторых других внутренне используемых атрибутах.
- Экстрактор атрибутов операции. Экстрактор отвечает за разбор специфичного для фреймворка представления операции и использует соответствующий класс операции для обновления атрибутов узла графа необходимыми атрибутами операции.

Расширения пользовательских операций для механизма вывода

Механизм вывода предоставляет механизм расширения для поддержки новых операций.

Каждый плагин устройства включает в себя библиотеку оптимизированных реализаций для выполнения известных операций, которые должны быть расширены для выполнения пользовательской операции. Расширение пользовательской операции реализуется в соответствии с целевым устройством:

- Расширение пользовательской операции для CPU. Скомпилированная общая библиотека (.so, .dylibor, .dll), необходимая плагину CPU для выполнения пользовательской операции на процессоре.
- Расширение пользовательская операция для GPU. Исходный код OpenCL (.cl) для ядра пользовательских операций будет скомпилирован для выполнения на графическом процессоре вместе с файлом описания операций (.xml), необходимым плагину GPU для ядра пользовательских операций.
- Пользовательская операция Расширение VPU. Исходный код OpenCL (.cl) для ядра пользовательской операции будет скомпилирован для выполнения на VPU вместе с файлом описания операции (.xml), необходимым плагином VPU для ядра пользовательской операции.

Кроме того, необходимо реализовать пользовательскую операцию nGraph в соответствии с пользовательской операцией nGraph, чтобы механизм вывода мог считывать IR-сигнал с помощью этой операции и правильно выводить форму и тип выходных тензоров.

2.3 nGraph

nGraph представляет нейронные сети в едином формате. Пользователь может создавать различные операции и объединять их в одну ngraph::Function.

Функция nGraph – это очень простая вещь. Она хранит общие указатели ngraph::op::Parameterngraph::op::Result и ngraph::op::Sink операции, которые являются входами, выходами и приёмниками графа. Приёмники графа не имеют потребителей и не входят в вектор результатов. Все остальные операции связываются друг с другом с помощью общих указателей: дочерняя операция связывается со своим родителем (жёсткая ссылка). Если операция не имеет приёмников и не является операцией результата или приёмника (счётчик общего указателя равен нулю), то она будет уничтожена и больше не будет доступна. Каждая операция в ngraph::Function имеет std::shared_ptr<ngraph::Node> тип.

ngraph::Op представляет любые абстрактные операции в представлении nGraph.

Набор операций представляет собой набор некоторых операций nGraph. nGraph::Opset – это класс, который предоставляет функциональность для работы с наборами операций. Набор пользовательских операций должен быть создан для их поддержки.

nGraph имеет два типа для представления формы:

- ngraph::Shape – представляет статические (полностью определенные) фигуры.
- ngraph::PartialShape - представляет собой динамические формы. Это означает, что ранг или некоторые измерения являются динамическими (неопределенными). ngraph::PartialShape может быть преобразован в ngraph::Shape использованием метода get_shape(), если все измерения статичны, в противном случае преобразование вызывает исключение.

Механизм вывода интегрирует nGraph для представления модели во время выполнения под обычным CNNNetwork API, который является экземпляром ngraph::Function.

Помимо обновления представления, nGraph поддерживает функции:

1. nGraph содержит несколько наборов операций, которые называются "opset1", "opset2" и т.д. Операции из этих наборов операций генерируются Оптимизатором модели и принимаются механизмом логического вывода.
2. Версия операции привязывается к каждой операции, а не ко всему формату IR-файла. IR по-прежнему версионен, но имеет другое значение.
3. Создание моделей во время выполнения без загрузки IR из xml/двоичного файла. Можно включить его, создав ngraph::Function передачу CNNNetwork.
4. Возможность изменения формы во время выполнения и постоянное сворачивание реализованы через код nGraph для большего количества операций по сравнению с предыдущими выпусками. В результате может быть изменено больше моделей.
5. Загрузка модели из формата ONNX без преобразования её в IR - механизм вывода.
6. Представление nGraph поддерживает динамические формы. Можно использовать CNNNetwork::reshape() метод для того, чтобы специализировать входные фигуры.

Полная картина взаимодействия компонентов представлена ниже.

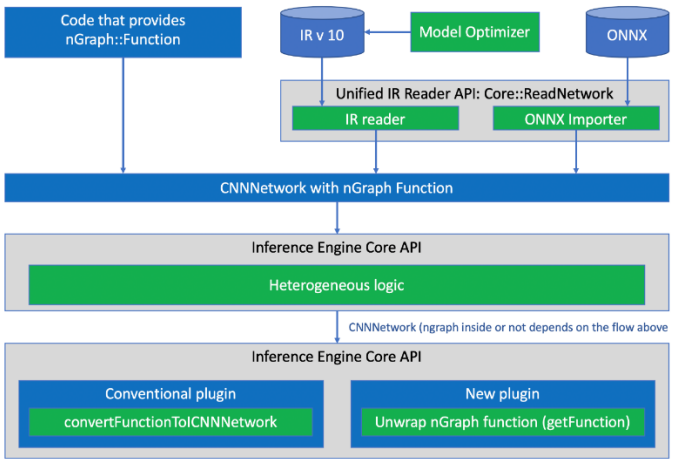


Рисунок 23 – Процесс взаимодействия компонентов OpenVino

IR версии 10 автоматически запускает поток nGraph внутри механизма вывода. Когда такой IR считывается в приложении, модуль вывода IR reader производит CNNNetwork и инкапсулирует ngraph::Function экземпляр.

Альтернативный метод подачи модели в механизм вывода – это создание модели во время выполнения. Это достигается созданием ngraph::Function построение с использованием классов операций nGraph и, возможно, пользовательских операций. На этом этапе код полностью независим от остального кода Механизма вывода и может быть построен отдельно. После создания экземпляра ngraph::Function, можно использовать его для создания CNNNetwork передавая его в новый конструктор для этого класса.

2.3.1 Intermediate Representation

Общая структура нейронных сетей

Сеть глубокого обучения обычно представлена в виде направленного графа, описывающего поток данных от входных данных сети к результатам вывода. Входные данные могут быть представлены в виде фото-, видео-, аудиоинформации или некоторых предварительно обработанных данных, которые представляют объект из интересующей области удобным способом.

Ниже показана иллюстрация небольшого графа, представляющего модель, состоящую из одного свёрточного слоя и функции активации:

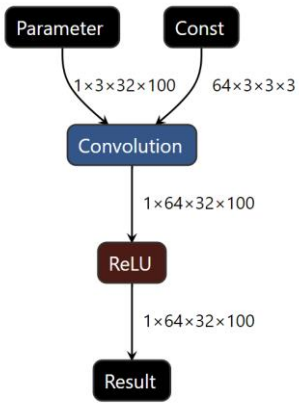


Рисунок 24 – Граф простой модели

Вершины графа представляют собой слои или экземпляры операций, таких как свёртка, объединение или поэлементные операции с тензорами. Термины слоя и операции используются взаимозаменяемо в документации OpenVINO и определяют, как обрабатываются входные данные для получения выходных данных для узла в графе. Операционный узел в графе может потреблять данные через один или несколько входных портов. Например, операция поэлементного сложения имеет два входных порта, которые принимают тензоры, сложенные вместе. Некоторые операции не имеют никаких входных портов, например, операция Const, которая знает данные, которые будут получены без какого-либо ввода. Ребро между операциями представляет собой поток данных или зависимость данных, подразумеваемую от одного операционного узла к другому операционному узлу.

Каждая операция производит данные на одном или нескольких выходных портах. Например, свертка производит выходной тензор с активациями на одном выходном порту. Операция разделения обычно имеет несколько выходных портов, каждый из которых производит часть входного тензора.

В зависимости от структуры глубокого обучения граф может также содержать дополнительные узлы, которые явно представляют тензоры между операциями. В таких представлениях узлы операций не связаны непосредственно друг с другом, а используют узлы данных в качестве промежуточных остановок для потока данных. Если узлы данных не используются, полученные данные связываются с портом вывода соответствующего узла операции, который производит данные.

Набор различных операций, используемых в сети, обычно фиксирован для каждой структуры глубокого обучения. Она определяет экспрессивность и уровень репрезентации, доступный в этой структуре. Может случиться так, что сеть, которая может быть представлена в одном фреймворке, трудно или невозможно представить в другом или она должна использовать значительно

отличающийся граф, поскольку операционные наборы, используемые в этих двух фреймворках, не совпадают.

Intermediate Representation OpenVINO

OpenVINO toolkit представляет собственный формат представления графов и собственный набор операций. График представлен двумя файлами: XML - файлом и двоичным файлом. Это представление обычно называют Промежуточным представлением или IR.

XML-файл описывает топологию сети с использованием <layer> тега для рабочего узла, а <edge> тег для подключения к потоку данных. Каждая операция имеет фиксированное количество атрибутов, определяющих логику операции, используемый для узла. Например, Convolution операция имеет такие атрибуты, как dilation,stride, pads_begin и pads_end.

XML-файл не имеет больших постоянных значений, таких как веса свёртки. Вместо этого он ссылается на часть сопровождающего двоичного файла, который хранит такие значения в двоичном формате. Пример XML-файла представлен в Приложении 2.

IR не использует явные узлы данных, описанные в предыдущем разделе. Напротив, свойства данных, таких как тензорные измерения и их типы данных, описываются как свойства входных и выходных портов операций.

Набор операций

Операции в операционном наборе OpenVINO выбираются на основе возможностей, поддерживаемых фреймворков глубокого обучения и аппаратных возможностей целевого устройства вывода. Он состоит из нескольких групп операций:

- Обычные слои глубокого обучения, такие как свертка, MaxPool, MatMul (также известный как FullyConnected).
- Различные функции активации, например, ReLU, Tanh, PReLU.
- Общие поэлементные арифметические тензорные операции, такие как Сложение, Вычитание, Умножение.

- Операции сравнения, которые сравнивают два числовых тензора и производят булевы тензоры, например, меньше, равно, больше.
- Логические операции, имеющие дело с булевыми тензорами, например, And, Xor, Not.
- Операции перемещения данных, имеющие дело с частями тензоров: Concat, Split, StridedSlice, Select.
- Специализированные операции, реализующие сложные алгоритмы, предназначенные для моделей определенного типа: DetectionOutput, RegionYolo, PriorBox.

Версии IR и версий операционного набора

Выразительность операций в OpenVINO сильно зависит от поддерживаемых фреймворков и возможностей целевого оборудования. По мере того, как фреймворки и аппаратные возможности растут с течением времени, операционный набор постоянно развивается для поддержки новых моделей. Для поддержания обратной совместимости и растущих требований как IR-формат, так и операционный набор имеют управление версиями.

Версия IR определяет правила, которые используются для чтения XML и двоичных файлов, представляющих модель. Они определяют схему XML и совместимый набор операций, которые могут быть использованы для описания операций.

Исторически существуют две основные эпохи IR версии.

1. Более старый включает в себя версии IR от версии 1 до версии 7 без управления версиями набора операций. В течение этой эпохи набор операций эволюционно рос, накапливая все больше типов слоев и расширяя существующую семантику слоев. Изменение набора операций для этих версий означало увеличение IR-версии.
2. OpenVINO 2020.1 – это отправная точка следующей эпохи. С появлением IR версии 10 в OpenVINO 2020.1 управление версиями операционного набора отслеживается отдельно от управления

версиями IR. Кроме того, набор операций был значительно переработан в результате интеграции nGraph в OpenVINO.

Первый поддерживаемый набор операций в новой эпохе – это opset1. Число после opset будет увеличиваться каждый раз, когда добавляются новые операции или удаляются старые операции в новом выпуске.

Операции из новой версии охватывают больше операторов TensorFlow и ONNX в форме, более близкой к исходной семантике операций из фреймворков по сравнению с набором операций, используемым в предыдущих версиях IR (7 и ниже).

Когда вводится новый набор операций, значительная часть операций остается неизменной, и она просто стирается из предыдущего набора операций в новый. Целью эволюции версий набора операций является добавление новых операций и, возможно, изменение небольшой части существующих операций (исправление ошибок и расширение семантики). Однако такие изменения затрагивают только новые версии операций из нового набора операций, в то время как старые операции используются путем указания, соответствующей версии. Когда старая версия задана, поведение сохраняется неизменным от указанной версии, чтобы обеспечить обратную совместимость со старыми IRS.

Один xml файл с IR может содержать операции из разных операционных наборов. Операция, включенная в несколько opset, может быть связана с версией любого opset, включающим эту операцию. Например, тот же Convolution можно использовать с version = "opset1"и version = "opset2" потому, что оба opset имеют одни и те же операции Convolution.

2.4 Inference Engine

Другим ключевым компонентом OpenVINO является Inference Engine, который управляет загрузкой и компиляцией, оптимизированной нейросетевой модели, выполняет операции вывода на входных данных и выводит результаты. Механизм вывода может выполняться синхронно или

асинхронно, а его существующая архитектура плагинов управляет соответствующими компиляциями для выполнения на нескольких устройствах Intel, включая как процессоры workhorse, так и специализированные платформы обработки графики и видео.

Перед выполнением вывода с помощью механизма вывода модели должны быть преобразованы в формат механизма вывода с помощью оптимизатора моделей или встроены непосредственно во время выполнения с помощью nGraph API.

После использования Оптимизатора модели для создания промежуточного представления (IR) необходимо использовать Inference Engine для вывода результата для заданных входных данных.

Inference Engine – это набор библиотек C++, предоставляющих общий API для предоставления решений вывода на платформе по вашему выбору: CPU, GPU или VPU. API механизма вывода для чтения Промежуточного представления предоставляет функционал по установки форматов ввода и вывода и выполнения модели на устройствах. В то время как библиотеки C++ являются основной реализацией. Также доступны библиотеки C и привязки Python.

Inference Engine использует архитектуру плагинов. Плагин Inference Engine – это программный компонент, содержащий полную реализацию вывода на определенном аппаратном устройстве Intel: CPU, GPU, VPU и т.д. Каждый плагин реализует унифицированный API и предоставляет дополнительные аппаратные API.

2.4.1 Модули в компоненте Inference Engine

Библиотеки ядра механизма вывода

Приложение должно быть связано с основными библиотеками механизма вывода:

- Linux OS: libinference_engine.so, который зависит отlibinference_engine_transformations.so, libtbb.so, libtbbmalloc.so и libngraph.so.
- Windows OS: inference_engine.dll, который зависит отinference_engine_transformations.dll, tbb.dll, tbbmalloc.dll и ngraph.dll.
- macOS: libinference_engine.dylib, который зависит от libinference_engine_transformations.dylib, libtbb.dylib, libtbbmalloc.dylib и libngraph.dylib.

Необходимые заголовочные файлы C++ находятся в include каталоге.

Эта библиотека содержит классы для:

- Создание объекта ядра механизма вывода для работы с устройствами и чтения сети (InferenceEngine::Core)
- Манипулирование сетевой информацией (InferenceEngine::CNNNetwork)
- Выполнение и передача входов и выходов (InferenceEngine::ExecutableNetwork и InferenceEngine::InferRequest)

Библиотеки плагинов для чтения сетевого объекта

Начиная с выпуска 2020.4, Inference Engine ввел концепцию CNNNetwork плагинов reader. Такие плагины могут быть автоматически динамически загружены механизмом вывода во время выполнения в зависимости от формата файла:

- Linux* OS:
  - libinference_engine_ir_reader.so – для чтения сети из IR
  - libinference_engine_onnx_reader.so – для чтения сети из формата модели ONNX
- Windows* OS:
  - inference_engine_ir_reader.dll – читать сеть из IR
  - inference_engine_onnx_reader.dll для чтения сети из формата модели ONNX

Для каждого поддерживаемого целевого устройства Механизм вывода предоставляет плагин DLL/общую библиотеку, которая содержит полную

реализацию вывода на этом конкретном устройстве. Доступны следующие плагины:

1. CPU – Intel® Xeon® with Intel® AVX2 and AVX512, Intel® Core™ Processors with Intel® AVX2, Intel® Atom® Processors with Intel® SSE
2. GPU – Intel® Processor Graphics, including Intel® HD Graphics and Intel® Iris® Graphics
3. MYRIAD – Intel® Neural Compute Stick 2 powered by the Intel® Movidius™ Myriad™ X
4. GNA – Intel® Speech Enabling Developer Kit, Amazon Alexa* Premium Far-Field Developer Kit, Intel® Pentium® Silver J5005 Processor и другие
5. HETERO – Автоматическое разделение сетевого вывода между несколькими устройствами (например, если устройство не поддерживает определенные уровни)
6. MULTI – Одновременный вывод одной и той же сети на несколько устройств параллельно.

Все библиотеки плагинов также зависят от основных библиотек механизма вывода.

2.4.2 Примитивы памяти механизма вывода

Blobs

InferenceEngine::Blob является основным классом, предназначенным для работы с памятью. С помощью этого класса можно читать и записывать в память, получать информацию о структуре памяти и т.д.

Правильный способ создания Blob объекты с определенной компоновкой – это использование конструкторов с InferenceEngine::TensorDesc.

Layouts

InferenceEngine::TensorDesc – это специальный класс, который предоставляет описание формата макета.

Этот класс позволяет создавать плоские макеты с использованием стандартных форматов (например, InferenceEngine::Layout::NCDHW, InferenceEngine::Layout::NCHW, InferenceEngine::Layout::NC, InferenceEngine::Layout::C и т.д.), а также неплоские макеты с использованием InferenceEngine::BlockingDesc.

Для создания сложного макета следует использовать InferenceEngine::BlockingDesc что позволяет определить заблокированную память со смещениями и шагами.

2.4.3 Вывод Bfloat16

Вычисления Bfloat16 (называемые BF16) – это формат с плавающей запятой с 16 битами. Это усеченная 16-битная версия 32-битного формата FP32 с плавающей запятой одинарной точности IEEE 754. BF16 сохраняет 8 битов экспоненты как FP32, но снижает точность знака и мантиссы с 24 бит до 8 бит.

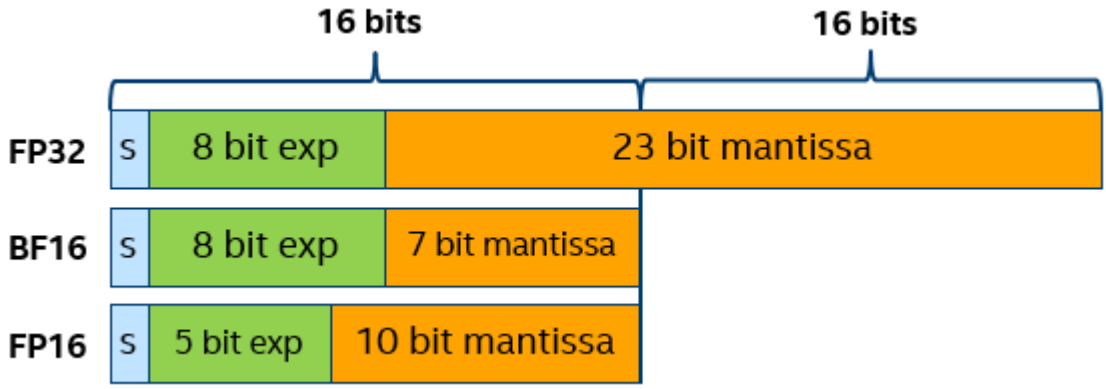


Рисунок 25 – Сравнение форматов

Сохранение битов экспоненты удерживает BF16 в том же диапазоне, что и FP32 (от~1e-38 до ~3e38). Это упрощает преобразование между двумя типами данных: вам просто нужно пропустить или сбросить до нуля 16 младших битов. Усеченная мантисса иногда приводит к меньшей точности, но, согласно исследованиям, нейронные сети более чувствительны к размеру экспоненты, чем размер мантиссы. Кроме того, во многих моделях точность необходима близкая к нулю, но не так сильно на максимальном диапазоне. Еще одной полезной особенностью BF16 является возможность кодирования INT8 в BF16 без потери точности, поскольку диапазон INT8 полностью вписывается в поле мантиссы BF16. Он уменьшает поток данных при преобразовании входных данных изображения INT8 в BF16 непосредственно

без промежуточного представления в FP32 или в комбинации уровней вывода INT8 и BF16.

Механизм вывода с выводом bfloat16, реализованным на процессоре, должен поддерживать собственную avx512_bf16 инструкцию и, следовательно, формат данных bfloat16. Вывод bfloat16 можно использовать в режиме моделирования на платформах с Intel Advanced Vector Extensions 512 (Intel AVX-512), но это приводит к значительному снижению производительности по сравнению с FP32 или avx512_bf16 использованием собственных команд

Есть два способа проверить, может ли процессорное устройство поддерживать вычисления bfloat16 для моделей:

1. Запросить набор команд через system lscpu | grep avx512_bf16 или cat /proc/cpuinfo | grep avx512_bf16.
2. Использование Query API с METRIC_KEY(OPTIMIZATION_CAPABILITIES), который должен вернуться BF16 в список параметров оптимизации процессора.

Текущее решение Inference Engine для вывода bfloat16 использует библиотеку математического ядра Intel для глубоких нейронных сетей (Intel MKL-DNN) и поддерживает вывод значительного числа слоев в режиме вычисления BF16.

Снижение Точности Вывода

Снижение точности для повышения производительности широко используется для оптимизации логического вывода. Использование типа данных bfloat16 на процессоре впервые открывает возможность подхода оптимизации по умолчанию. Воплощение такого подхода заключается в использовании оптимизационных возможностей текущей платформы для достижения максимальной производительности при сохранении точности расчетов в допустимых пределах.

Использование данных Bfloat16 обеспечивает следующие преимущества, повышающие производительность:

- Более быстрое умножение двух чисел BF16 из-за более короткой мантиссы данных bfloat16.
- Нет необходимости поддерживать денормалы и обрабатывать исключения, так как это оптимизация производительности.
- Быстрое преобразование float32 в bfloat16 и наоборот.
- Уменьшенный размер данных в памяти приводит к тому, что более крупные модели помещаются в те же границы памяти.
- Уменьшенный объем данных, которые должны быть переданы, в результате сокращается время перехода данных.

Для оптимизации по умолчанию на процессоре исходная модель преобразуется из FP32 или FP16 в BF16 и выполняется внутренне на платформах с собственной поддержкой BF16. В этом случае KEY_ENFORCE_BF16 устанавливается значение YES.

Чтобы отключить внутренние преобразования BF16, установите значение KEY_ENFORCE_BF16 в NO. В этом случае модель выводится как есть без изменений с точностями, которые были установлены на каждом краю слоя.

Низкоточные 8-битные целочисленные модели не могут быть преобразованы в BF16, даже если оптимизация bfloat16 установлена по умолчанию.

Режим моделирования Bfloat16 доступен на платформах CPU и Intel AVX-512, которые не поддерживают собственную avx512_bf16 инструкцию. Симулятор не гарантирует адекватной производительности.

Информация о точности слоя хранится в счетчиках производительности, доступных из API механизма вывода. Слои имеют следующие метки:

1. Суффикс BF16 для слоев, которые имели входной тип данных bfloat16 и были вычислены с точностью BF16
2. Суффикс FP32 для слоев, вычисленных с 32-битной точностью

2.4.4 Низко точный 8-разрядный целочисленный Вывод

Низкоточный 8-битный вывод оптимизирован для:

- Процессоры архитектуры Intel со следующими набори команд:
  - Intel Advanced Vector Extensions 512 Инструкций Векторной нейронной Сети (Intel AVX-512 VNNI)
  - Intel Advanced Vector Extensions 512 (Intel AVX-512)
  - Intel Advanced Vector Extensions 2.0 (Intel AVX2)
  - Intel Streaming SIMD Extensions 4.2 (Intel SSE4.2)
- Графика процессора Intel:
  - Intel Iris Xe Graphics
  - Графика Intel Iris Xe MAX
- Модель должна быть квантована. Можно использовать квантованную модель из предварительно обученных моделей Intel OpenVINO Toolkit или квантовать модель самостоятельно. Для квантования можно использовать:
  - Инструмент оптимизации после обучения поставляется вместе с дистрибутивом Intel пакета выпуска OpenVINO toolkit.
  - Фреймворк сжатия нейронных сетей доступен на GitHub[https://github.com/openvinotoolkit/nncf]

Много исследований было сделано в области глубокого обучения с идеей использования низко точных вычислений во время вывода, чтобы увеличить конвейеры глубокого обучения и получить более высокую производительность. Например, одним из популярных подходов является уменьшение точности значений активаций и весов с fp32 точности до меньших значений, например, до fp11 или int8.

8-разрядные вычисления (int8) обеспечивают лучшую производительность по сравнению с результатами вывода с более высокой точностью (например, fp32), поскольку они позволяют загружать больше данных в одну инструкцию процессора. Обычно стоимость значительного

повышения - это снижение точности. Однако доказано, что падение точности может быть незначительным и зависит от требований задачи, так что инженер-прикладник может установить максимальное падение точности, которое является приемлемым.

Рабочий процесс низко точного 8-битного целочисленного Вывода

Для 8-битных целочисленных вычислений модель должна быть квантована. Квантованные модели можно загрузить из Обзора предварительно обученных моделей Intel OpenVINO Toolkit. Если модель не квантована, можно использовать инструмент оптимизации после обучения для квантования модели. Процесс квантования добавляет слои FakeQuantize на активациях и весах для большинства слоев.

8-битный конвейер вывода включает в себя два этапа:

1) Автономная стадия, или квантование модели.

На этом этапе слои FakeQuantize добавляются перед большинством слоев, чтобы иметь квантованные тензоры перед слоями таким образом, чтобы падение точности с низкой точностью для 8-битного целочисленного вывода удовлетворяло заданному порогу. Выход этого этапа – это квантованная модель. Точность квантованной модели не изменяется, квантованные тензоры находятся в исходном диапазоне точности (fp32). FakeQuantize слой имеет levels атрибут, который определяет количество квантов. Количество квантов определяет точность, которая используется при выводе. Для int8 диапазона levels значение атрибута должно быть 255 или 256. Для квантования модели можно использовать Инструмент оптимизации после обучения поставляется вместе с дистрибутивом Intel пакета выпуска OpenVINO toolkit.

Когда вы передаете квантованный IR плагин OpenVINO, плагин автоматически распознает его как квантованную модель и выполняет 8-битный вывод. Обратите внимание, если вы передаете квантованную модель другому плагину, который не поддерживает 8-битный вывод, но поддерживает все операции из модели, модель выводится с точностью, которую поддерживает этот плагин.

2) Этап выполнения.

Этот этап является внутренней процедурой плагина OpenVINO. На этом этапе квантованная модель загружается в плагин. Плагин использует Low Precision Transformation компонент для обновления модели, чтобы вывести ее с низкой точностью

2.4.5 Использование динамического пакетирования (Batching)

Функция динамической пакетной обработки позволяет динамически изменять размер пакета для вызовов вывода в пределах заданного размера пакета. Эта функция может быть полезна, когда размер пакета заранее неизвестен, а использование сверхбольшого размера пакета нежелательно или невозможно из-за ограничений ресурсов. Например, по распознаванию лиц распознавать возраст, пол или настроение человека является типичным сценарием использования.

Можно активировать динамическое пакетирование, установив KEY_DYN_BATCH_ENABLED флаг YES в конфигурации, которая передается плагину при загрузке сети. Эта конфигурация создает ExecutableNetwork объект, который позволит динамически устанавливать размер пакета во всех его запросах вывода с помощью SetBatch() метода. Размер пакета, заданный в переданном CNNNetwork объекте, будет использоваться в качестве максимального ограничения размера пакета.

В настоящее время существуют определенные ограничения на использование Динамического дозирования:

- Используйте динамическую пакетную обработку только с плагинами CPU и GPU.
- Используйте динамическое дозирование только для топологий, состоящих из определенных слоев:
    - Convolution
    - Deconvolution
    - Activation
    - LRN
    - Pooling

- FullyConnected
- SoftMax
- Split
- Concatenation
- Power
- Eltwise
- Crop
- BatchNormalization
- Copy

Не следует использовать слои, которые могут произвольно изменяют форму тензора (например, выравнивать, переставлять, изменять форму), слои, относящиеся к топологиям обнаружения объектов (ROIPooling, ProirBox, DetectionOutput), и пользовательские слои. Анализ топологии выполняется в процессе загрузки сети в плагин, и, если топология неприменима, генерируется исключение.

2.5 Конфигурация

Обзор механизма вывода Глубокого Обучения

Deep Learning Inference Engine является частью Intel Deep Learning Deployment Toolkit (Intel DL Deployment Toolkit) и OpenVINO toolkit. Механизм вывода облегчает развертывание решений для глубокого обучения, предоставляя унифицированный, аппаратно-агностический API.

Ниже приведены три основных этапа процесса развертывания, которые можно оптимизировать:

- Обученные модели преобразуются из определенного фреймворка, такого как TensorFlow, или формата, такого как ONNX, в формат промежуточного представления (IR).
- Вывод/выполнение модели

После преобразования Inference Engine потребляет IR для выполнения вывода. В то время как API Inference Engine сам по себе является целевым агностиком, внутренне он имеет понятие плагинов, которые являются

специфичными для устройств библиотеками, облегчающими аппаратное ускорение.

- Интеграция с продуктом

После проверки вывода модели с помощью образцов код механизма вывода обычно интегрируется в реальное приложение или конвейер.

Рекомендации к сбору метрик для оценивания производительности

Данные о производительности приходят в самых разных формах. Например, одним из наиболее распространенных показателей производительности является латентность, представляющая время, необходимое для выполнения единицы работы (например, время вывода для одного изображения).

Оценивая производительность вашей модели с помощью Механизма логического вывода, нужно измерить правильный набор операций. Для этого рассмотрены следующие советы:

- Избегать включения разовых затрат, таких как загрузка модели.
- Отследить отдельно операции, которые происходят вне Механизма вывода, такие как декодирование видео.

В асинхронном случае производительность отдельного запроса вывода обычно вызывает меньше беспокойства. Вместо этого обычно выполняются несколько запросов асинхронно и измеряется пропускная способность в изображениях в секунду, разделяя количество изображений, которые были обработаны за время обработки. Напротив, для задач, ориентированных на задержку, время до одного кадра более важно.

При сравнении производительности механизма вывода с фреймворком или другим эталонным кодом убедитесь, что обе версии максимально похожи:

- Необходимо обернуть точное выполнение вывода
- Время загрузки модели трека рассчитать отдельно.
- Убедиться, что входные данные идентичны для механизма вывода и фреймворка. Например, Caffe позволяет автоматически заполнять

входные данные случайными значениями. Обратите внимание, что это может дать другую производительность, чем на реальных изображениях.

- Аналогично, для корректного сравнения производительности убедиться, что шаблон доступа, например, входные макеты, является оптимальным для механизма вывода (в настоящее время это NCHW).
- Любая предварительная обработка на стороне пользователя должна отслеживаться отдельно.
- Обязательно использовать те же параметры среды, которые рекомендуют разработчики фреймворка, например, для TensorFlow. Во многих случаях вещи, более дружественные к машине, такие как соблюдение NUMA также могут хорошо работать для механизма вывода.
- Если применимо, необходимо использовать batching с помощью механизма вывода.
- По возможности требуется такая же точность. Например, TensorFlow допускает FP16 поддержку, поэтому при сравнении с ним обязательно протестируйте Inference Engine с FP16.

Так же нужно строить свои выводы о производительности на воспроизводимых данных. Выполните измерения производительности с большим количеством вызовов одной и той же процедуры. Поскольку первая итерация почти всегда значительно медленнее последующих, можно использовать агрегированное значение времени выполнения для конечных прогнозов:

- Если, прогрев не помогает или время выполнения все еще меняется, можно попробовать выполнить большое количество итераций, а затем усреднить или найти среднее значение результатов.
- Для значений времени, которые слишком сильно варьируются, необходимо использовать geomean.

Выполнение на нескольких устройств

Инструментарий OpenVINO поддерживает автоматическое выполнение на нескольких устройств. Рекомендаций для выполнения нескольких устройств:

- MULTI обычно работает лучше всего, когда самое быстрое устройство указано первым в списке устройств. Это особенно важно, когда параллелизм недостаточен.
- Настоятельно рекомендуется запрашивать оптимальное количество запросов вывода непосредственно из экземпляра ExecutionNetwork
- Обратите внимание, что, например, выполнение CPU+GPU лучше работает с определенными флагами. Одним из конкретных примеров является отключение опроса драйверов GPU, который, в свою очередь, требует нескольких потоков GPU (что уже является значением по умолчанию), чтобы амортизировать более медленное завершение вывода от устройства к хосту.
- Логика нескольких устройств всегда пытается сэкономить на копиях данных (например, входных данных) между агностическими запросами вывода, обращенными к пользователю, и специфичными для устройства "рабочими" запросами, которые фактически планируются за сценой. Чтобы облегчить экономию копирования, рекомендуется запускать запросы в том порядке, в котором они были созданы (с помощью CreateInferRequest ExecutableNetwork).

2.5.1 Оптимизация для Конкретных Устройств

Механизм вывода поддерживает несколько целевых устройств (CPU, GPU, Intel Movidius Myriad 2 VPU, Intel Movidius Myriad X VPU, Intel Vision Accelerator Design with Intel Movidius Vision Processing Units (VPU) и FPGA), и каждое из них имеет соответствующий плагин. Если нужно оптимизировать конкретное устройство, вы должны иметь в виду следующие советы по повышению производительности.

CPU

CPU плагин полностью полагается на библиотеку Intel Math Kernel Library for Deep Neural Networks (Intel MKL-DNN) для ускорения основных примитивов, например, сверток или полностью подключенных.

Единственный информацию, которую можно получить из этого – это то, как ускоряются основные примитивы (и вы не можете изменить это). Например, на основных машинах вы должны увидеть вариации jit_avx2 при осмотре внутренние счетчики производительности вывода (и дополнительный постфикс '_int8' для вывода int8). Если вы опытный пользователь, можно дополнительно проследить выполнение процессора с помощью Intel VTune.

Внутренне Inference Engine имеет уровень потоковой абстракции, который позволяет компилировать версия с открытым исходным кодом с помощью либо Intel Threading Building Blocks (Intel TBB), который с недавних пор используется по умолчанию, либо OpenMP в качестве альтернативного решения параллелизма. При использовании вывода на ЦП это особенно важно для выравнивания модели потоковой передачи с остальной частью вашего приложения (и любыми сторонними библиотеками, которые вы используете), чтобы избежать избыточной подписки.

Начиная с R1 2019, инструментарий OpenVINO поставляется предварительно скомпилированным с Intel TBB, поэтому любые настройки OpenMP API или среды (например, OMP_NUM_THREADS) не имеет никакого эффекта. Некоторые настройки (например, количество потоков, используемых для вывода на CPU) все еще возможны с помощью Параметры конфигурации процессора.

Другие общие рекомендации:

- Обычно пакетирование повышает производительность процессора. Однако необходимость собирать кадры в пакет может усложнить логику приложения. Вместо этого можно сохранить отдельный запрос вывода для каждой камеры или другого источника ввода и обрабатывать запросы параллельно. Дополнительные сведения см. в следующем разделе.

- Если ваше приложение одновременно выполняет вывод нескольких моделей на одном процессоре, убедитесь, что вы не переподписываете машину. См. Раздел Аспекты производительности одновременного выполнения нескольких запросов.
- Обратите внимание, что гетерогенное выполнение может неявно нагружать процессор. Для получения более подробной информации обратитесь к Гетерогенность.
- Считать 8-битный целочисленный вывод на ЦП.

GPU

Механизм вывода опирается на Вычислительную библиотеку глубоких нейронных сетей (clDNN) для ускорения свёрточных нейронных сетей на графических процессорах Intel. Внутренне clDNN использует OpenCL для реализации ядер. Таким образом, применимы многие общие советы:

- Предпочитайте FP16 вместо FP32, так как оптимизатор модели может генерировать оба варианты, а FP32 по умолчанию.
- Попробуйте сгруппировать отдельные задания вывода с помощью пакетов.
- Обратите внимание, что использование графического процессора приводит к единовременным накладным расходам (порядка нескольких секунд) на компиляцию ядер OpenCL. Компиляция происходит при загрузке сети в плагин GPU и не влияет на время вывода.
- Если ваше приложение одновременно использует вывод на ЦП или иным образом сильно загружает хост, убедитесь, что потоки драйверов OpenCL не простаивают. Можно использовать параметры конфигурации процессора, чтобы ограничить количество потоков вывода для плагина процессора.
- В сценарии только для графического процессора драйвер графического процессора может занимать ядро процессора с циклическим

опросом для завершения. Если вас беспокоит загрузка ЦП, рассмотрите KEY_CLDND_PLUGIN_THROTTLE вариант конфигурации.

Обратите внимание, что при отключении опроса этот параметр может снизить производительность GPU, поэтому обычно этот параметр используется с несколькими Потоки GPU.

FPGA

Ниже перечислены наиболее важные советы по эффективному использованию FPGA:

- Точно так же, как для разновидностей VPU Intel Movidius Myriad, для FPGA важно скрыть накладные расходы на коммуникацию, выполняя несколько запросов вывода параллельно.
- Так как первая итерация вывода с FPGA всегда значительно медленнее, чем последующие, убедитесь, что вы выполняете несколько итераций.
- Производительность FPGA сильно зависит от битового потока.
- Количество запросов вывода на исполняемую сеть ограничено пятью, поэтому “канальный” параллелизм (сохранение отдельного запроса вывода на вход камеры/видео) не будет работать за пределами пяти входов. Вместо этого вам нужно добавить входные данные в некоторую очередь, которая будет внутренне использовать пул (5) запросов.
- В большинстве сценариев ускорение FPGA достигается за счет гетерогенного выполнения.
- Для выполнения FPGA с несколькими устройствами обратитесь к документации плагина FPGA.

2.5.2 Гетерогенность

Гетерогенное выполнение (состоящее из выделенного плагина Inference Engine “Hetero”) позволяет планировать сетевой вывод на несколько устройств.

Основными случаями для выполнения сети в гетерогенном режиме являются следующие:

1. Вычислите самые тяжелые части сети с ускорителем, возвращаясь к процессору для слоев, которые не поддерживаются ускорителем.

Это особенно полезно, когда определенные пользовательские ядра реализуются только для процессора (и гораздо сложнее или даже невозможно реализовать для ускорителя).

2. Необходимо использовать все доступные вычислительные устройства более эффективно, например, запустив ветви сети на разных устройствах.

Выполнение через гетерогенный плагин состоит из трех различных этапов:

3. Применение настройки сходства для слоев, то есть привязка их к устройствам.

- Это может быть сделано автоматически с использованием резервных приоритетов или на основе каждого уровня.
- Настройка сходства производится перед загрузкой сети в (гетерогенный) плагин, так что это всегда статическая настройка по отношению к исполнению.

4. Загрузка сети в гетерогенный плагин, который внутренне разбивает сеть на подграфы.

5. Выполнение запросов infer. Со стороны пользователя это выглядит идентично случаю с одним устройством, в то время как внутренне подграфы выполняются фактическими плагинами/устройствами.

Преимущества гетерогенного исполнения в значительной степени зависят от детализации коммуникаций между устройствами. Если передача/преобразование данных с одного устройства части на другое занимает больше времени, чем выполнение, гетерогенный подход имеет мало

смысла или вообще не имеет смысла. Использование Intel VTune помогает визуализировать поток выполнения на временной шкале.

Точно так же, если подграфов слишком много, синхронизация и передача данных могут поглотить всю производительность. В некоторых случаях можно определить (более грубое) сродство вручную, чтобы избежать многократной отправки данных туда и обратно во время одного вывода.

Общее “эмпирическое правило” сродства состоит в том, чтобы держать вычислительно интенсивные ядра на ускорителе, а "склеванные" или вспомогательные ядра на процессоре. Обратите внимание, что это включает в себя соображения детализации. Например, запуск какой-либо пользовательской активации (которая происходит после каждой свертки с ускорителем) на процессоре может привести к снижению производительности из-за слишком большого количества преобразований типов данных и/или макетов, даже если сама активация может быть чрезвычайно быстрой. В этом случае, возможно, имеет смысл рассмотреть возможность реализации ядра для ускорителя. Преобразования обычно проявляются как выдающиеся (по сравнению с выполнением только на ЦП) записи "Переупорядочения".

FPGA

Поскольку ПЛИС рассматривается как ускоритель вывода, большинство проблем с производительностью связано с тем фактом, что из-за запасного варианта процессор все еще может использоваться довольно интенсивно.

Однако в большинстве случаев процессор выполняет только небольшие/легкие слои, например, постобработку (SoftMax в большинстве классификационных моделей или DetectionOutputв топологиях на основе SSD). В этом случае ограничение количества потоков процессора конфигурацией `KEY_CPU_THREADS_NUM` приведет к дальнейшему снижению загрузки процессора без значительного снижения общей производительности.

Кроме того, если вы все еще используете версию OpenVINO более раннюю, чем R1 2019, или если вы перекомпилировали Inference Engine с

помощью OpemMP (скажем, для обратной совместимости), то установка переменной KMP_BLOCKTIME окружения на что-то меньшее, чем стандартные 200 мс (мы предлагаем 1 мс), особенно полезна. Необходимо использовать KMP_BLOCKTIME=0 если подграф процессора мал.

GPU/CPU

Следующие советы приведены для того, чтобы дать общее руководство по оптимизации выполнения на GPU/CPU устройствах.

- Как правило, производительность графического процессора лучше на тяжелых слоях (например, свертках) и больших входных данных. Таким образом, если время вывода сети уже слишком мало (~1 мс времени выполнения), то использование графического процессора вряд ли даст толчок.
- Типичная стратегия для начала заключается в том, чтобы сначала протестировать сценарии только для CPU и GPU (-d CPU или -d GPU). Если есть определенные ядра, которые не поддерживаются графическим процессором, лучше всего попробовать тот HETERO:GPU, CPU, который автоматически применяет разделение по умолчанию (на основе поддержки слоев плагинов). Затем можно поиграть с ручными настройками связности (например, чтобы еще больше минимизировать количество подграфов).
- Общее сродство “эмпирическое правило” состоит в том, чтобы держать вычислительно-интенсивные ядра на ускорителе, а "слитые" (или вспомогательные) ядра на процессоре. Обратите внимание, что это включает в себя соображения детализации. Например, запуск некоторой (пользовательской) активации на процессоре приведет к слишком большому количеству конверсий.
- Рекомендуется провести анализ производительности, чтобы определить перегруженные ядра, которые должны быть первыми кандидатами на разгрузку. В то же время часто более эффективно разгрузить некоторую последовательность ядер разумного размера, а не отдельные ядра, чтобы

минимизировать планирование и другие накладные расходы во время выполнения.

- Обратите внимание, что графический процессор может быть занят другими задачами (например, рендерингом). Аналогично, процессор может отвечать за общие процедуры ОС и другие потоки приложений. Кроме того, высокая частота прерываний из-за многих подграфов может повысить частоту одного устройства и уменьшить частоту другого.
- На производительность устройства может повлиять динамическое масштабирование частоты. Например, запуск медленных слоев на обоих устройствах одновременно может в конечном итоге привести к тому, что одно или оба устройства прекратят использование технологии Intel Turbo Boost. Это может привести к снижению общей производительности, даже по сравнению со сценарием с одним устройством.
- Смешивание выполнения FP16(GPU) и FP32(CPU) приводит к преобразованиям и, следовательно, к проблемам производительности. Если вы видите много тяжелых (по сравнению с выполнением только на ЦП) переупорядочений, подумайте о реализации реальных ядер графических процессоров.

2.5.3 Пользовательские Kernels

Механизм вывода поддерживает пользовательские ядра CPU, GPU и VPU. Как правило, пользовательские ядра используются для быстрой реализации отсутствующих слоев для новых топологий. Не следует переопределять реализацию стандартных слоев, особенно на критическом слоях, например, сверток. Кроме того, переопределение существующих слоев может отключить некоторые существующие оптимизации производительности, такие как слияние.

Обычно проще начать с расширения процессора и переключиться на графический процессор после отладки с помощью пути процессора. Иногда, когда пользовательские слои находятся в самом конце конвейера, проще

реализовать их как обычную постобработку в приложении, не оборачивая их в виде ядер. Это особенно верно для ядер, которые плохо подходят для графического процессора, например, для сортировки выходных ограничивающих прямоугольников. Во многих случаях можно сделать такую постобработку на процессоре.

Существует множество случаев, когда последовательность пользовательских ядер может быть реализована как «супер» ядро, позволяющее экономить на доступе к данным.

Наконец, при гетерогенном исполнении можно выполнить подавляющее большинство интенсивных вычислений с помощью ускорителя и сохранить пользовательские части на процессоре. Компромисс заключается в детализации/стоимости связи между различными устройствами.

2.5.4 Подключение механизма вывода к приложениям

Для вывода на ЦП существует несколько вариантов привязки потоков:

- Опция привязки (по умолчанию) сопоставляет потоки с ядрами и лучше всего работает для статических/синтетических сценариев, таких как бенчмарки. Он использует только доступные ядра (определяемые через маску процесса) и связывает потоки циклическим способом.
- Привязка "NUMA" может работать лучше в реальных сценариях, оставляя гораздо больше места для планирования ОС. Основное ожидаемое использование – это конфликтующие сценарии, например, несколько приложений с большим объемом вывода (CPU), выполняемых одновременно на одной машине.

Если вы создаете конвейер уровня приложения с помощью сторонних компонентов, таких как GStreamer, общие рекомендации для машин NUMA следующие:

- По возможности необходимо использовать по крайней мере один экземпляр конвейера на узел NUMA:

- Закрепить весь экземпляр конвейера к определенному узлу NUMA на самом внешнем уровне (например, необходимо использовать Kubernetes и/или numactl команду с правильными настройками перед фактическими командами GStreamer).
- Отключить любое отдельное закрепление компонентов конвейера (например, установите CPU_BIND_THREADS в значение "NO").
- Ограничить каждый экземпляр по количеству потоков вывода. Необходимо использовать CPU_THREADS_NUM или другие средства (например, виртуализацию, Kubernetes и т.д.), чтобы избежать избыточной подписки.

- Если закрепление экземпляра/закрепление всего конвейера невозможно или нежелательно, нужно ослабить привязку потоков вывода только к "NUMA".
  - Это менее ограничительно по сравнению с закреплением потоков по умолчанию к ядрам, но позволяет избежать штрафов NUMA.

По умолчанию Inference Engine использует Intel TBB. Таким образом, вывод процессора OpenVINO использует тот же пул потоков, предоставленный TBB. Но в приложении есть и другие потоки, поэтому избыточная подписка возможна на уровне приложения.

Эмпирическое правило состоит в том, что нужно попытаться иметь общее количество активных потоков в вашем приложении, равное количеству ядер в вашей машине. Имейте в виду запасное ядро(ы), которое может также понадобиться драйверу OpenCL под плагином GPU.

Один конкретный обход для ограничения количества потоков для механизма вывода использует параметры конфигурации процессора.

Чтобы избежать дальнейшей избыточной подписки, необходимо использовать одну и ту же потоковую модель во всех модулях/библиотеках, которые использует ваше приложение. Обратите внимание, что сторонние компоненты могут иметь свою собственную резьбу. Например, использование механизма вывода, который теперь скомпилирован с TBB по умолчанию,

может привести к проблемам с производительностью при смешивании в одном приложении с другой вычислительной библиотекой, но скомпилированной с использованием OpenMP. Можно попробовать скомпилировать версия с открытым исходным кодом из механизма вывода, чтобы использовать OpenMP. Но обратите внимание, что в целом TBB предлагает гораздо лучшую композиционность, чем другие потоковые решения.

Если код (или сторонние библиотеки) использует GNU OpenMP, то Intel OpenMP (если вы перекомпилировали с его помощью Inference Engine) должен быть инициализирован первым. Это может быть достигнуто путем связывания вашего приложения с Intel OpenMP вместо GNU OpenMP или с помощью LD_PRELOAD на Linux OS.

Во многих случаях сеть ожидает предварительно обработанный образ, поэтому нужно убедиться, что вы не выполняете ненужные шаги в коде.

Оптимизатор модели может эффективно рассчитывать средние и нормализованные (масштабные) значения в модель (например, веса первой свертки).

Или если обычные 8-битные изображения на канал являются родными носителями информации (например, декодированные кадры), не следует конвертировать их в FP32, т.к. это то, что плагины могут ускорить. Необходимо использовать InferenceEngine::Precision::U8 в качестве входного формата.

Во многих случаях можно напрямую поделиться входными данными с механизмом вывода.

2.5.6 Асинхронный API механизма вывода

Асинхронный API механизма вывода может улучшить общую частоту кадров приложения. Пока акселератор занят выводом, приложение может продолжать делать что-то на хосте, а не ждать завершения вывода.

Однако есть важные ограничения по производительности: например, задачи, которые выполняются параллельно, должны стараться избегать избыточной подписки на общие вычислительные ресурсы. Если вывод выполняется на FPGA и процессор по существу простаивает, имеет смысл вычислять на CPU параллельно. Однако несколько запросов infer могут переопределить это. Гетерогенное выполнение может неявно использовать процессор, обратитесь к Гетерогенности.

Кроме того, если вывод выполняется на графическом процессоре (GPU), то для кодирования, например, результирующего видео, на том же GPU параллельно может потребоваться небольшое усиление, поскольку устройство уже занято.

Для процессора, лучшее решение использовать режим «пропускная способность» процессора. Если задержка вызывает больше беспокойства, можно попробовать опцию EXCLUSIVE_ASYNC_REQUESTS конфигурации, которая ограничивает количество одновременно выполняемых запросов для всех (исполняемых) сетей, которые совместно используют определенное устройство только одним.

Для FPGA и GPU фактическая работа сериализуется плагином и/или драйвером в любом случае.

Наконец, для любого вида VPU использование нескольких запросов является обязательным условием для достижения хорошей пропускной способности.

В системе логического вывода нет понятия приоритетов запросов. Он остается на стороне пользователя (например, не ставить в очередь запрос с низким приоритетом, пока не будет ждать другой более высокий приоритет). Для синхронизации между исполняемыми сетями (очередями) в коде приложения потребуется дополнительная логика.

2.6 DL Workbench

DL Workbench – это веб-графическая среда, которая позволяет легко использовать вышеперечисленные компоненты инструментария OpenVINO toolkit. Позволяет визуализировать, точно настраивать и сравнивать производительность моделей глубокого обучения на различных конфигурациях архитектуры.

Если есть достаточный опыт работы с нейронными сетями, DL Workbench предоставляет удобный веб-интерфейс пользовательского интерфейса для оптимизации вашей модели и подготовки ее к производству.

Интуитивно понятный веб-интерфейс DL Workbench позволяет легко использовать различные компоненты OpenVINO toolkit:

- Загрузчик моделей для загрузки моделей из Intel Open Model Zoo с предварительно подготовленными моделями для различных задач
- Оптимизатор моделей для преобразования моделей в формат промежуточного представления (IR)
- Инструментарий оптимизации после обучения для калибровки модели, а затем выполнения ее в точности INT8
- Проверка точности для определения точности модели
- Инструмент бенчмарка для оценки производительности глубокого обучения вывода на поддерживаемых устройствах

Кроме того, DL Workbench предоставляет Jupyter Playground учебные пособия по использованию OpenVINO, его Python API и его компонентов, которые помогают анализировать и оптимизировать модели. Игровая площадка позволяет быстро начать работу с OpenVINO в предварительно настроенной среде.

DL Workbench поддерживает несколько расширенных сценариев профилирования:

1. Параллельные асинхронные запросы вывода (потоки для процессоров) и пакетная настройка целевого устройства

2. INT8 калибровка модели
3. Виноградская алгоритмическая настройка

3. ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ INTEL OPENVINO TOOLKIT

В данном разделе будет описан весь процесс по оптимизации модели и ускорение ее под процессоры Intel при помощи инструментария OpenVino.

Как говорилось в предыдущих разделах самый простой процесс при работе с OpenVino заключает в себе три этапа:

1. Получить обученную модель.
2. Пропустить обученную модель через оптимизатор модели, чтобы преобразовать модель в IR, который состоит из пары файлов .xml и .bin, которые используются для механизма вывода.
3. Воспользоваться API-интерфейс Inference Engine в приложении, чтобы выполнить вывод для IR (оптимизированной модели) и выполнить вывода результата.

3.1 Обученная модель

При использовании OpenVino предполагается, что уже существует некая обученная модель, работу которой и необходимо ускорить.

Инструментарий OpenVino может работать только с определенными наиболее популярными фреймворками, в которые входят:

1. Caffe
2. TensorFlow 1
3. TensorFlow 2
4. MXNet
5. Kaldi
6. ONNX

OpenVINO toolkit предоставляет набор предварительно обученных моделей, которые можно использовать для обучения и демонстрации или для разработки программного обеспечения глубокого обучения.

Для этого достаточно воспользоваться специальным скриптом «downloader.py». Он загружает файлы моделей из сетевых источников и, при необходимости, исправляет их, чтобы сделать их более удобными для использования с Model Optimizer. Пример:

- Для получения всего списка доступных моделей:

./downloader.py --print_all

- Для загрузки определенной модели:

/downloader.py --name face-detection-retail-0004

3.1.1 Примечание

Для определенных фреймворков необходимо провезти дополнительные манипуляции прежде чем их оптимизировать и конвертировать.

TensorFlow

Когда нейронная сеть определена в Python, необходимо создать файл графа вывода. Обычно графы строятся в форме, позволяющей моделировать обучение. Это означает, что все обучаемые параметры представлены в виде переменных на графе. Чтобы иметь возможность использовать такой граф с оптимизатором модели, такой граф должен быть заморожен. Пример кода для замораживания графа и сбрасывания его в файл:

```
      import tensorflow as tf
      from tensorflow.python.framework import graph_io
      frozen = tf.graph_util.convert_variables_to_constants(sess,
sess.graph_def, ["name_of_the_output_node"])
      graph_io.write_graph(frozen, './', 'inference_graph.pb', as_text=False)
```

Где:

- sess – является экземпляром объекта сеанса TensorFlow, в котором определена топология сети.
- ["name_of_the_output_node"] – является ли список имен выходных узлов в графе. frozen граф будет включать только те узлы из оригинала sess.graph_def которые прямо или косвенно используются для вычисления заданных выходных узлов.

- ./ – является каталогом, в котором должен быть создан файл графа вывода.
- inference_graph.pb – имя сгенерированного графа вывода.
- as_text – указывает, должен ли создаваемый файл быть в удобочитаемом текстовом формате или двоичном.

3.2 Оптимизатор модели

Использование оптимизатора необходимо перед использованием механизма вывода, т.к. как говорилось выше он создает промежуточное представление, с которым работает Inference Engine.

3.2.1 Настройка оптимизатора модели

Необходимо настроить Оптимизатор модели для фреймворка, который использовался для обучения модели.

Можно настроить все фреймворки одновременно, либо установить зависимости для отдельного фреймворка. Скрипты, поставляемые вместе с инструментом, устанавливают все необходимые зависимости и обеспечивают самый быстрый и простой способ настройки оптимизатора модели:

- install_prerequisites.sh – настройка для всех фреймворков
- install_prerequisites_caffe.sh – настройка для всех Caffe
- install_prerequisites_tf.sh – настройка для всех Caffe
- install_prerequisites_tf2.sh – настройка для всех Caffe
- install_prerequisites_mxnet.sh – настройка для всех MXNet
- install_prerequisites_kaldi.sh – настройка для всех Kaldi
- install_prerequisites_onnx.sh – настройка для всех ONNX

3.2.2 Преобразование модели

В качестве основной утилиты для преобразования модели используется mo.py (<INSTALL_DIR>/deployment_tools/model_optimizer). Она запускает оптимизатор модели и преобразует модель в промежуточное представление (IR). Пример простого использования оптимизатора:

```
python3 mo.py --input_model INPUT_MODEL
```

mo.py – это универсальная точка входа, которая может работать с любым фреймворком, используя входную модель, с помощью стандартного расширения файла модели:

1. .caffemodel - Модели Caffe
2. .pb - Модели TensorFlow
3. .params - Модели MXNet
4. .onnx - Модели ONNX
5. .nnet - Модели Kaldi.

Если файлы модели не имеют стандартных расширений, можно использовать аргумент --framework {tf, caffe, kaldi, onnx, mxnet} опцию для явного указания типа фреймворка.

Далее будут перечислены ряд рекомендаций для создания IR-представления.

Средние и масштабные значения

Обычно нейросетевые модели обучаются с нормализованными входными данными. Это означает, что значения входных данных преобразуются в определенный диапазон, например, [0, 1]или [-1, 1]. Иногда средние значения (средние изображения) вычитаются из значений входных данных в рамках предварительной обработки. Существует два случая реализации предварительной обработки входных данных:

1. Операции предварительной обработки входных данных являются частью топологии. В этом случае приложение, использующее фреймворк для вывода топологии, предварительно не обрабатывает входные данные.
2. Операции предварительной обработки входных данных не являются частью топологии, и предварительная обработка выполняется в приложении, которое снабжает модель входными данными.

В первом случае Оптимизатор модели генерирует IR с требуемыми слоями предварительной обработки, и для запуска модели может быть использован стандартный механизм вывода.

Во втором случае информация о средних/масштабных значениях должна быть предоставлена Оптимизатору модели для встраивания ее в сгенерированную IR-информацию. Оптимизатор модели предоставляет ряд параметров командной строки для их указания: --scale, --scale_values, --mean_values, --mean_file.

Если указаны как средние, так и масштабные значения, сначала вычитается среднее, а затем применяется масштаб. Входные значения делятся на значения шкалы.

Входная размерность

Бывают ситуации, когда размерность входных данных для модели не фиксирована, как для полностью сверточных нейронных сетей. В этом случае, например, модели TensorFlow содержат значения -1 в атрибуте shape Placeholder операции. Механизм вывода не поддерживает входные слои с неопределенным размером, поэтому, если входные формы не определены в модели, Оптимизатор модели не сможет преобразовать модель.

Решение состоит в том, чтобы предоставить входную размерность с помощью параметра --input --input_shape или предоставить размер пакета с помощью параметра командной строки -b, если модель содержит только один вход с неопределенным размером пакета. В последнем случае Placeholder размерности модели TensorFlow выглядит следующим [-1, 224, 224, 3] образом.

Цветовые каналы

Входные данные для приложения могут иметь порядок ввода цвета RGB или BRG. Например, выборка механизма вывода загружают входные изображения в порядке каналов BGR. Однако модель может быть обучена на изображениях, загруженных с противоположным порядком (например, большинство моделей TensorFlow обучаются с изображениями в порядке

RGB). В этом случае результаты вывода с использованием образцов механизма вывода могут быть неверными. Решение состоит в том, чтобы обеспечить --reverse_input_channels параметр командной строки. Взяв этот параметр, Оптимизатор модели выполняет первую свертку или другую канально-зависимую операцию модификации весов, так что выходные данные этих операций будут похожи на то, как изображение передается с порядком каналов RGB.

Прочее

Далее перечислены иные аргументы утилиты mo.py, которые могут регулировать различные аспекты конвертации и оптимизации:

1. --model_name – будущее название нейронной сети в IR-представлении;
2. --output_dir – директория, в которую будетзапиано IR-представление;
3. --log_level – уровень логирования;
4. --output – имя выходной операции модели;
5. --data_type - тип данных для всех промежуточных тензоров и весов, включает FP16, FP32, half, float;
6. --disable_fusing – отключите слияние линейных операций с сверткой;
7. --disable_resnet_optimization – отключить оптимизацию resnet;
8. --extensions – расширения оптимизатора модели;
9. --silent – отключить вывод всех сообщений кроме уровня Error;
10. --disable_weights_compression – отключить сжатие и сохранить оригинальную точность весов;
11. --progress – включить отображение хода преобразования модели;
12. --stream_output – переключите отображение хода преобразования модели в многострочный режим.
13. --transformations_config – необходимо использовать файл конфигурации с описанием преобразований.

Для конкретных фреймворков существуют дополнительные аргументы, позволяющие настраивать специфичные для фреймворка поведения.

3.3 Demo приложение

Для демонстрации эффективности использования OpenVINO было выбрано приложение, основной функцией которой является распознавание объектов на изображении. Данная задача является одной из самых актуальных и чьи решения являются очень чувствительными к вычислительным ресурсам. По этой причине демонстрация является наиболее представительной.

Для распознавания в приложении будет использоваться модель YOLO 3 [https://github.com/mystic123/tensorflow-yolo-v3] обученная на фреймворке TenserFlow. Модель показывает отличную точность и способна распознавать до 80 различных объектов.

В качестве сравнительно характеристики будет использоваться время на обработку одного изображения(кадра), изменения в точности не учитываются так как архитектурно модель не будет изменена.

3.5.1 Процесс преобразования и интеграции в приложение

Так как в качестве экспериментального фреймворка был выбран TensorFlow, то прежде чем преобразовывать модель в IR-формат необходимо заморозить модель. Для этого воспользуемся скриптом convert_weights_pb.py, в котором реализован код по замораживанию существующей модели. Сам код был описан в предыдущих главах. После чего генерируется файл с замороженной моделью:

frozen_darknet_yolov3_model.pb

Теперь можно переходить к преобразованию в IR-представление. Для этого необходимо воспользоваться оптимизатором моделей при помощи скрипта mo.py:

```
python mo_tf.py
    --input_model .\frozen_darknet_yolov3_model.pb
    --transformations_config .\yolo_v3.json
    --batch 1
```

Где yolo_v3.json – это вспомогательный файл для трансформации слоев, он поставляется вместе с инструментарием OpneVINO.

В конечном итоге оптимизатор модели генерирует два файла:

- frozen_darknet_yolov3_model.bin
- frozen_darknet_yolov3_model.xml

Это и есть IR-представление, которое можно использовать в приложении.

Сама интеграция IR-модели может происходит в несколько этапов (рисунок 26). Но в приложении буде представлена укороченная версия.

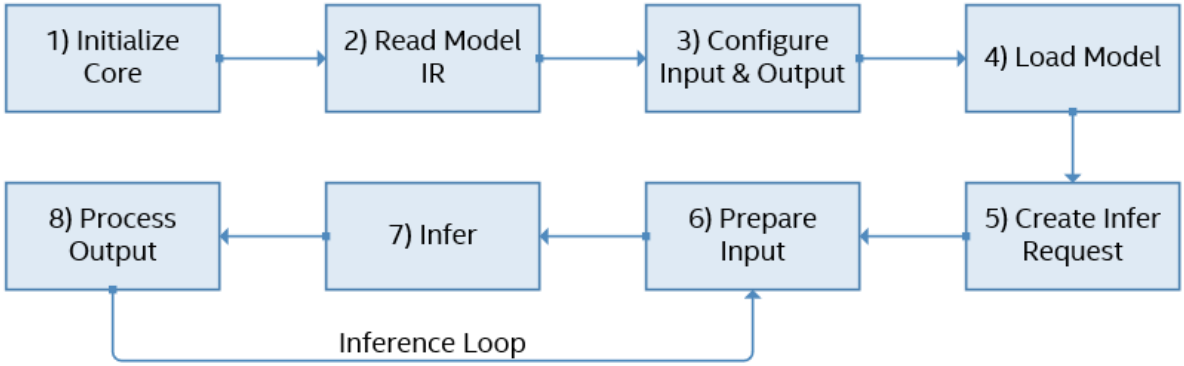


Рисунок 26 – Этапы интеграции IR

Полный код программы предложен в Приложении 3. Ниже будут приведены отдельные участки кода из реализации приложения, использующего OpenVINO.

1. Инициализация ядра

```
ie = IECore()
```

2. Чтение модели

```
model_name = 'frozen_darknet_yolov3_model'
net = ie.read_network(model_name + '.xml', model_name + ".bin")
```

Для чтения подходят IR-модели, созданные оптимизатором модели (поддерживается формат .xml) или ONNX модели.

3. Настройка ввода/вывода.

Получить информацию о входных и выходных слоях можно получить из свойств загруженной нейронной сети:

```
net.input_info
net.outputs
```

При необходимости можно установить числовой формат (точность) и структуру для входов и выходов.

Также можно разрешить ввод любого размера. Для этого необходимо отметить каждый вход как изменяемый, установив желаемый алгоритм изменения размера внутри соответствующей входной информации.

Также на данном этапе можно производить базовые преобразования цветовых форматов. По умолчанию Inference Engine предполагает, что входной цветовой формат является BGR и преобразования цветовых форматов отключены.

Если вы пропустите этот шаг, будут установлены значения по умолчанию:

- для входных данных не задан алгоритм изменения размера
- входной цветовой формат – RAW это означает, что входные данные не нуждаются в преобразовании цвета
- точность ввода и вывода – FP32
- макет ввода – NCHW

4. Загрузите модель

```
exec_net = ie.load_network(network=net, num_requests=2, device_name='CPU')
```

Создается исполняемая сеть из сетевого объекта. Исполняемая сеть связана с одним аппаратным устройством.

Можно создавать столько сетей, сколько необходимо, и использовать их одновременно (вплоть до ограничения аппаратных ресурсов).

Четвертый параметр – это конфигурация плагина. Это карта пар: (имя параметра, значение параметра).

5. Асинхронный запуск модели

```
exec_net.start_async(request_id=request_id, inputs={'inputs': in_frame})
```

StartAsync возвращает немедленно и запускает вывод без блокировки основного потока, Inferблокирует основной поток и возвращается, когда

вывод завершен. Необходимо вызвать Wait, чтобы он стал доступен для асинхронного запроса.

Есть три способа его использования:

- указать максимальную продолжительность блокировки в миллисекундах. Метод блокируется до истечения указанного тайм-аута или до получения результата, в зависимости от того, что наступит раньше.
- RESULT_READY - ждет, пока результат вывода не станет доступен
- STATUS_ONLY - немедленно возвращает статус запроса. Он не блокирует и не прерывает текущий поток.

Несколько запросов для одного ExecutableNetwork выполняются последовательно один за другим в порядке FIFO.

6. Получения результата

Для этого необходимо пройтись по все Blob:

```
output = exec_net.requests[cur_request_id].outputs
for layer_name, out_blob in output.items():
      pass
```

3.5.3 Результат

После всех манипуляций можно сравнить производительность приложений с использованием исходной модели и модели запущенной с использованием библиотеки OpenVINO.

В качестве входного изображения выступает фото городской улицы:

Рисунок 26 – Исходное изображение для детектирования

Исходная модель производит детектирование объектов на изображение в среднем за 11 с.

В это время приложение, которое использует максимально эффективно ресурсы процессора при помощи библиотеки OpenVino, детектирует объекты на изображении в среднем за 1 с.

Очевидно, что результат на лицо. Использование OpenVINO позволяет нам сильно ускорить производительность приложения за счет более эффективного использования инструкции процессора. Кроме этого инструмент позволяет из коробки производить асинхронные вычисления, что так же влияет на производительность.

Более тонкое конфигурирование и оценка эффективности будет произведена в следующем разделе.

4. АНАЛИЗ ЭФФЕКТИВНОСТИ OPENVINO TOOLKIT

Для более полного понимания, на сколько эффективно OpenVINO справляется с оптимизацией моделей и ускорением их на целевых вычислительных устройствах компании Intel необходимо протестировать несколько моделей разных фреймворков.

Испытуемые модели и их характеристики представлены в таблице:

Таблица 1 – Модели и характеристики для анализа эффективности

Названия моделей	Фреймворк	Входной слой	Общее количество нейронов	Flop
frozen_darknet_yolov3_model	TensorFlow	(1,224,224,3)	$25{,}533\times10^{6}$	$8{,}22\times10^{9}$
yolo-v2-tf	TensorFlow	(1,416,416,3)	$64{,}923\times10^{6}$	$65{,}99\times10^{9}$
squeezenet1.1	Caffe	(1,3,227,227)	$1{,}236\times10^{6}$	$0{,}78\times10^{9}$
resnet-50-tf	TensorFlow	(1,608,608,3)	$50{,}953\times10^{6}$	$63{,}06\times10^{9}$
mobilenet-ssd	Caffe	(1,224,224,3)	$5{,}783\times10^{6}$	2.325×10^{9}
googlenet-v1-tf	TensorFlow	(1,224,224,3)	$6{,}62\times10^{6}$	$3{,}02\times10^{9}$

Их объединяет то, что они предназначены для работы с изображениями. Это даст более однозначную оценку эффективности инструмента, так как решают одну задачу и подвержены одним факторам.

В качестве входных данных будет выступать не полный набор изображений COCO, который включает в себя 5000 фотографий различных объектов.

В работе используется два параметра для измерений, которые являются ключевыми элементами для успешного применения систем с глубоким обучением:

- Пропускная способность – измеряет количество предсказаний, сделанных в пределах порога задержки (например, количество кадров в секунду – FPS). Данная метрика лучше всего подходит для систем, основанных на потоковой обработке данных в реальном времени, к примеру видеопоток. В таких системах как правило реализованы параллельная или асинхронная обработка.
- Задержка – это синхронная обработка запроса в миллисекундах. Каждый запрос вывода (например, preprocess, infer, postprocess) можно завершить только до запуска следующего. Эта метрика

производительности актуальна в сценариях использования, где требуется как можно скорее обработать одно входное изображения. избегания препятствий для автономных транспортных средств.

Точность предсказания моделей не имеет смысла сравнивать, так как влияние OpenVino на топологию нейронной сети и на ее веса незначительны или его вовсе отсутствует.

Также стоит подчеркнуть, что пропускная способность подразумевает использование асинхронной обработки данных. По этой причине FPS, не смотря на рост задержки, также может расти.

Эксперимент будет проходить на компьютере со следующими характеристиками:

Таблица 2 – Характеристики компьютера

Тип ЦП	Mobile QuadCore Intel Core i7-4710HQ, 3233 MHz (33 x 98)
Системная плата	MSI MegaBook GE60 2PL (MS-16GH)
Чипсет системной платы	Intel Lynx Point HM86, Intel Haswell
DIMM1	Kingston MSI16D3LS1KBG/8G 8 ГБ DDR3-1600 DDR3 SDRAM (11-11-11-28 @ 800 МГц) (10-10-10-27 @ 761 МГц) (9-9-9-24 @ 685 МГц) (8-8-8-22 @ 609 МГц) (7-7-7-19 @ 533 МГц) (6-6-6-16 @ 457 МГц) (5-5-5-14 @ 380 МГц)
DIMM3	AMD R538G1601S2SL
Тип BIOS	AMI (07/19/2014)
GPU	Intel HD Graphics 4600
GPU	NVIDIA GeForce GTX 850M

Графический процессор Intel HD Graphics версии 4600 не поддерживается OpenVino и по этой причине не будет участвовать в

эксперименте. NVIDIA GeForce GTX 850M так же не поддерживается, так как не является продуктом компании Intel.

Для более эффективной работы в исследовании будет применяться программное обеспечение от OpenVino Workbench, так как оно дает широкий функционал по настройке оптимизации модели по сравнению результат между другими моделями.

4.1 Результаты

Прежде чем переходить к анализу стоит отметить, что измерение производительности включает в себя множество переменных и чрезвычайно сильно зависит от конкретного случая использования и приложения.

В анализе производилось сравнение между моделями, оптимизированными под разный формат нейронных сетей (INT8, FP32) и оригиналом. На графике 1 и 2 представлены время обработки одной фотографии моделью.

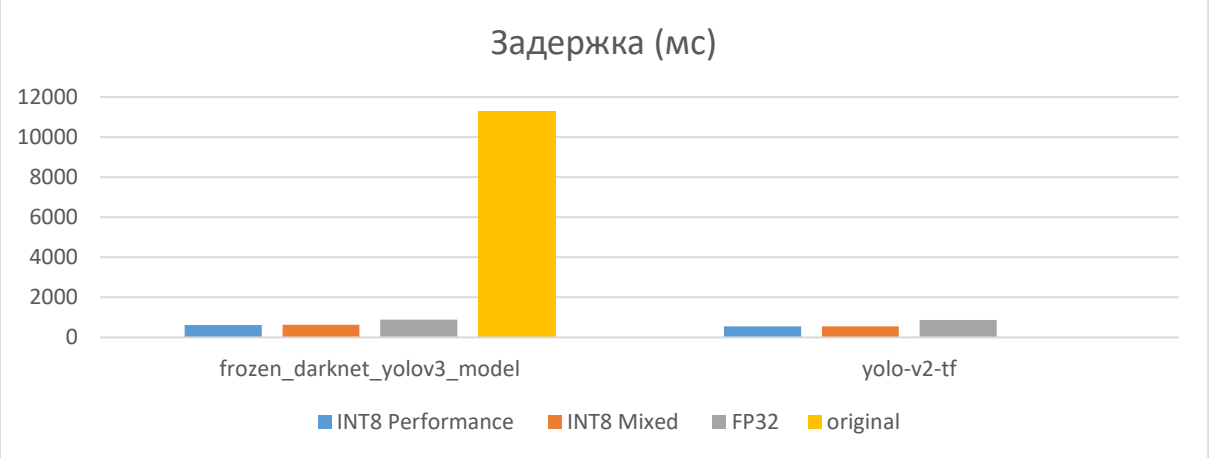


Рисунок 27 – График задержки frozen_darknet_yolov3_model и yolo-v2-tf

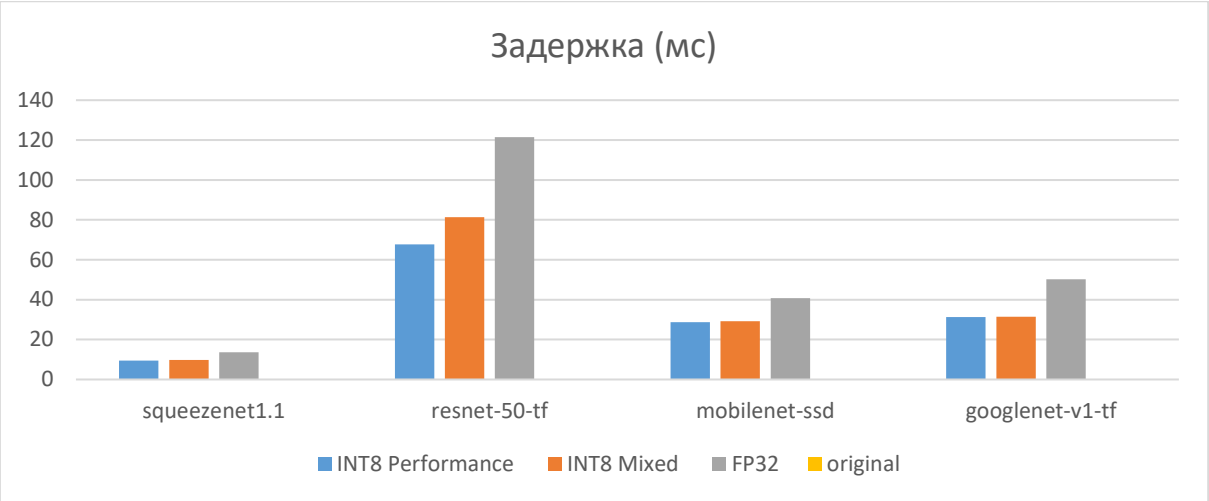


Рисунок 28 – График задержки squeezenet1.1, resnet-50-tf, mobilenet-ssd, googlenet-v1-tf

Если рассматривать с точки зрения пропускной способности, то график примет другую форму.

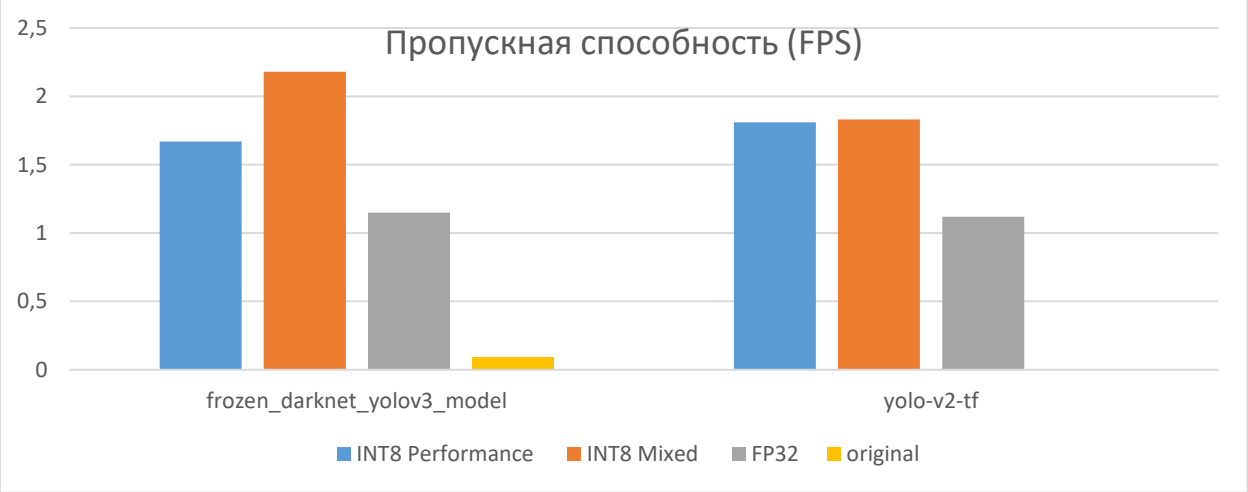


Рисунок 29 – График пропускной способности frozen_darknet_yolov3_model и yolo-v2-tf

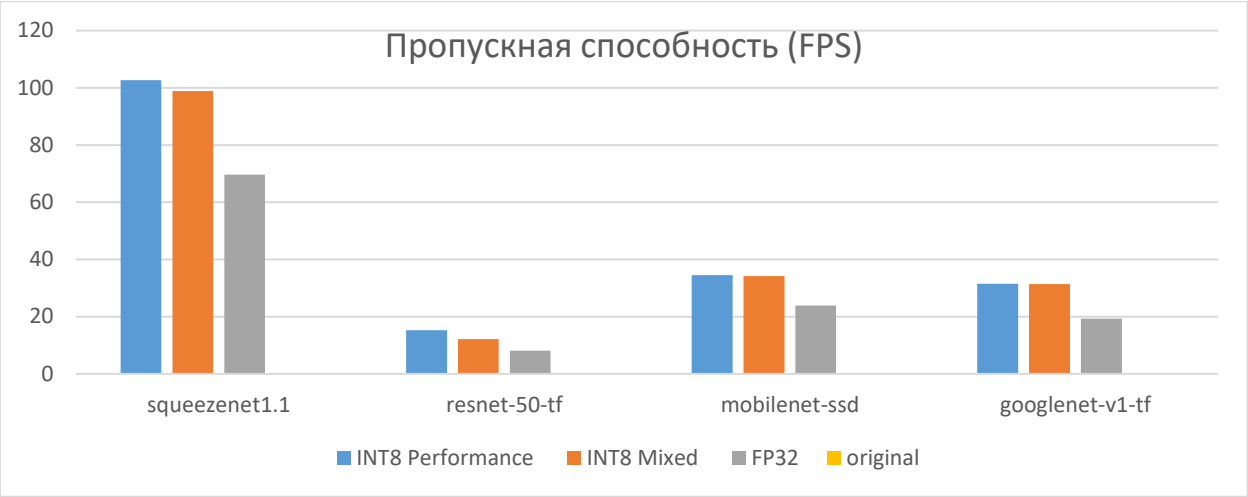


Рисунок 30 – График пропускной способности squeezenet1.1, resnet-50-tf, mobilenet-ssd, googlenet-v1-tf

Из графиков видно, что производительность моделей при использовании OpenVino даже без дополнительных оптимизаций сама по себе очень хороша. Но при использовании формата пониженной точности на некоторых моделях производительность увеличивается еще больше. И это ожидаемо, ведь теперь процессор может хранить большее инструкций в кэше.

Также как указано на графике в исследованиях использовалось два вида понижения точности: INT8 Performance и INT8 Mixed.

INT8 Performance – означает то, что как веса, так и активации калибруются в симметричном режиме. Эта предустановка является опцией по умолчанию, поскольку она обеспечивает максимальное ускорение производительности и не зависит от целевой платформы.

INT8 Mixed – означает, что веса и активации калибруются соответственно в симметричном и асимметричном режимах. Данный вид может привести к более точной модели за счет снижения производительности. В зависимости от целевой платформы и модели падение производительности обычно колеблется от 5 до 15%. Например, необходимо использовать эту предустановку, если модель имеет сверточные или полностью связанные слои как с отрицательными, так и с положительными активациями, например, модель с активациями без ReLU.

Помимо этого, оптимизировать можно и другие параметры модели, такие как размер пакета и количество потоков.

Для модели frozen_darknet_yolov3_model в формате пониженной точности INT8 Mixed были проведены такие оптимизации. На рисунках 31 и 32 показана зависимость задержки и пропускной способности в зависимости от размера пакета.

Рисунок 31 – Зависимость задержки от размера пакета

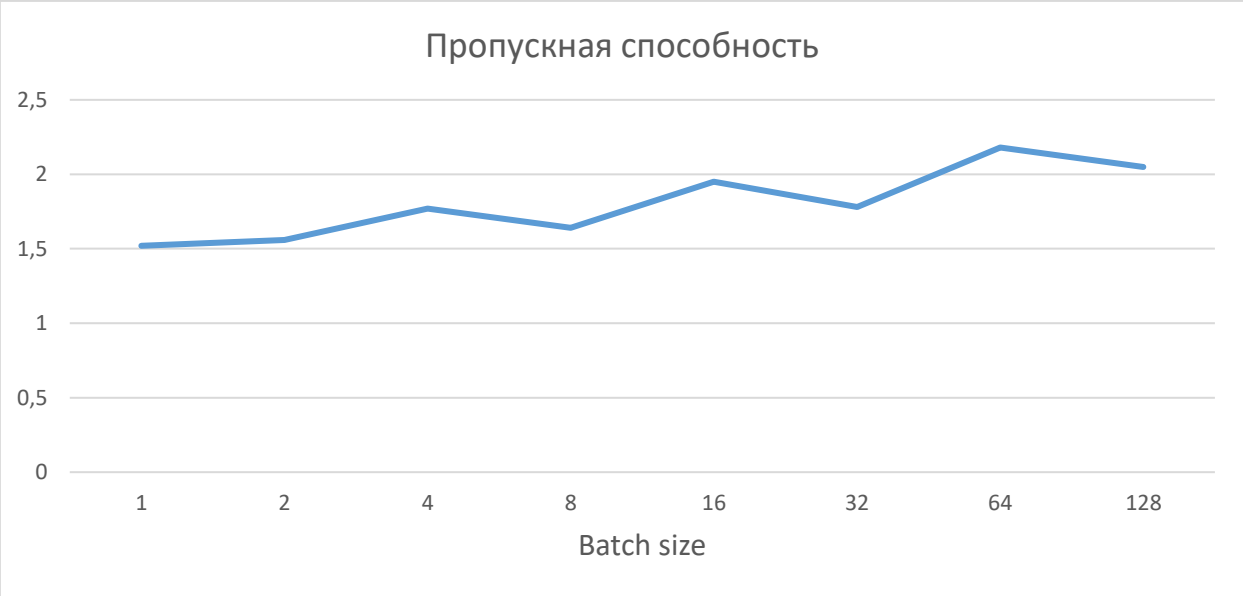


Рисунок 32 – Зависимость пропускной способности от размера пакета

Как можно видеть на рисунке 31 задержка растет линейно с ростом размера пакета. В это время пропускная способность также возрастает.

На следующих рисунках 33 и 34 показана зависимость характеристик от количества потоков. Стоит отметить, что размер пакета при этом равен 1.

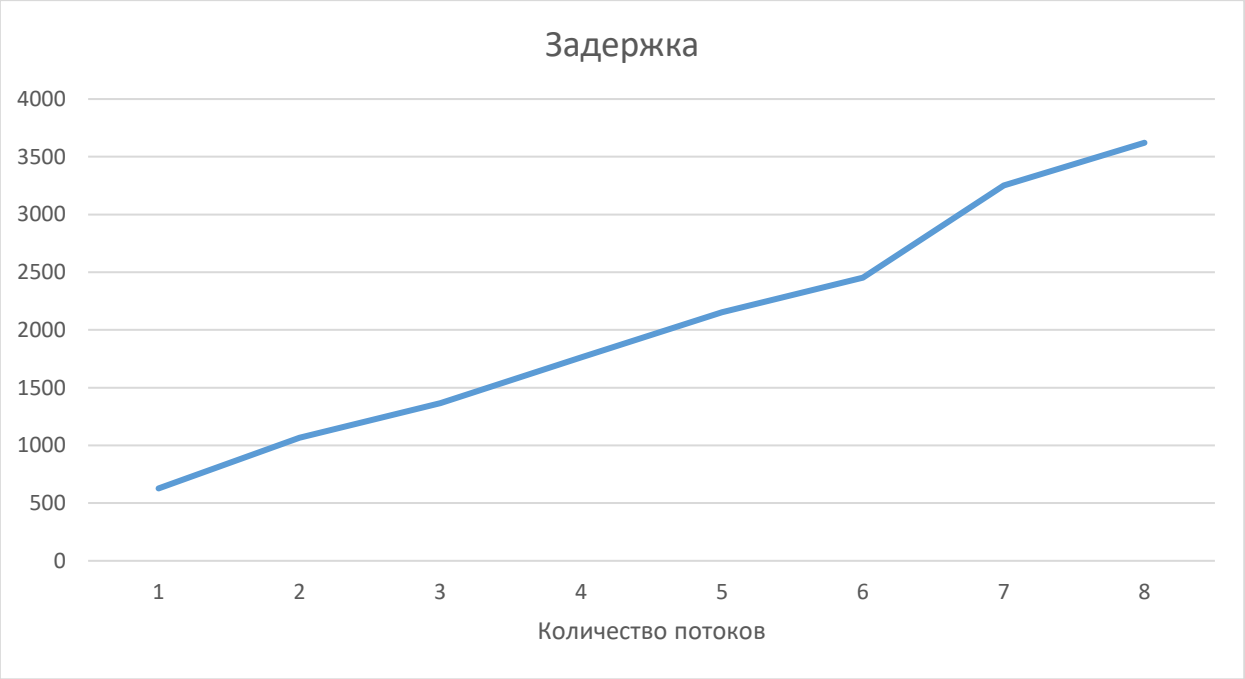


Рисунок 33 – Зависимость задержки от количества потоков

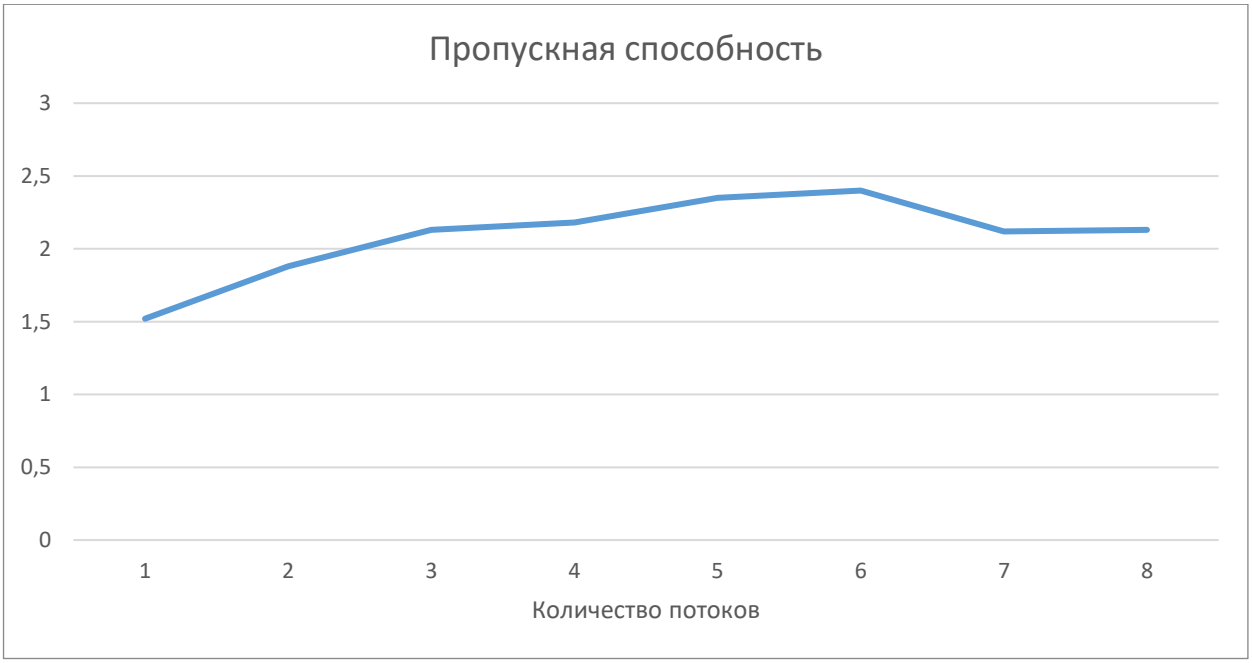


Рисунок 34 – Зависимость пропускной способности от количества потоков

Так же, как и на графике с размерами пакета характер зависимости от количества потоков также линейный и возрастает. Но пропускная способность после 6 потоков перестает давать прирост и на оборот начинает падать.

Представленные выше графики позволяют нам сделать вывод, что OpenVino предоставляет отличный функционал по ускорению работы моделей, основанных на сверточных нейронных сетях.

5. ОЦЕНКА И ЗАЩИТА РЕЗУЛЬТАТОВ ИНТЕЛЛЕКТУАЛЬНОЙ ДЕЯТЕЛЬНОСТИ

5.1 Описание результатов интеллектуальной деятельности

Данный подход предполагает использование библиотеки Intel OpenVino Toolkit с открытым исходным кодом. Использование библиотеки позволяет увеличить производительность систем искусственного интеллекта основанных на глубоких нейронных сетях.

Вследствие этого можно утверждать, что данный исследуемый объект интеллектуальной собственности (ОИС) является секретом производства (ноу-хау), на который в свою очередь в соответствии с п. 1 статьи 1465 главы 75 части 4 Гражданского кодекса Российской Федерации распространяются авторские права.

5.2 Оценка рыночной стоимости результатов интеллектуальной деятельности

Так как применение технологии OpenVino распространяется только на сверточные нейронные сети, а они в свою очередь являются основным компонентом в системах компьютерного зрения, то основным сегментом рынка является Computer Vision.

Рынок компьютерного зрения сегментируется по компонентной, программной, продуктовой, прикладной вертикали. Рост среди сегментов помогает анализировать нишевые очаги роста и стратегии, чтобы приблизиться к рынку и определить основные области применения и разницу на целевых рынках.

- По компонентному составу рынок компьютерного зрения сегментируется на аппаратное обеспечение, камеры, частоту кадров, формат, стандарт, сенсор, захваты кадров, оптику, светодиодное освещение, процессоры, ПЛИС, ЦСП, микроконтроллеры и микропроцессоры, блоки визуальной обработки и другие.

- Основываясь на программном обеспечении, рынок компьютерного зрения сегментируется на традиционное программное обеспечение и программное обеспечение для глубокого обучения.
- В зависимости от продукта рынок компьютерного зрения сегментируется на ПК-и смарт-камеры.
- Основываясь на вертикали, рынок компьютерного зрения делится на промышленный и непромышленный. Промышленный сегмент далее подразделяется на автомобильный, электронный и полупроводниковый, потребительский, стекло, металлы, дерево и бумагу, фармацевтические препараты, пищевая промышленность и упаковка, резина и пластмассы, полиграфия, машиностроение, производство солнечных панелей и текстиль. Непромышленный сегмент далее подразделяется на здравоохранение, почту и логистику, интеллектуальные транспортные системы безопасности и наблюдения, сельское хозяйство, потребительскую электронику, автономные и полуавтономные транспортные средства, спорт и развлечения и розничную торговлю.

Аналитический центр Tadviser совместно с компанией «Системы компьютерного зрения» (входит в группу «Ланит») провел исследование российского рынка компьютерного зрения. По итогам 2018 г. объем этого рынка составил 8 млрд руб., а концу 2023 г. он может увеличиться в 4,8 раза.

С появлением глубокого обучения технологии, системы компьютерного зрения стали более способны воспроизводить человеческое зрение. Например, в здравоохранении искусственный интеллект используется в визуализации черепа для мониторинга острых неврологических событий и выявления неврологических заболеваний. Диагностика заболеваний в здравоохранении, биометрический анализ в области безопасности, контроль качества в производстве, обнаружение препятствий и управление автономными транспортными средствами на транспорте, среди прочего, являются важными

областями применения компьютерного зрения в различных отраслях промышленности.

Аппаратная составляющая лидировала на рынке и составила более 70,0% мировой выручки в 2019 году. Высокая доля объясняется наличием новейших аппаратных платформ, обеспечивающих удобное соединение компонентов и расширенные возможности, такие как быстрая обработка, много-мегапиксельное разрешение и полностью цифровая обработка данных. Аппаратные компоненты системы компьютерного зрения включают процессоры и сопроцессоры, камеры, осветительные и оптические компоненты, а также захваты кадров. Кроме того, высокопроизводительные аппаратные компоненты облегчили установку зрительных систем и служат широкому спектру разнообразных приложений, предлагая различные сетевые архитектуры.

Кроме того, ожидается, что сегмент программного обеспечения продемонстрирует заметный сдвиг в своем спросе, зарегистрировав средний показатель 8,6% за прогнозируемый период. Основные задачи, выполняемые программным обеспечением компьютерного зрения, включают классификацию изображений, обнаружение объектов, отслеживание объектов и извлечение изображений на основе содержимого. Однако многим организациям не хватает ресурсов и вычислительной мощности для обработки огромного количества визуальных данных, что может затруднить рынок программного обеспечения для приложений компьютерного зрения. Несколько технологических гигантов работают над устранением ограничений для разработки программного обеспечения компьютерного зрения. Например, корпорация IBM помогает компаниям разрабатывать приложения для компьютерного зрения, предлагая услуги по разработке программного обеспечения для компьютерного зрения. Эти сервисы предоставляют готовые модели глубокого обучения, доступные из облачного источника, для решения проблем как вычислительных ресурсов, так и развития.

Сегмент обеспечения качества и инспекции лидировал на рынке и в 2019 году составил более 28,0% мировой выручки. Эта высокая доля объясняется быстрым внедрением автоматизации технологических процессов в обрабатывающей промышленности для повышения производительности труда. Компьютерное зрение в сочетании с алгоритмами глубокого обучения позволяют автоматизировать контроль каждого продукта на производственной линии. Например, канадская компания Matrox, занимающаяся цифровой визуализацией, предоставляет решения и продукты для обеспечения качества и контроля производителям оборудования и интеграторам, занимающимся анализом изображений, машинным зрением и медицинскими визуализационными приложениями. Suntory PepsiCo Vietnam Beverage, компания по производству потребительских товаров, внедрила на своих заводах решение на основе изображений Matrox Imaging для оптимизации контроля качества кодовых этикеток.

Кроме того, ожидается значительный рост сегмента прогнозного технического обслуживания в течение прогнозируемого периода. Прогнозное техническое обслуживание сочетает в себе алгоритмы машинного обучения с устройствами Интернета вещей (IoT) для мониторинга данных о машинах и связанных с ними компонентах. Он часто использует датчики для сбора данных и идентификации сигналов, чтобы принимать точные решения до того, как произойдет поломка активов или компонентов. Например, Tesla, американская компания по производству электромобилей и экологически чистой энергии, использует ультразвуковые датчики для самостоятельного вождения и предотвращения несчастных случаев в своих автономных автомобилях.

Продавцы на рынке сосредоточены на увеличении клиентской базы, чтобы получить конкурентное преимущество в отрасли. Поэтому поставщики предпринимают ряд стратегических инициатив, таких как партнерство, поглощения и слияния, а также сотрудничество с другими ключевыми игроками отрасли. Например, в апреле 2020 года китайская компания

компьютерного зрения Clobotics приобрела AtSite A/S, поставщика решений для визуального контроля. Clobotics выполнила эту сделку, чтобы разработать технологию контроля морских ветряных турбин с поддержкой искусственного интеллекта. Например, в январе 2020 года американская компания социальных сетей Snap Inc. завершила приобретение украинского стартапа компьютерного зрения под названием AI Factory. Предполагалось, что это приобретение поможет приложению социальных сетей, предлагаемому Snap Inc., добавить новую функцию под названием Cameos, которая превращает фотографию человека в короткое анимированное видео. Некоторые из выдающихся игроков на рынке компьютерного зрения включают в себя:

- Cognex Corporation
- Intel Corporation
- Keyence Corporation
- Matterport, Inc.
- National Instruments
- OMRON Corporation
- Sony Semiconductor Solutions Corporation
- Teledyne Digital Imaging Inc.
- Teledyne Optech
- Texas Instruments Incorporated

Перечисленные выше факты, предположения и анализ рынка в целом дают нам основания предполагать, что Computer Vision сейчас большой спрос, а в будущем данный сегмент станет одним из самых востребованных на рынке.

Так как на данный момент сложно определить доходная часть, так как для этого требуются реальные цифры с производства. Отсюда вытекает и ограничение по расчёту потенциальной выгоды от разработки и её реализации в будущем, однако можно предположить затраты, направленные на проведение НИР, разработку и изготовление устройства.

По этой причине имеет смысл для расчёта рыночной стоимости использовать затратный подход, в пределах которого будут использованы два метода подсчёта:

- метод удельных затрат.

Для расчета рыночной стоимости ОИС по методу удельных затрат необходимо знать среднюю годовую численность персонала, которая участвует в НИР и время проведения НИР.

В таблице 3 представлены сведения о затратах на заработную плату рабочим, в сумму которых помимо непосредственной зарплаты и налоговых вычетов также входят страховые взносы на социальное, медицинское и пенсионное отчисления. Данные взяты на основании информации, представленной федеральной службой государственной статистики.

Таблица 3 – Совокупность затрат на заработную плату рабочим

№ п/п	Работник	Затраты на заработную плату, руб.
1	Инженер (младший научный сотрудник)	50000
2	Инженер (младший научный сотрудник)	50000
3	Инженер (старший научный сотрудник)	80000
4	Руководитель проекта	120000

В соответствии с таблицей 3 среднегодовая заработная плата на одного рабочего будет рассчитана следующим образом:

$$ЗП_{ср} = \frac{\sum З_{pi}}{n} \cdot 12$$

Где $З_{pi}$ – затраты на заработную плату i-ому рабочему; n – общее количество работников.

$$ЗП_{ср} = \frac{50000 + 50000 + 80000 + 120000}{4} \cdot 12 = 900000 \text{ рублей}$$

Общая сумма затрат на заработную плату рабочим рассчитывается по формуле:

$$З_{ЗП} = Т_{НИР} \cdot Ч_{НИР} \cdot ЗП_{ср}$$

где ТНИР – продолжительность НИР, лет; ЧНИР – среднегодовая численность сотрудников НИР.

Учитывая, что среднегодовая численность сотрудников НИР не отличается от общего количества работников общая сумма затрат на зарплату составляет:

$$З_{ЗП} = 1 \cdot 4 \cdot 900000 = 3600000 \text{ рублей}$$

Итоговая стоимость НИР рассчитывается по следующей формуле:

$$З = \frac{З_{ЗП}}{q_{ЗП}}$$

где qзп – удельный вес заработной платы, который зависит от общего числа работников и среднегодовой численности сотрудников НИР.

В результате, учитывая, что при полной занятости сотрудников НИР удельный вес зарплаты $q_{ЗП}$= 1, итоговая стоимость НИР составляет 3600000 рублей.

5.3 Правовая защита результатов интеллектуальной деятельности

Предложенных подход на основании «статьи 1465 ГК РФ. Секрет производства (ноу-хау)» можно расценивать как «Секреты производства (ноу-хау)», так как не противоречит не одному из перечисленных критериев, указанных в статье:

1) Секретом производства (ноу-хау) признаются сведения любого характера (производственные, технические, экономические, организационные и другие) о результатах интеллектуальной деятельности в научно-технической сфере и о способах осуществления профессиональной деятельности, имеющие действительную или потенциальную коммерческую ценность вследствие неизвестности их третьим лицам, если к таким сведениям

у третьих лиц нет свободного доступа на законном основании и обладатель таких сведений принимает разумные меры для соблюдения их конфиденциальности, в том числе путем введения режима коммерческой тайны.

2) Секретом производства не могут быть признаны сведения, обязательность раскрытия которых либо недопустимость ограничения доступа, к которым установлена законом или иным правовым актом.

Согласно «статье 1225 ГК РФ. Охраняемые результаты интеллектуальной деятельности и средства индивидуализации» секрет производства попадает под защиту авторских прав.

По этой причине, на данный объект исследования распространяется пункт 2 статьи 1255 ГК РФ, описывающий права автора произведения, при этом они вступают в силу с момента обнародования объекта. Авторскими правами являются:

- исключительное право на произведение;
- право авторства;
- право автора на имя;
- право на неприкосновенность произведения;
- право на обнародование произведения.

Данные права раскрываются в статьях 1256, 1257, 1265, 1266, 1267, 1268, 1269, 1270.

Более полное описание права на секрет производства (ноу-хау) описаны в статьях 1465, 1466, 1467, 1468, 1469, 1470, 1471, 1472.

Исключительное авторское право, согласно статье 1466 ГК РФ, подразумевает право автора на использование секрета производства любым из способов, описанных в статье 1229 ГК РФ.

Статья 1229 определяет, что обладатель авторского права вправе использовать результат своей интеллектуальной деятельности любым не

противоречащим закону способом. Правообладатель имеет право разрешать или запрещать другим своё творение. Если использование происходит без ведома правообладателя и его разрешения, что является нарушением и влечёт ответственность, установленную Гражданским кодексом или иными законами, кроме случаев, допускающих использование без разрешения.

Понятие использования даётся в статье 1270: «Использованием произведения независимо от того, совершаются ли соответствующие действия в целях извлечения прибыли или без такой цели, считается, в частности:

1) воспроизведение произведения, то есть изготовление одного и более экземпляра произведения или его части в любой материальной форме, в том числе в форме звуко- или видеозаписи, изготовление в трех измерениях одного и более экземпляра двухмерного произведения и в двух измерениях одного и более экземпляра трехмерного произведения. При этом запись произведения на электронном носителе, в том числе запись в память ЭВМ, также считается воспроизведением. Не считается воспроизведением краткосрочная запись произведения, которая носит временный или случайный характер и составляет неотъемлемую и существенную часть технологического процесса, имеющего единственной целью правомерное использование произведения либо осуществляемую информационным посредником между третьими лицами передачу произведения в информационно-телекоммуникационной сети, при условии, что такая запись не имеет самостоятельного экономического значения;
2) распространение произведения путем продажи или иного отчуждения его оригинала или экземпляров;
3) публичный показ произведения, то есть любая демонстрация оригинала или экземпляра произведения непосредственно либо на экране с помощью пленки, диапозитива, телевизионного кадра или иных технических средств, а также демонстрация отдельных кадров

аудиовизуального произведения без соблюдения их последовательности непосредственно либо с помощью технических средств в месте, открытом для свободного посещения, или в месте, где присутствует значительное число лиц, не принадлежащих к обычному кругу семьи, независимо от того, воспринимается произведение в месте его демонстрации или в другом месте одновременно с демонстрацией произведения;

4) импорт оригинала или экземпляров произведения в целях распространения;
5) прокат оригинала или экземпляра произведения;
6) публичное исполнение произведения, то есть представление произведения в живом исполнении или с помощью технических средств (радио, телевидения и иных технических средств), а также показ аудиовизуального произведения (с сопровождением или без сопровождения звуком) в месте, открытом для свободного посещения, или в месте, где присутствует значительное число лиц, не принадлежащих к обычному кругу семьи, независимо от того, воспринимается произведение в месте его представления или показа либо в другом месте одновременно с представлением или показом произведения;
7) сообщение в эфир, то есть сообщение произведения для всеобщего сведения по радио или телевидению, за исключением сообщения по кабелю. При этом под сообщением понимается любое действие, посредством которого произведение становится доступным для слухового и (или) зрительного восприятия независимо от его фактического восприятия публикой. При сообщении произведений в эфир через спутник под сообщением в эфир понимается прием сигналов с наземной станции на спутник и передача сигналов со спутника, посредством которых произведение может быть доведено до всеобщего сведения независимо от его фактического приема

публикой. Сообщение кодированных сигналов признается сообщением в эфир, если средства декодирования предоставляются неограниченному кругу лиц организацией эфирного вещания или с ее согласия;

8) сообщение по кабелю, то есть сообщение произведения для всеобщего сведения по радио или телевидению с помощью кабеля, провода, оптического волокна или аналогичных средств. Сообщение кодированных сигналов признается сообщением по кабелю, если средства декодирования предоставляются неограниченному кругу лиц организацией кабельного вещания или с ее согласия;
    a. ретрансляция, то есть прием и одновременное сообщение в эфир (в том числе через спутник) или по кабелю полной и неизменной радио- или телепередачи либо ее существенной части, сообщаемой в эфир или по кабелю организацией эфирного или кабельного вещания;
9) перевод или другая переработка произведения. При этом под переработкой произведения понимается создание производного произведения (обработки, экранизации, аранжировки, инсценировки и тому подобного). Под переработкой (модификацией) программы для ЭВМ или базы данных понимаются любые их изменения, в том числе перевод такой программы или такой базы данных с одного языка на другой язык, за исключением адаптации, то есть внесения изменений, осуществляемых исключительно в целях функционирования программы для ЭВМ или базы данных на конкретных технических средствах пользователя или под управлением конкретных программ пользователя;
10) практическая реализация архитектурного, дизайнерского, градостроительного или садово-паркового проекта;
11) доведение произведения до всеобщего сведения таким образом, что любое лицо может получить доступ к произведению из любого

места и в любое время по собственному выбору (доведение до всеобщего сведения);

Наказание за нарушение исключительных прав на результаты интеллектуальной деятельности установлены статьёй 1252 ГК РФ. Таким образом, защита прав производится предъявлением требований в порядке, определённом статьёй:

- о признании права;
- о пресечении действий, нарушающих право или создающих угрозу его нарушения;
- о возмещении убытков;
- об изъятии материального носителя;
- о публикации решения суда о допущенном нарушении с указанием действительного правообладателя - к нарушителю исключительного права [20].

Обладатель прав вправе, в соответствии со статьёй 1311 ГК РФ, потребовать возмещение убытков от нарушителя в следующем размер:

1. в размере от десяти тысяч рублей до пяти миллионов рублей, определяемом по усмотрению суда исходя из характера нарушения;
2. в двукратном размере стоимости контрафактных экземпляров фонограммы;
3. в двукратном размере стоимости права использования объекта смежных прав, определяемой исходя из цены, которая при сравнимых обстоятельствах обычно взимается за правомерное использование такого объекта тем способом, который использовал нарушитель;

Право автора на имя определяется пунктом 1 статьи 1265 ГК РФ: «Право авторства – право признаваться автором произведения и право автора на имя - право использовать или разрешать использование произведения под своим именем, под вымышленным именем (псевдонимом) или без указания имени, то есть анонимно, неотчуждаемы и непередаваемы, в том числе при передаче

другому лицу или переходе к нему исключительного права на произведение и при предоставлении другому лицу права использования произведения. Отказ от этих прав ничтожен.» [24]

Право автора на неприкосновенность его произведения устанавливается статьёй 1266 ГК РФ, которая говорит о недопущении без согласия автора внесения изменений в его произведение, его, сокращение, дополнение,

ЗАКЛЮЧЕНИЕ

В ходе исследовательской работы был выяснено, что использование OpenVino позволило увеличить производительность работы демонстрационного приложения в десять раз. Для этого достаточно конвертировать модель в промежуточное представление (IR) при помощи Model Optimizer и использовать это представление в Inference Engine.

Model Optimizer – первый этап на пути эффективного использования ресурсов. Можно выделить две основные функции, которые он выполняет. Первая это оптимизация предоставленной модели путем удаления лишних слоев (к примеру, Dropout) или объединения узлов графа вычислений в один, если это возможно. Вторая функция – это конвертирование модели в промежуточное представление (Intermediate Representation), с которым работает Inference Engine.

Inference Engine – это компонент через который и происходит взаимодействие с вычислительными устройствами. Именно этот компонент обеспечивает эффективное использование ресурсов процессора, а именно правильное использование инструкций процессора, обеспечивая высокую производительность. Также в его обязанности входит оркестрация потока данных между вычислительными устройствами, ведь OpenVino позволяет использовать гетерогенный подход. Другими словами, можно запускать модель сразу на нескольких устройствах или ядрах.

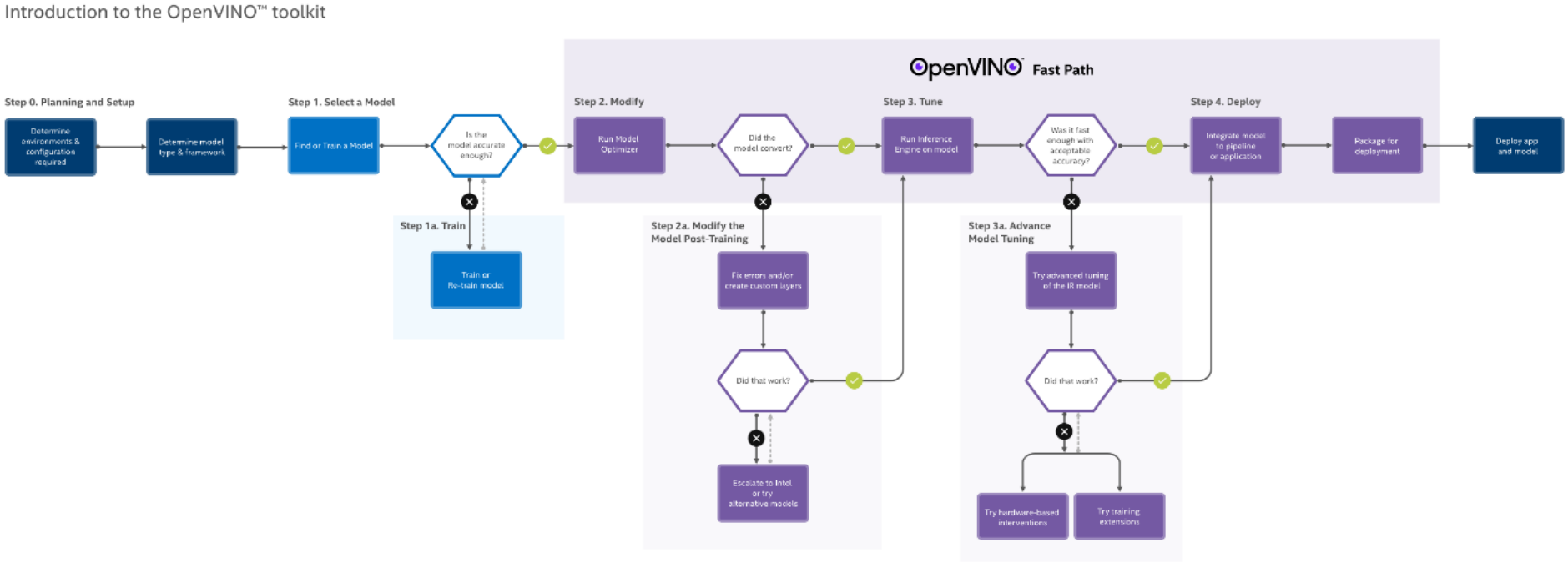
Кроме этого OpenVino предоставляет широкий инструментарий для конфигурирования оптимизаций над моделями и сравнения этих моделей с между собой. Все это можно сделать в инструменте OpenVino Workbench, который предоставляет хороший визуальный интерфейс для этого.

Также стоит отметить, что сама интеграция OpenVino в приложения не несет никаких проблем, ведь для корректной работы приходится оперировать небольшим набором классов и функций. В сравнении с кодом итого приложения – это сущие мелочи.

В совокупности предоставленные возможности дают прекрасный повод для использования OpenVino в своих приложениях. Тем более, что технология изначально нацелена на использование уже существующих моделей. А это значит, что не придётся переделывать все под новую технологию.

ЛИТЕРАТУРА

ПРИЛОЖЕНИЕ 1



ПРИЛОЖЕНИЕ 2

```
<?xml version="1.0" ?>
<net name="model_file_name" version="10">
    <layers>
        <layer id="0" name="input" type="Parameter" version="opset1">
            <data element_type="f32" shape="1,3,32,100"/> <!-- attributes of operation -->
            <output>
                <!-- description of output ports with type of element and tensor dimensions -->
                <port id="0" precision="FP32">
                    <dim>1</dim>
                    <dim>3</dim>
                    <dim>32</dim>
                    <dim>100</dim>
                </port>
            </output>
        </layer>
        <layer id="1" name="conv1/weights" type="Const" version="opset1">
            <!-- Const is only operation from opset1 that refers to the IR binary file by specifying offset and size in bytes relative to the beginning of the file. -->
            <data element_type="f32" offset="0" shape="64,3,3,3" size="6912"/>
            <output>
                <port id="1" precision="FP32">
                    <dim>64</dim>
                    <dim>3</dim>
                    <dim>3</dim>
                    <dim>3</dim>
                </port>
```

```
            </output>
        </layer>
        <layer id="2" name="conv1" type="Convolution" version="opset1">
            <data auto_pad="same_upper" dilations="1,1" output_padding="0,0"
pads_begin="1,1" pads_end="1,1" strides="1,1"/>
            <input>
                <port id="0">
                    <dim>1</dim>
                    <dim>3</dim>
                    <dim>32</dim>
                    <dim>100</dim>
                </port>
                <port id="1">
                    <dim>64</dim>
                    <dim>3</dim>
                    <dim>3</dim>
                    <dim>3</dim>
                </port>
            </input>
            <output>
                <port id="2" precision="FP32">
                    <dim>1</dim>
                    <dim>64</dim>
                    <dim>32</dim>
                    <dim>100</dim>
                </port>
            </output>
        </layer>
        <layer id="3" name="conv1/activation" type="ReLU"
version="opset1">
```

```
            <input>
                <port id="0">
                    <dim>1</dim>
                    <dim>64</dim>
                    <dim>32</dim>
                    <dim>100</dim>
                </port>
            </input>
            <output>
                <port id="1" precision="FP32">
                    <dim>1</dim>
                    <dim>64</dim>
                    <dim>32</dim>
                    <dim>100</dim>
                </port>
            </output>
        </layer>
        <layer id="4" name="output" type="Result" version="opset1">
            <input>
                <port id="0">
                    <dim>1</dim>
                    <dim>64</dim>
                    <dim>32</dim>
                    <dim>100</dim>
                </port>
            </input>
        </layer>
    </layers>
    <edges>
```

```
        <!-- Connections between layer nodes: based on ids for layers and ports used in the descriptions above -->
        <edge from-layer="0" from-port="0" to-layer="2" to-port="0"/>
        <edge from-layer="1" from-port="1" to-layer="2" to-port="1"/>
        <edge from-layer="2" from-port="2" to-layer="3" to-port="0"/>
        <edge from-layer="3" from-port="1" to-layer="4" to-port="0"/>
      </edges>
      <meta_data>
        <!-- This section that is not related to a topology; contains auxiliary information that serves for the debugging purposes. -->
        <MO_version value="2019.1"/>
        <cli_parameters>
          <blobs_as_inputs value="True"/>
          <caffe_parser_path value="DIR"/>
          <data_type value="float"/>

          ...

          <!-- Omitted a long list of CLI options that always are put here by MO for debugging purposes. -->

        </cli_parameters>
      </meta_data>
    </net>
```