ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

На правах рукописи

Гончаренко Александр Игоревич

ВЫСОКОПРОИЗВОДИТЕЛЬНЫЕ НЕЙРОННЫЕ СЕТИ ГЛУБОКОГО ОБУЧЕНИЯ ДЛЯ УСТРОЙСТВ С НИЗКИМИ ВЫЧИСЛИТЕЛЬНЫМИ РЕСУРСАМИ

Специальность 1.2.2 —
«Математическое моделирование, численные методы и комплексы программ»

Диссертация на соискание учёной степени
кандидата технических наук

Научный руководитель:
доктор технических наук, ведущий научный сотрудник ИАиЭ СО РАН
Нежевенко Евгений Семёнович

Новосибирск — 2023

Оглавление

Стр.

Стр.

Список сокращений и определений

CPU - (Central Processing Unit) центральный вычислительный процессор

float32 - тип представления вещественных чисел одинарной точности, для записи которых выделяется 32 разряда.

FLOPS - (FLoating-point Operations Per Second) количество операций с плавающей запятой, которое вычислитель способен произвести за одну секунду.

FPGA - (Field Programmable Gate Array) полупроводниковое устройство, которое может быть сконфигурировано производителем или разработчиком после изготовления; наиболее сложная по организации разновидность программируемых логических интегральных схем. Используется на начальном этапе проектирования NPU как устройство для отладки архитектуры будущего процессора.

GPU - (Graphics Processing Unit) графический вычислительный процессор. Имеет массивно параллельную архитектуру и широко используется для вычисления нейронных сетей.

lstm - архитектура рекуррентной нейронной сети.

minifloat - тип представления вещественных чисел аналогичный float32, для записи которых выделяется менее 16 разрядов.

NPU - (Neural Processing Unit) специализированный подкласс процессоров для эффективного выполнения операций над многомерными тензорами.

Батч - (англ. batch) группа элементов данных, подаваемая в нейронную сеть для для параллельного вычисления. Как правило, значительно меньше всей обучающей выборки.

Датасет - (англ. dataset) структурированный набор элементов данных, к которым можно обращаться или по отдельности, или в комбинации, связанный с решаемой задачей.

Дистилляция - обучение нейронной сети с малым числом параметров за счёт передачи знаний от нейронной сети с большим числом параметров. Где под процедурой передачи знаний понимается обучение нейронной сети воспроизводству обобщений в данных, извлекаемых большей по числу параметров моделью.

Дообучение - процесс тонкой настройки весовых коэффициентов нейронной сети на наборе калибровочных данных, направленный на ее оптимизацию с целью повышения качества модели нейронной сети на новом наборе данных и/или адаптации к модификациям в архитектуре модели.

Квантование - отображение вещественного числового пространства в его компьютерной интерпретации типа float32 на пространство целых чисел как правило с меньшей разрядностью.

Квантование модели нейронной сети - процесс квантования параметров обученной модели нейронной сети таких, как веса и активации, а также определение параметров такого квантования и преобразование структуры нейронной сети с учетом особенностей арифметических операций над квантованными данными.

Конвертация - отображение вещественного числового пространства в его компьютерной интерпретации типа float32 на его облегченное представление иного типа или того же типа меньшей разрядности без предварительной калибровки, направленной на адаптацию к распределению типичных значений исходного набора чисел.

Матричное разложение - процедура представления матрицы в виде произведения матриц, обладающих некоторыми определёнными свойствами (например, ортогональностью, симметричностью, диагональностью).

НПА - нейросетевой поиск архитектур. Семейство подходов для автоматического выбора типа и порядка слоёв в нейронной сети с целью получения модели с наибольшей эффективностью.

Прунинг - (англ. pruning) процедура удаления весовых коэффициентов из обученной нейронной сети как по отдельности, так и в группе.

Сеть-учитель - в задаче дистилляции обученная нейронная сеть с большим числом обучаемых параметров.

Сеть-ученик - в задаче дистилляции обучаемая нейронная сеть с малым числом обучаемых параметров.

Введение

На сегодняшний день нейронные сети достигли значительного прогресса в решении задач, связанных с обработкой изображений [1], [2] , [3], распознавания речи [4] и обработки текста [5]. Для решения большинства задач подобного рода существуют методы оптимизации (обучения) нейронных сетей [6], [7], которые позволяют получить приемлемые метрики качества. Однако, вычислительная сложность таких обученных моделей может быть очень высока, что позволяет запускать их только на графических ускорителях (GPU), но не на устройствах с низкими вычислительными возможностями, такими как мобильные процессоры и FPGA. Необходимость же запускать нейронные сети на мобильных устройствах вызвана прежде всего следующими факторами:

1. **Широкая популярность мобильных телефонов.** На сегодняшний день, рынок мобильных платформ, равно как и мобильных приложений, растет и развивается [8]. Особую популярность приобретают голосовые помощники и медицинские приложения, анализирующие и ставящие диагнозы по фотографии. К сожалению, работа таких программ крайне ограничена на клиентской части приложения и требует пересылки данных на серверную составляющую, из чего следуют пункты далее.
2. **Безопасность данных.** Нередки случаи утечки данных со стороны крупных компаний. Вычисления на стороне клиента могли бы значительно снизить последствия взломов корпораций.
3. **Работа в условиях без широкополосного интернета.** Несмотря на тот факт, что доступ к интернету на сегодняшний день не является проблемой, все же существуют регионы, где ширина полосы связи недостаточна для качественной работы клиент-серверных приложений. Такими регионами является горная местность, где зачастую случаются несчастные случаи и при помощи дронов осуществляются поисково-спасательные операции. Качество таких работ повышается при помощи

специального ПО, которое позволяет находить объекты на изображении [9].

Все вышеперечисленные факторы указывают на то, что уменьшение вычислительной сложности нейронных сетей с целью запуска на мобильных платформах является актуальной задачей. На сегодняшний день существует ряд способов позволяющих выполнить подобную процедуру: нейросетевой поиск архитектур [10], прунинг [11], матричные разложения [12], квантование [13] и дистилляция [14]. Однако, у большинства разработанных алгоритмов есть ряд существенных недостатков. Эти алгоритмы либо тестировались на “больших” нейронных сетях [2], запуск которых не целесообразен на мобильных платформах, либо предполагают обучение целевой нейронной сети с нуля, либо тратят значительные ресурсы на дообучение моделей, что может быть затруднительно по следующим причинам:

1. **Нейронные сети требуют значительных вычислительных ресурсов для их тренировки.** Типичные времена обучения могут занимать от одного до 14 дней даже при использовании видеокарты;
2. **Отсутствие размеченных данных для обучения.** Нередко авторы различных научных статей в области машинного обучения запрещают использовать собранные ими наборы данных в коммерческих целях, но публикуют и разрешают пользоваться обученными моделями. При этом, публичные аналоги датасетов, доступные для использования, могут обладать разметкой худшего качества, что значительно снижает точность модели в процессе дообучения. Более того, исследователи-практики, работающие в частных компаниях, не предоставляют обучающие выборки на специфичных данных, способных принести коммерческую выгоду (например, персональные средства защиты или автотранспорт специального типа);
3. **Замедление процесса разработки комплексных приложений.** Зачастую для достижения оптимального качества при фиксированном времени требуется несколько итераций взаимодействия между специалистами по машинному обучению и системными программистами, поскольку невозможно заранее спрогнозировать какие архитектурные

особенности приведут к улучшению или ухудшению при фиксированной скорости. Длительное обучение нейронных сетей значительно замедляет данный процесс.

Альтернативным подходом является проектирование собственных архитектурных решений и использование особых типов данных с плавающей запятой, у которых разрядность порядка и мантиссы отличается от принятого стандарта IEEE-754. К сожалению, применимость такого подхода к распространенным архитектурам рекуррентных нейронных сетей была слабо исследована.

Целью диссертационной работы является разработка методов для ускорения вычисления нейронных сетей на мобильных платформах, не требующих от применяющего большого количества размеченных данных и вычислительных ресурсов для тонкой настройки сети, но при этом не допускающие значительного падения качества.

Для достижения поставленной цели необходимо решить следующие **задачи**:

1. Проанализировать существующие архитектуры нейронных сетей с целью определения возможности вычисления на мобильных платформах;
2. Исследовать существующие методы ускорения нейронных сетей;
3. На основе существующих решений разработать собственный метод ускорения нейронных сетей с незначительным падением точности относительно оригинальной архитектуры и не требующий размеченных данных для тонкой настройки;
4. Экспериментально подтвердить эффективность разработанного метода;
5. Исследовать возможность применения нестандартных типов данных с плавающей запятой;
6. Экспериментально найти подходящие разрядности порядка и мантиссы, не снижающие качественные показатели сетей;

Научная новизна:

1. Предложен и реализован новый алгоритм квантования для моделей произвольного типа на основе тонкой настройки масштабирующих коэффициентов для порогов квантования. При этом время, затраченное на тонкую настройку сети, после применения алгоритма значительно ниже (от 5 до 10 раз, в зависимости от архитектуры нейронной сети), чем в большинстве современных работ в данной области, при незначительном падении точности (менее 1%) относительно оригинальной модели;
2. Разработан и применен алгоритм перемасштабирования весовых коэффициентов для процедуры скалярного квантования для ограниченной функции активации ReLU6, наиболее распространенной в мобильных архитектурах нейронных сетей;
3. Предложена и реализована процедура нахождения разрядности для специализированных типов данных. Предложенный механизм не требует дополнительной тонкой настройки сети, что позволяет упростить внедрение нейронных сетей в сложные программно-аппаратные комплексы. Данная процедура применялась к разнообразным архитектурам сверточных нейронных сетей, что может свидетельствовать о ее универсальности;
4. Предложенный подход нахождения разрядности для специализированных типов данных впервые применен к глубокой рекуррентной сети, что может свидетельствовать о его универсальности и позволяет его использовать в программно-аппаратных комплексах на программируемых интегральных схемах для проектирования аппаратных нейросетевых ускорителей.

Теоретическая значимость полученных результатов заключается в предложенном в диссертации новом подходе устранения влияния выбросов посредством обучения порогов квантования, что ведет к уменьшению шага дискретизации при квантовании и, как следствие, снижению количества ошибок, возникающих в нейронной сети. Предложенный подход был теоретически проанализирован при помощи метода обратного распространения ошибки.

Практическая значимость полученных результатов заключается в реализации разработанного подхода в виде программного комплекса для обучения свёрточных нейронных сетей на ЭВМ. Получаемые таким образом нейросетевые модели могут использоваться для произвольных задач на маломощных вычислителях. В частности, описанный в данной диссертации подход позволил ускорить алгоритм детектирования лица на конечном устройстве пользователя с мобильным ARM процессором. Также разработанный алгоритм стал частью программной платформы EENNT, позволяющей оптимизировать вычислительную сложность нейронных сетей произвольной архитектуры.

Процедура подбора разрядности порядка и мантиссы была использована как один из основных модулей для исследования оптимизации архитектуры аппаратных ускорителей на основе систолического массива.

Технология, основанная на разработанном подходе, внедрена в продуктах компаний:

1. “ООО Диалоговые системы”. Разработанный подход применялся для ускорения нейронной сети для распознавания речи в программной платформе для создания интеллектуальных диалоговых агентов “W11”;
2. “ООО Экспасофт”. Разработанные методы легли в основу программного комплекса ускорения нейронных сетей “Expasoft Embedded Neural Network Technology” или “EENNT”, имеющий свидетельство о государственной регистрации программы для ЭВМ и включенный реестр отечественного ПО с номером реестровой записи 8738;
3. “ООО ИВА ТЕХНОЛОДЖИС”. Разработанные подходы внедрены в программно-аппаратный комплекс “Микропроцессор IVA TPU”.

Основные положения, выносимые на защиту:

1. Использование масштабирующих коэффициентов для порогов квантования в качестве параметров сети и их тонкая настройка позволяет корректировать шаг дискретизации и таким образом более точно квантовать значения, близкие к нулю;
2. Алгоритм квантования нейронных сетей позволяет использовать неразмеченные данные для тонкой настройки;

3. Процедура перемасштабирования последовательных слоев нейронной сети с учетом нелинейной функции активации позволяет производить выравнивание итоговых распределений между каналами и увеличивает точность скалярного квантования до тонкой настройки;
4. Экспериментально подтвержденная процедура подбора порядка и мантиссы для нестандартных типов данных с плавающей запятой позволяет сократить разрядность вычислений до 11 бит для глубоких сверточных и рекуррентных архитектур без применения процедуры тонкой настройки;

Достоверность полученных результатов обеспечивается корректным использованием математического аппарата при разработке и анализе методов и корректным проведением большого числа тестов на реальных данных. Для измерения точности системы использовались объективные метрики, продемонстрировавшие результаты непротиворечивые друг другу и теоретическим выкладкам.

Апробация работы. Основные положения и результаты диссертационной работы докладывались и получили одобрение на Международных конференциях Artificial Neural Networks and Machine Learning (ICANN) и 15th International Work-Conference on Artificial Neural Networks (IWANN). Используя предложенный алгоритм, автор занял первые места в двух из трех номинациях на конкурсе LPIRC [15].

Личный вклад. Автор разработал и реализовал подходы к оптимизации производительности нейронных сетей. Разработанные подходы описаны в главах: “Глава 2. Метод дообучения порогов как комбинация дистилляции и квантования” и “Глава 4. Процедура нахождения оптимальной разрядности порядка и мантиссы”. Также автором проведен теоретический анализ разработанной процедуры дообучаемых порогов на основе алгоритма обратного распространения ошибки.

Публикации. Материалы диссертации опубликованы в 4 печатных работах в рецензируемых журналах [16—19]. Все работы изданы в индексируемых

Scopus журналах. Работа [19] издана в журнале, относящемуся к квартили Q1, а работы [16—18] - в относящихся к квартилям Q3 и Q2.

Объем и структура работы. Диссертация состоит из введения, 4 глав и заключения. Полный объём диссертации составляет 98 страниц, включая 37 рисунков и 6 таблиц. Список литературы содержит 94 наименования.

Первая глава носит обзорный характер и описывает способы ускорения нейронных сетей, которые применяются научным сообществом на сегодняшний день. На основе анализа были выявлены проблемные моменты применения уже существующих методов с точки зрения существующих подходов разработки комплексного программного обеспечения.

Во второй главе описывается разработанный метод обучаемых порогов как эффективный метод ускорения нейронных сетей, не требующих значительных ресурсов для тонкой настройки. Дан анализ того, как происходит обновление порогов с точки зрения алгоритма обратного распространения ошибки.

В третьей главе проведен анализ самых распространенных операций, присутствующих в нейронных сетях с целью выбора наиболее перспективных для аппаратной реализации.

В четвертой главе показывается, как на основе симуляции типа данных с плавающей запятой с произвольной разрядностью можно моделировать потерю точности при конвертации. Была экспериментально показана эффективность разработанной процедуры.

Глава 1. Способы уменьшения вычислительной сложности нейронных сетей

Задача классификации изображений является одной из первых успешных задач, решаемых при помощи нейронных сетей. Типом нейронных сетей, которые успешно решают эти задачи, являются сверточные нейронные сети [1; 2; 20]. Первой архитектурой и типичным представителем такой сети является нейронная сеть AlexNet:

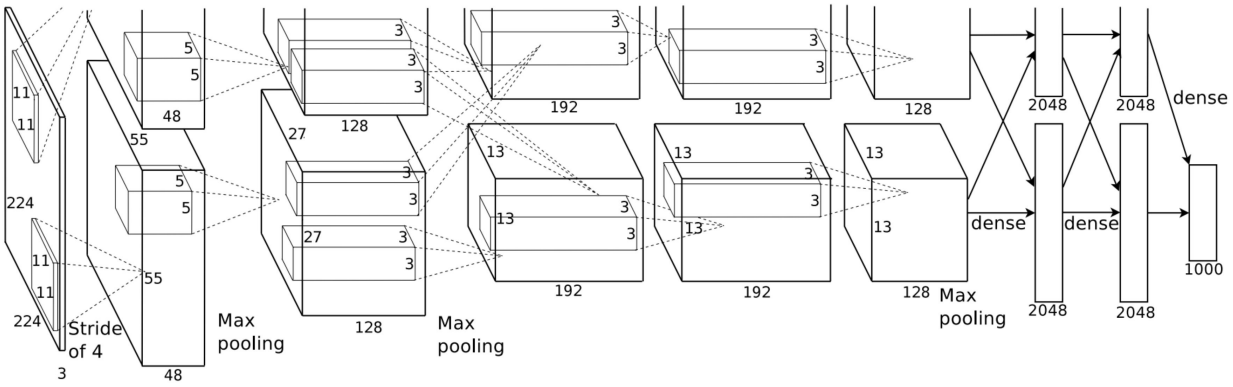

Рисунок 1.1 — Архитектура нейронной сети AlexNet [1].

В основе работы всех сверточных сетей лежит операция свертки, которая задается формулой ниже:

$$G_{kln} = \sum_{i,j,m} K_{i,j,m} I_{k+i-1,l+j-1,m}$$

где K - это ядро свертки, а I - входные данные для сверточного слоя. Пусть входная карта признаков F имеет размерность [1, ImH, ImW, ImC], а сверточный слой имеет размерность [k, k, ImC, N]. Тогда количество операций сложения-умножения будет равняться $ImH * ImW * h * w * ImC * N$ при условии применения процедуры дополнения изображения до исходного пространственного размера.

В этой главе будут рассмотрены архитектуры нейронных сетей, пригодные для запуска на мобильных платформах для задачи классификации, как отправная точка для конструирования вычислительно эффективных типов нейронных сетей. Также будет произведен обзор методов оптимизации их вычислительной сложности для дальнейшего ускорения: квантование, прунинг, дистилляция и

канонические разложения. Дополнительно будет приведен способ автоматического конструирования мобильных архитектур нейронных сетей - нейросетевой поиск архитектур.

1.1 Разработка вычислительно эффективных архитектур на примере задачи классификации

Первопроходцами в данной области можно считать авторов работ [2], [21]. В работе [2] предложена следующая идея: заменить фильтры большего пространственного разрешения, на два последовательных фильтра меньшего пространственного разрешения, сохранив таким образом разрешение выходной карты признаков, но уменьшив вычислительную сложность. Например, для сверточного слоя с пространственной размерностью 5x5 мы получим $25 * ImH * ImW * ImC * N$ операций сложения-умножения, а для двух сверточных слоев 3x3, следующих друг за другом мы получим количество операций $9 * ImH * ImW * ImC * (ImC + N)$. Таким образом, уменьшение количества операций будет равно:

$$\frac{25 * ImH * ImW * ImC * N}{9 * ImH * ImW * ImC * (ImC + N)} = \frac{25 * N}{9 * (ImC + N)} = \frac{25}{9}(1 - \frac{ImC}{ImC + N})$$

где $N \geqslant ImC$.

Несмотря на тот факт, что данная архитектура не является подходящей для мобильных устройств в силу высокой вычислительной сложности, именно авторы работы [2] заложили в основу идею разложения процедуры свертки на группу операций, которые в конечном итоге работают так же хорошо, как и исходная свертка, но при этом суммарно имеют меньшее число сложений и умножений.

Исследователи из компании Google в работе [21] предложили методику уменьшения количества каналов входной карты признаков при помощи свертки с пространственным разрешением 1x1 и обработки параллельным образом

нескольких типов сверточных слоев с последующим соединением результатов свертки. Такой блок изображен на рисунке 1.2.

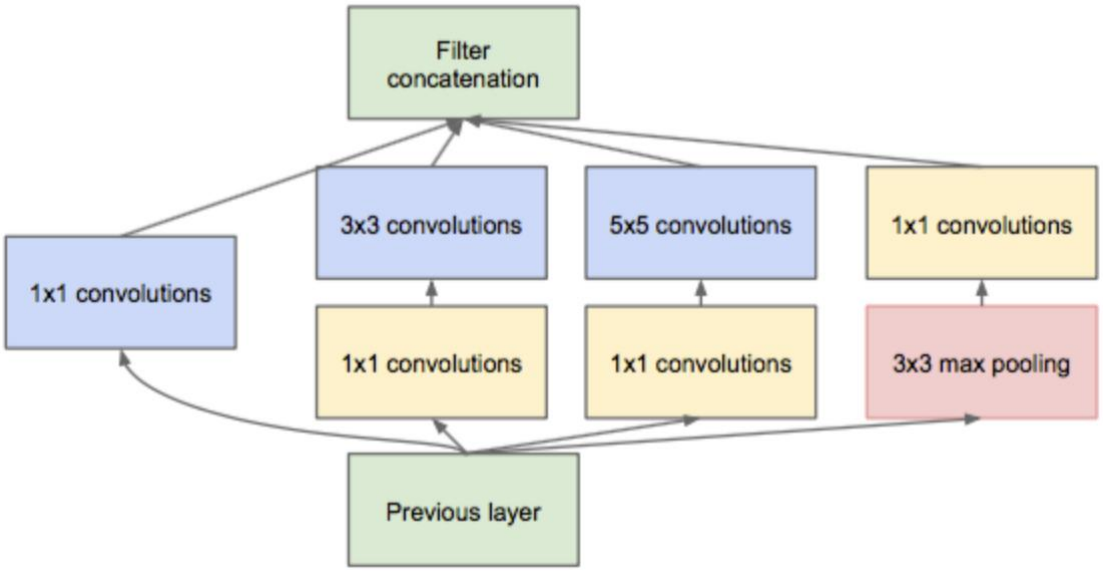

Рисунок 1.2 — Блок из сети GoogleNet [21].

Подход к уменьшению количества операций в нейронных сетях был развит в работах [22—24]. В [22], авторы соединили идеи параллельной обработки одинаковых карт признаков и сокращения вычислений при сохранении пространственного разрешения путем введения так называемой асимметричной свертки. При подобной процедуре фильтр $k \times k$ заменяется на два последовательных фильтра с пространственным разрешением $[1 \times k]$ и $[k \times 1]$. В работе авторов представлено большое количество подобных блоков (см. рис 1.3).

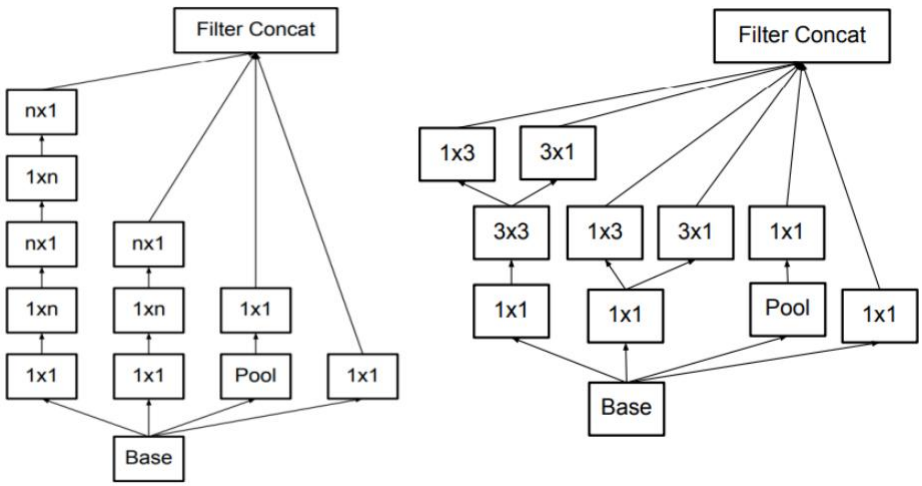

Рисунок 1.3 — Блоки из сети InceptionV3 [22].

Авторы работы [23] (и ее дальнейшего улучшения [24]) подошли к сокращению вычислений несколько с иной стороны. Авторы посчитали, что пространственное сокращение работает не так эффективно, как если уменьшать количество вычислений по каналам. Так появилась архитектура MobileNet. Основная идея заключается в том, чтобы использовать depthwise-separable свертку, которая отличается тем, что не осуществляет суммирование по каналам, а обрабатывает каждый канал входной карты признаков отдельно:

$$G_{kln} = \sum_{ij} K_{i,j,n} I_{k+i-1,l+j-1,n}$$

После подобной свертки идет обычная свертка 1x1 для того, чтобы все же осуществить перемешивание каналов в пространственном измерении. Принимая во внимание предыдущие обозначения, заметим, что применение подобной процедуры позволяет достигнуть следующего ускорения:

$$\frac{k^2 * ImH * ImW * ImC * N}{ImH * ImW * ImC * (k^2 + N)} = \frac{k^2 * N}{k^2 + N}$$

Архитектура, представленная в работе [24], идейно такая же, как в [23] с добавлениями некоторых современных улучшений [25] и называется MobileNet-v2. На сегодняшний день эта архитектура де-факто является стандартом для обработки изображений на мобильных устройствах. Структура базового блока для MobileNet-v2 изображена на рисунке ниже:

На основе архитектуры MobileNet-v2 была разработана более эффективная архитектура MNasNet [26], правда, для вполне конкретной мобильной платформы (Google Pixel Phone 2), но нередко данная нейросеть демонстрирует отличное качество и на других мобильных устройствах. Архитектура MNasNet изображена на рисунке 1.5. Особенностью этой архитектуры является то, что она получена на основе нейросетевого поиска архитектур - метода, при котором архитектура нейронной сети строится автоматически.

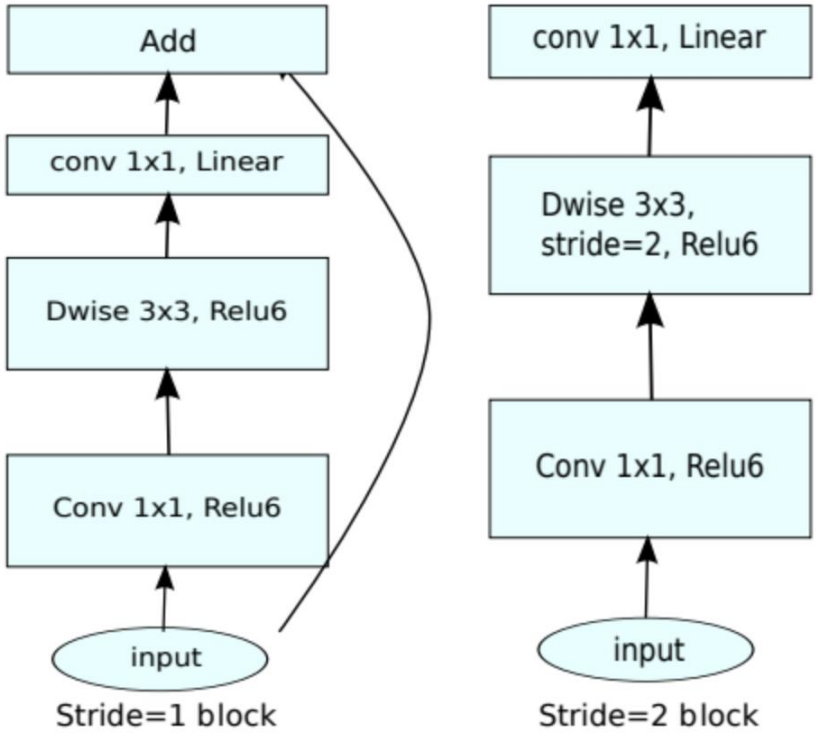

Рисунок 1.4 — Блок из сети MobileNetv2 [24].

1.2 Нейросетевой поиск архитектур

В предыдущем разделе описывался исторический процесс изобретения оптимальных архитектур с точки зрения вычислительной сложности. Как можно было заметить, это достаточно трудоемкий процесс, который требует профессионализма и высокой квалификации специалиста. Однако, даже придумав оптимальный слой и/или блок нет никаких гарантий, что комбинация составляющих частей может дать оптимальную архитектуру нейронной сети. Действительно, ведь при создании нейросети необходимо учитывать влияние слоев друг на друга, а также характеристики этих слоев (число выходных/выходных фильтров, размер ядра свертки и т.д.), называемых гиперпараметрами.

Специалисты, работающие в области нейросетевого поиска архитектур (НПА) стремятся автоматизировать данный процесс, чтобы исключить человеческий фактор и, таким образом, строить более точные / “легкие” нейронные сети. При таком подходе, в зоне ответственности специалиста находится только выбор оптимального пространства поиска, иначе говоря, списка возможных

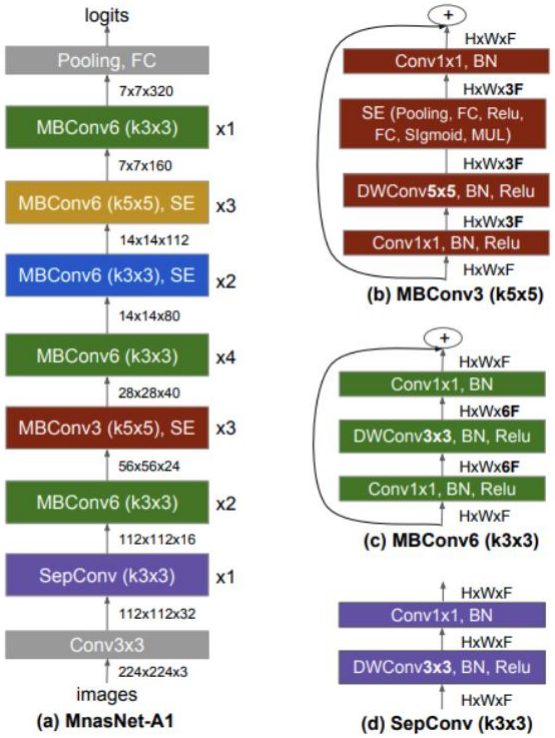

Рисунок 1.5 — Архитектура нейронной сети MNasNet [26]. a) архитектура сети целиком, b), c), d) - устройство отдельных блоков. .

типов слоев, из которых будет построена нейронная сеть. Методы НПА можно разделить на две больших группы:

1. Недифференцируемые [10; 27; 28], [26];
2. Дифференцируемые [29—31]

Первый класс методов, то есть недифференцируемый нейросетевой поиск архитектур, появился исторически раньше и был предложен специалистами из компании Google [10]. Работу алгоритмов из данного класса можно разделить на несколько этапов:

1. Этап генерации вариантов при помощи “алгоритма” поиска;
2. Обучение предложенных архитектур;
3. Оценка качества архитектур из пункта 2;
4. Обновление “алгоритма” поиска.

На этапе 1 может быть сгенерирована как вся нейросеть слой за слоем, так и конкретный блок, который будет являться базовым для строительства сети. При этом, самое существенное отличие между методами заключается, как правило, в этапе 1, а именно в выборе “генерирующего” алгоритма. На сего-

дняшний день доминируют два подхода: обучение с подкреплением [10; 26; 27] и эволюционные алгоритмы [28].

В алгоритмах с обучением с подкреплением в качестве генерирующего алгоритма используется нейронная сеть с рекуррентной архитектурой типа LSTM [32], которая в специализированной литературе называется сетью-контроллером. Контроллер предсказывает дочернюю сеть или блок, который проходит 2 и 3 этапы, после чего веса контроллера уточняются на основе методов обучения с подкреплением [33], поскольку при данной постановке задачи НПА у нас нет возможности напрямую обновлять веса контроллера при помощи методов, основанных на градиентном спуске.

Альтернативным вариантом генерации сети может являться использование эволюционных алгоритмов [28]. Эволюционные или генетические алгоритмы были созданы на основе биологических представлений о том, как должны развиваться живые организмы. Объект или группа объектов, оптимизируемая в эволюционном подходе должна быть описана неким математическим образом. В случае нейронной сети необходимо закодировать связи между операциями в блоке поиска. Такое строковое описание объекта называется вектором генов. Необходимо также ввести еще два понятия: мутации и скрещивание. Скрещивание - это процедура обмена частями вектора генов между двумя объектами в выборке потомков, а мутация - это процедура случайного изменения случайных признаков в векторе генов. Эти две процедуры призваны повысить разнообразие генерируемых выборок с целью большего исследования возможных вариантов архитектур.

Тогда, применительно к задаче НПА данные процедуры работают следующим образом:

1. Генерируется текущая популяция дочерних архитектур из набора родителей;
2. Обучение предложенных вариантов нейронной сети;
3. Оценка качества архитектур из пункта 2;
4. Отбор наиболее перспективных потомков;
5. Мутации и скрещивание потомков;
6. Обновление набора родителей.

Основным недостатком для данного класса методов НПА является требование огромного количества вычислительных ресурсов. Для примера, в работах [27], [28] требовалось примерно 2000 и 3000 GPU-дней соответственно, что накладывает существенные ограничения на возможность использования данных алгоритмов.

Альтернативой недифференцируемому поиску является второй тип НПА - дифференцируемый. Прорывной работой в этой области служит статья [29]. Основная идея заключается в том, чтобы не выбирать операцию за операцией при построении сети, обучая каждый раз новую архитектуру сети, а учитывать все операции одновременно, но каждую с каким-то весом. Математически, это выглядит следующим образом:

$$o(x) = \sum_{i=0}^{N} \frac{exp(p^i)}{\sum_{j=0}^{N} exp(p^j)} o^i(x)$$

где $o(x)$ - это выход комбинации, $o^i(x)$ - это выход одной конкретной операции из пространства поиска, а p^i - это обучаемые параметры. При этом, у нас появляется два принципиально разных типа параметров: “обычные” параметры, то есть весовые коэффициенты в сверточных и других слоях, и “архитектурные” параметры p^i, величина которых определяет важность той или иной операции. Нейронная сеть, построенная таким образом называется суперсетью [31].

Процесс обучения в данном случае происходит двухуровневым образом [29], то есть:

1. Сначала выполняется шаг методом стохастического градиентного спуска [34] для “обычных” параметров слоев при зафиксированных “архитектурных” параметрах;
2. Затем выполняется шаг методом стохастического градиентного спуска [34] для “архитектурных” параметров при зафиксированных “обычных” параметрах;
3. В случае сходимости алгоритма, необходимо выбрать те операции, вес у которых наибольший;
4. Выполнить процедуру тонкой настройки найденной архитектуры нейронной сети.

Конструируя процедуру НПА таким образом можно будет избавиться от лишних процедур обучения дочерних сетей, что значительно ускорит получение конечного результата. Графически процесс настройки и изменения архитектурных параметров изображен на рисунке 1.6.

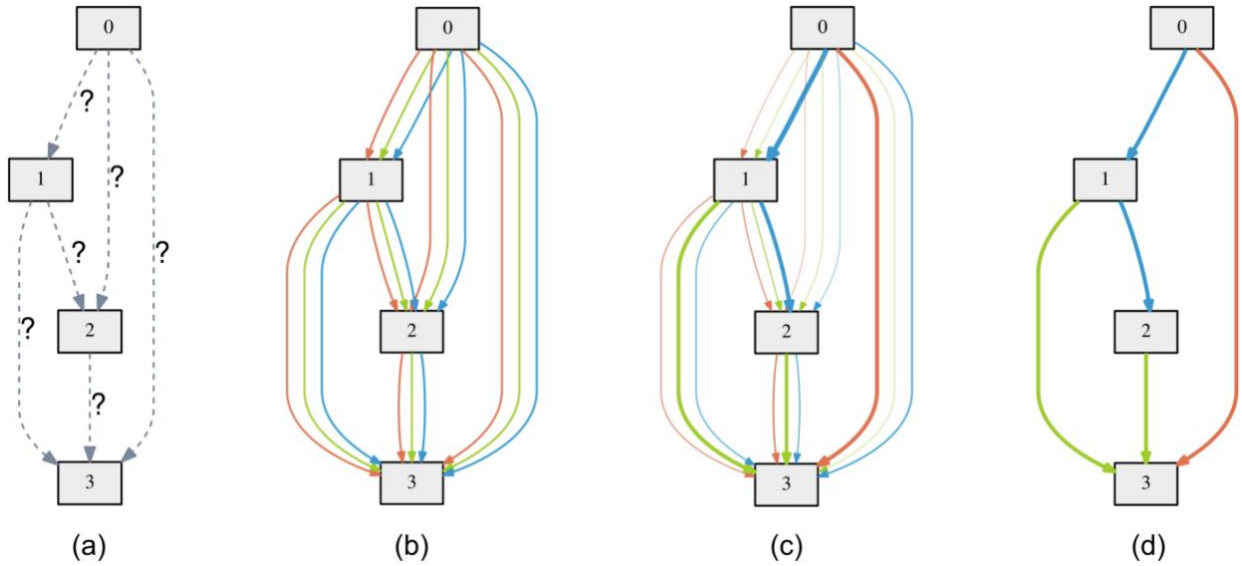

Рисунок 1.6 — Визуализация процесса настройки метода DARTS для поиска оптимального блока [29]. a) Блок представлен в виде ациклического направленного графа, где в качестве граней - тип операции, а узлы - это результирующие тензора b) инициализация алгоритма, цветом обозначены операции разного типа, а их толщина обозначает важность c) изменение важности операций в процессе обучения d)выбор финальной архитектуры

Дифференцируемая процедура поиска сократила время работы до десятков GPU-дней, однако может быть применима только лишь к модельным задачам с небольшим размером входных изображений. Действительно, при обучении “обычных” параметров, нам необходимо держать в памяти все выходы операций для обратного распространения ошибки, что делает невозможным применить данный метод НПА к реальным задачам напрямую.

Дальнейшие развития данного направления, показанные в работах [30; 31], базируются на том, что нет необходимости вводить архитектурные параметры, как таковые, а достаточно в процессе обучения суперсети случайным образом выбирать операции, получая на каждом этапе обучения новые архитектуры, параметры которых делятся между собой. В обученной таким образом суперсети

будет множество путей, каждый из которых в принципе является валидным, но при этом, не все одинаково хороши. После этого можно применить, как обучение с подкреплением, так и генетические алгоритмы для выбора оптимального пути и, таким, образом получения сети наилучшего качества.

Дифференцируемые методы, безусловно, дали значительный толчок в этом направлении, однако и они не лишены недостатков:

1. По прежнему необходима высокая квалификация специалиста, выполняющего обучение супер сети;
2. Время на обучение суперсети увеличивается в разы по сравнению с обучением базовой сети;
3. Невозможно ускорить уже обученные архитектуры.

1.3 Прунинг

Прунинг или прореживание нейронных сетей - это способ оптимизации архитектур обученных нейронных сетей, в основе которого лежит идея в том, что не все весовые коэффициенты в нейронной сети могут быть полезны с точки зрения итогового качества.

Процедуры прунинга можно разделить на два больших типа: структурированный или неструктурированный [35]. При неструктурированном прунинге при процедуре оптимизации сети удаляются любые слабо влияющие весовые коэффициенты, в то время как при структурированном удаляются и фильтры (для сверточных слоев), и нейроны (для полносвязных слоев) целиком. И в том, и в другом типе прунинга для оценки “полезности” необходимо задать так называемый критерий прунинга, то есть числовое выражение “полезности”. Наиболее очевидным в данном случае является L1/L2 нормы весов [11] или фильтров [36]. Действительно, поскольку сверточные и полносвязные слои могут быть представлены как матричные умножения, то наименьшие весовые коэффициенты вносят наименьший вклад в итоговую сумму.

Для структурированного прунинга большую распространенность получил критерий на основе слоя пакетной нормализации (англ. batch normalization, BN) [37—39]. Математически, данный слой выглядит следующим образом:

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \varepsilon}} * \gamma + \beta$$

где x - это входной тензор, $\mathrm{E}[x], \mathrm{Var}[x]$ - среднее и дисперсия соответственно, а γ, β - параметры сети, обучаемые методом обратного распространения ошибки. При этом, на этапе работы сети на исполнения вместо величин $\mathrm{E}[x], \mathrm{Var}[x]$ используются их скользящие средние версии, накопленные на всем этапе обучения [40], либо вычисленные по всем датасету в конце обучения [37].

Следует обратить внимание, что коэффициенты γ, β - это вектора, размерность которых совпадает с числом каналов у входного тензора, а значит умножение на эти параметры выполняется поканально. Таким образом становится ясно, как эти величины могут быть использованы в качестве критерия для прунинга: чем меньшее значение γ^i, тем менее значим канал, на который он умножается.

Хотелось бы отметить тот факт, что в основе приведенных выше критериев лежат интуитивные соображения, а не теоретическое обоснование. Авторы работы [41] попытались исправить данный недостаток и предложили свой критерий прунинга. Данный критерий выводится из того факта, что при удалении малозначимых весов изменения функции потерь должны быть минимальные и разложения этой функции потерь в ряд Тейлора:

$$S_{\theta_i} = \theta_i * \frac{\partial \mathcal{L}}{\partial \theta_i}$$

где θ_i - какой-либо параметр в нейронной сети. Весовые коэффициенты, у которых значение критерия S минимальное, считаются малопригодными и могут быть удалены из сети. В продолжении [42] своих исследований авторы выводят, как подобный критерий можно применить к группе весов и таким образом выполнять структурированный прунинг.

Обсуждая все вышеперечисленные способы прунинга, мы подразумевали, что процесс удаления весовых коэффициентов происходит один раз (в англоязычной литературе авторы именуют такие процедуры термином one-shot) и до

желаемого нами уровня ускорения с последующей процедурой тонкой настройки. Однако, можно попробовать выполнять удаление и дообучение итеративно, чтобы давать нейронной сети возможность вернуться в оптимальное состояние. Подобный подход значительно улучшает итоговое качество сети, что было продемонстрировано в работе [11], результаты которой приведены на рисунке 1.7.

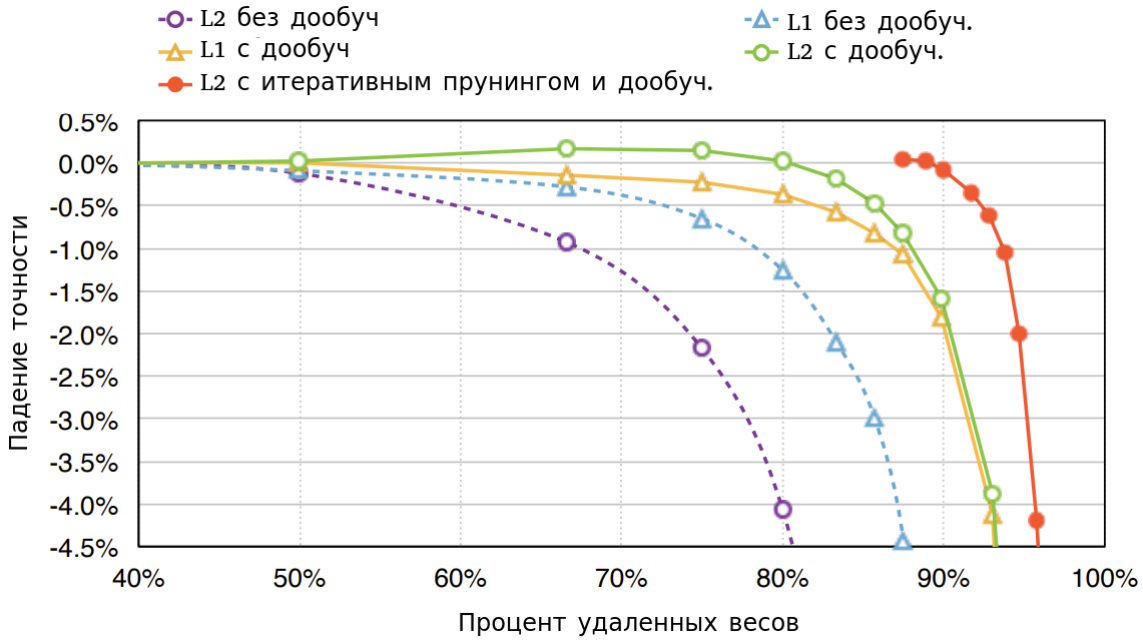

Рисунок 1.7 — Преимущество итеративного прунинга на сети AlexNet [1] на наборе данных [43].L1/L2 - тип критерия прунинга.

Прунинга на сегодняшний день является одним из самых доступных подходов ускорения нейронных сетей для рядового пользователя. Однако, у этих методов есть ряд недостатков:

1. Хорошо работают только с избыточными сетями;
2. Итеративная процедура, которая обеспечивает лучшую точность по сравнению с one-shot процедурой, требует много вычислительных ресурсов;
3. Процент удаленных весовых коэффициентов на каждом отдельном слое становится дополнительным гиперпараметром, который требует отдельной настройки.

1.4 Квантование

Несмотря на высокую вычислительную эффективность мобильных нейронных сетей, можно добиться увеличения скорости их работы путем перевода вычислений в целочисленные типы данных, поддерживаемые мобильным процессором. Подобная процедура называется в научной литературе “квантование” нейронных сетей [13], [44], [45].

В общем случае, процедура квантования нейронной сети подразумевает дискретизацию как весов, так и входных значений каждого слоя. Отображение из пространства float32 величин в пространство знаковых целочисленных величин с n значащими разрядами int-n для весовых коэффициентов задается следующими формулами [17]:

$$T_w = |W|_{max} \tag{1.1}$$

$$S_W = \frac{2^{n-1} - 1}{T_W} \tag{1.2}$$

$$W_{int} = \lfloor S_W * W \rceil \tag{1.3}$$

$$W_q = max(min(W_{int}, 2^{n-1} - 1), -2^{n-1} + 1) \tag{1.4}$$

где $\lfloor \rceil$ - это округление до ближайшего целого, $|W|_{max}$ - максимальное значение по модулю от всего тензора W, а функции min и max применяются к каждому элементу тензора. Величину T_W будем называть порогом квантования для весов, а также введем функцию

$$clip(x, a, b) = max(min(x, b), a)$$

где min и max действуют так же, как в формуле 1.4.

Квантование входных карт признаков I выполняется аналогичным образом. При этом, в связи с распространением функции активации $ReLU(x) = max(x, 0)$ квантоваться они могут как в беззнаковые целые (что дает большую точность), так и в знаковые целые числа (что унифицирует процедуру квантования):

$$S_I = \frac{2^n - 1}{T_I} \tag{1.5}$$

$$I_{int} = \lfloor S_I * I \rceil \tag{1.6}$$

$$I_q = max(min(I_{int}, 2^n - 1), 0) \tag{1.7}$$

Необходимо отметить, что существует два способа считать порог квантования T_I: статически и динамически. Динамическое вычисление порога квантования подразумевает вычисление его каждый раз для каждого тензора, подаваемого на вход слоя, что может быть сопряжено с дополнительными затратами вычислительных ресурсов.

Альтернативой этому является статическое квантование, когда T_I рассчитывается заранее на предзаготовленном наборе типичных данных, называемых калибровочными, а сам процесс называется калибровкой [46]. Статическое квантование лишено накладных расходов и работает быстрее, чем динамическое, однако у него имеется существенный недостаток, который обсуждается подробно в разделе 2.2 и может привести к снижению точности.

После того, как входы и веса нейронной сети квантованы, процедура свертки или матричного умножения выполняется обычным образом. Однако необходимо упомянуть, что для результирующего значения операции необходимы аккумуляторы большей разрядности чем операнды (в работе [44] используют схему, когда веса и активации квантуются в 8 бит, а аккумуляторы представляют собой 32-битные значения). Из этого можно сделать вывод, что пользователю необходим специальный программный комплекс, в котором подобная процедура была бы реализована.

Методы квантования можно разделить на два больших класса: методы, работающие **без дополнительного обучения** [45] или **с обучением** [44].

Немногие фреймворки предоставляют возможность квантования нейронных сетей без тонкой настройки. Яркими представителями являются TensorRT [45], квантование от фреймворка Tensorflow [46] и фреймворк distiller [47] от NervanaSystems. Однако, в последних двух расчет коэффициентов квантования происходит непосредственно во время работы нейронной сети, что может значительно снизить скорость работы на мобильном устройстве. Поэтому, наиболее распространенными являются методы с дообучением.

Большинство исследователей в этой области стараются создать процедуру обучения нейронной сети таким образом, чтобы в процессе квантования падение точности были незначительными. Приоритетными в данной теме можно считать работы [44; 48—50]. Авторы используют State Through Estimator [51] (далее, STE) для обучения весов нейронных сетей в 2 или 3 битное представление, однако, точность таких сетей значительно ниже, чем у float32 аналогов.

Необходимо отметить использование STE. Эта методика “пропускает” градиент через недифференцируемые операции (рис. 1.8), таким образом позволяя применять методы обучения, основанные на градиентном спуске.

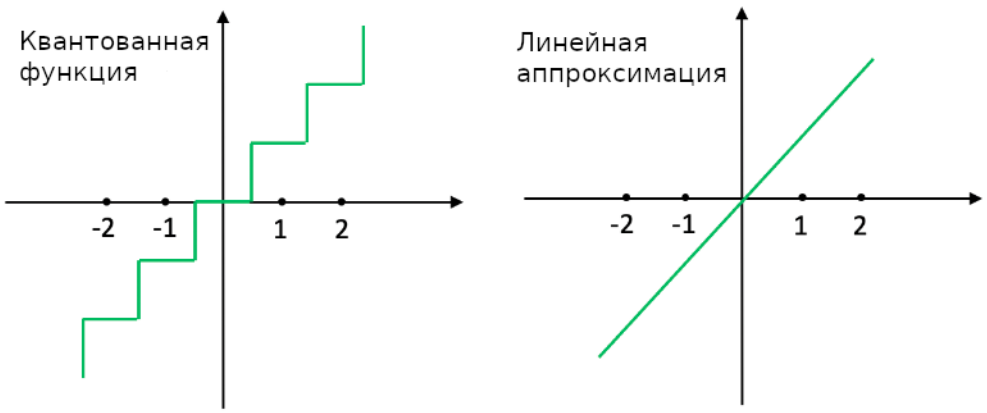

Рисунок 1.8 — Представление значений функции квантования и её линейной аппроксимации для вычисления градиента.

Одними из перспективных статей в области квантования являются [52; 53], где качество обученных моделей практически не отличаются от оригинальных архитектур. Более того, авторы в [52] поднимают отдельный вопрос касательно создания ансамбля квантованных сетей, что является интересной темой и может принести пользу для практического применения нейронных сетей, квантованных в двоичное представление.

Отдельно стоит отметить работу [44], в которой авторы предоставили целый фреймворк для модификации архитектуры сети, с возможностью последующего запуска обученных квантованных моделей на мобильных устройствах. Также в этой работе впервые в публичном поле появился термин fake-quantization, описывающий процесс симуляции квантования. Математически, это последовательно применяемые процедуры квантования (формулы 1.1

- 1.4) и деквантования, то есть возвращение в исходное пространство чисел float32:

$$W_{fq} = \frac{W_q}{S_W} \tag{1.8}$$

Также авторы предоставили методику модификации сверточных слоев с учетом последующих слоев Batch Normalization [37] (рис. 1.9). Это позволяет процедуру квантования свести к более простой версии. Необходимо, однако, учитывать, что эта модификация выполняется в режиме реального времени в процессе обучения, так что программная реализация подобного алгоритма требует высокой квалификации исполнителя.

В [53] авторы применяют процедуру обучения порога, что отчасти похоже на методику, предложенную в данной диссертационной работе. Однако есть два недостатка, не позволяющих использовать работу авторов для быстрого импорта уже заранее обученных нейронных сетей на мобильные устройства: это тренировка порога на полном наборе ImageNet [43] и отсутствие демонстрации качества на архитектурах сетей, которые являются де-факто стандартными для мобильных платформ.

Таким образом, можно сформулировать следующие недостатки методов квантования:

- Квантование сети без дообучения происходит быстро, поскольку квантуется заранее обученная модель, однако даже с использованием продвинутых статистических методов точность может существенно уступать оригинальной для мобильных архитектур нейронных сетей.
- Основным недостатком квантования с обучением является высокая трудоемкость процедуры тонкой настройки, что существенно ограничивает его применимость.

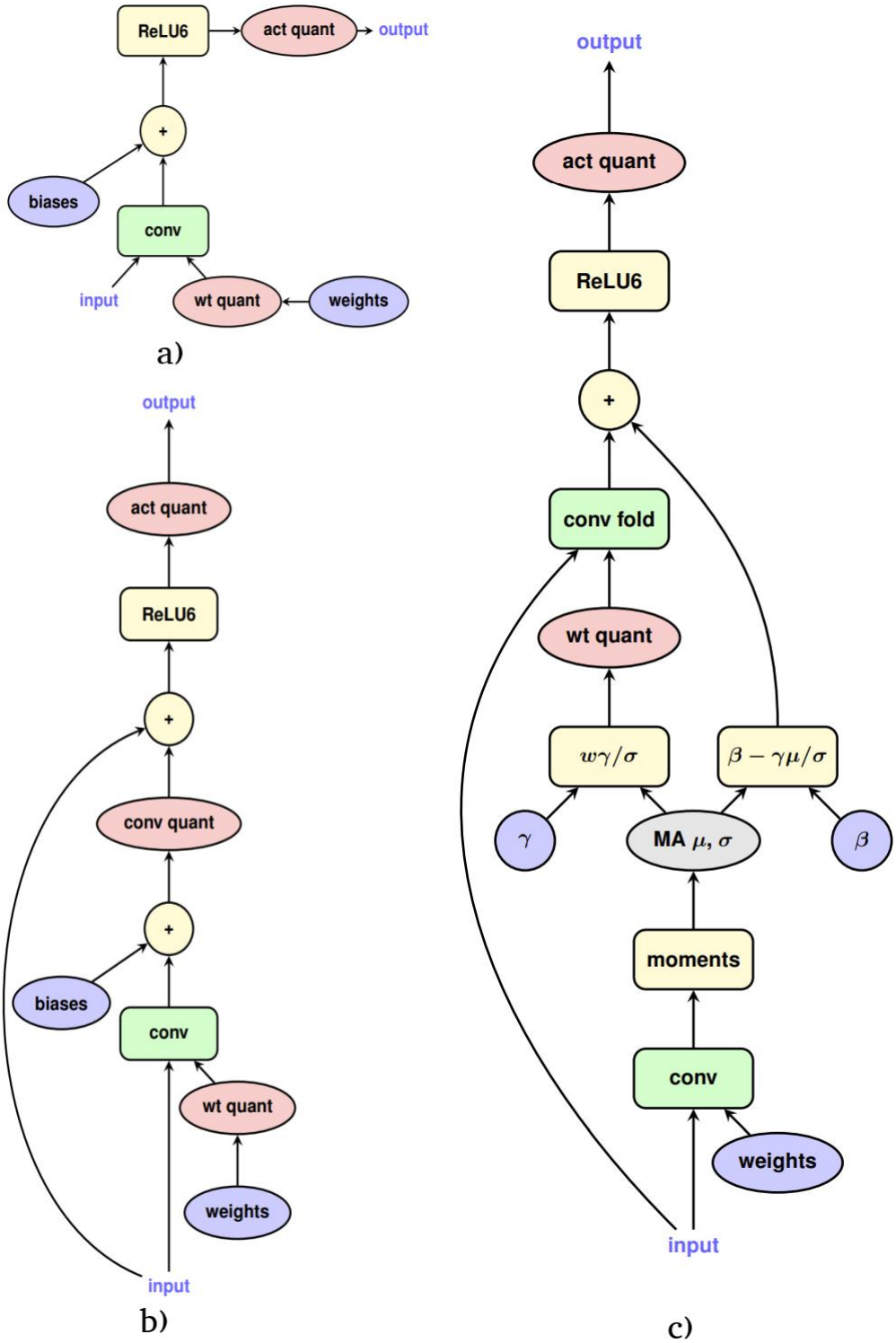

Рисунок 1.9 — Процедура симуляции квантования для наиболее распространенных блоков в нейронных сетях. Розовый овал - непосредственное применение указанного алгоритма. a) квантование обычного сверточного слоя b) квантование сверточного слоя с остаточными связями [25] c) Процедура сплавления слоя Batch Normalization [37] со сверточным слоем

1.5 Дистилляция

Дистилляция (или процедура переноса знаний) - это процесс обучения одной нейронной сети (сеть-ученик) при помощи другой, заранее обученной сети (сеть-учитель). При дистилляции ученик стремится предсказать не истинную разметку на примере, а то, что предсказал учитель на этом примере.

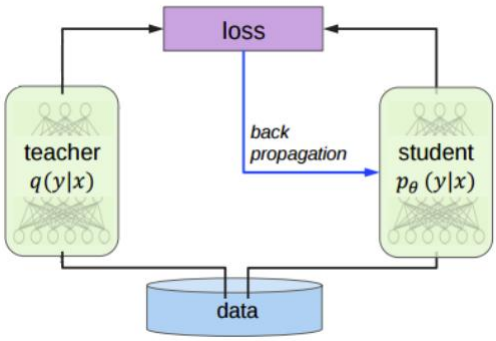

Рисунок 1.10 — Схематическое изображение процесса дистилляции

Впервые этот термин был введен в работе английского исследователя Джоффри Хинтона [14]. В своей статье автор убедительно показывает, что процедура дистилляции увеличивает точность сети-ученика не только на задаче обработки изображений, но еще и на задаче распознавания речи. Разберем интуицию, стоящую за этим подходом на примере задачи классификации изображений.

Сеть-учитель в результате тренировки усваивает некоторые зависимости между разными типами объектов в обучающей выборке и стремится передать эту информацию ученику. Возьмем обученную на датасете Imagenet [43] нейронную сеть ResNet-50 [25], выберем класс English Setter, построим предсказания данной нейронной сети на этом классе (рис. 1.11). Процедура построения была осуществлена следующим образом:

1. Из набора данных, который не участвовал в обучении сети, датасета ImageNet [43] выбирались примеры с разметкой, соответствующей классу English Setter;

2. Нейронная сеть запускалась на выбранном поднаборе, при этом выходом считался результат вычисления функции softmax:

$$softmax(z_k) = \frac{exp(z_k)}{\sum_{j=0}^{K} exp(z_j)}$$

где K - это число классов в обучающем наборе данных. Данная функция применялась для каждого элемента выходного вектора нейронной сети $z_k, k \in [0, K]$;

3. Полученные вектора предсказаний усреднялись по классам, в результате чего и получилась "средняя" вероятность, изображенная на рисунке 1.11.

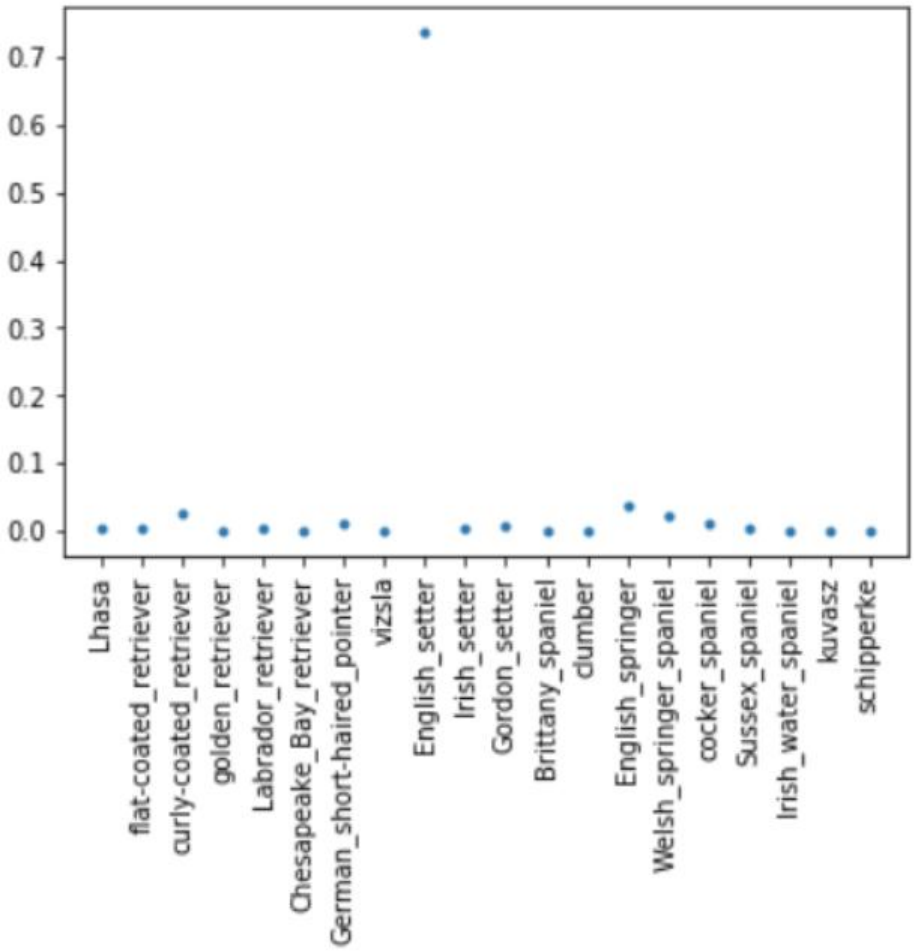

Рисунок 1.11 — Усредненные предсказания нейронной сети ResNet-50 для класса English Setter на датасете ImageNet.

Можно заметить, иногда тестируемая нейронная сеть ошибается и предсказывает класс объекта не верно. При этом, если визуализировать второй класс по величине вероятности предсказания, то это будет класс English Springer. Эти

породы собак изображены на рисунке 1.12. Как можно увидеть, эти виды собак очень похожи, а значит сеть выучивает какие-то признаки, которые являются общими и значимыми для этих классов. Следовательно, подав в качестве разметки предсказания учителя, мы “заставим” сеть обратить на эти признаки внимание. Выучивая значимые признаки у сети увеличивается обобщающая способность, что ведет к повышению точности.

Рисунок 1.12 — Представители пород English Setter и English Springer

Немного ранее появилась работа [54], которая предлагает выполнить процедуру переноса знаний не только с последнего слоя нейронной сети, но еще и с промежуточных слоев, используя MSE-функцию потерь. При этом, в [14] было показано, что для предпоследнего слоя это является частным случаем дистилляции по выходам сети с центрированным распределением признаков.

Дистилляцию можно воспринимать не только как способ сократить размеры сети, но еще и способ увеличить точность моделей. Это использовалось в работах [55; 56], где дистилляция была успешно скомбинирована с процедурой квантования в следующем виде: в качестве модели-учителя использовалась float32 модель, а моделью-учеником являлась квантованная нейронная сеть. При такой парадигме обучения наблюдается не только улучшение точности в работе квантованных сетей, но еще и появляется возможность снизить разрядность квантованных данных при приемлемом уровне падения точности (см. рис 1.13).

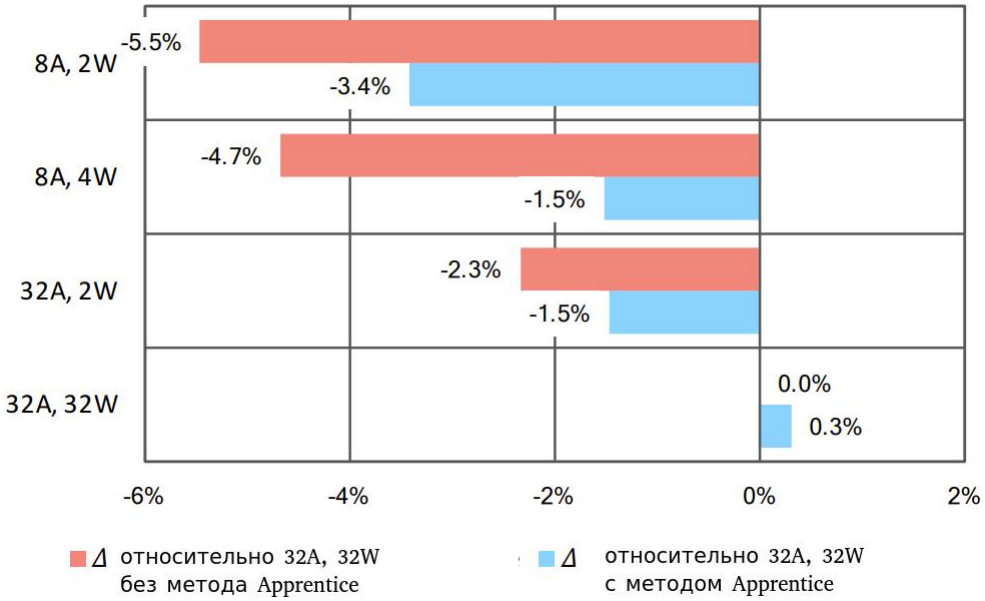

Рисунок 1.13 — Преимущества дистилляции при квантовании ResNet-50. A - обозначает выходы слоев, а W - весовые коэффициенты. Цифры обозначают количество бит, в которое выполняется процедура квантования.

Преимуществом метода дистилляции - является эмпирический факт, что она позволяет увеличить точность практически во всех задачах, где применяется. Однако, на пользователе по прежнему лежит обязанность выбрать тип сети-ученика, что может потребовать высокой квалификации и глубокого понимания области.

1.6 Использование нестандартных типов данных

Другим способом ускорения нейронных сетей может являться использование особых типов чисел с плавающей запятой, у которых разрядность мантиссы и порядка отличается от общепринятого стандарта IEEE-754 [57]. Представление числа в памяти (рисунок 1.14) содержит последовательно: один бит, определяющий знак числа, и определенное число бит, содержащих смещенный показатель порядка числа. Оставшиеся биты формируют значение мантиссы числа. Обозначение minifloat<e,m> показывает, что под представление порядка выделено e бит, а под мантиссу — m бит. В числе одинарной точности отводит-

ся 8 бит под представление порядка и 23 бита под представление мантиссы. В стандартном числе половинной точности используется 5 бит и 10 бит для порядка и мантиссы соответственно.

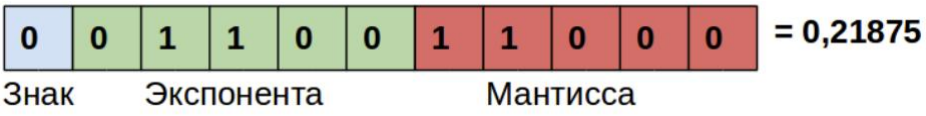

Рисунок 1.14 — Пример представления числа в памяти

Соответствие стандарту при этом заключается в сохранении правил получения десятичного числа из бинарного представления (формула ниже), а также в наличии специальных состояний, обозначающих ± 0 , $\pm \inf$ и $\pm NaN$ (not a number).

$$F = (-1)^s 2^{E-2^{e-1}+1}(1 + \frac{M}{2^m})$$

В формуле выше, e, m - число бит, которые выделены под порядок и мантиссу, а E, M - это значения, хранящиеся в порядке и мантиссе, и переведенные в десятичное представление. Иногда, выражение вида $2^{e-1}-1$ называют сдвигом или смещением [57]. Стандарт также предусматривает два состояния числа с плавающей запятой: нормализованное и денормализованное. Особенность денормализованных чисел — это иная логика интерпретации мантиссы. При значении показателя порядка, равном нулю, к мантиссе не прибавляется неявная единица:

$$F = (-1)^s 2^{E-2^{e-1}+2}(\frac{M}{2^m})$$

Это позволяет получить дополнительную точность представления чисел в окрестности нуля, однако ведет к усложнению аппаратной реализации и уменьшению быстродействия.

Известно, что нейронные сети способны без потери точности работать на современных видеокартах с использованием типа данных float16, причем не только на исполнение, но еще и обучаться в этом типе данных. Стандарт чисел с плавающей запятой половинной точности зафиксирован в IEEE-754 и имеет широкую аппаратную и программную поддержку. Кроме того, в тензорных

процессорах компании Google активно используется нестандартный тип данных bfloat16 [46], который характеризуется более широким диапазоном значений порядка числа.

В работе [58] показано, что нейронные сети GoogleNet и DeepSpeech-v1 сохраняют качество своей работы при 12-ти разрядном minifloat, в котором 5 бит отведено под представление порядка числа и 7 бит — под представление мантиссы. В работе ARM Research [59] провели эксперименты со сверточными нейронными сетями AlexNet, VGG-16, GoogleNet. Эти эксперименты показали, что перечисленные нейронные сети способны работать практически без потери точности при использовании 10-ти разрядных fixed-point активаций и 8-ми разрядных minifloat.

Аналитики данных из исследовательского центра Томаса Ватсона [60] компании IBM провели объемную работу по исследованию возможности обучения весов нейронной сети. У них получилось успешно тренировать нейронную сеть, используя 8-ми разрядное представление чисел с плавающей запятой для весов нейронных сетей AlexNet, ResNet-18, ResNet-50. Обученные для minifloat сети работают без потери точности с float16 аккумулятором для промежуточных вычислений.

1.7 Матричные и тензорные разложения

Матричным разложением называется представление матрицы в виде произведения нескольких. Наиболее известные матричные разложения встречаются в курсе линейной алгебры: сингулярное разложение или SVD (от англ. Singular Value Decomposition) [61], жорданова нормальная форма, QR-разложение и другие [62].

Особого внимания заслуживает SVD, так как зачастую возникает в очень многих областях машинного обучения:

– классический метод главных компонент для сокращения размерности основан на SVD [63];

– в рекомендательных алгоритмах [64];

– в цифровой обработке изображений [65];

– в биоинформатике [66]

и во многих других областях, поэтому и задачу ускорения нейронных сетей подобный алгоритм не мог обойти стороной. Сингулярное разложение матрицы W задается следующей формулой:

$$W = U\Sigma V^T$$

где матрицы U и V ортогональные (то есть $U^T * U = V^T * V = 1$), а Σ - это диагональная матрица с неотрицательными элементами (называемыми сингулярными числами). Количество ненулевых элементов на главной диагонали будем обозначать r и назовем рангом матрицы.

Основная идея ускорения на основе сингулярного разложения заключается в следующем [67]:

1. Выбрать слой, действие которого представимо в виде матричного умножения (то есть $O = I * W$, где I входная матрица, W матрица весовых коэффициентов);
2. Выполнить процедуру сингулярного разложения: $W = U\Sigma V^T$;
3. Удалить наименьшие сингулярные числа, уменьшив таким образом ранг матрицы $\hat{r} < r$;
4. Удалить столбцы матрицы U, соответствующие удаленным сингулярным числам.
5. Выполнить процедуру умножения $Z = \Sigma V^T$
6. Матричное умножение на W заменить на два последовательных умножения, то есть $O = I * U * Z$

Пусть I - это матрица размера [M, N], а W - матрица размера [N, Q]. Выполняя процедуру, описанную выше, получаем следующие величины ускорения:

$$\frac{M * N * r + M * r * Q}{M * N * Q} = \frac{r}{Q} + \frac{r}{N}$$

Подобную операцию можно применить и к сверточным слоям, однако необходимо использовать уже не матричные, а тензорные разложения, такие как: каноническое разложение (иногда используется термин CP-разложение) [12],

разложение Такера [68] или разложение в "тензорный поезд" [69], [70]. Детально вдаваться в эти методы не будем, однако приведем изображение (1.15) из работы [68], которое иллюстрирует недостатки этих подходов:

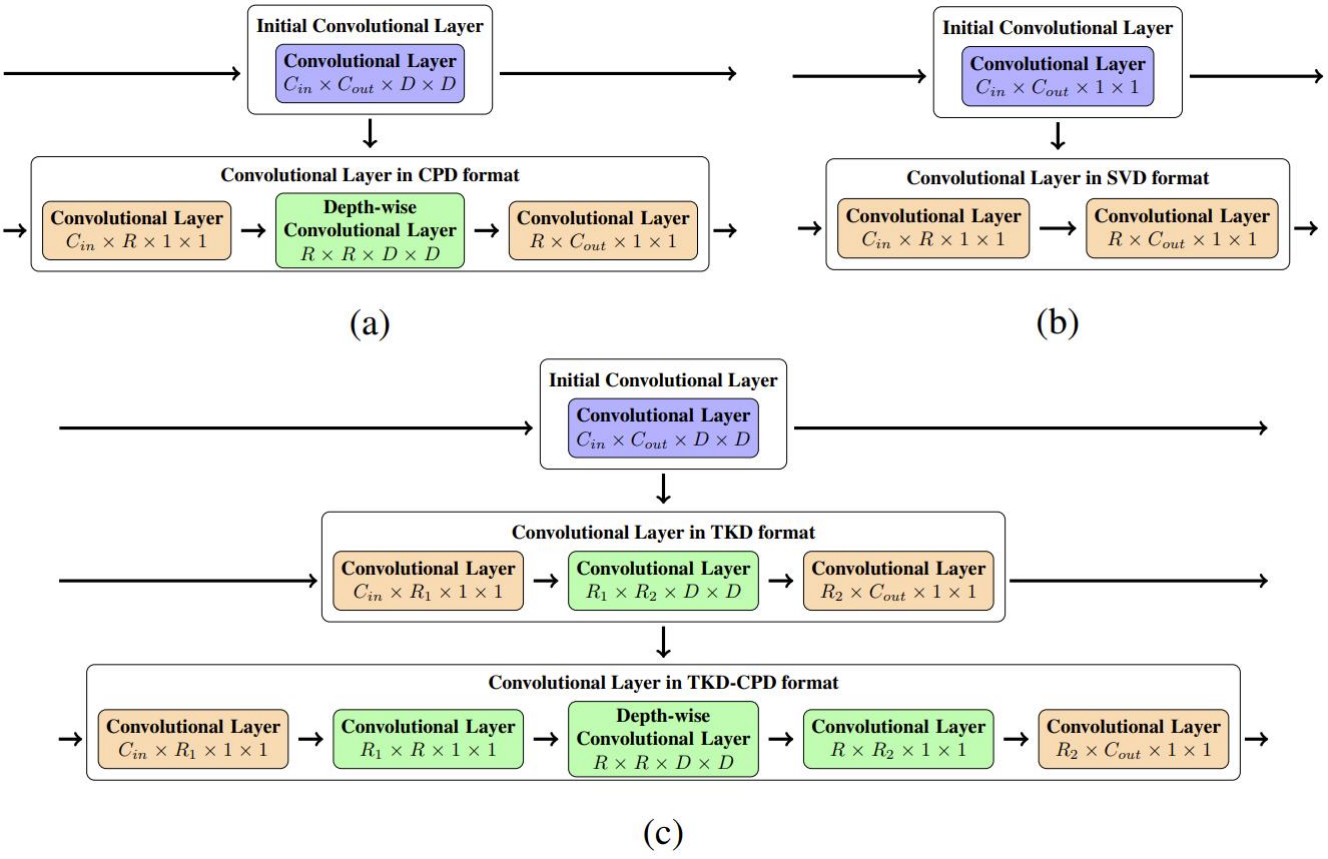

Рисунок 1.15 — Визуализация сверточных слоев после разложений различного типа и их объяснение в терминах “типов” сверток. a) каноническое разложение b) сингулярное c) разложение Такера и каноническое, примененные последовательно

Видно, что после применения CP-разложения к сверточному слою мы получаем блок, аналогичный блоку в [24], что значительно ограничивает применимость этих подходов к ускорению сетей с depth-wise сверткой, которые, как говорилось уже ранее, являются де-факто стандартом для работы на мобильных устройствах.

1.8 Выводы по первой главе

Разработка комплексных программных продуктов, где нейронная сеть - это одна из составляющих, является трудоемким делом не только с точки зрения самого процесса разработки, но еще и с точки зрения процесса управления. Действительно, если специалист по нейронным сетям ошибется и выберет неподходящие гиперпараметры обучения, некорректно размеченный обучающий датасет или неподходящую архитектуру сети, то процесс разработки может затянуться на недели, так как это характерные величины для обучения нейронных сетей. Таким образом не все алгоритмы, описанные выше, могут быть эффективно использованы для задачи ускорения нейронных сетей:

- Алгоритмы на основе матричных разложений не является эффективными, так как полученная сеть имеет архитектуру, которая совпадает большинством предобученных мобильных нейронных сетей;
- Алгоритмы НПА крайне сложны в процессе выбора гиперпараметров, а также крайне требовательны к вычислительным ресурсам;
- Алгоритмы прунинга требуют подбора гиперпараметров, а именно процент удаления каналов на каждом слое;
- Если рассматривать дистилляцию как алгоритм ускорения, а не как способ повышения точности, то тогда все проблемы с гиперпараметрами проявляют себя еще более остро: какую сеть выбрать в качестве ученика, какой метод дистилляции выбрать и т.д.

В то же время, у процедур квантования и использования нестандартных типов данных есть потенциал в плане использования постфактум, после процедуры обучения. Однако, существующие методы квантования требуют аналогичного процесса обучения сети, что в свою очередь может быть затруднительно для небольших команд разработчиков. В следующей главе будет предложен метод, позволяющий сократить процедуру обучения при сохранении точности квантованных архитектур. Разработанный метод может быть применен к произвольной архитектуре, что позволяет применять в большинстве современных приложениях.

Нестандартные типы данных в свою очередь не настолько применимы с практической точки зрения, однако при использовании в проектировании микропроцессоров такой подход является мощным инструментом для выбора разрядности будущего устройства. Действительно, выбрав оптимальный тип данных можно сократить площадь на кристалле для одного базового элемента матричного умножителя, сложителя-умножителя, что в свою очередь позволит освободить место на кристалле для дополнительной памяти и/или большего числа базовых единиц. Однако, использование нестандартных типов данных было проверено только на задачах компьютерного зрения, при этом применимость к современным мобильным архитектурам нейронных сетей не была исследована. Так же не была исследована применимость этого метода к другим популярным задачам, таким как распознавание речи.

Таким образом, основной целью диссертации является исследование применимости типов данных с сокращенной разрядностью на мобильных архитектурах нейронных сетей и на задаче распознавания речи, а также разработка оптимального алгоритма квантования с тонкой настройкой и дистилляцией.

Глава 2. Метод дообучения порогов как комбинация дистилляции и квантования

2.1 Описание процесса квантования нейронных сетей в программном пакете TensorflowLite

В разделе 1.4 было указано, что для выполнения квантованных вычислений требуется специальное программное обеспечение. При этом, некоторые особенности могут отличаться из-за так называемой схемы квантования - набора правил, по каким должно происходить квантование.

На сегодняшний день для запуска нейронных сетей на мобильных телефонах с процессором ARM де-факто стандартом является программный пакет TensorflowLite (далее будем использовать сокращение tflite), поэтому приведем схему квантования, принятую для него.

В tflite используется **несимметричное скалярное квантование**, то есть:

$$W_c = clip(W, T_{wl}, T_{wr}) \tag{2.1}$$

$$S_W = \frac{2^n - 1}{T_{wr} - T_{wl}} \tag{2.2}$$

$$W_q = \lfloor S_W * W - T_{wl} \rceil \tag{2.3}$$

$$\tag{2.4}$$

где функция *clip* работает аналогичным образом, как и в 1.4, $n = 8$, а T_{wl}, T_{wr} будем называть порогами квантования. Можно заметить, что все значения после процедуры квантования переводятся в целочисленные значения от 0 до 255, при этом симметрия относительно 0 отсутствует в данном квантованном пространстве. Скалярное квантование S_W в данном случае означает, что коэффициент S_W является скаляром, то есть один на весь тензор.

Необходимо отметить, что данная процедура превращается в симметричное квантование, описанное в 1.4 в формулах 1.2 - 1.4, при замене $T_{wl} = -T_{wr}$

Симметричное квантование, описанное в прошлых разделах, крайне просто реализуется на конкретных устройствах, однако оно не оптимально использует доступный спектр целых значений, что значительно снижает точность квантованных моделей, именно поэтому выбор авторов tflite пал на несимметричный подход.

2.2 Проблема выбросов в задаче квантования нейронных сетей

В своей статье [44] авторы отмечают, что качество квантованных моделей может быть значительно ниже float32 аналога. Из-за значений, которые отстоят далеко от основной массы (называемых выбросами) и которые выбираются в качестве порогов квантования, может происходить “размытие” данных вблизи нуля, что в свою очередь может негативно сказаться на качестве работы квантованной модели. На рисунке 2.1 продемонстрировано, что число значений, попавших в бины вблизи нуля, значительно увеличилось.

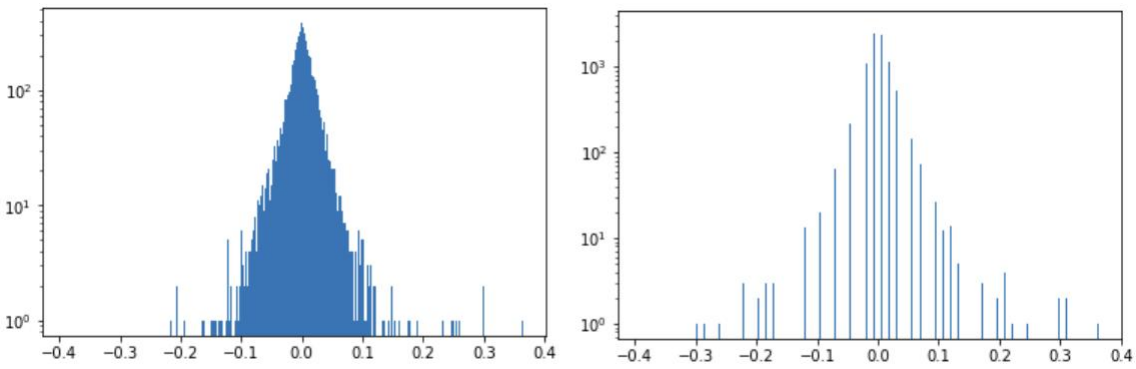

Рисунок 2.1 — Распределение весов нейронной сети ResNet-50 до процедуры квантования (слева) и после (справа)

Выбросы могут возникать как из-за особенностей калибровочного датасета (несбалансированность классов, нетипичные входные данные), так и могут быть неотъемлемой частью работы сети (например, сформировавшиеся выбросы у весов в процессе обучения сети или реакция отдельных нейронов на те признаки, которые принимают наибольшее значение). Из вышесказанного следует, что полностью избавиться от выбросов нельзя, так как они связаны с

фундаментальными особенностями нейронных сетей. Однако если постараться найти баланс между величиной максимального значения и “размытием” остальных величин при квантовании, то можно добиться лучшего качества у квантованной нейронной сети.

Авторы фреймворка tflite попытались решить эту проблему, добавив скользящие средние на значения порогов квантования, что приводит к усложнению процедуры обучения, а также не решает проблему полностью, поскольку при симуляции квантования эти самые пороги обновляются независимо от весов и недифференцируемым образом. В следующем разделе мы представим процедуру, способную исправить указанные проблемы.

2.3 Дифференцируемый порог квантования

Ряд исследователей и специалистов в области глубокого машинного обучения показали, что можно использовать STE [51] для введения производной разрывных функции, таких как $\lfloor\rceil, sign, clip$ [49], [71], [48]. Таким образом, используя STE, величину, являющуюся аргументом для этих функций, можно сделать дифференцируемой и настраивать методом градиентного спуска [6]. Автором данной диссертации предлагается рассмотреть порог квантования в качестве переменной, обучение которой способно напрямую привести к оптимальному качеству квантованной сети. При этом, предлагается обучать пороги с учетом fake quantization процедуры (см. 1.8), то есть учитывать потери квантования при процессе тонкой настройки.

Читателю предлагается рассмотреть предложенную методику сначала на простом примере с симметричным квантованием, а потом перейти уже к несимметричным порогом по схеме квантования tflite.

Масштаб порога. Для ускорения процедуры обучения предлагается перестать обновлять все остальные переменные, такие как веса сверточных слоев и полносвязных слоев, а обновлять только пороги квантования. В качестве начального значения порогов необходимо выбрать значение, вычисленное в

процессе калибровки для активаций, а для весов - максимальное абсолютное значение. При этом, порог квантования T из формул 1.1, 1.2 заменяется на

$$T_{adj} = clip(\alpha, min_\alpha, max_\alpha) \cdot T_{max} \tag{2.5}$$

где α - обучаемый параметр, который принимает значения от min_α до max_α. Значения данных параметров находятся эмпирически и равны 0.5 и 1.0 соответственно. Введение масштабирующего коэффициента упрощает обучение сети, так как обновления порогов непосредственно должны проводиться с разным learning rate для разных слоев нейронной сети, поскольку они могут иметь различный порядок значений (например, величина значений на промежуточных слоях сети VGG [2] может увеличиваться до 7 раз по сравнению с значениями на первых слоях). Таким образом, процедура в случае симметричного квантования будет выглядеть следующим образом:

$$T_{adj} = clip(\alpha, 0.5, 1) \cdot T_i \tag{2.6}$$

$$S_I = \frac{2^n - 1}{T_{adj}} \tag{2.7}$$

$$I_q = \lfloor I \cdot S_I \rceil \tag{2.8}$$

$$I_{fq} = \frac{I_q}{S_I} \tag{2.9}$$

Аналогичная процедура выполняется и для весов, чтобы при необходимости "поджать" выбросы в весовых коэффициентах. В данной схеме квантования у нас есть две недифференцируемые функции: $\lfloor\ \rceil$ и clip. Предлагается задать производные данных функций следующим образом (в разделе 2.5 будет приведен анализ с точки зрения алгоритма обратного распространения ошибки о допустимости такого подхода):

$$I_q = \lfloor I \rceil;\ \frac{dI_q}{dI} = 1 \tag{2.10}$$

$$X_c = clip(X, a, b);\ \frac{dX_c}{dX} = \begin{cases} 1, if\, X \in [a, b] \\ 0, otherwise \end{cases} \tag{2.11}$$

Квантование отступов выполняется аналогично работе [44]:

$$b_q = clip(\lfloor S_i \cdot S_w \cdot b \rceil, -(2^{31}-1), 2^{31}-1)$$

Предложенную методику лучше всего использовать с небольшими модификациями, изложенными ниже.

Сплавление слоя Batch Normalization. На сегодняшний день при обучении большинства нейронных сетей используется процедура batch normalization (BN), которая ускоряет сходимость нейронных сетей [37]. Прежде чем перейти к квантованию весов нейронной сети предлагается выполнить процедуру сплавления BN с весами сети, аналогично работе [44]. После этой процедуры получаем новые весовые коэффициенты, которые вычисляются по формулам ниже:

$$W_{fold} = \frac{\gamma W}{\sqrt{\sigma^2 + \varepsilon}} \tag{2.12}$$

$$b_{fold} = \beta - \frac{\gamma \mu}{\sqrt{\sigma^2 + \varepsilon}} \tag{2.13}$$

Предлагается применять процедуру квантования уже к весам, которые были сплавлены с BN слоями, поскольку это упрощает процедуру квантования и ускоряет работу нейронной сети в принципе, уменьшая количество операций сложения умножений. В дальнейшем, когда будут упоминаться веса нейронной сети, в данной диссертации будет иметься в виду W_{fold} (там, где это не оговорено иначе).

Векторное квантование. Иногда, из-за большого разброса значений весов, лучше выполнить процедуру дискретизации более мягко, квантуя разные фильтры сверточного слоя с разными порогами. Таким образом, вместо одного коэффициента квантования на весь сверточный слой (скалярное квантование), мы имеем группу коэффициентов (векторное квантование). Данная процедура не должна внести особой сложности при реализации на устройствах, однако позволяет существенно повысить точность работы квантованной модели. Значительное повышение точности наблюдается у моделей, в архитектуре которых используются Depth-wise separable свертки (яркими представителями являются сети MobileNet-v1 [23] и MobileNet-v2 [24]).

Обучение на неразмеченных данных. В большинстве работ, связанных с квантованием сетей, используется размеченный датасет для обучения порогов квантования или непосредственно весов сети. В данной работе предлагается отказаться от исходной разметки обучающих данных, что значительно ускорит переход от обученной неквантованной сети к квантованной, так как снизит требования к обучающему набору данных. Предлагается оптимизировать RMSE между выходами квантованной и оригинальной сети до применения функции softmax, при этом не меняя параметров оригинальной сети.

$$H(z^T, z^S) = \sqrt{\sum_{i=1}^{N} \frac{(z_i^T - z_i^S)^2}{N}} \tag{2.14}$$

где z^T, z^S - это выходы сети учителя и ученика соответственно, а N - количество изображений в батче.

Вышеописанную технику можно рассматривать с точки зрения дистилляции, где в качестве сети ученика используется квантованная модель, а оригинальная модель в качестве сети-учителя.

Хотелось бы дополнительно отметить, что выбор функции потерь RMSE был обусловлен еще и тем, чтобы не накладывать ограничения на решаемую нейронной сетью задачу. Действительно, ведь RMSE можно применить как к задаче классификации, так и к любым другим задачам [3].

Обучение несимметричных порогов. Рассмотрев процедуру обучения для симметричного квантования, теперь можно перейти к несимметричному. Для несимметричных порогов есть левая (T_l) и правая (T_r) границы распределения. Если обучаемые пороги ввести аналогично формуле 2.5, то может возникнуть ситуация, при которой левая граница больше чем правая (например, в силу неопытности пользователя, который неправильно подобрал гиперпараметры обучения или нестабильности обучения в силу шумной природы данных).

Для процедуры квантования удобнее перейти к двум другим обучаемым величинам: левой границе и ширине диапазона значений. Процедура обучения левой границы заключается в следующем. Мы должны ввести “смещение” для левой границы, поскольку если значение левой границы равняется 0, то мас-

штабирование такой величины никакого эффекта не даст. Она равняется:

$$R = T_r - T_l \tag{2.15}$$

$$T_{l\_adj} = T_l + clip(\alpha_T, min_{\alpha_T}, max_{\alpha_T}) \cdot R \tag{2.16}$$

Коэффициенты $min_{\alpha_T}, max_{\alpha_T}$ задаются эмпирически и равняются в случае знаковых переменных -0.2 и 0.4, а в случае беззнаковых 0 и 0.4. Ширина распределения подбиралась аналогичным способом, а значения min_{α_R}, max_{α_R} тоже задавались эмпирически и равнялись 0.5 и 1 соответственно:

$$R_{adj} = clip(\alpha_R, min_{\alpha_R}, max_{\alpha_R}) \cdot R \tag{2.17}$$

Тогда, схема квантования, приведенная в формулах 2.1 - 2.3 будет выглядеть следующим образом:

$$W_c = clip(W, T_{l\_adj}, T_{l\_adj} + R_{adj}) \tag{2.18}$$

$$S_W = \frac{2^n - 1}{R_{adj}} \tag{2.19}$$

$$W_q = \lfloor S_W * W - T_{l\_adj} \rceil \tag{2.20}$$

2.4 Результаты экспериментов

Исследуемые архитектуры. Процедура квантования для архитектур, которые обладают высокой избыточностью, не несет практической значимости, поскольку данные разновидности нейронных сетей плохо пригодны для мобильных устройств общего назначения. Было принято решение провести исследования как на тех нейронных сетях, которые де-факто считаются стандартом для мобильных устройств (MobileNet-v2 [24]), так и на относительно новых (MNasNet [26]).

Метрика качества и набор данных для тестирования. Поскольку указанные выше архитектуры нейронных сетей решают задачу классификации, то и тестирование производилось на датасете [43], а в качестве основной метрики была выбрана top-1 точность, то есть доля правильных предсказаний сети

на валидационной выборке. Набор данных [43] представляет собой изображения разного входного разрешения с общим количеством классов 1000. Типичные примеры изображений приведены на рисунке 2.2. Необходимо уточнить, что в [43] присутствуют не только объекты принципиально разные, но и, например, разные виды змей, которые обозначаются как разные классы для усложнения задачи распознавания (см. рисунок 2.3). Все архитектуры тестировались с разрешением входного изображения 224x224, то есть входное изображение интерполировалось до указанного разрешения при помощи билинейной интерполяции [72].

Рисунок 2.2 — Типичные объекты из набора данных [43]

Процедура тонкой настройки. Как уже было упомянуто ранее в разделе “Обучение на неразмеченных данных”, в качестве функции потерь использовался корень из среднеквадратичной ошибки (RMSE) между оригинальной и квантованной сетями. Для обучения использовался оптимизатор

Рисунок 2.3 — Объекты похожего типа, но отмеченные как разные классы [43]

Adam [6], а к learning rate применялся косинусный отжиг (см. рисунок 2.4) со сбросом параметров оптимизатора. Начальное значение learning rate равняется 0.001.

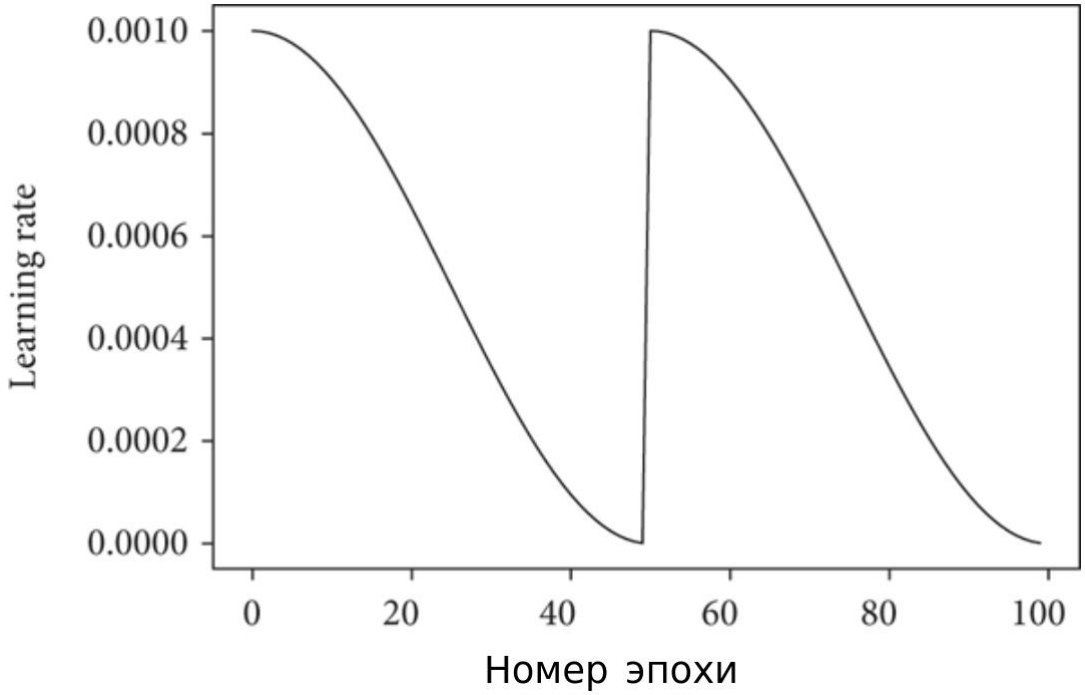

Рисунок 2.4 — Пример косинусного расписания.

Таблица 1 — Квантование в 8 бит в скалярном режиме

Architecture	Symmetric thresholds, %	Asymmetric thresholds, %	Original accuracy, %
MobileNet v2	8.1 ±2.8	19.86 ±1.2	71.55
MNas-1.0	72.42 ±0.3	73.46 ±0.1	74.34
MNas-1.3	74.92 ±0.12	75.30 ±0.17	75.79

Таблица 2 — Квантование в 8 бит в векторном режиме

Architecture	Symmetric thresholds, %	Asymmetric thresholds, %	Original accuracy, %
MobileNet v2	71.11 ±0.22	71.39 ±0.34	71.55
MNas-1.0	73.96 ±0.25	74.25 ±0.17	74.34
MNas-1.3	75.56 ±0.1	75.72 ±0.11	75.79

Обучение происходило не на всем наборе данных ImageNet [43], а на 100000 случайных изображениях, тестирование проводилось на всем валидационном наборе. В качестве калибровочных данных бралось 100 изображений из обучающего набора. Тренировка проходила от 6 до 8 эпох в зависимости от сети, что значительно меньше чем в работах [44], [53]. Качество квантования сетей представлены в таблицах 1 и 2.

По результатам эксперимента, скалярное квантование в MobileNet v2 показало крайне низкую точность. Это может быть связано с тем, как выглядит распределение масштабирующих коэффициентов в слоях Batch Normalization, а также с использованием функция активации $ReLU6(x) = clip(x, 0, 6)$, негативное влияние которой на процесс квантования сети указывалось в работе [73]. Необходимо отметить, что если использовать векторные пороги квантования, то точность квантованной MobileNet v2 и других исследуемых нейронных сетей практически совпадает с оригинальной.

Для реализации был выбран фреймворк Tensorflow [46], поскольку он обладает достаточной гибкостью и удобством для дальнейшего использования

моделей на мобильных устройствах, а заранее обученные сети взяты с репозиториев Slim [74] и Tensorflow [75].

Время float32 модели MobileNet-V2 на телефоне Xiaomi Redmi Note 4X (процессор Qualcomm Snapdragon 625) составило 265±30 мс, ее квантованной скалярной версии 184±25 мс, а время работы ее векторно-квантованной версии составило 190±30 мс. Для float32 модели MNasNet-1.0 времена на том же телефоне равны 293±30 мс, 210±35мс, 225±35 мс соответственно. Таким образом, предложенные мобильные архитектуры удалось ускорить в 1.44 и 1.4 раза соответственно. Необходимая для воспроизведения результатов кодовая база доступна на сервисе хранения программного кода GitHub [76].

Поскольку дообучаемые пороги меняют диапазон возможных значений, то уменьшается шаг дискретизации нейронной сети, что повышает точность после процедуры дообучения. В уменьшении шага дискретизации (как весов, так и входных тензоров) можно убедиться, построив гистограммы(см. рис. 2.5 и 2.6). Действительно, на рисунках, приведенных ниже, поменялись максимальные диапазоны для левой и правой границы, а также уменьшилась высота бинов, что говорит о более “эффективном” использовании диапазона квантования.

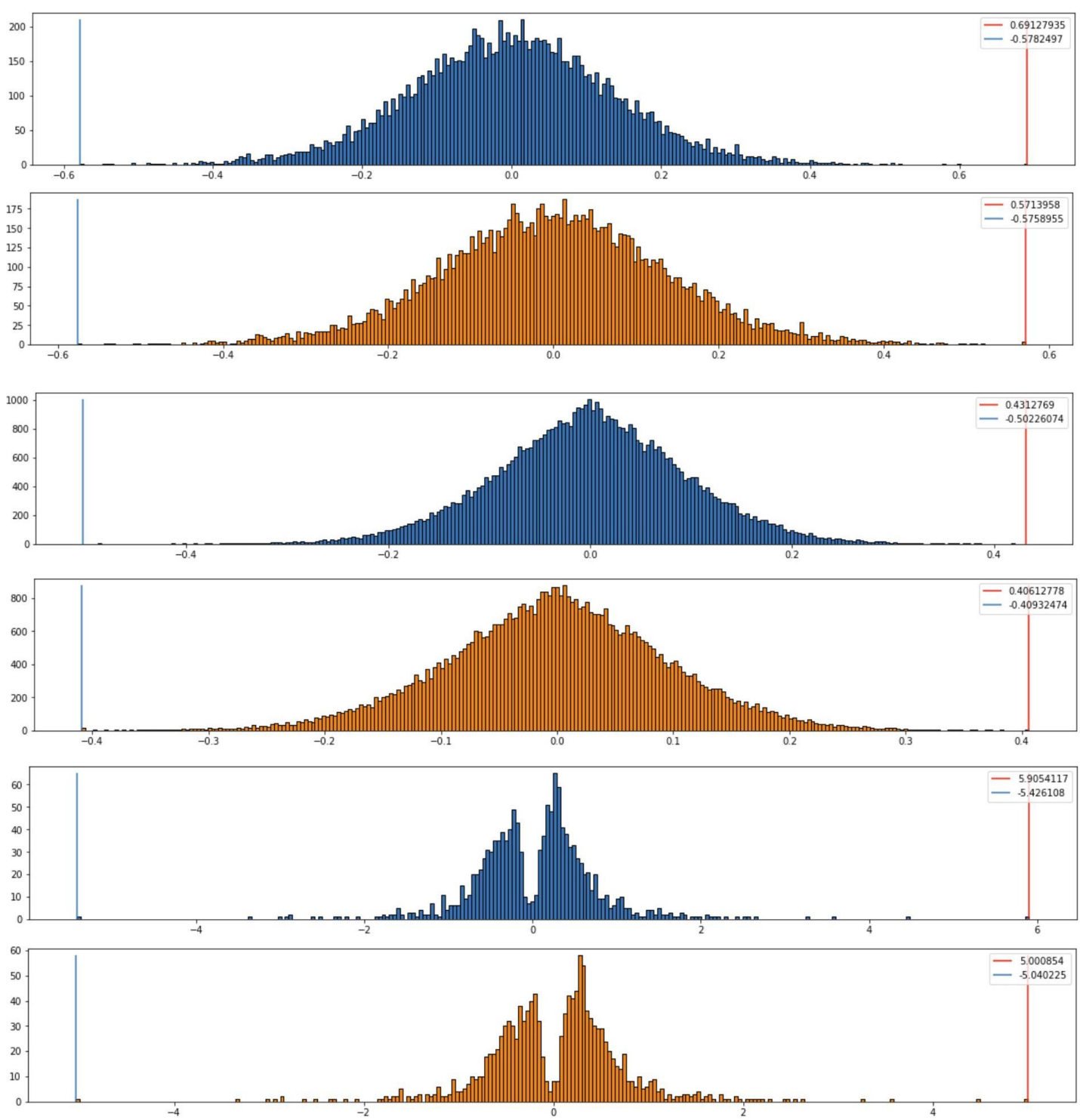

Рисунок 2.5 — Гистограммы весов до и после процедуры тонкой настройки порогов квантования различных слоев у сети MobileNet-v2. Синим изображены гистограммы оригинальной сети, а оранжевым - квантованной.

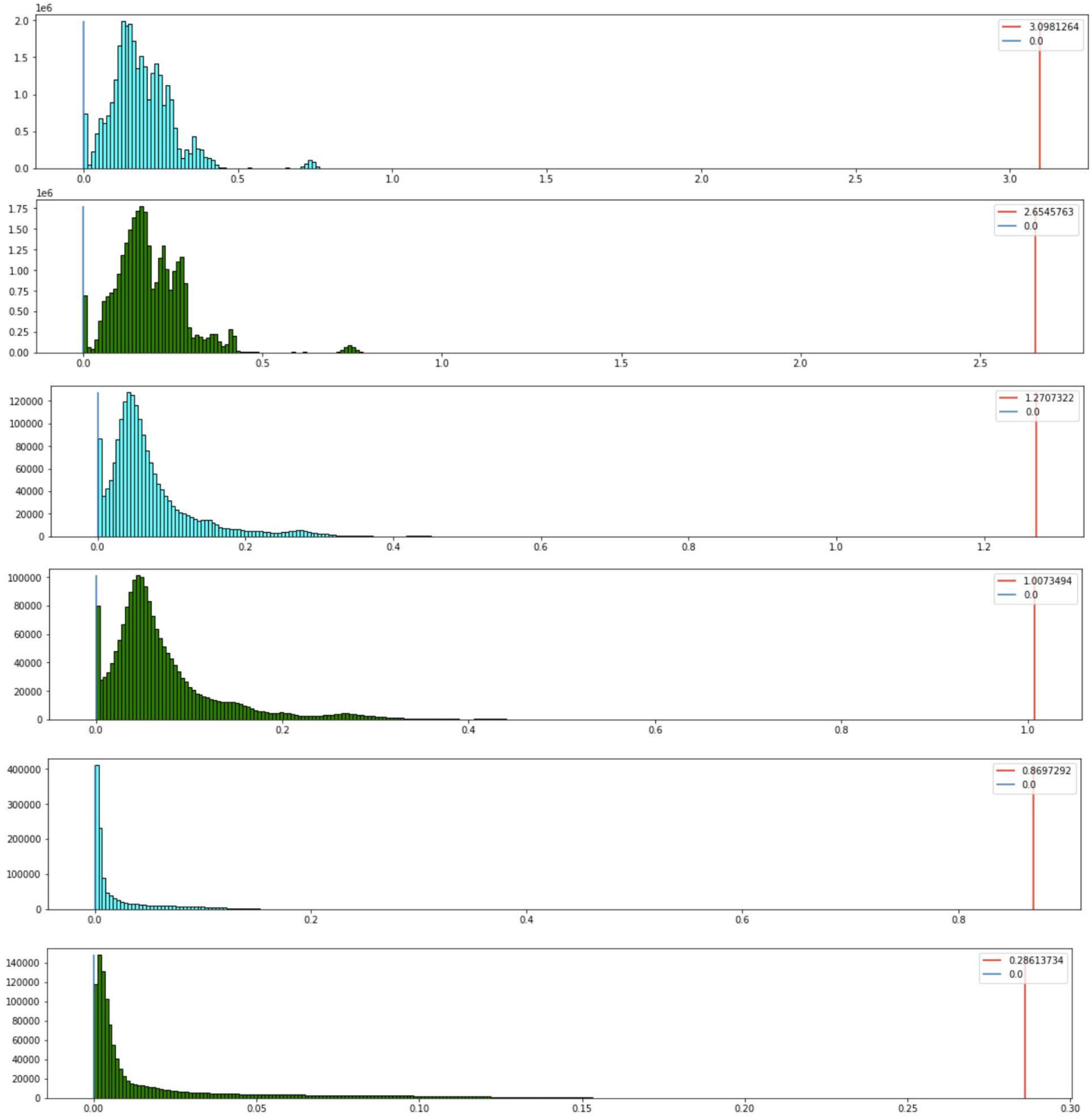

Рисунок 2.6 — Гистограммы активаций до и после процедуры тонкой настройки порогов квантования различных слоев у сети MobileNet-v2. Бирюзовым изображены гистограммы оригинальной сети, а зеленым - квантованной.

2.5 Анализ алгоритма быстрых дообучаемых порогов с точки зрения обратного распространения ошибки

Рассмотрим предложенный алгоритм с точки зрения обратного распространения ошибки. Это позволит понять, как предложенная процедура влияет на нейронную сеть. Введем следующие ограничения:

– Квантование будет симметричное;
– Квантование будет выполнено в случае полносвязного слоя;
– Квантование будет применимо только для весов;
– У полносвязного слоя будет один нейрон на выходе.

Выше приведенные условия не умаляют общности, при этом модификация анализа на другие режимы работы (векторное, асимметричное квантование) и другие типы слоев (сверточные) не составляет труда, однако приводиться в данной диссертационной работе не будет. Также для простоты предлагается рассмотреть производную не по обучаемому порогу, а по множителю S_W

Пусть W - весовые коэффициенты данного полносвязного слоя, I - входная матрица, f - функция активации, а $\mathcal{L}$ - функция потерь. Тогда симуляция квантования в данном слое будет выглядеть следующим образом:

$$\hat{W} = W \cdot S_W \tag{2.21}$$

$$W_q = \lfloor \hat{W} \rceil \tag{2.22}$$

$$W_{fq} = \frac{W_q}{S_W} \tag{2.23}$$

$$O = W_{fq} I + b \tag{2.24}$$

Рассмотрим, чему будет равна производная $\frac{\partial \mathcal{L}}{\partial S_W}$:

$$\frac{\partial \mathcal{L}}{\partial S_W} = \frac{\partial \mathcal{L}}{\partial O}\frac{\partial O}{\partial S_W} = \frac{\partial \mathcal{L}}{\partial O}\frac{\partial O}{\partial W_{fq}}\frac{\partial W_{fq}}{\partial S_W} = \frac{\partial \mathcal{L}}{\partial O} I^T \frac{\partial W_{fq}}{\partial S_W} \tag{2.25}$$

$$\frac{\partial W_{fq}}{\partial S_W} = \frac{\partial}{\partial S_W}\frac{\lfloor W \cdot S_W \rceil}{S_W} = -\frac{\lfloor W \cdot S_W \rceil}{S_W^2} + \frac{1}{S_W}\frac{\partial \lfloor W \cdot S_W \rceil}{\partial S_W} = \tag{2.26}$$

$$-\frac{\lfloor W \cdot S_W \rceil}{S_W^2} + \frac{1}{S_W}\frac{\partial W_q}{\partial \hat{W}}\frac{\partial \hat{W}}{\partial S_W} = -\frac{\lfloor W \cdot S_W \rceil}{S_W^2} + \frac{1}{S_W} W = \tag{2.27}$$

$$\frac{1}{S_W}[W - \frac{\lfloor W \cdot S_W \rceil}{S_W}] = \frac{1}{S_W}[W - W_{fq}] \tag{2.28}$$

где равенство в строке 2.27 выполняется за счет ранее введенной производной (см. равенство 2.10) на основе STE [51].

Итого, получаем:

$$\frac{\partial \mathcal{L}}{\partial S_W} = \frac{1}{S_W}\frac{\partial \mathcal{L}}{\partial O} I^T [W - W_{fq}] \tag{2.29}$$

Как можно заметить, градиент прямо пропорционален разнице между оригинальными весами во float32 пространстве и их fake quant версией, на что и был расчет при добавлении процедуры симуляции квантования. Аналогичное выражение несложно получить и для входного тензора I, добавив к нему процедуру симуляции квантования и заметив симметрию между W и I.

2.6 Описание процедуры перемасштабирования каналов для скалярного квантования.

Как было указано в таблице 1, скалярное квантование даже с дообучаемыми порогами допускает сильное снижение точности квантуемой сети.

Авторы работы [73] заметили это еще на архитектуре MobileNet-v1 [23] и связывают это с двумя факторами:

– С диапазоном значений на Depthwise слоях;

– Функцией активации ReLU6.

В качестве доказательства, авторы приводят значения весовых коэффициентов после сплавления на первом Depthwise слое:

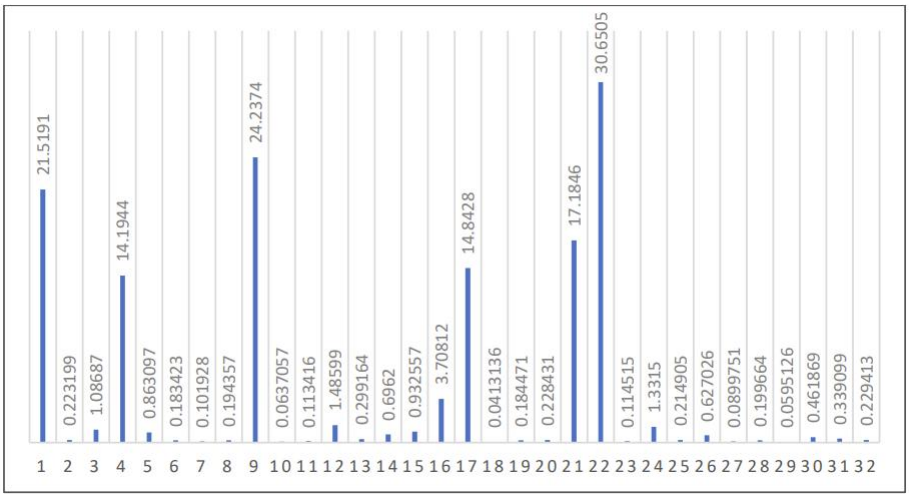

Рисунок 2.7 — Диапазон значений для первого слоя у сети MobileNet-v1

Аналогичное изображение можно вывести и для архитектуры MobileNet-v2, исследованной в прошлом разделе:

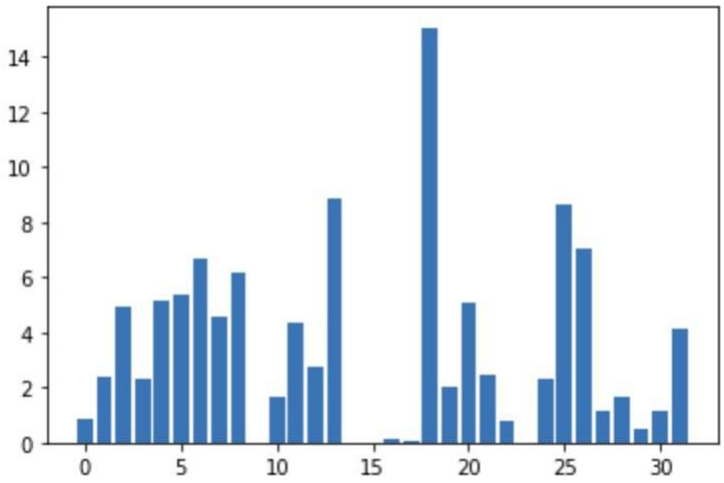

Рисунок 2.8 — Диапазон значений для одного из слоев у сети MobileNet-v2

Как можно увидеть, есть фильтры, отличие которых от “типичных” представителей составляет как минимум один порядок. Этим можно объяснить успешное применение векторного квантования, но не скалярного даже с дообучаемыми порогами. Действительно, при таком распределении величин весов очень сложно найти такое значение порога, чтобы с одной стороны, используя его в качестве границы, не было накопления большого числа значений на “хвосте” распределения, а с другой стороны чтобы не позволять маленьким значениям “размыться” в процессе квантования.

Для преодоления этой проблемы можно воспользоваться следующим свойством нейронных сетей. Пусть W_1, W_2 - весовые коэффициенты двух по-

следовательных слоев, X - входные данные, s - некоторая неотрицательная константа, а функция активация между слоями $f(x) = max(x, 0)$, то есть ReLU. Тогда:

$$f(XW_1)W_2 = f(XW_1\frac{s}{s})W_2 = f(X * \frac{W_1}{s})W_2 * s$$

поскольку $f(s * x) = max(x * s, 0) = s * max(x, 0)$ ввиду неотрицательности s. Данное свойство называется эквивариантностью [77]. Воспользовавшись данным свойством, можно ввести процедуру перемасштабирования отдельных каналов так, чтобы веса лежали в одном диапазоне. Графически это изображено на рисунке 2.9, для двух последовательных сверток (Depthwise и свертки с ядром 1) из архитектуры Mobilenet-v2.

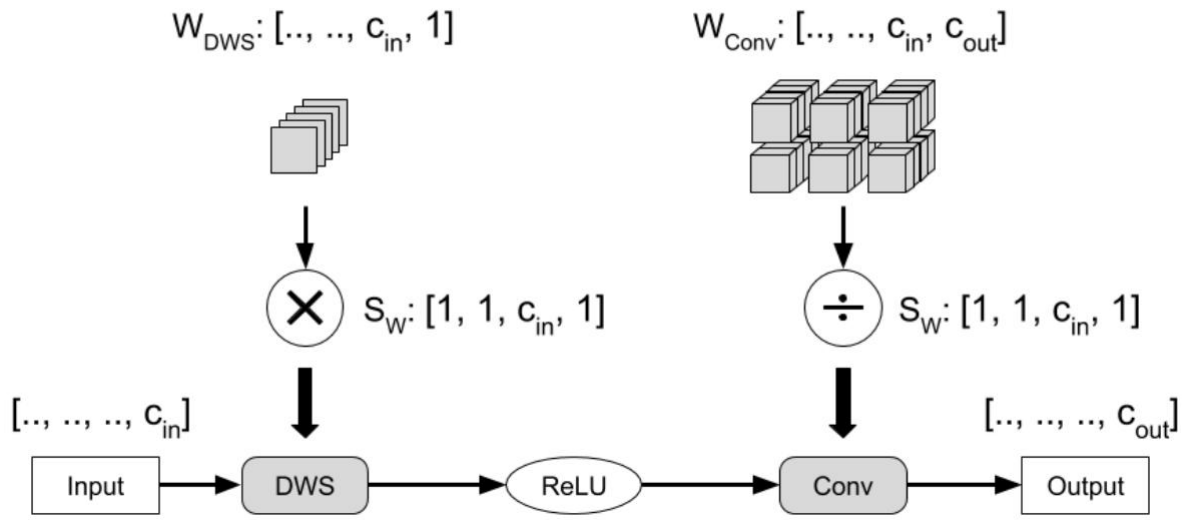

Рисунок 2.9 — Процедура перемасштабирования внутри блока сети MobileNet-v2. c_{in} обозначает количество каналов входного тензора.

Однако в случае с MobileNet-v2 условие эквивариантности для функции активации ReLU6 выполняется не для всех значений s. Например, для $x = 7, s = 10$ равенство не выполняется.

$$ReLU6(s * x) = ReLU6(60) = 6 \neq s * ReLU6(x) = 10 * ReLU6(7) = 60$$

При этом, если $s * x < 6$ и $0 < x < 6$, то тогда свойство эквивариантности сохраняется. Это значит, что выбрав каналы, у которых максимальное значение отстоит достаточно далеко от 6 (тогда ReLU6 для него действует как ReLU), и

масштабировав его с правильным коэффициентом, чтобы итоговый результат не превышал 6, можно добиться перемасштабирования значения весов и таким образом повысить точность скалярного квантования.

Более подробно процедура перемасштабирования для архитектуры MobileNet-v2 выглядит следующим образом:

1. Используя набор калибровочных данных, определите максимальное значение каждого из каналов после свертки с Depthwise слоем и до применения ReLU6.
2. Выберите каналы, значения на которых превышают 6.
3. Отметьте такие каналы как "заблокированные". Фильтры, соответствующие "заблокированным" каналам, должны оставаться неизменными. Более того, предлагается "заблокировать" каналы, выходы которых не превышают 6, но достаточно близки к 6, поскольку выбрав другой калибровочный набор данных мы вполне могли бы превысить пороговое значение. Предлагается выбрать в качестве порогового значения 5.9.
4. Найдите максимальное абсолютное значение $T^i_{w\_fixed} = |W^i_{fixed}|_{max}$ в соответствии с формулой 1.1 для каждого из заблокированных каналов, а затем усредните между всеми: $T_s = \frac{1}{n}\sum_0^n T^i_{w\_fixed}$. Полученное значение будет контролирующим для всех остальных (незаблокированных) фильтров;
5. Найдите максимальное абсолютное значение весов в $T^j_{w\_free} = |W^j_{free}|_{max}$ в незаблокированных каналах;
6. Найдите соответствующие перемасштабирующие факторы для остальных незаблокированных каналов по формуле $S^j = \frac{T_s}{T^j_{w\_free}}$;
7. Ограничьте значения этих перемасшиабирующих факторов так, чтобы результирующее значения после свертки не превышали 6.

После применения процедуры перемасштабирования для сети MobileNet-v2 в скалярном режиме квантования точность сети до тонкой настройки достигла значения 66.8 ± 1.2, где дисперсия обусловлена разными калибровочными данными. Для дальнейшего повышения точности была необходима тонкая настройка сети, в результате которой удалось добиться точности 71.01 ± 0.4.

2.7 Заключение по второй главе

В данной главе был предложен подход к квантованию нейронных сетей за счёт внедрения дообучаемых порогов квантования. Было экспериментально подтверждено, что предложенный подход приводит к незначительному падению точности квантованных архитектур, а также требует меньшее число обучающих данных, что позволяет сократить время до получения квантованной сети с нескольких дней до нескольких часов. Теоретический анализ на основе алгоритма обратного распространения ошибки показал, что обновление порогов квантования учитывает симуляцию квантования и минимизирует разницу между оригинальными тензорами и тензорами после процедуры fake quantization. Наличие дистилляции между квантованной и оригинальной сетью позволяет использовать данный метод в задачах, где разметка данных дорогостоящая, при этом выбранная функция потерь RMSE не ограничивает тип архитектуры и задачи, к которой данный подход может быть применен. Исходный код с реализацией предложенного метода и экспериментами представлен в открытом доступе на GitHub [76].

Глава 3. Анализ особенностей проектирования современных аппаратных архитектур для исполнения нейронных сетей

В предыдущей главе данной диссертационной работы был приведен метод квантования нейронных сетей, который позволяет запускать нейронные сети на довольно распространенных мобильных платформах. Однако, процессоры архитектуры ARM не были специальным образом разработаны под исполнение нейронных сетей. Проанализировав характерные особенности работы нейронных сетей, специалисты при участии специалистов по нейронным сетям разработали класс устройств, способных исполнять нейронные сети более эффективно, чем процессоры общего назначения - так называемые NPU. В данной главе будет приведен анализ наиболее типичных представителей аппаратных ускорителей, а также особенностей работы нейронных сетей с целью нахождения "узких" мест подобных устройств и дальнейшей оптимизации.

3.1 Анализ распространенных слоев нейронных сетей на возможность представления в унифицированном виде

Созданием специализированных аппаратных архитектур под конкретные задачи специалисты были заняты давно [78—80]. В связи с растущим применением искусственного интеллекта в разных областях науки и техники, специалисты озадачились процессом создания ускорителей для задачи вычисления нейронных сетей.

Поскольку вариантов архитектур нейронных сетей изобретено весомое количество, то возникает необходимость создания устройства, способного работать с большинством архитектур из представленных в публичном поле. Для создания подобного устройства необходимо проанализировать самые часто встречающиеся операции в нейронных сетях с целью поддержания их аппаратным ускорителем. Необходимо отметить, что помимо сверточных архитектур,

представленных в разделе 1.1, существуют еще и рекуррентные сети, описание и анализ которых будет приведен в 3.1.1.

3.1.1 Описание принципа работы рекуррентной ячейки

Задача обработки последовательности, где каждый следующий элемент в той или иной степени зависит от предыдущего не теряет актуальности и по сей день. Анализ текста, анализ временных рядов и прочие задачи могут быть решены при помощи так называемых рекуррентных сетей, то есть нейронных сетей с обратной связью. Однако долгое время применение подобных архитектур было затруднено в связи со сложностью обучения [81].

Архитектура сети, способная решать вышеуказанные задачи и способная учиться методом обратного распространения ошибки, была разработана в 1997 году Юргеном Шмитхубером [32]. В работе Шмитхубера был сделан ряд нововведений, которые позволили рекуррентным сетям использоваться наравне с обычными сетями прямого распространения.

Одной из основных идей является внедрение так называемых вентилей, которые могут пропускать, а могут блокировать информацию, поступающую в ячейку. Вентилем можно назвать выражение вида: $g = y\sigma(XW)$, где σ - это функция сигмоиды, вычисляемая по формуле: $\sigma(z) = \frac{1}{1+e^{-z}}$. В классической архитектуре имеется 4 вентиля: входной вентиль (представленный умножением W_i), выходной вентиль (W_o), вентиль забывания (W_f) и вентиль обновления скрытого состояния (W_g). Также было предложено разделить состояния, которые переносятся между временными отсчетами на два: c и h. Исходя из архитектурных особенностей ячейки, состояние c должно запоминать контекст, а состояние h должно представлять некоторый способ добавить обратную связь. Уравнения, которыми задается работа LSTM ячейки представлены ниже, а схема работы ячейки приведена на рисунке 3.1:

$$
\begin{aligned}
i &= \sigma(W_i x_{k-1} + R_i h_{k-1} + b_i) \\
f &= \sigma(W_f x_{k-1} + R_f h_{k-1} + b_f) \\
o &= \sigma(W_o x_{k-1} + R_o h_{k-1} + b_o) \\
g &= \tanh(W_g x_{k-1} + R_g h_{k-1} + b_g) \\
c' &= f * c + i * g \\
h_k &= o * \tanh(c')
\end{aligned}
$$

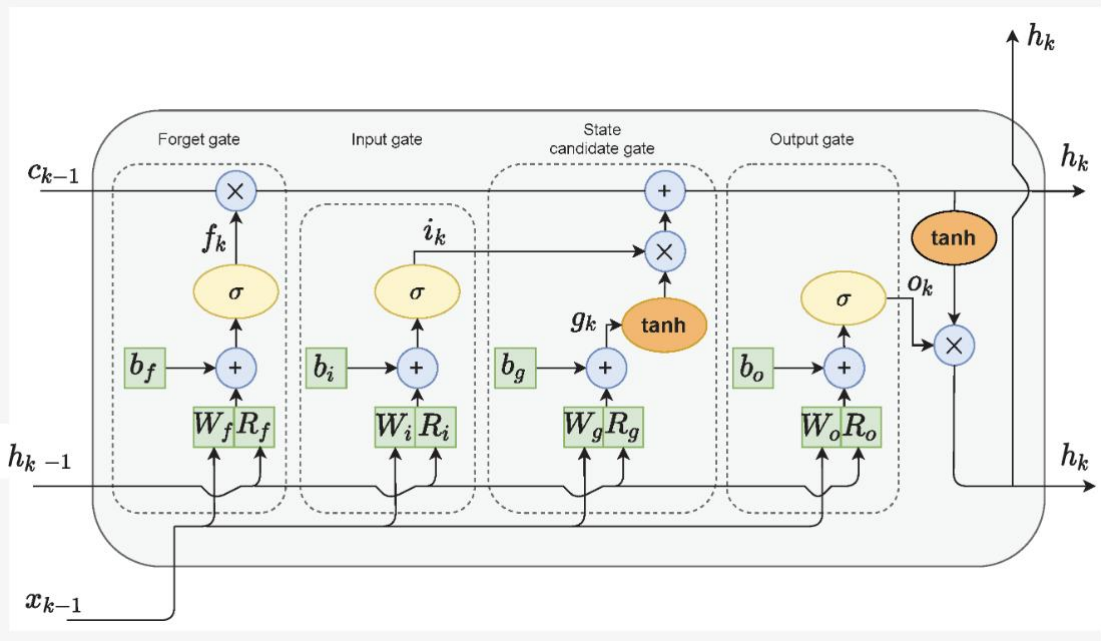

Рисунок 3.1 — Ячейка долгой краткосрочной памяти. Зелеными квадратами обозначены матричные умножения с соответствующими коэффициентами.

Хотелось бы отметить, что зачастую 4 матричных умножения в вентилях на рисунке 3.1 заменяют на следующую комбинацию операций:

1. Соединение векторов скрытого состояния h_{k-1} и входной последовательности x_{k-1} в один вектор;
2. Матричное умножение между вектором из прошлого пункта и матрицей большей размерности;
3. Разъединение результата умножения и дальнейшее распространение по каналам.

Действительно, из формул выше можно заметить, что два матричных умножения для любого вентиля можно заменить на одно. Например, для входного вентиля это можно сделать посредством соединения по столбцам матриц W, R_i и по строкам вектора скрытого состояния h_{k-1} и входного вектора x_{k-1}.

Обращая внимание на то, что аналогичным образом выполняются замены для всех вентилей, а также на тот факт, что все 4 матричных умножения выполняются с одним входом (соединенные h_{k-1} и x_{k-1}), можно соединить уже все 4 матрицы для разных вентилей по столбцами и получить вычисление всех i, o, g, f за одно умножение матриц большего размера.

Пусть операция $\oplus_i$ - будет обозначать процедуру конкатенации двух объектов, а i - размерность, по которой эта процедура происходит. Для примера, у вектора одна размерность, по которой можно провести конкатенацию (поэтому нижний индекс мы будем опускать и просто будем обозначать такую операцию как $\oplus$), а у матрицы - две.

$$
\begin{aligned}
&x = x_{k-1} \oplus h_{k-1} \\
&W_i = W_i \oplus_2 R_i \\
&W_f = W_f \oplus_2 R_f \\
&W_g = W_g \oplus_2 R_g \\
&W_o = W_o \oplus_2 R_o \\
&W = W_i \oplus_1 W_f \oplus_1 W_g \oplus_1 W_o \\
&i, f, g, o = Wx \\
&g = \tanh(g) \\
&i, f, o = \sigma(i), \sigma(f), \sigma(o) \\
&c' = f * c + i * g \\
&h_k = o * \tanh(c')
\end{aligned}
$$

Помимо матричного умножения в LSTM-ячейке присутствуют операции следующего типа:

– 3 операции поэлементного умножения векторов;

– 1 операция поэлементного сложения векторов;

– Операция вычисления функций $tanh$, σ.

Оценим влияние операции каждого типа на итоговую вычислительную сложность LSTM-ячейки. Для вычисления нелинейных функций активации мы будем использовать таблицы поиска [82], что позволяет нам не учитывать вычислительную сложность этих функций при расчете итоговой вычислительной сложности ячейки. Пусть размерность вектора h равняется H, а размерность вектора x - равняется X. Тогда размер матрицы W равняется [4H, H+X]. А

количество операций сложения-умножения будет равно $(X + H) * 4H$. На все операции векторного типа мы затратим 4H сложений-умножений, что в $X + H$ раз меньше.

Таким образом, основной операцией, которая занимает всю вычислительную сложность у LSTM - ячейки является матричное умножение. В разделе ниже будет показано, что и операция свертки представима в виде матричного умножения.

3.1.2 Анализ потребления вычислительных ресурсов операцией свертки в современных нейронных сетях

В разделе 1.1 уже были приведены некоторые блоки современных архитектур нейронных сетей и была показана вычислительная сложность для некоторых из них. В частности, была приведена оценка вычислительной сложности для архитектуры MobileNet-v1 [23]. В архитектуре MobileNet-v2 [24] основным улучшением, которое позволило значительно увеличить точность, является применение так называемых остаточных связей, позволяющих делать сеть более глубокими за счет более эффективного обратного распространения градиента. Идея остаточных связей были взяты из архитектуры ResNet [25], разновидности блоков которой приведены ниже (см рис. 3.2).

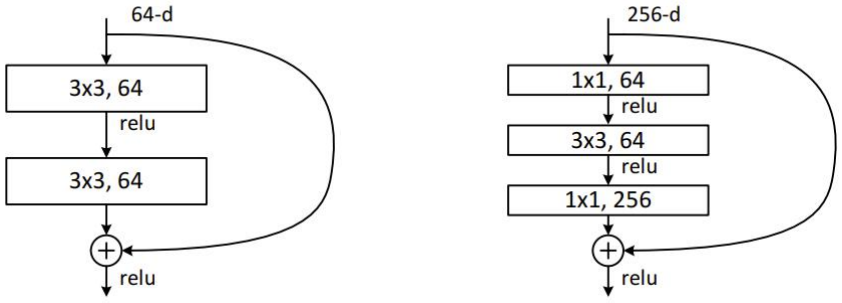

Рисунок 3.2 — Варианты блоков нейронной сети ResNet [25].

Блок справа наиболее распространен и представляет собой так называемую архитектуру бутылочного горлышка из-за того, что количество каналов у карты признаков, которая приходит на свертку 3x3, меньше чем на входе у

первой и выходе у третьей свертки. Таким образом, необходимо удостовериться, что даже в такой структуре подавляющее большинство вычислительных ресурсов будет затрачиваться на операцию свертки.

Действительно, если следовать нотации из раздела 1.1, то количество сложений-умножений, затрачиваемых на все операции свертки равняется $2*ImH*ImW*ImC*N+2*ImH*ImW*N*N*3*3$, в то время как на операцию поэлементного сложения тензоров (которая собственно и есть остаточная связь) затрачивается $ImH*ImW*ImC$. Поскольку величина N как правило равняется $0.25*ImC$, то получаем, что на операцию свертки затрачивается во столько раз больше операций сложений умножения:

$$\frac{2*ImH*ImW*ImC*N+2*ImH*ImW*N*N*3*3}{ImH*ImW*ImC}=1.625*N$$

Тенденция, что операция свертки занимает почти все вычислительные ресурсы в сети сохраняется и в мобильных архитектурах. Приведем в качестве примера ранее упомянутый MobileNet-v2 (в этой сети $N=6ImC$):

$$\frac{2*ImH*ImW*ImC*N+2*ImH*ImW*N*3*3}{ImH*ImW*ImC}=$$

$$2*N+2*6*3*3=12*Im+108$$

При этом, хотелось бы отметить, что на свертки 1x1 затрачивается больше сложений-умножений, чем на depthwise-separable свертки.

Также есть вариант сверточного слоя, который является в каком-то смысле промежуточным между обычным сверточным слоем и depthwise-separable, он называется групповой сверткой [83]. Основное отличие этого типа свертки от обычной заключается в том, что все каналы у входной карты признаков разбиваются на группы, и между группами они не пересекаются, а внутри группы эта операция работает как обычная свертка (см. рис 3.3).

Depthwise-separable свертку можно рассматривать как вариант групповой свертки, у которой число групп совпадает с числом входных каналов [40]. Впервые групповая свертка была применена в архитектуре нейронной сети ResNext [83]. Авторы в этой статье доказали взаимную заменяемость этого типа свертки

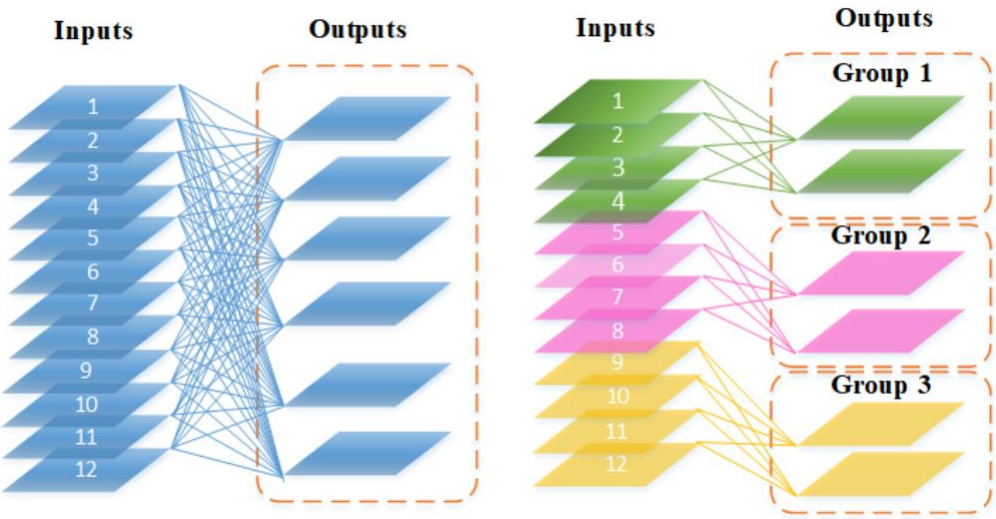

Рисунок 3.3 — Визуализация работы групповой свертки. Слева для наглядности изображена обычная свертка.

и отдельных сверток, которые затем агрегированы в одну, а также вывели, что вычислительная сложность групповой свертки в g раз меньше, чем у обычной свертки с аналогичным числом входных и выходных каналов (рис. 3.4).

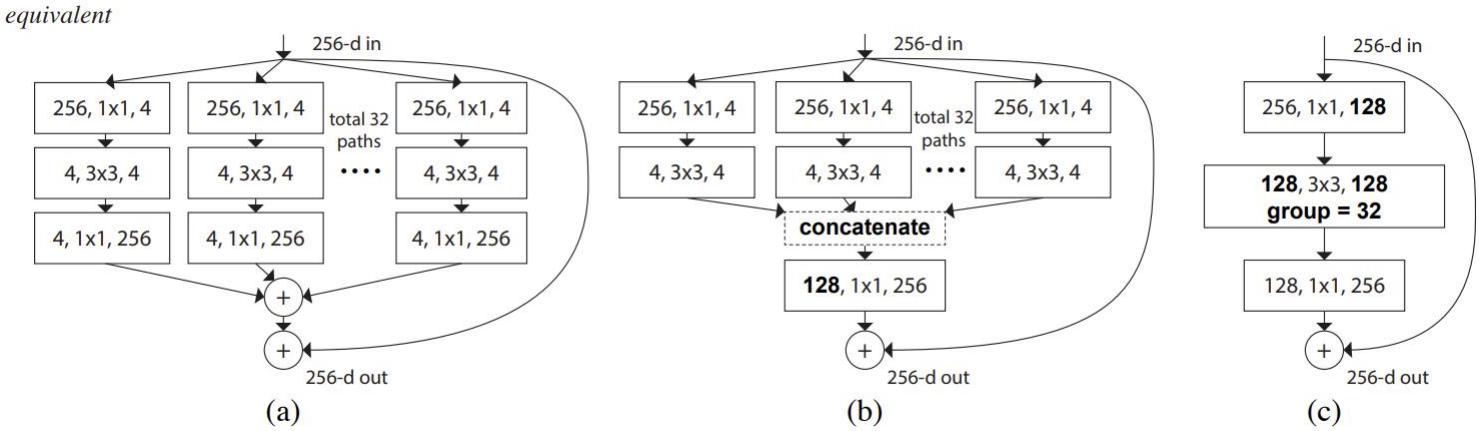

Рисунок 3.4 — Визуализация ResNext блока (c) и ему эквивалентных (a,b).

Поскольку в двух крайних случаях (обычная свертка и depthwise separable) мы получили, что операция свертки потребляет основное количество вычислительных ресурсов, то и для групповой свертки подобное будет верно, как для промежуточного варианта. Таким образом, во всех современных сверточных нейронных сетях операция свертки потребляет значительно больше вычислительных ресурсов по сравнению с операциями другого типа.

3.1.3 Представление операции свертки в виде матричного умножения

Процедура свертки фигурирует в большом спектре областей науки и техники, так что неудивительно, что существует ряд способов для ее вычисления:

– При помощи Фурье преобразования [84];

– При помощи схемы Винограда [85];

– При помощи im2col преобразования [86];

Для реализации на аппаратной платформе необходимо выбрать такой способ вычисления, который бы не отличался от матричного умножения, поскольку это основная операция, фигурирующая как в полносвязных слоях, так и в LSTM ячейках, что было показано в предыдущей главе. Третий способ вычисления из списка, называемый im2col, полностью удовлетворяет требованиям. Рассмотрим его более подробно.

im2col - это сокращение фразы на английском языке “image to column”. Основная идея заключается в том, чтобы сформировать из исходного изображения матрицу, расположив по столбцам те пиксели, которые находятся в “цветовой оси” исходного изображения. Необходимо учесть, однако, что такая процедура выполняется не просто с пикселями исходного изображения, а с каждым “кубиком”, который участвует в процедуре свертки, порождая таким образом некоторую избыточность по памяти. Весовые коэффициенты фильтров же трансформируются в матрицу размером [K*K*ImC, N], если следовать обозначениям, введенным в разделе 1.1. Визуализация данного метода приведена на изображении 3.5.

Таким образом, основной операцией, которую необходимо реализовать в аппаратном ускорителя, является матричное умножение. Специалистам по аппаратным вычислениям уже давно известно, что для матричного умножения есть эффективная структура, называемая систолический массив [87]. На основе нее и сделаны все современные NPU, анализ устройства которых представлен в разделе ниже.

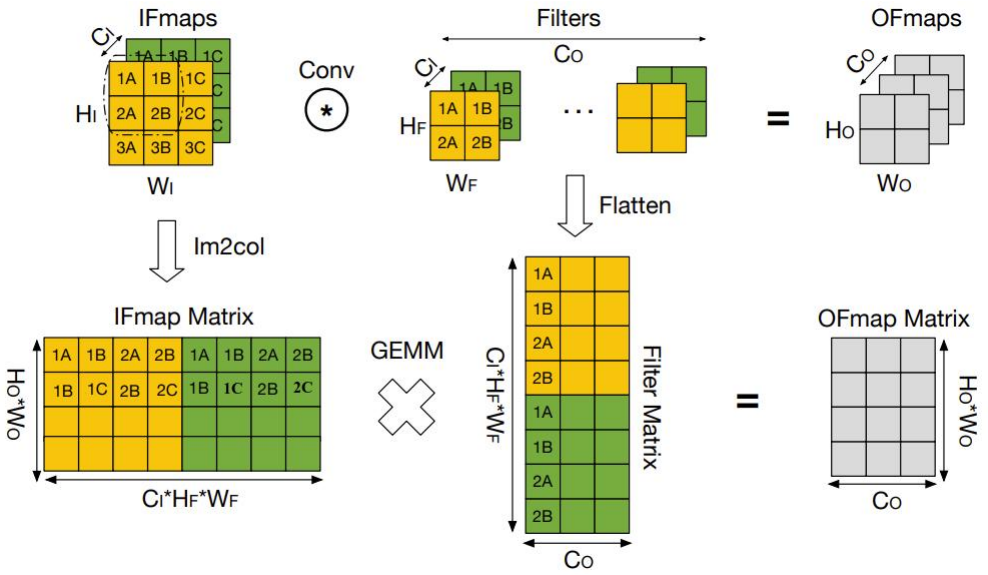

Рисунок 3.5 — Иллюстрация процедуры im2col [86]

3.2 Описание принципов работы аппаратных ускорителей нейронных сетей на основе систолического массива

Одними из первых в публичном поле о промышленном применении NPU заговорила корпорация Google [88], но позже стали появляться проекты от других фирм и независимых исследователей [89]. Как уже было сказано ранее, не смотря на обилие возможных решений, “ядром” подобных проектов является так называемая архитектура систолического массива. Она изображена на рисунке 3.7.

Данная аппаратная архитектура называется так потому, что информация внутри одного конкретного блока (называемого “tile” на 3.7) распространяется как бы “волнами” , чем-то напоминая прилив (см. рис. 3.6). Базовой ячейкой в систолическом массиве является PE (от англ. processing element) - элемент, который выполняет математическую операцию $c+ = a * b$ и пропускает множители по горизонтали и вертикали.

Процедура, эффективно выполняемая систолическим массивом, - это умножение матриц, которая, как показано в разделах 3.1.1 и 3.1.3 является основной в большинстве современных архитектур. Систолический массив, однако, является не единственным аппаратным элементом, от которого зависит производительность NPU. Приведем схематичное изображение тензорного процессора

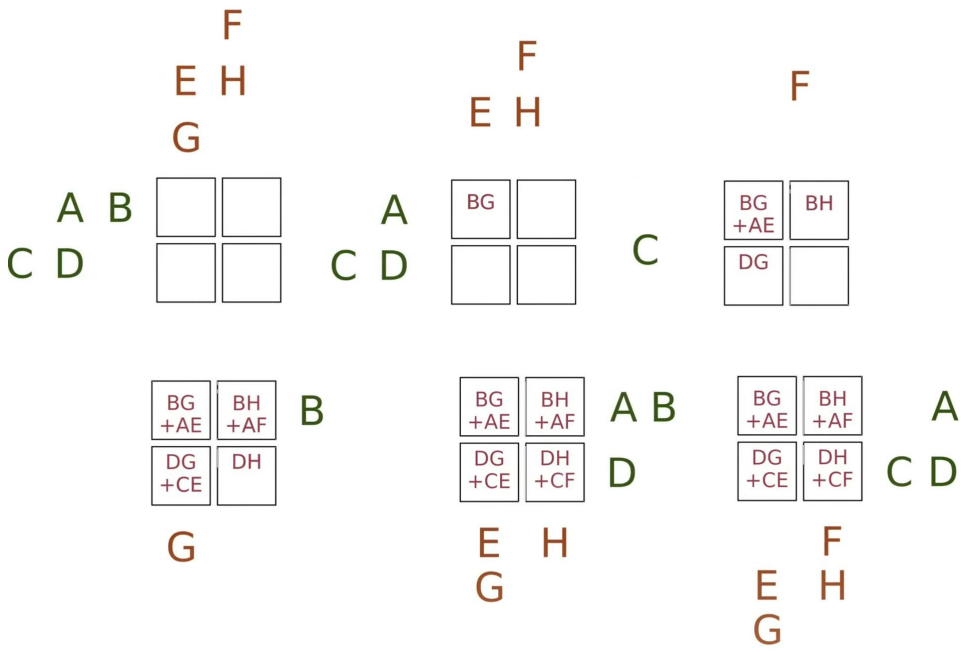

Рисунок 3.6 — Принцип работы систолического массива.

от компании Google (рис. 3.8). Можно заметить, что помимо систолического массива, обозначенного на изображении как Matrix Multiply Unit, есть еще и другие элементы, отвечающие за:

– Хранение данных;
– Загрузку весовых коэффициентов и новых входных данных или отправку вычисленного результата;
– Модуль, вычисляющий функции активации;
– Модуль, выполняющий нормализацию / пулинг данных, иногда заменяемый на модуль выполнения векторных операций [89];
– Модули, управляющие последовательностью инструкций.

Инженерное искусство при разработке NPU заключается в том, чтобы на кристалле разместить оптимальным образом как базовый элемент тензорного процессора, так и дополнительные модули, а также подобрать параметры этих модулей наиболее эффективно. При этом, чем меньше площади на кристалле занимает систолический массив, тем больше потенциальных возможностей для увеличения производительности NPU: добавить больше PE элементов, увеличить память, увеличить пропускную способность загружающего/передающего канала. Ту площадь, которую занимает систолический массив определяет размер его элементарной ячейки, а ее размер в свою очередь определяется

разрядностью регистров, участвующих в математических операциях. Таким образом, выбор оптимальной разрядности для регистров в PE является важной задачей при проектировании тензорных процессоров.

Зачастую, команды-разработчики не располагают людскими и финансовыми ресурсами, чтобы держать в штате специалистов по нейронным сетям, которые могли бы заниматься созданием и поддержкой программного комплекса для квантования нейронный сетей (даже с учетом предложенного алгоритма в главе 2). Именно поэтому остро стоит задача выбора типа данных для NPU-ускорителей, такого что:

- Количество бит, затрачиваемое на один регистр в PE было бы наименьшим;
- Имелась бы возможность работы с произвольными архитектурами нейронных сетей;
- Перевод в этот тип данных не требовал бы постоянного участия пользователя.

Вычисления в числах с плавающей запятой с сокращенной разрядностью удовлетворяет указанным выше условиям, что делает перспективным данный подход с точки зрения исследователя.

3.3 Заключение по третьей главе

В данной главе были проанализированы современные архитектуры нейронных сетей, в которых выявлены самые ресурсозатратные операции с точки зрения количества арифметических вычислений. После было показано, что эти операции являются частными случаями матричного умножения, для вычисления которого эффективной аппаратной архитектурой является систолический массив. Далее, было выяснено, что оптимизация систолического массива может повысить производительность NPU, при этом квантование и целочисленные вычисления не являются подходящим вариантом, поскольку требуют постоянной поддержки программного комплекса.

Предложенный ниже алгоритм вычисления оптимальной разрядности порядка и мантиссы, а также найденный при помощи него тип данных minifloat<e,m> позволяет оптимизировать количество бит, затрачиваемое на один PE элемент. В 1.6 уже был приведен обзор того, как некоторые нестандартные типы данных применялись к архитектурам нейронных сетей для распознавания изображений, однако для мобильных архитектур нейронных сетей подобный анализ не был проведен. Также он не был проведен и для рекуррентных сетей, что будет исправлено в следующем разделе данной диссертации.

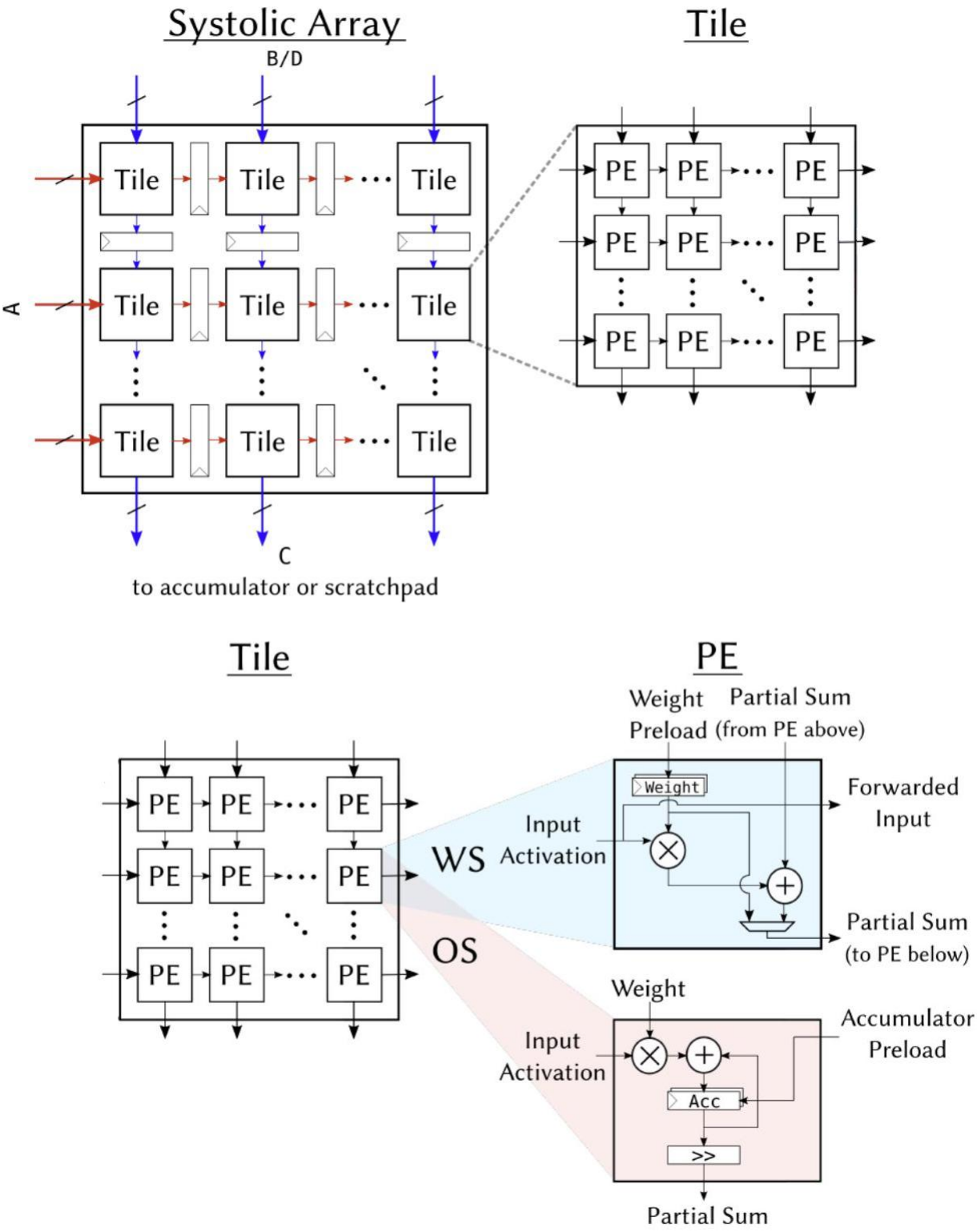

Рисунок 3.7 — Иллюстрация архитектуры систолического массива. Изображение взято из [90]

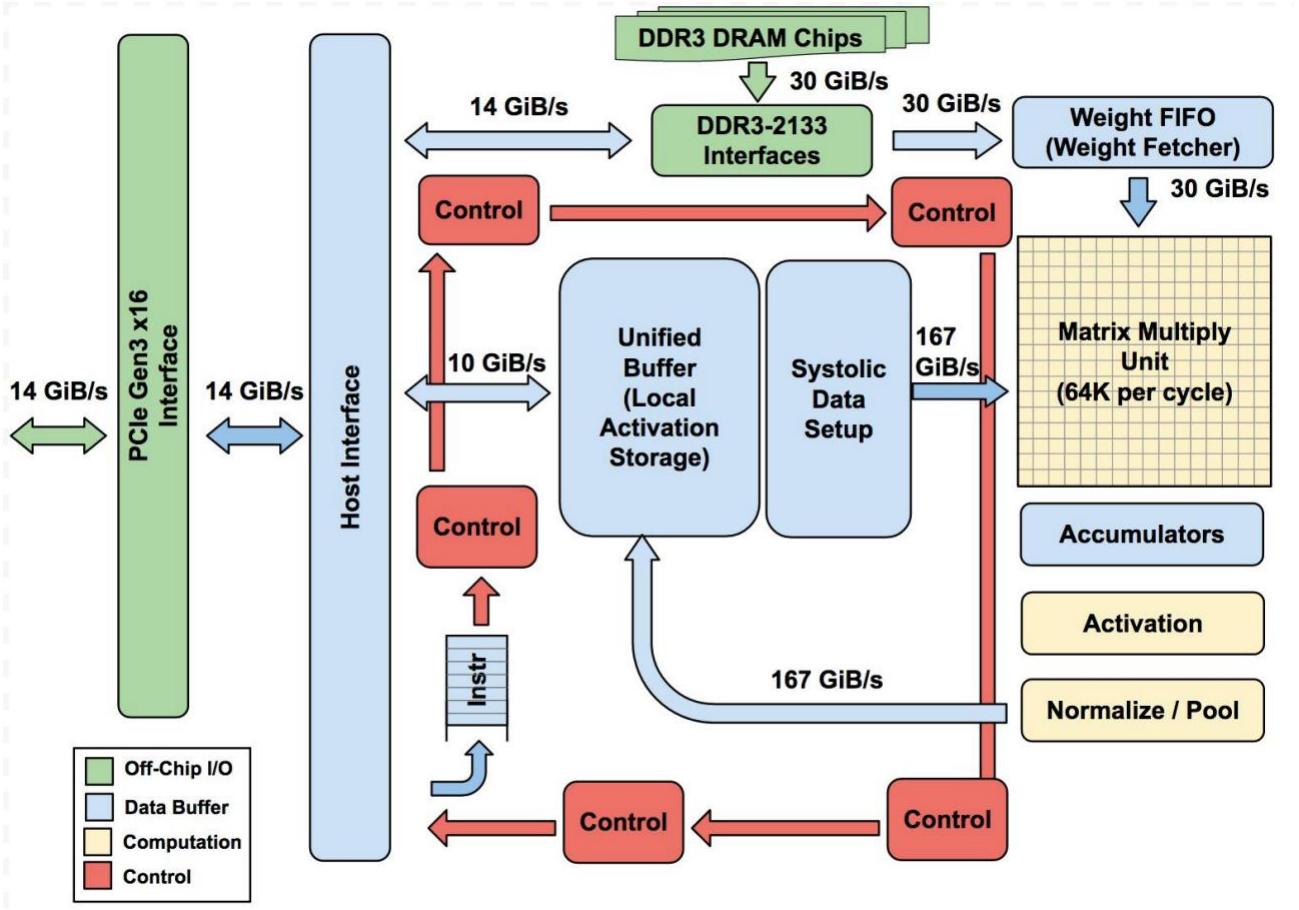

Рисунок 3.8 — Тензорный процессор от компании Google [88]. Желтым изображены элементы, выполняющие математические операции, синим - элементы памяти, зеленым - элементы, выполняющие отправку/получение данных, красным - управляющие модули.

Глава 4. Процедура нахождения оптимальной разрядности порядка и мантиссы

В предыдущей главе данной диссертационной работы был приведен анализ аппаратных ускорителей, а также была поставлена задача нахождения оптимального типа данных, способного работать не только со сверточными нейронными сетями, но и рекуррентными. В данной главе будет представлена методика подбора оптимальной разрядности порядка и мантиссы, а также экспериментальная апробация предложенной процедуры.

4.1 Описание метода нахождения оптимальной разрядности порядка и мантиссы

Алгоритм конвертации из числа одинарной точности в minifloat<e,m> заключается в отбрасывании избыточных бит мантиссы с округлением к ближайшему. При этом выполняется проверка переполнения порядка числа. Обратный алгоритм получения числа одинарной точности из minifloat тривиален, поскольку число одинарной точности обладает большей емкостью показателей порядка и мантиссы. Соответственно, достаточно извлечь из минифлоата эти показатели и сохранить в порядке и мантиссе числа float32, добавив противоположные сдвиги для порядка. В настоящей работе рассматриваются только нормализованные числа. Кроме того, зафиксирован алгоритм округления чисел “к ближайшему” (round nearest) при выполнении всех операций.

Поскольку тип данных minifloat<e,m> не поддерживается большинством программных пакетов для нейронных сетей, то для использования вышеописанного алгоритма конвертации необходимо “встроиться” в процедуру вычисления сверточных и полносвязных слоев.

Для этого тензоры весовых коэффициентов и входных карт признаков приводятся к типу данных исследуемого minifloat<e,m>, а затем выполняется преобразование типа к float16. Данное последовательное преобразование

допустимо, поскольку результаты вычислений все равно будут храниться в аккумуляторах типа float16. Выходной тензор текущего сверточного слоя является входным тензором активаций для следующего слоя, в котором также будет производиться преобразование типа. Таким образом, основным типом данных для активаций и весов нейронной сети является минифлоат<e,m>. Данная процедура изображена графически на рисунке 4.1

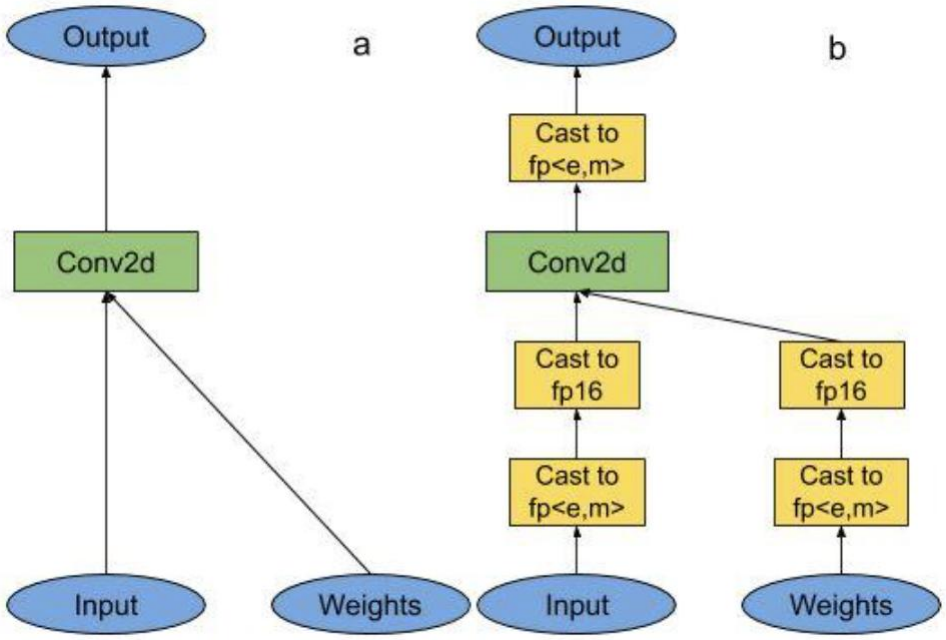

Рисунок 4.1 — Пример исходного (a) и модифицированного (b) сверточного слоя для процедуры конвертации.

Таким образом, две последовательные операции преобразования типа позволяют получить данные в низкоразрядном представлении minifloat и проводить дальнейшие вычисления свертки, сложения и активации с типом данных float16, который поддерживается большинством фреймворков [40; 46].

Конвертация для полносвязных слоев выполняется аналогичным образом.

4.2 Нахождение оптимальной разрядности порядка и мантиссы для сверточных нейронных сетей

Для проведения численных экспериментов по классификации изображений были выбраны следующие сверточные нейронные сети: GoogleNet [21],

Таблица 3 — Точность работы GoogleNet [21] при различных разрядностях e, m. Точность float32 версии равна 70.99

	m=2	m=3	**m=4**	**m=5**	m=6	m=7	m=8	m=9	m=10
e=3	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14
e=4	50.85	67.45	69.43	69.71	70.09	70.02	70.08	70.08	70.09
e=5	52.55	67.79	**70.08**	**70.66**	70.7	70.94	70.99	70.93	70.97

ResNet-50 [25], MobileNet-v2 [24]. Эти нейронные сети охватывают широкий класс сетей глубокого обучения: ResNet-50 является одной из распространенных нейронных сетей, с помощью которой решаются задачи компьютерного зрения, а MobileNet-v2 и GoogleNet являются представителями “мобильных” архитектур, а значит есть риск, что даже незначительное уменьшение разрядности порядка и/или мантиссы может сильно снизить точность указанных нейронных сетей. Обученные Tensorflow модели нейронных сетей взяты из открытых источников [74].

Классификация изображений выполнена на тестовом наборе изображений Imagenet [43], описание которого приводилось в разделе 2.4. Процедура стандартизации разрешения тестовых изображений включала в себя билинейную интерполяцию до разрешения 256x256 и выбор центральной части размером 224x224.

Объектами исследования являются подготовленные типы minifloat с общей разрядностью от 6 до 16 бит включительно. При этом количество бит под представление порядка меняется в диапазоне от 3 до 5, а размер мантиссы варьируется от 2 до 9 бит. Сравнение осуществлено с исходным качеством работы нейронных сетей на типе данных float32. Для нейронных сетей GoogleNet, ResNet-50, MobileNet-v2 основной метрикой качества является top-1 точность (top-1 accuracy), а качество при различных конфигурациях minifloat представлено в таблицах 3-5.

Из результатов, представленных в таблицах выше, видно, что наименьший minifloat, при котором сохраняется приемлемое качество данных нейронных сетей — это 11-ти разрядный minifloat <5,5>. Это значит, что количество бит представления мантиссы можно с минимальными потерями точности

Таблица 4 — Точность работы ResNet [25] при различных разрядностях e, m. Точность float32 версии равна 74.04

	m=2	m=3	**m=4**	**m=5**	m=6	m=7	m=8	m=9	m=10
e=3	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11
e=4	0.08	0.1	0.11	0.11	0.11	0.11	0.11	0.11	0.11
e=5	66.1	71.21	**73.77**	**73.9**	73.86	73.98	74.08	73.98	74.08

Таблица 5 — Точность работы MobileeNet-v2 [24] при различных разрядностях e, m. Точность float32 версии равна 72.83

	m=2	m=3	m=4	**m=5**	m=6	m=7	m=8	m=9	m=10
e=3	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
e=4	12.63	56.69	61.97	66.43	69.18	69.39	69.48	69.43	69.45
e=5	17.85	59.64	69.93	**71.82**	72.63	72.72	72.74	72.88	72.8

сократить до пяти относительно представления float16. При разрядности мантиссы, равной четырем, наблюдается заметный спад в 2 процента точности на MobileNet-v2, что для ряда задач может быть уже неприемлемой потерей качества. В то же время необходимо отметить, что нейронные сети GoogleNet и ResNet являются более устойчивыми к низкому количеству бит мантиссы и качественно работают с minifloat<5,4>.

Что касается разрядности порядка, то с количеством бит менее 5-ти хорошо работает только GoogleNet. Одновременно с этим, сеть ResNet не способна работать с 4-мя битами порядка, поскольку в процессе вычислений происходит переполнение чисел.

4.3 Нахождение оптимальной разрядности порядка и мантиссы для рекуррентных нейронных сетей

4.3.1 Описание архитектуры нейронной сети DeepSpeech

Как уже было сказано в подразделе 3.1.1, LSTM ячейки используются для анализа временных рядов, что обычно связывают с задачами, тип данных в которых является табличным. Типичными представителями таких задач являются задачи, связанные с предсказанием спроса, цены на какой-либо товар или количества посетителей магазина. Однако, интерпретировать результаты на подобных задачах может быть сложно, поэтому тестирование алгоритма нахождения мантиссы и порядка было проведено на более понятной для человека задаче - на задаче распознавания речи.

При этом, отдельное исследование LSTM ячейки в отрыве от других слоев может привести к тому, что при встраивании ее в глубокую нейронную сеть накопленная ошибка может испортить качество предсказания.

Поэтому было принято решение исследовать оптимальную разрядность у нейронной сети DeepSpeech, архитектура которой представлена на изображении 4.2.

Данная нейронная сеть состоит из 4 полносвязных слоев, цель которых углубить архитектуру, двух LSTM ячеек, которые обрабатывают прямую и инвертированную последовательность с целью большего удержания контекста, а также выходного полносвязного слоя. Подобная комбинация слоев обладает всеми желательными свойствами для надежного тестирования процедуры подбора порядка и мантиссы:

- Сеть обладает достаточной глубиной для возможного накопления ошибки;
- В архитектуре сети есть несколько LSTM ячеек, соединенных нестандартным образом;
- Архитектура обучена на хорошо интерпретируемой задаче.

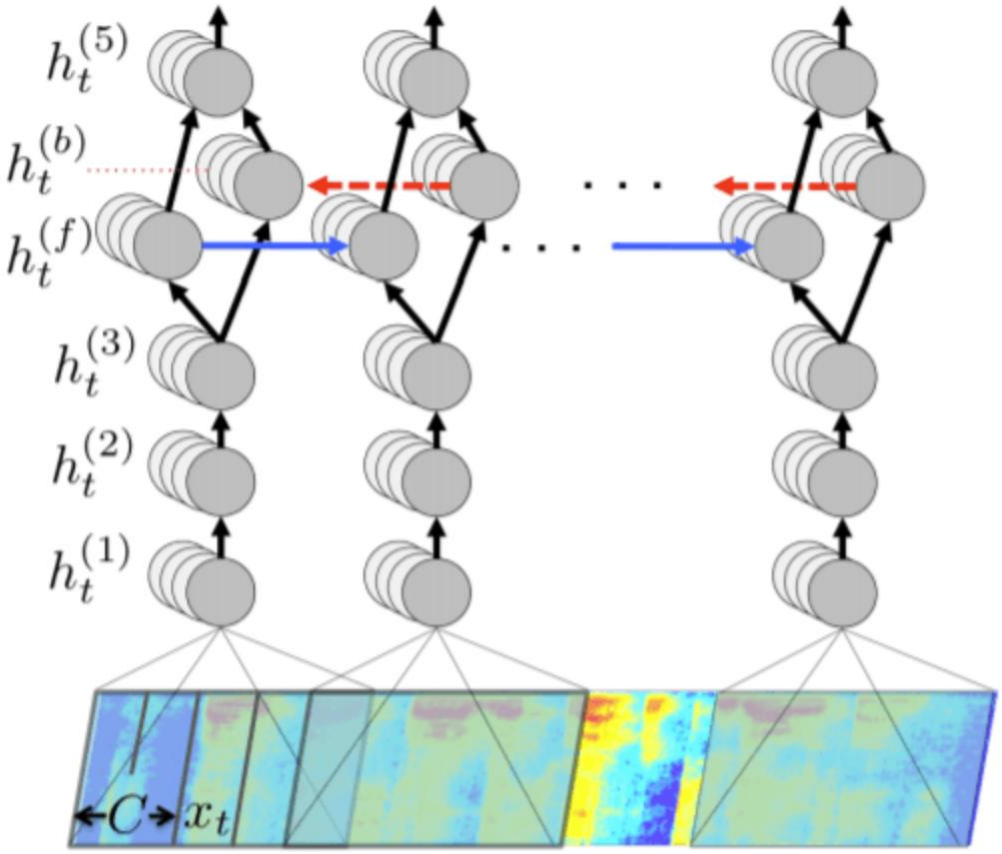

Рисунок 4.2 — Архитектура сети DeepSpeech

4.3.2 Описание процедуры обработки входного звукового сигнала

Для большего качества рекомендуется распознавание проводить не в "сыром" виде (так называемый waveform), а предварительно построив мел-кепстральные коэффициенты или mfcc (от англ. mel-frequency cepstral coefficients) [91].

В основе этого построения лежит так называемая мел-шкала. Разработана она была так, чтобы имитировать нелинейное восприятие звука человеческим ухом, будучи более различимой на более низких частотах и менее различимой на более высоких частотах. Графически, величина мел в зависимости от частоты изображена на рисунке 4.3. Есть приближенная формула, которая выражает эту зависимость: $m = 2595log_{10}(1 + \frac{f}{700})$.

Итого, построение выглядит следующим образом [92]:

1. Исходное аудио делится на окна с перекрытием. Ширину окна принимают равным от 20 до 40 мс, поскольку считается, что изменение

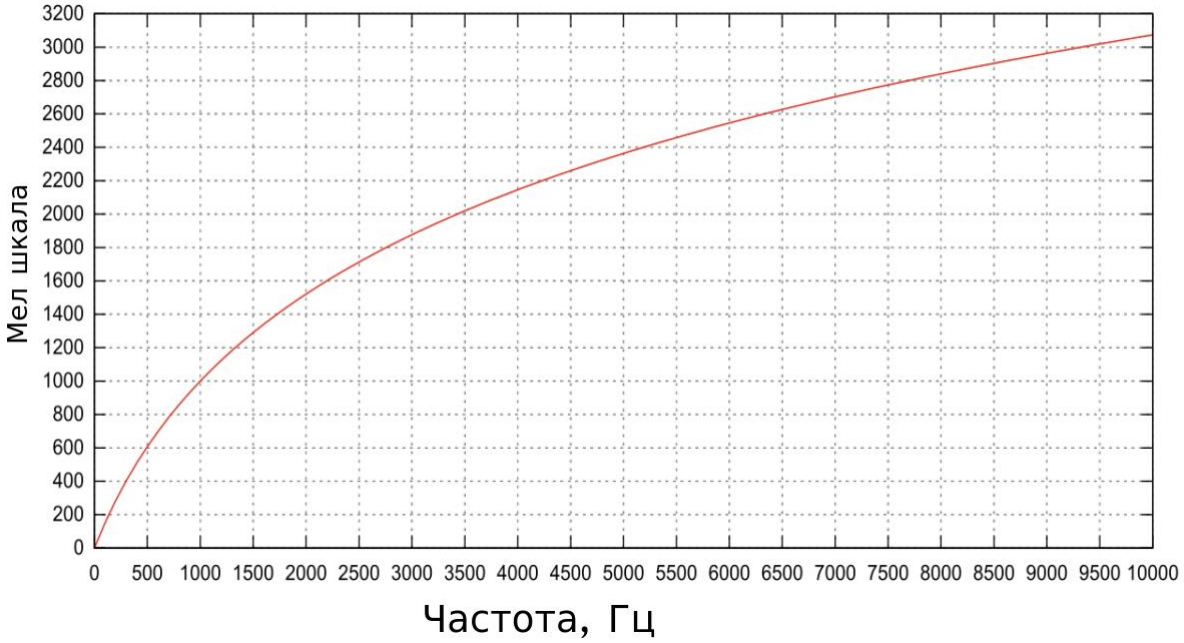

Рисунок 4.3 — График зависимости мел от частоты.

речевого сигнала на таком промежутке не сильно значительное. Величина перекрытия обычно берется от 10 до 20 мс. Дальнейшие пункты выполняется для каждого окна в отдельности;

2. Каждое окно умножается на оконную функцию Хэмминга: $\omega(n) = 0.54 - 0.46cos(\frac{2*\pi*n}{N-1}); 0 \leqslant n \leqslant N-1$. Таким образом происходит сглаживание сигнала на краях окна, дабы не возникал эффект утечки от последующего преобразования Фурье. N - количество отсчетов, которые попадают в одно окно;
3. Выполнить преобразование Фурье над полученным сигналом: $X_j(k) = \sum_{n=0}^{N} x_j(n)\omega(n)e^{-\frac{w*\pi*N}{N}kn}; 0 \leqslant k \leqslant N$. Индекс j в данном случае обозначает номер окна;
4. Вычисляем периодограмму для каждого окна по формуле $P_j(k) = \frac{|X_j(k)|^2}{N}$;
5. Далее необходимо выполнить вычисление блока мел-фильтров. Для этого треугольные фильтры (обычно 26 штук) умножаются на периодограмму и суммируются. Каждый треугольный фильтр моделируется

с помощью следующей функции:

$$H_m(k) = \begin{cases} 0, k < f(m-1) \\ \frac{k-f(m-1)}{f(m)-f(m-1)}, f(m-1) \leqslant k < f(m) \\ \frac{f(m+1)-k}{f(m+1)-f(m)}, f(m) \leqslant k \leqslant f(m+1) \\ 0, k > f(m+1) \end{cases}$$

m - это точки частот, распределенные равномерным образом в мел-шкале и которые определяются диапазоном частот сигнала по шкале Герца (что можно узнать из спектрограммы). Пример фильтров можно увидеть на рисунке 4.4;

6. Применяем логарифмирование к коэффициентам, полученным в предыдущем пункте: $S_j(m) = ln \sum_{k=0}^{N-1} P_j(k) H_m(k), 0 \leqslant m < M$
7. Далее, применяем дискретное косинусное преобразование и получаем мел-кепстральные коэффициенты: $c_j(n) = \sum_{m=0}^{M-1} S_j(m) cos(\pi * n(m + 0.5)/M); 0 \leqslant n < M.$

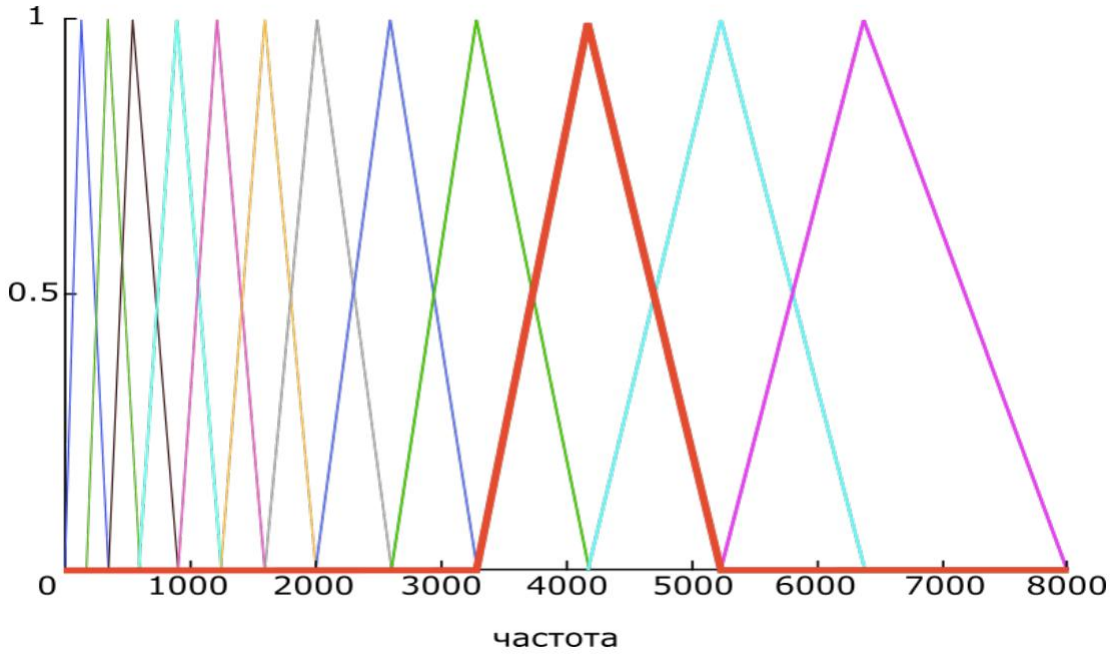

Рисунок 4.4 — Типичный пример мел фильтров. Фильтры собираются в области низких частот, обеспечивая более высокое "разрешение"того диапазона, где человек лучше распознает изменения высоты тона.

Вышеописанная процедура является стандартной и включена в большинство программных пакетов, в том числе и в [46], который использовался в данной диссертационной работе.

4.3.3 Экспериментальные результаты

Тестирование качества распознавания речи проведено на размеченных аудиозаписях LibriSpeech [93]. Данный датасет содержите около 1000 часов аудиокниг на английском языке. Все аудиозаписи были выполнены профессиональными дикторами, количество уникальных речевых образцов составляет 2484. Датасет сбалансирован относительно мужских и женских голосов, а также голосов спикеров. Аудио записывалось в сыром виде (без применения алгоритмов сжатия) с частотой 16 кГц. Примеры мел-спектрограмм для записей из данного набора приведены на рисунке 4.5:

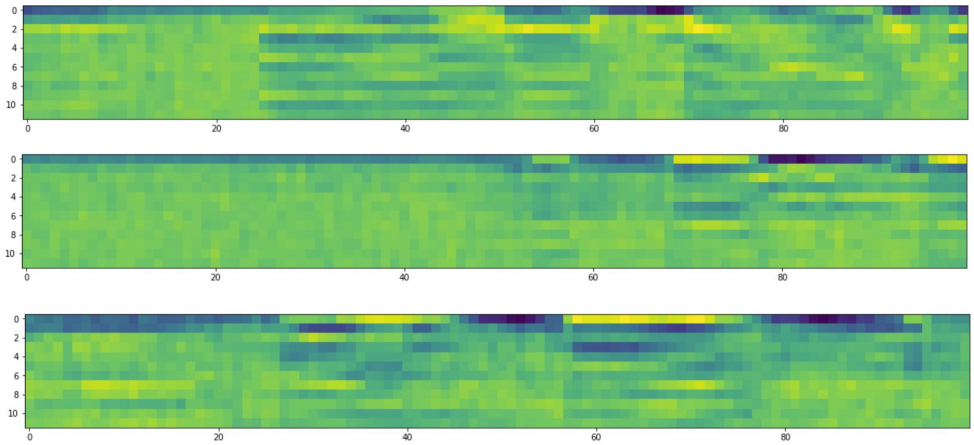

Рисунок 4.5 — Примеры mfcc для аудиозаписей 8842-302196-0006.flac, 8842-302201-0010.flac, 8842-302201-0012.flac. Код для обработки взят из [94].

Базовой метрикой в задаче распознавания речи является WER (от англ. word error rate) которая показывает, по сути, долю неправильно распознанных слов и вычисляется по следующей формуле:

$$WER = \frac{N_{insert} + N_{subst} + N_{delete}}{N_{words}}$$

Таблица 6 — Качество работы DeepSpeech-v1 [4] при различных разрядностях e, m. Показатель WER у float32 версии равен 0.1548

	m=2	m=3	m=4	m=5	m=6	m=7	m=8	m=9	m=10
e=3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
e=4	0.1881	0.1645	0.1595	0.1553	0.1576	0.1570	0.1560	0.1565	0.1566
e=5	0.1891	0.1615	0.1546	0.1527	0.1558	0.1576	0.1565	0.1548	0.1567

где N_{insert} вставки слов, которых нет в исходной аудиозаписи, N_{subst} - замены слов на некорректные, N_{delete} - пропуск слова, а N_{words} - общее число слов в аудио-дорожке.

Полученные значения качества показывают, что для нейронной сети Deepspeech использование 9-ти разрядного minifloat<4,4> ведет к потере 0,0047 единиц WER, что в большинстве реальных приложений является допустимым результатом. Использование minifloat разрядности 8 ведет к заметному падению качества распознавания речи. Однако, полученная метрика для minifloat <5,2> и minifloat <4,3> дает надежду, что дообучение весов улучшит качество работы и позволит использовать эти типы данных.

4.4 Заключение по четвертой главе

Исходя из полученных результатов экспериментов можно судить, что для нейронных сетей наибольшую важность имеет диапазон, в котором производится вычисление, нежели точность самих вычислений. Это подтверждается фактом существования значения размера порядка, после которого качество работы нейронной сети значительно ухудшается. При уменьшении размера мантиссы точность исследованных нейронных сетей либо уменьшается постепенно, либо не уменьшается вообще. При этом разрядность мантиссы определяется вычислительной сложностью каждой конкретной архитектуры. Например, самая сложная с вычислительной точки зрения нейронная сеть, DeepSpeech-v1, допускает снижения мантиссы вплоть до 4 бит (как и порядка), в то время как

мобильная архитектура MobileNet-v2 не позволяет уменьшить количество бит в представлении как мантиссы, так и порядка менее 5-ти без значительной потери точности. Полученные результаты свидетельствуют о том, что подход к ускорению нейронных сетей, заключающийся в использовании специализированных типов данных с плавающей запятой, оправдывает себя в случае избыточных нейронных сетей и требует аккуратного применения в случае мобильных видов архитектур нейронных сетей. Необходимо отметить, что при дополнительной тонкой настройке моделей возможно дополнительное сокращение разрядности числа с плавающей запятой на 1-2 бита, поскольку падение точности было не настолько велико, что демонстрируют результаты из приведенных выше таблиц.

Заключение

В процессе выполнения данной диссертационной работы достигнуты следующие результаты:

1. Был разработан алгоритм квантования с дообучаемыми порогами, который позволил оптимизировать современные мобильные архитектуры нейронных сетей с незначительной потерей (<1%) в точности;
2. Разработанный программный код, позволяющий воспроизвести результаты экспериментов, а также получить версии квантованных архитектур, был опубликован в открытом доступе в репозитории GitHub;
3. Предложенный алгоритм квантования позволил ускорить нейронные сети с архитектурами MobileNet-v2 и MNasNet-1.0 в 1.44 и 1.4 раз соответственно на процессоре типа ARM от компании Qualcomm на наборе данных ImageNet;
4. Разработанная методика нахождения разрядности порядка и мантиссы позволила сократить разрядность числового типа данных с 32 бит до 10 бит для сверточных нейронных сетей с архитектурами GoogleNet и ResNet-50 с падением точности менее 1% на наборе данных ImageNet и до 11 бит для архитектуры MobileNet-v2;
5. Разработанная методика была апробирована на рекуррентной нейронной сети с архитектурой DeepSpeech - v1. Для данной нейронной сети аналогичным образом удалось сократить разрядность с 32 бит до 9 при увеличении метрики WER на 0,0047;
6. Предложенные в данной работе методы реализованы и внедрены в виде ключевых модулей в компаниях ООО “Диалоговые системы”, ООО “Экспасофт” и ее клиентов, о чём имеются соответствующие акты о внедрении, прикреплённые к данной диссертации.

Список литературы

1. *Krizhevsky, A.* ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing 25 [Текст] / A. Krizhevsky, I. Sutskever, G. Hinton. — 2012.
2. *Simonyan, K.* Very deep convolutional networks for large-scale image recognition [Текст] / K. Simonyan, A. Zisserman // arXiv preprint arXiv:1409.1556. — 2014.
3. Rich feature hierarchies for accurate object detection and semantic segmentation [Текст] / R. Girshick [и др.] // Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition. — 2014. — С. 580—587.
4. Deep Speech: Scaling up end-to-end speech recognition [Текст] / A. Hannun [и др.] // arXiv preprint arXiv:1412.5567. — 2014.
5. Attention Is All You Need [Текст] / A. Vaswani [и др.] // Proceedings of the 31st International Conference on Neural Information Processing Systems. — 2017. — С. 6000—6010.
6. *Kingma, D.* Adam: A method for stochastic optimization [Текст] / D. Kingma, J. L. Ba // Proceedings of 3rd International Conference on Learning Representations. — 2015.
7. *Zeiler, M.* Adadelta: An adaptive learning rate method [Текст] / M. Zeiler // arXiv preprint arXiv:1212.5701. — 2012.
8. URL: https://www.perficientdigital.com/insights/our-research/mobile-vs-desktop-usage-study. — (Дата обр. 22.04.2020).
9. URL: https://tass.ru/obschestvo/6757981. — (Дата обр. 22.04.2020).
10. *Zoph, B.* Neural architecture search with reinforcement learning [Текст] / B. Zoph, Q. V. Le // arXiv preprint arXiv:1611.01578. — 2016.

11. Learning both weights and connections for efficient neural network [Текст] / S. Han, J. Pool, J. Tran, W. Dally // Advances in neural information processing systems. — 2015. — Т. 28.

12. Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition [Текст] / V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, V. Lempitsky // Proceeding of International Conference on Learning Representations. — 2015.

13. *Han, S.* Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding [Текст] / S. Han, H. Mao, W. Dally // Proceeding of 4th International Conference on Learning Representations. — 2016.

14. Distilling the knowledge in a neural network [Текст] / G. Hinton, O. Vinyals, J. Dean [и др.] // arXiv preprint arXiv:1503.02531. — 2015.

15. URL: https://lpcv.ai/competitions/2018. — (Дата обр. 25.06.2023).

16. *Гончаренко, А. И.* ИССЛЕДОВАНИЕ ПРИМЕНИМОСТИ НИЗКОРАЗРЯДНЫХ ПРЕДСТАВЛЕНИЙ ЧИСЕЛ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ ДЛЯ ЭФФЕКТИВНЫХ ВЫЧИСЛЕНИЙ В НЕЙРОННЫХ СЕТЯХ [Текст] / А. И. Гончаренко, А. Ю. Кондратьев // Автометрия. — 2020. — Т. 56, № 1. — С. 93—99.

17. On practical approach to uniform quantization of non-redundant neural networks [Текст] / A. Goncharenko, A. Denisov, S. Alyamkin, E. Terentev // Lecture Notes in Computer Science. — 2019. — Т. 11728. — С. 349—360.

18. Trainable thresholds to uniform quantization of non-redundant neural networks [Текст] / A. Goncharenko, A. Denisov, S. Alyamkin, E. Terentev // Lecture Notes in Computer Science. — 2019. — Т. 11507. — С. 302—312.

19. Low-power computer vision: Status, challenges, and opportunities [Текст] / S. Alyamkin, ..., A. Goncharenko, G. Xuyang [и др.] // IEEE Journal on Emerging and Selected Topics in Circuits and Systems. — 2019. — Т. 9, № 2. — С. 411—421.

20. URL: http : / / yann . lecun . com / exdb / lenet / index . html. — (Дата обр. 22.04.2020).

21. Going deeper with convolutions [Текст] / C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich // Proceedings of the IEEE conference on computer vision and pattern recognition. — 2015. — С. 1—9.

22. Rethinking the Inception Architecture for Computer Vision [Текст] / C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna // Proceedings of the IEEE conference on computer vision and pattern recognition. — 2016. — С. 2818—2826.

23. Mobilenets: Efficient convolutional neural networks for mobile vision applications [Текст] / A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam // arXiv preprint arXiv:1704.04861. — 2017.

24. Mobilenetv2: Inverted residuals and linear bottlenecks [Текст] / M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen // Proceedings of the IEEE conference on computer vision and pattern recognition. — 2018. — С. 4510—4520.

25. Deep residual learning for image recognition [Текст] / K. He, X. Zhang, S. Ren, J. Sun // Proceedings of the IEEE conference on computer vision and pattern recognition. — 2016. — С. 770—778.

26. Mnasnet: Platform-aware neural architecture search for mobile [Текст] / M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, Q. V. Le // Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. — 2019. — С. 2820—2828.

27. Learning transferable architectures for scalable image recognition [Текст] / B. Zoph, V. Vasudevan, J. Shlens, Q. V. Le // Proceedings of the IEEE conference on computer vision and pattern recognition. — 2018. — С. 8697—8710.

28. Regularized evolution for image classifier architecture search [Текст] / E. Real, A. Aggarwal, Y. Huang, Q. V. Le // arXiv preprint arXiv:1802.01548. — 2018.

29. *Liu, H.* Darts: Differentiable architecture search [Текст] / H. Liu, K. Simonyan, Y. Yang // arXiv preprint arXiv:1806.09055. — 2018.

30. *Cai, H.* Proxylessnas: Direct neural architecture search on target task and hardware [Текст] / H. Cai, L. Zhu, S. Han // arXiv preprint arXiv:1812.00332. — 2018.

31. Once-for-all: Train one network and specialize it for efficient deployment [Текст] / H. Cai, C. Gan, T. Wang, Z. Zhang, S. Han // Proceeding of International Conference on Learning Representations. — 2020.

32. *Hochreiter, S.* Long short-term memory [Текст] / S. Hochreiter, J. Schmidhuber // Neural Computation. — 1997.

33. Proximal policy optimization algorithms. [Текст] / J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov // arXiv preprint arXiv:1812.00332. — 2017.

34. *Ruder, S.* An overview of gradient descent optimization algorithms [Текст] / S. Ruder // arXiv preprint arXiv:1609.04747. — 2016.

35. *Anwar, S.* Structured pruning of deep convolutional neural networks. [Текст] / S. Anwar, K. Hwang, W. Sung // Journal on Emerging Technologies in Computing Systems. — 2017.

36. Learning both weights and connections for efficient neural network [Текст] / H. Li, A. Kadav, I. Durdanovic, H. Samet, H. Peter // arXiv preprint arXiv:1608.08710. — 2016.

37. *Ioffe, S.* Batch normalization: accelerating deep network training by reducing internal covariate shift [Текст] / S. Ioffe, C. Szegedy // Proceedings of the 32nd International Conference on International Conference on Machine Learning. — 2015. — С. 448—456.

38. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers [Текст] / J. Ye, X. Lu, Z. Lin, J. Z. Wang // Proceeding of International Conference on Learning Representations. — 2018.

39. Variational convolutional neural network pruning. [Текст] / C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, Q. Tian // Proceedings of the IEEE conference on computer vision and pattern recognition. — 2019. — С. 2780—2789.

40. Pytorch: An imperative style, high-performance deep learning library [Текст] / A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga [и др.] // Advances in neural information processing systems. — 2019. — Т. 32.

41. Pruning convolutional neural networks for resource efficient inference [Текст] / P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz // arXiv preprint arXiv:1611.06440. — 2016.

42. Importance Estimation for Neural Network Pruning [Текст] / P. Molchanov, A. Mallya, S. Tyree, I. Frosio, J. Kautz // Proceedings of the IEEE conference on computer vision and pattern recognition. — 2019. — С. 11264—11272.

43. Imagenet: A large-scale hierarchical image database [Текст] / J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei // 2009 IEEE conference on computer vision and pattern recognition. — Ieee. 2009. — С. 248—255.

44. Quantization and training of neural networks for efficient integer-arithmetic-only inference [Текст] / B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko // Proceedings of the IEEE conference on computer vision and pattern recognition. — 2018. — С. 2704—2713.

45. URL: https://developer.nvidia.com/tensorrt. — (Дата обр. 22.04.2020).

46. Tensorflow: Large-scale machine learning on heterogeneous distributed systems [Текст] / M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin [и др.] // arXiv preprint arXiv:1603.04467. — 2016.

47. URL: https : / / github . com / NervanaSystems / distiller. — (Дата обр. 22.04.2020).

48. *Courbariaux, M.* Training deep neural networks with low precision multiplications [Текст] / M. Courbariaux, Y. Bengio, J.-P. David // Proceeding of 3rd International Conference on Learning Representations. — 2015.

49. *Courbariaux, M.* Xnor-net: Imagenet classification using binary convolutional neural networks [Текст] / M. Courbariaux, Y. Bengio, J.-P. David // European Conference on Computer Vision. — Springer. 2016. — С. 525—542.

50. Binarized neural networks [Текст] / I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio // Advances in neural information processing systems. — 2016. — Т. 29. — С. 4107—4115.

51. *Bengio, Y.* Estimating or propagating gradients through stochastic neurons for conditional computation [Текст] / Y. Bengio, N. Léonard, A. Courville // arXiv preprint arXiv:1308.3432. — 2013.

52. *McDonnell, M.* Training wide residual networks for deployment using a single bit for each weight [Текст] / M. McDonnell // Proceeding of International Conference on Learning Representations. — 2018.

53. NICE: Noise Injection and Clamping Estimation for Neural Network Quantization [Текст] / C. Baskin, N. Liss, Y. Chai, E. Zheltonozhskii, E. Schwartz, R. Giryes, A. Mendelson, A. M. Bronstein // arXiv preprint arXiv:1810.00162. — 2018.

54. Fitnets: Hints for thin deep nets [Текст] / A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, Y. Bengio // arXiv preprint arXiv:1412.6550. — 2014.

55. WRPN: Wide Reduced-Precision Networks [Текст] / A. Mishra, E. Nurvitadhi, J. J. Cook, D. Marr // arXiv preprint arXiv:1709.01134. — 2017.

56. *Mishra, A.* Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy [Текст] / A. Mishra, D. Marr // arXiv preprint arXiv:1711.05852. — 2017.

57. *Goldberg, D.* What every computer scientist should know about floating-point arithmetic [Текст] / D. Goldberg // ACM Computing Surveys. — 1991.

58. *Кондратьев А. Ю. Гончаренко А.И., З. В.* Исследование применимости облегченных типов данных с плавающей запятой для нейронных сетей [Текст] / З. В. Кондратьев А. Ю. Гончаренко А.И. // Интеллектуальный анализ сигналов, данных и знаний : методы и средства. — 2018. — С. 514—522.

59. *Lai, L.* Deep Convolutional Neural Network Inference with Floating-point Weights and Fixed-point Activations [Текст] / L. Lai, N. Suda, V. Chandra // arXiv preprint arXiv:1703.03073. — 2017.

60. Training deep neural networks with 8-bit floating point numbers [Текст] / N. Wang, J. Choi, D. Brand, C.-Y. Chen, K. Gopalakrishnan // arXiv preprint arXiv:1812.08011. — 2018.

61. *Jacobi, C. G. J.* Über ein leichtes Verfahren die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen [Текст] / C. G. J. Jacobi // Journal für die reine und angewandte Mathematik. — 1857. — Т. 30. — С. 51—94.

62. *Lu, J.* Numerical Matrix Decompositions [Текст] / J. Lu // arXiv preprint arXiv:2107.02579. — 2021.

63. *Pearson, K.* On lines and planes of closest fit to systems of points in space [Текст] / K. Pearson // Philosophical Magazine. — 1901. — Т. 2. — С. 559—572.

64. *Rendle, S.* Factorization machines [Текст] / S. Rendle //. — 2010. — С. 995—1000.

65. *Goncharenko, A.* Blind watermarking algorithm for a sequence of TV images [Текст] / A. Goncharenko, I. Tarantsev, K. Lysakov // Optoelectronics, Instrumentation and Data Processing. — 2018. — Т. 52. — С. 341—346.

66. *Min, W.* Group-sparse SVD Models and Their Applications in Biological Data [Текст] / W. Min, J. Liu, S. Zhang // arXiv preprint arXiv:1807.10956. — 2018.

67. Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation [Текст] / E. Denton, W. Zaremba, J. Bruna, Y. LeCun, R. Fergus // arXiv preprint arXiv:1404.0736. — 2014.

68. Stable Low-rank Tensor Decomposition for Compression of Convolutional Neural Network [Текст] / A.-H. Phan, K. Sobolev, K. Sozykin, J. Ermilov Dmitry Gusak, P. Tichavsky, V. Glukhov, I. Oseledets, A. Cichocki // In European Conference on Computer Vision. — Springer. 2020. — С. 522—539.

69. Towards Efficient Tensor Decomposition-Based DNN Model Compression with Optimization FrameworkConvolutional Networks for Efficient Evaluation [Текст] / M. Yin, Y. Sui, S. Liao, B. Yuan // arXiv preprint arXiv:2107.12422. — 2021.

70. *Oseledets, I.* Tensor-train decomposition [Текст] / I. Oseledets // SIAM Journal on Scientific Computing. — 2010. — Т. 31. — С. 2130—2145.

71. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients [Текст] / S. Zhou [и др.] // arXiv preprint arXiv:1606.06160. — 2016.

72. *Gonzalez, R. C.* Digital image processing [Текст] / R. C. Gonzalez, R. E. Woods. — Prentice Hall, 2008. — URL: http://www.amazon.com/Digital-Image-Processing-3rd-Edition/dp/013168728X.

73. A quantization-friendly separable convolution for mobilenets [Текст] / T. Sheng, C. Feng, S. Zhuo, X. Zhang, L. Shen, M. Aleksic // arXiv preprint arXiv:1803.08607. — 2018.

74. URL: https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet. — (Дата обр. 22.04.2020).

75. URL: https://tensorflow.org/lite/guide/hosted_models. — (Дата обр. 22.04.2020).

76. URL: https://tinyurl.com/fatgithub. — (Дата обр. 22.04.2020).

77. Naturally Occurring Equivariance in Neural Networks [Текст] / C. Olah, N. Cammarata, C. Voss, L. Schubert, G. Goh // Distill. — 2020. — https://distill.pub/2020/circuits/equivariance.

78. *Xiu, L.* Design of a H.264 Main Profile Video Decoding Coprocessor [Текст] / L. Xiu, Z. Li, W. Xiu // Lecture Notes in Electrical Engineering. — 2011. — Т. 87. — С. 355—360.

79. *Yardi, S.* HELLAS: a specialized architecture for interactive deformable object modeling [Текст] / S. Yardi, B. Bishop, T. Kelliher // ACM-SE 44: Proceedings of the 44th annual Southeast regional conference. — 2006. — С. 56—61.

80. A pipelined processor architecture for regular expression string matching [Текст] / Q. Li, J. Li, J. Wang, B. Zhao, Y. Qu // Microprocessors and Microsystems. — 2012. — Т. 36, № 6. — С. 520—526.

81. Overcoming the vanishing gradient problem in plain recurrent networks [Текст] / Y. Hu, A. Huber, J. Anumula, S.-C. Liu // arXiv preprint arXiv:1801.06105. — 2018.

82. *Kumar Meher*, *P.* An optimized lookup-table for the evaluation of sigmoid function for artificial neural networks [Текст] / P. Kumar Meher // 2010 18th IEEE/IFIP International Conference on VLSI and System-on-Chip. — 2010. — С. 91—95.

83. Aggregated Residual Transformations for Deep Neural Networks [Текст] / S. Xie, R. B. Girshick, P. Dollar, Z. Tu, K. He // arXiv preprint arXiv:1611.05431. — 2016.

84. *Osgood*, *B.* The Fourier Transform and Its Applications [Текст] / B. Osgood // Lecture notes for Electrical Engineering. — 2011.

85. *Lavin*, *A.* Fast Algorithms for Convolutional Neural Networks [Текст] / A. Lavin, S. Gray // arXiv preprint arXiv:1509.09308. — 2015.

86. Characterizing and Demystifying the Implicit Convolution Algorithm on Commercial Matrix-Multiplication Accelerators [Текст] / Y. Zhou, M. Yang, C. Guo, J. Leng, Y. Liang, Q. Chen, M. Guo, Y. Zhu // arXiv preprint arXiv:2110.03901. — 2021.

87. ERIDANUS: Efficiently Running Inference of DNNs Using Systolic Arrays [Текст] / B. Asgari, R. Hadidi, H. Kim, S. Yalamanchili // IEEE Micro. — 2019.

88. *Jouppi*, *N.* In-Datacenter Performance Analysis of a Tensor Processing Unit [Текст] / N. Jouppi, D. H. ... Yoon // arXiv preprint arXiv:1704.04760. — 2017.

89. AI Accelerator Survey and Trends [Текст] / A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, J. Kepner // arXiv preprint arXiv:2109.08957. — 2021.

90. Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration [Текст] / H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, Y. S. Shao // Proceedings of the 58th Annual Design Automation Conference (DAC). — 2021.

91. *Huzaifah, M.* Comparison of Time-Frequency Representations for Environmental Sound Classification using Convolutional Neural Networks [Текст] / M. Huzaifah // arXiv preprint arXiv:1706.07156. — 2017.

92. URL: https://alphacephei.com/ru/lecture1.pdf. — (Дата обр. 30.10.2023).

93. Librispeech: an ASR corpus based on public domain audio books [Текст] / V. Panayotov, G. Chen, D. Povey, S. Khudanpur // 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). — IEEE. 2015. — С. 5206—5210.

94. URL: https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html. — (Дата обр. 22.04.2020).

Список рисунков

Список таблиц