УДК 004.272.2 DOI: 10.17587/it.25.747-756

Р. А. Соловьев, д-р техн. наук, вед. науч. сотр., turbo@ippm.ru,
Д. В. Тельпухов, д-р техн. наук, зав. отделом, turbo@ippm.ru,
А. Г. Кустов, инженер-исследователь, turbo@ippm.ru,
Т. Ю. Исаева, канд. техн. наук, науч. сотр., turbo@ippm.ru,
А. А. Волков, инженер-исследователь, turbo@ippm.ru,
Институт проблем проектирования в микроэлектронике РАН

Применение методов модулярной арифметики при разработке аппаратных реализаций нейронных сетей[1]

Обсуждается использование методов модулярной арифметики для реализации нейронных сетей на аппаратном уровне в СБИС и ПЛИС. Рассматриваются широкоизвестные мобильные нейронные сети, которые имеют высокую точность и подходят для реализации "в железе". Рассмотрены и проанализированы основные трудности при их реализации в базисе модулярной арифметики. Предложен ряд методов для решения этих проблем, таких как использование сверток со значением шага больше 1 вместо используемых ранее MaxPooling слоев, использование нестандартных активаций, содержащих только операции сложения, вычитания и умножения, а также эффективный алгоритм реализации операции округления. Дополнительно предложен полный маршрут проектирования и переноса нейронной сети MobileNet на аппаратный уровень в модулярном базисе.

Ключевые слова: *нейронные сети, модулярная арифметика, система остаточных классов, операция округления, аппаратная реализация*

Введение

Нейронные сети хорошо справляются с множеством задач, связанных с классификацией и обработкой изображений, аудио- и видеоданных, в некоторых случаях даже лучше человека [1, 2]. Большинство современных архитектур имеют в своем составе сверточные (convolution) блоки (например, VGG [1], Inception [3], ResNet [4], U-Net [5]), поэтому такие сети называются сверточными (convolutional neural nets (CNN)). Вычислительная сложность во время классификации так велика, что с вычислениями плохо справляются даже мощные процессоры общего назначения CPU. Для полноценной работы с современными нейросетями используют мощные и дорогие GPU (видеокарты) [6]. Особенно эта проблема актуальна при обработке видеоинформации в реальном времени.

Некоторые структуры нейросетей при очень высокой точности классификации изображений обладают свойствами, которые позволяют легко переносить их на аппаратную платформу. Существует направление, связанное с проектированием нейронных сетей для использования в мобильных устройствах: MobileNets [7], SqueezeNet [8]. Эти варианты отличаются малым числом весов и относительно небольшим числом арифметических операций. Однако они также выполняются на программном уровне и используют для работы вычисления с плавающей точкой. К сожалению, в ряде случаев, например, в задачах обработки видеоинформации в реальном времени, даже при использовании мобильных сетей не всегда удается обеспечить непрерывную обработку видеопотока со скоростью 30 кадров в секунду без использования существенной оптимизации.

Для использования нейросетей на этапе, когда у нас есть обученная модель в реальном устройстве, можно применить набор оптимизаций, чтобы ускорить выполнение вычислений в несколько раз. Для этого уже существует

[1]Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 17-07-00404.

ряд методик, например, компрессия весов [9] или вычисления на малобитных данных [10].

Поскольку потребности в аппаратуре для работы с нейронными сетями постоянно растут, то требуется разработка специальных аппаратных блоков для использования в СБИС и ПЛИС для ускорения расчетов. Ускорение расчетов может достигаться за счет:

- аппаратной реализации свертки. Аппаратный блок для вычисления свертки работает быстрее, чем свертка, выполненная на программном уровне;
- перехода от вычислений с плавающей запятой к фиксированной запятой;
- уменьшения размерности вычислений с сохранением приемлемой точности;
- сокращения части нейронной сети с сохранением точности классификации;
- модификации структуры нейронной сети с незначительным уменьшением точности (или даже без нее) с увеличением скорости работы и уменьшением размера аппаратной реализации и хранимых весов.

Дополнительное ускорение расчетов или снижение потребляемой энергии при расчетах с использованием нейронных сетей, как показали современные исследования, может достигаться за счет использования модулярной арифметики.

В статье [11] предлагается использовать вариацию рекурсивной модулярной арифметики [26] для представления весов и карт признаков. Для этого массивный 48-битный MAC-юнит заменяется на большой набор 4-битных модулярных MAC-юнитов, которые реализованы в виде Lookup-таблиц в ПЛИС. За счет существенного уменьшения размерности модулей до 4 бит удалось значительно увеличить тактовую частоту устройства со 100...200 МГц до 400 МГц, а также существенно увеличить производительность устройства в целом.

В статье [12] предлагается усовершенствованный MAC-Unit на базе модулярной арифметики и системы модулей вида $\{2^{l_1}, 2^{l_2}-1, \ldots, 2^{l_N}-1\}$. Немного уменьшая точность сети за счет округления весов и промежуточных вычислений, авторам удалось увеличить скорость работы в 1,5...2,4 раза и сократить потребление энергии в 3—9 раз. Точность классификации нейронной сети AlexNet при этом упала всего с 79 до 75 %.

Авторы работы [13] представили архитектуру сверточной нейронной сети с вычислениями в системе остаточных классов с применением модулей специального вида, за счет чего достигли увеличения производительности на 37 % по сравнению с двоичной реализацией.

Указанные исследования показывают значительный потенциал модулярной арифметики при проектировании аппаратной реализации нейронных сетей. В данной статье исследуются методы, которые подойдут для реализации нейронных сетей в небольших специализированных микроэлектронных устройствах, в которых использование GPU является нерентабельным по причине существенного энергопотребления и высокой стоимости.

1. Модулярная арифметика для вычислений на базе нейронных сетей. Преимущества и недостатки

Типовая структура сверточной нейронной сети приведена на рис. 1. Позже появились некоторые усовершенствованные структуры с разветвлениями [3, 4], но суть осталась та же: размер изображения от слоя к слою уменьшается, а число фильтров растет. В конце сверточной сети образуется набор признаков, которые подаются на классификационный слой (или слои), и выходные нейроны сигнализируют вероятность принадлежности изображения к конкретному классу.

Для ускорения вычислений обычно переходят от вычислений на программном уровне к аппаратной реализации, а также от вычислений с плавающей точкой к фиксированной точке [14, 15]. В данном случае открываются существенные возможности с точки зрения использования полезных свойств модулярной арифметики [26—28] при проектировании устройства, реализующего нейронную сеть:

1) модулярная арифметика плохо справляется с вычислениями на базе плавающей точки, однако при работе с фиксированной точкой,

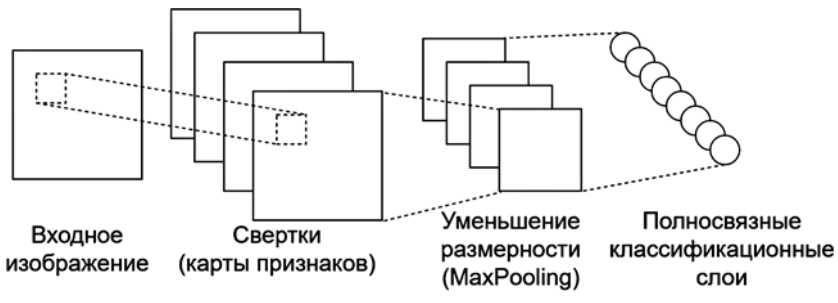

Рис. 1. Типовая структура сверточной нейронной сети

где заранее известен динамический диапазон вычислений, может быть подобран подходящий модулярный базис;

2) веса предобученной нейронной сети известны заранее, поэтому их можно хранить сразу в модулярном виде на базе известного модулярного базиса, т. е. не потребуется аппаратная реализация прямого преобразователя для весов, объем которых очень большой;

3) основные вычислительные блоки нейронной сети — сверточные и полносвязные слои, которые требуют 90 и более процентов вычислительной мощности [16], — имеют в своем составе только операции сложения и умножения. Именно на этих операциях модулярная арифметика показывает свое преимущество перед позиционной;

4) входные данные в нейронную сеть подаются в виде цветного изображения небольшого размера. Затем над этими данными выполняется большой набор арифметических операций по мере движения данных от слоя к слою по нейронной сети. Результат при этом в случае классификации — это небольшой набор выходных чисел. Это означает, что прямые и обратные преобразователи будут выполнять совсем немного вычислений по сравнению с основным модулярным трактом.

Однако есть несколько проблем при использовании некоторых типов слоев, которые могут привести к уменьшению производительности при использовании модулярной арифметики:

1) в нейронных сетях для уменьшения размера промежуточных карт признаков используется блок MaxPooling или AvgPooling. Их главная проблема с точки зрения модулярной арифметики в первом случае — наличие операции Max, которая является частным случаем сравнения чисел и представляет собой сложную операцию в модулярной арифметике и требует отдельного блока большого размера. Во втором случае требуется вычисление среднего, что влечет за собой операцию деления на 4, которая тоже может вызывать трудности. Однако эта проблема может быть решена использованием вместо MaxPooling сверточного блока с параметром strides, равным 2, что уже широко используется в современных сетях, например в MobileNet. В случае использования сверточного блока вместо MaxPooling остаются только операции сложения и умножения;

2) слои типа Batch Normalization содержат деление, но после сокращения констант остаются только одна операция сложения и одна умножения. Дополнительно в случае если слой Batch Normalization идет сразу после сверточного блока, а именно так он располагается в большинстве современных нейронных сетей, его можно удалить, пересчитав веса у сверточного слоя;

3) основную проблему представляют слои активации нейронов, которые присутствуют почти после всех сверточных слоев и добавляют нелинейность. Самые популярные активации в современных сетях — Rectified linear unit (RELU) из-за его простоты с точки зрения имплементации в позиционной арифметике и Sigmoid или Softmax, которые обычно используется на выходных слоях сети:

$$\text{RELU} \to f(x) = \begin{cases} 0, x < 0; \\ x, x \geqslant 0; \end{cases}$$

$$\text{Sigmoid} \to f(x) = \frac{1}{1 + \mathbf{e}^{-x}}.$$

RELU крайне проста для позиционной арифметики, но из-за операции сравнения становится сложной в модулярной арифметике. По-другому сравнение с нулем называется операцией определения знака [17]. Sigmoid вычислительно сложна в обоих случаях из-за наличия функции $\mathbf{e}^{-x}$ и деления.

Одним решением проблемы может быть разработка функции активации, которая содержит только "дружественные" для модулярной арифметики операции, либо создание эффективной аппаратной реализации одной из существующих активаций. Одной из таких активаций является RELU. Для определения знака можно использовать эффективные алгоритмы [18]. Однако более эффективным методом является подход с разработкой функций активации для модулярной арифметики.

2. Нестандартные функции активации

В работе были изучены функции активации, которые содержат только операции сложения и умножения. Экспериментальным путем было показано, что функция активации $f(x) = x^2 - kx$ при некоторых значениях k хорошо сходится. Была построена сеть MobileNet, в которой все слои RELU заменены на предложенную функцию, и показано, что для задачи классификации автомобилей на наборе Open Images Dataset [24] нейронная сеть обучается и показывает точность классификации до 93 % (рис. 2). Точность на валидационных данных чуть выше, чем на тренировочном наборе из-за того, что к входным данным применялись очень сильные аугментации.

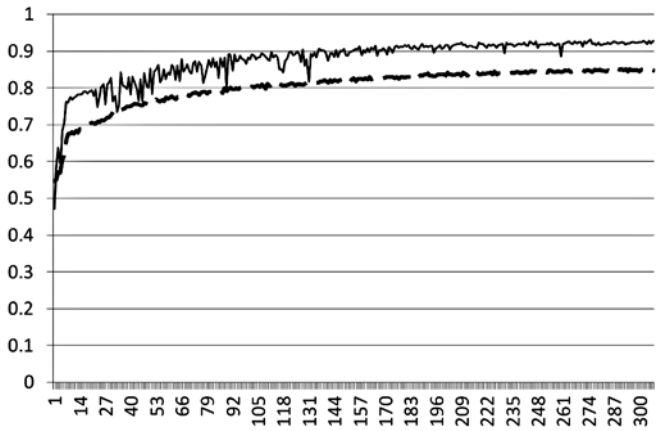

Рис. 2. Процесс тренировки MobileNet с функциями активации $f(x) = x^2 - 50x$. По горизонтали номер эпохи, по вертикали точность классификации. Штриховая линия — точность на тренировочных данных, сплошная линия — точность на валидационных данных

Предложенная функция активации позволяет избавиться от операции сравнения при реализации нейронной сети и тем самым устранить узкое место для аппаратной реализации на базе модулярной арифметики.

3. Переход от вычислений с плавающей точкой к вычислениям с фиксированной точкой

В нейронных сетях вычисления традиционно проводятся с плавающей точкой на GPU (быстро) или CPU (медленно), например, с использованием типа float32. При реализации на аппаратном уровне вычисления с плавающей точкой работают медленнее, чем с фиксированной точкой из-за сложностей с контролем мантиссы и показателя степени при различных операциях.

Рассмотрим первый слой нейросети типа Convolution (рис. 3, см. третью сторону обложки) — в большинстве сверточных нейросетей этот слой является основным.

На входе слоя находится двумерная матрица (исходное изображение, значения которой находятся на интервале [0; 1).

Известно также, что если $a \in [-1; 1]$ и $b \in [-1; 1]$, то произведение $ab \in [-1; 1]$.

Формула для расчета конкретного пикселя в позиции (i, j) второго слоя с учетом того, что используется Convolution с размером 3 × 3:

$$n_{ij} = b + w_{00}p_{i-1,j-1} + w_{01}p_{i-1,j+0} + w_{02}p_{i-1,j+1} + \\ + w_{10}p_{i+0,j-1} + w_{11}p_{i+0,j+0} + w_{12}p_{i+0,j+1} + \\ + w_{20}p_{i+1,j-1} + w_{21}p_{i+1,j+0} + w_{22}p_{i+1,j+1}.$$

Поскольку веса $w_{i,j}$ и смещение b известны, то можно рассчитать потенциальный минимум mn и максимум mx значений на входе второго слоя. Пусть $M = \max(|mx|, |mn|)$. Если разделить $w_{i,j}$ и b на значение M, то можно гарантировать, что при любой конфигурации входных данных значение на втором слое не превысит 1. Назовем *M коэффициентом редукции слоя*. На втором слое получается такая же ситуация, как и на первом, а именно на входе слоя значение из промежутка [−1; 1] и рассуждения можно повторить.

Можно легко показать, что для нейронной сети на последнем слое после всех редукций весов позиция максимума на последнем нейроне не изменится, т. е. нейронная сеть будет работать эквивалентно нейронной сети без редукции с точки зрения вычислений с плавающей точкой.

Выполнив редукцию на каждом слое, мы можем перейти от вычислений с плавающей точки к вычислениям с фиксированной точкой, поскольку мы точно знаем диапазон значений на каждом этапе вычислений. Для представления чисел размерности N бит будем использовать следующую нотацию:

$$xb = \lfloor x \cdot 2^N \rfloor.$$

Если $z = x + y$, тогда сложение $z' = xb + yb = \lfloor x \cdot 2^N \rfloor + \lfloor y \cdot 2^N \rfloor = \lfloor (x + y) \cdot 2^N \rfloor = \lfloor z \cdot 2^N \rfloor = \lfloor zb \rfloor$. Умножение $z' = xb \cdot yb = \lfloor x \cdot 2^N \rfloor \cdot \lfloor y \cdot 2^N \rfloor = \lfloor (x \cdot y) \cdot 2^N \cdot 2^N \rfloor = \lfloor z \cdot 2^N \cdot 2^N \rfloor = \lfloor zb \cdot 2^N \rfloor$, т. е. после умножения требуется разделить результат на 2^N, чтобы получить реальное значение, или же просто выполнить сдвиг на N позиций.

Если перебирать все возможные входные изображения и ориентироваться на потенциальный минимум и максимум, то коэффициенты редукции могут оказаться очень большими, и точность будет теряться от слоя к слою довольно быстро, что может потребовать большой разрядности фиксированной точки для хранения весов и промежуточных результатов вычислений. Чтобы этого не произошло, можно использовать все (или некоторую часть) изображения тренировочного набора как наиболее вероятные, чтобы найти максимальные и минимальные значения на каждом слое. Как показали эксперименты, использование тренировочного набора позволяет сильно уменьшить коэффициенты редукции. При этом желательно брать коэффициенты с небольшим запасом, например, увеличивая значение максимума на несколько процентов.

Однако при некоторых условиях возможно переполнение и выход за границы рассчитанного диапазона. Для этого в аппаратной реализации требуется детектор таких случаев и замена переполненных значений на максимум для данного слоя. Впрочем, этого можно достигнуть лишь небольшой модификацией блока выполнения свертки.

При вычислениях с фиксированной точкой с ограниченной разрядностью весов и промежуточных вычислений неизбежно возникают ошибки округления, которые накапливаются от слоя к слою и могут привести к некорректной работе нейронной сети. Некорректной работой мы считаем ошибку классификации при сравнении с математической моделью, а не с реальным ответом.

При использовании модулярной арифметики основной проблемой при вычислениях становится операция округления после всех операций сложений и умножений в сверточном блоке. Если же постараться избежать этой операции, то размерность чисел и требуемый динамический диапазон будут сильно расти от слоя к слою, и потребуется слишком большой набор модулей, что сделает применение модулярной арифметики менее эффективным. Для операции округления в модулярной арифметике существуют специальные быстрые методы [19—22].

4. Метод проектирования нейронной сети на базе модулярной арифметики

Рассмотрим последовательность действий, требуемых для подготовки аппаратной реализации нейронной сети на конкретном примере. Возьмем в качестве нейронной сети известную и хорошо зарекомендовавшую себя на практике сеть MobileNet [7]. Она имеет большое число слоев (около 100), не содержит разветвлений и отличается небольшим числом весов, что позволяет хранить их в памяти даже для небольших аппаратных устройств. Точность классификации на наборе ImageNet достигает 70,6 %.

Всего MobileNet включает в себя восемь типов слоев:

1) ZeroPadding2D — добавляет ободок из нулей вокруг изображения. Слой требуется для выполнения свертки таким образом, чтобы размер выходной карты признаков совпадал с размером входной карты признаков;

2) Conv2D (kernel = 3 × 3) — двумерная свертка с ядром размера 3 × 3;

3) Conv2D (kernel = 1 × 1) — двумерная свертка с ядром размера 1 × 1;

4) DepthwiseConv2D — упрощенная версия Conv2D (меньше сложений, умножений и соответственно меньше матриц с весами);

5) Activation (Relu6) — функция активации. Является усовершенствованной функцией RELU, в ней появляется дополнительное ограничение сверху. Все значения, большие 6, приравниваются 6. Это не дает потенциальным максимальным значениям уходить далеко от 0 и очень полезно для представления чисел с фиксированной точкой;

6) GlobalAvgPooling — слой, который используется в нейронной сети только один раз, является предпоследним слоем. У него нет весов, и он находит среднее значение от карт признаков предыдущего слоя. Генерирует вектор признаков изображения (длина вектора обычно от 512 до 4096);

7) Dense (Fully Connected) — последний полносвязный слой сети. Выполняет классификацию изображения. В нашей задаче он состоит из двух нейронов. Первый сигнализирует о том, что объект отсутствует, второй — о том, что объект присутствует. На выходе этого слоя используется активация Softmax, которая возвращает вероятности наличия искомых объектов;

8) BatchNormalization — слой, выполняющий нормализацию и идущий в связке с каждым сверточным слоем.

Шаг 1. Работа начинается с подготовки тренировочного набора данных. Для простоты положим, что нам требуется разделить изображения на два класса по какому-то признаку, например, определить, есть ли на изображении человек или нет, что может быть полезно для камеры слежения или мобильных телефонов.

Шаг 2. На данном шаге требуется выбрать размер входного изображения для нейронной сети, от которого будет зависеть ее конфигурация. Типовые размеры изображений подходящие для MobileNet: 128 × 128, 160 × 160, 192 × 192 и 224 × 224 пикселей.

Шаг 3. Тренируем сеть на нашем наборе данных, используя любой подходящий фреймворк: tensorflow, pytorch, caffe, keras и т. д. Цель — получить веса для сети, которые представляют собой набор чисел с плавающей точкой.

Шаг 4. Связки слоев Conv2D + BatchNormalization и DepthwiseConv2D + BatchNormalization можно сократить, заменив одним Conv2D и одним DepthwiseConv2D слоями, пересчитав веса по формулам метода BatchNorm Fusion. На этом этапе MobileNet уменьшится

со 100 до 70 слоев. Сеть будет работать быстрее, а значения на выходе сети будут эквивалентны полной версии.

Замечание 1: после редукции слоев сеть уже не подходит для обучения, точнее обучение будет гораздо менее эффективно, чем на полной сети.

Замечание 2: в начальной версии сети у слоев Conv2D и DepthwiseConv2D не было смещения (bias), после редукции он у них появится.

Шаг 5. Значение 6 у функции активации RELU(6) можно заменить на 1, тогда значения карт признаков после каждой активации будут нормироваться на интервале от 0 до 1, так же как и у входного изображения. Сделать это не сложно. Для этого достаточно все веса только первого слоя разделить на 6, затем смещение у каждого слоя разделить на 6 и заменить все RELU(6) в сети на функцию RELU(1). Позиция максимума на последнем слое не изменится, т. е. классификация будет выполнена аналогично начальной модели.

Шаг 6. На данном этапе у нас получилось нормировать входное изображение и все значения промежуточных карт признаков на промежуток от 0 до 1, однако веса и смещения в общем случае могут превышать эти значения. Поэтому нам требуется найти максимум и минимум для них, чтобы определить, сколько значений старших разрядов нам потребуется хранить дополнительно. Для этого можно использовать формулы

$$ConvW = \lceil \log_2 \max(weights) \rceil ConvB = \\ = \lceil \log_2 \max(bias) \rceil,$$

где *weights* — все значения весов на всех слоях; *bias* — все значения смещений на всех слоях; *ConvW* — число дополнительных битов, которые потребуются для хранения каждого веса; *ConvB* — число дополнительных битов, которые потребуются для хранения каждого смещения.

Если сеть обучена на большом наборе данных и имеет хорошую точность классификации, то *ConvW* и *ConvB* не должны быть очень большими по сравнению с размерностью карт признаков и входных изображений. На нашей тестовой сети было рассчитано $ConvW = 7$ и $ConvB = 3$.

Шаг 7. Далее для перехода к вычислениям с фиксированной точкой необходимо определить, сколько битов *N* достаточно хранить, чтобы вычисления не теряли точность, т. е. классификатор выдавал бы одинаковые результаты при вычислениях с плавающей и фиксированной точкой (рис. 4).

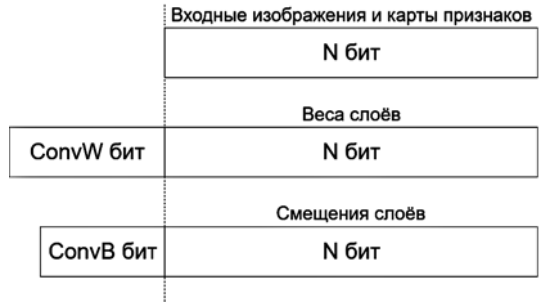

Рис. 4. Одинаковая размерность весов, смещений и карт признаков

Рис. 5. Разная размерность весов, смещений и карт признаков

Можно немного усложнить задачу и найти оптимальные размерности отдельно для карт признаков, весов и смещений (рис. 5).

Для этого можно использовать алгоритм следующего типа. Устанавливаем переменную *N* в некоторое большое значение, на котором сеть заведомо будет работать точно. Пропускаем все валидационные изображения через сеть, и сравниваем результат классификации с математической моделью с плавающей точкой. Если точность выше требуемой, уменьшаем *N* до тех пор, пока не найдем *N*, где точность ниже заданной. Затем пробуем отдельно уменьшить размерность весов *M* и смещений *K*, пока не найдем оптимум. Эта операция известна как квантование с калибровкой (quantization with calibration) [25].

Для разных датасетов для тестовой сети MobileNet из этого исследования значения *N*, *M* и *K* могут немного различаться. Значения варьировались $N \in [10, 12]$, $M \in [9, 11]$, $K \in [8, 10]$ (без учета знака чисел).

Шаг 8. Поскольку планируется выполнять операцию округления после каждого сверточного блока, нам требуется найти максимальное значение промежуточных вычислений, которое может быть достигнуто для заданных параметров. Стратегия выполнения округления может быть разной. Например, можно вы-

полнять округление после каждой элементарной операции сверточного блока: 9 умножений и 9 сложений (включая смещение). Но так как в модулярной арифметике операция округления является вычислительно затратной, то лучше выполнять ее после расчета карты признаков для следующего слоя или даже один раз за несколько слоев. Поэтому во всей сети требуется найти максимальное число сложений и умножений, которое будет выполнено, прежде чем будет выполнена операция округления. Если округление выполняется после каждого отдельного сверточного слоя, то рассчитать размерность динамического диапазона D можно по следующей формуле:

$$D = \lceil \log_2 S \rceil + N + (ConvW + M),$$

где S — требуемое число сложений. Округление после каждого сверточного слоя нужно делать на значение 2^M бит, аналогичное сдвигу на M позиций влево в позиционной записи числа.

Шаг 9. В зависимости от вычисленного динамического диапазона выбирается модулярный базис.

Шаг 10. После выбора системы модулей для представления чисел требуется преобразовать все веса и смещения сначала в двоичный вид и запись с фиксированной точкой и затем в модулярный вид на базе выбранной системы модулей и сохранить в ОЗУ устройства.

Шаг 11. MobileNet после редукции имеет структуру, состоящую из последовательности блоков "свертка" + "активация", т. е. в модулярном виде потребуется набор операций модульного канала, состоящих только из сложений и умножений. Рекомендуется сначала выполнить операцию активации RELU (1), т. е. сравнение с 0 и 1, чтобы уменьшить размерность в операции округления. Операции сравнения и округления можно выполнять параллельно с модульными вычислениями, как только будут готовы результаты вычислений первых пикселей для карт признаков. Иными словами, для вычисления результатов для всей сети потребуется набор из трех последовательных блоков операций:

- массивный набор быстрых модульных вычислений на сверточном блоке, включающий только параллельные умножения и сложения малой размерности;
- операция сравнения с 0 и 1;
- операция сдвига на M позиций в двоичной записи числа.

5. Метод округления чисел с фиксированной точкой в модулярной арифметике

За основу алгоритма для округления чисел в аппаратной реализации нейронной сети были взяты два метода из публикаций [23] и [20]. В первой статье предлагается метод для реализации операции расширения модулярного базиса (base extension) на основе Lookup-таблиц (LUT). Во второй статье приводится метод для округления на произвольное число K, взаимно простое со всеми модулями выбранного базиса. Метод также основан на базе LUT. За счет использования таблиц и небольшого арифметического блока скорость операции округления значительно увеличивается.

Рассмотрим подробнее метод расширения модулярного базиса. Пусть задан модулярный базис из взаимно простых оснований $\{p_1, p_2, ..., p_n\}$ и число $X = \{x_1, x_2, ..., x_n\}$ в рамках этого базиса. Применив китайскую теорему об остатках, можно получить:

$$X = \left| \sum_{i=1}^{n} M_i \left| \frac{x_i}{M_i} \right|_{p_i} \right|_M = \sum_{i=1}^{n} M_i \left| \frac{x_i}{M_i} \right|_{p_i} - eM,$$

где $e \in [0; n)$ и $M_i = \dfrac{M}{p_i}$.

В работе [23] доказано, что для любого модуля p_{n+1} значение $|X|_{p_{n+1}} = x_{n+1}$ может быть рассчитано по следующей формуле:

$$x_{n+1} = |X_E - eM|_{p_{n+1}} = \left| M^{(p)} \left| \sum_{i=1}^{n} a_{i,p} \right|_{\frac{M}{M^{(p)}}} + \sum_{i=1}^{p} \frac{M^{(p)}}{p_i} \left| \frac{M}{M^{(p)}} \left| \frac{x_i}{M_i} \right|_{p_i} \right|_{p_i} - eM \right|_{p_{n+1}}. \quad (1)$$

В этой формуле: $M^{(p)} = \prod_{j=1}^{p} p_j$, $M_i = \dfrac{M}{p_i}$, $a_{i,p} = \left\lfloor \dfrac{M_i \left| \frac{x_i}{M_i} \right|_{p_i}}{M^{(p)}} \right\rfloor$ и $e = \{0, 1\}$. Также в этой формуле p может принимать любое значение на промежутке от 1 до n: $1 \leqslant p < n$. Значение e равно 1 только при выполнении следующих условий:

$$\begin{cases} \left| \sum_{i=1}^{n} a_{i,p} \right|_{\frac{M}{M^{(p)}}} \geqslant \dfrac{M}{M^{(p)}} - p + 1; \\ |X_E|_M \leqslant (p-1)M^{(p)} - M^{(p)} \sum_{i=1}^{p} \dfrac{1}{p_i}. \end{cases} \quad (2)$$

Значения $a_{i,p}$ могут быть предварительно рассчитаны для заданного базиса и затем получены из таблиц. Значения $a_{i,p}$ лежат на интервале: $0 \leqslant a_{i,p} \leqslant \frac{M}{M^{(p)}} - \frac{M_i}{M^{(p)}}$. Тоже самое касается значений $b_i = \frac{M^{(p)}}{p_i}\left|\frac{M}{M^{(p)}}\left|\frac{x_i}{M_i}\right|_{p_i}\right|_{p_i}$, которые могут быть рассчитаны предварительно. Наибольшую сложность представляет вычислений условий, при которых $e = 1$.

Приняв $T = \left|\sum_{i=1}^{n} a_{i,p}\right|_{\frac{M}{M^{(p)}}}$, формулу (1) можно переписать в виде

$$x_{n+1} = \left|M^{(p)}T + \sum_{i=1}^{p} b_i - eM\right|_{p_{n+1}} = \\ = \left|\left|M^{(p)}T\right|_{p_{n+1}} + \left|\sum_{i=1}^{p} b_i\right|_{p_{n+1}} - e\left|M\right|_{p_{n+1}}\right|_{p_{n+1}}, \quad (3)$$

а условие (2) — в виде

$$\begin{cases} T \geqslant \frac{M}{M^{(p)}} - p + 1; \\ \left|M^{(p)}T + \sum_{i=1}^{p} b_i\right|_M \leqslant (p-1)M^{(p)} - M^{(p)}\sum_{i=1}^{p}\frac{1}{p_i}. \end{cases} \quad (4)$$

Предложенный метод позволяет быстро выполнить операцию расширения базиса. Теперь рассмотрим метод для округления на базе расширения модулярного базиса, предложенный в работе [20], т. е. требуется найти такое значение Y, что $Y = \frac{X}{K}$. Данную формулу можно переписать следующим образом: $Y = \frac{X - |X|_K}{K}$ или в модулярном виде:

$$y_i = |Y|_{p_i} = \left|\frac{X - |X|_K}{K}\right|_{p_i} = \left|x_i - |X|_K\right|_{p_i}\left|K^{-1}\right|_{p_i}\Big|_{p_i}. \quad (5)$$

Если выбрать K взаимнопростым со всеми базисными модулями, то $\left|K^{-1}\right|_{p_i}$ всегда существует. Это хорошо подходит для нашего метода, поскольку мы можем выбрать значение $K = 2^L$, если все базисные модули — это простые числа, большие 2. Если K зафиксировано, то значения $\left|K^{-1}\right|_{p_i}$ могут быть рассчитаны заранее и сохранены в таблицах. Для вычисления потребуется только модулярное вычитание и модулярное умножение на константу, эффективные реализации которых хорошо известны.

Заключение

В статье был предложен метод разработки аппаратной реализации нейронной сети на базе модулярной арифметики. Предложен маршрут реализации для известной мобильной нейронной сети MobileNet. Отмечены узкие места данной реализации, связанные с операциями сравнения и округления в модулярной арифметике. Предложен эффективный алгоритм округления для реализации в аппаратном модуле, а также функция активации, которая не содержит операцию сравнения.

Список литературы

1. **Simonyan K., Zisserman A.** Very deep convolutional networks for large-scale image recognition // arXiv preprint arXiv: 1409.1556. 2014.

2. **He K., Zhang X., Ren S., Sun J.** Delving deep into rectifiers: Surpassing human-level performance on imagenet classification // Proceedings of the IEEE international conference on computer vision. 2015. P. 1026—1034.

3. **Szegedy C., Liu W., Jia Y., Sermanet P., Reed S., Anguelov D., Rabinovich A.** Going deeper with convolutions. // Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1—9).

4. **He K., Zhang X., Ren S., Sun J.** Deep residual learning for image recognition // Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. P. 770—778.

5. **Ronneberger O., Fischer P., Brox T.** U-net: Convolutional networks for biomedical image segmentation // International Conference on Medical image computing and computer-assisted intervention. Springer, Cham, 2015. P. 234—241.

6. **Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., Kudlur M.** TensorFlow: A System for Large-Scale Machine Learning // OSDI. 2016. Т. 16. С. 265—283.

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

8. **Iandola F. N., Han S., Moskewicz M. W., Ashraf K., Dally W. J., Keutzer K.** SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size //arXiv preprint arXiv:1602.07360. 2016.

9. **Han S., Mao H., Dally W. J.** Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding // arXiv preprint arXiv:1510.00149. 2015.

10. **Jouppi N.** Google supercharges machine learning tasks with TPU custom chip //Google Blog, May. 2016. Vol. 18.

11. **Nakahara H., Sasao T.** A deep convolutional neural network based on nested residue number system // Field Programmable Logic and Applications (FPL), 2015 25th International Conference on. IEEE, 2015. P. 1—6.

12. **Arrigoni V., Rossi B., Fragneto P., Desoli G.** Approximate operations in Convolutional Neural Networks with RNS data representation // European Symposium on Artificial Neural Networks (ESANN 2017).

13. **Червяков Н. И., Ляхов П. А., Калита Д. И., Валуева М. В.** Архитектура сверточной нейронной сети с вычислениями

в системе остаточных классов с модулями специального вида // Нейрокомпьютеры: разработка, применение. 2017. № 1. С. 3—15.

14. **Solovyev R., Kustov A., Telpukhov D., Rukhlov V., Kalinin A.** Fixed-Point Convolutional Neural Network for Real-Time Video Processing in FPGA // 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). 2019. P. 1605—1611.

15. **Solovyev R. A., Kalinin A. A., Kustov A. G., Telpukhov D. V., Ruhlov V. S.** FPGA implementation of convolutional neural networks with fixed-point calculations. 2018. arXiv preprint arXiv:1808.09945.

16. **Cong J., Xiao B.** Minimizing computation in convolutional neural networks // International conference on artificial neural networks. Springer, Cham, 2014. P. 281—290.

17. **Brönnimann H., Emiris I., Pan V. Y., Pion S.** Sign determination in residue number systems // Theoretical Computer Science. 1999. Vol. 210. P. 173—197.

18. **Van Vu T.** Efficient implementations of the Chinese remainder theorem for sign detection and residue decoding // IEEE Transactions on Computers. 1985. Vol. 100, N. 7. P. 646—651.

19. **Hitz M. A., Kaltofen E.** Integer division in residue number systems // IEEE transactions on computers. 1995. Vol. 44, N. 8. P. 983—989.

20. **Kong Y., Phillips B.** Fast scaling in the residue number system // IEEE transactions on very large scale integration (VLSI) systems. 2009. Vol. 17, N. 3. P. 443—447.

21. **Low J. Y. S., Chang C. H.** A VLSI Efficient Programmable Power-of-Two Scaler for {2^{n}-1, 2^{n}, 2^{n} + 1} RNS // IEEE Transactions on Circuits and Systems I: Regular Papers. 2012. Vol. 59, N. 12. P. 2911—2919.

22. **Smyk R., Czyżak M., Ulman Z.** Scaling of signed residue numbers with mixed-radix conversion in FPGA with extended scaling factor selection //Computer Applications in Electrical Engineering. 2013. Vol. 11.

23. **Barsi F., Pinotti M. C.** Fast base extension and precise scaling in RNS for look-up table implementations // IEEE transactions on signal processing. 1995. Vol. 43, N. 10. P. 2427—2430.

24. **Kuznetsova A., Rom H., Alldrin N., Uijlings J., Krasin I., Pont-Tuset J., Ferrari V.** The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. 2018. arXiv preprint arXiv:1811.00982.

25. **Goncharenko A., Denisov A., Alyamkin S., Terentev E.** Fast adjustable threshold for uniform neural network quantization. 2018. arXiv preprint arXiv:1812.07872.

26. **Стемпковский А. Л., Амербаев В. М., Соловьев Р. А.** Принципы рекурсивных модулярных вычислений // Информационные технологии. 2013. № 2. С. 22—27.

27. **Амербаев В. М., Соловьев Р. А., Тельпухов Д. В.** Реализация библиотеки модульных арифметических операций на основе алгоритмов минимизации логических функций // Известия Южного федерального университета. Технические науки. 2013. Т. 7, № 144.

28. **Амербаев В. М., Соловьев Р. А., Тельпухов Д. В., Балака Е. С.** Построение обратных преобразователей модулярной арифметики с коррекцией ошибок на базе полиадического кода // Нейрокомпьютеры: разработка, применение. 2014. № 9. С. 30—35.

R. A. Solovyev, Ph.D., Chief Researcher, e-mail: turbo@ippm.ru,
D. V. Telpukhov, Ph.D., Head of the Department, **A. G. Kustov,** Engineer,
T. U. Isaeva, Ph.D., Researcher, **A. A. Volkov,** Engineer,
Institute for Design Problems in Microelectronics

Application of Modular Arithmetic Methods in the Development of Hardware Implementations of Neural Networks

The article proposes to use the methods of modular arithmetic for the implementation of neural networks at the hardware level in VLSI and FPGAs. Widely known mobile neural networks that are highly accurate and suitable for implementation in hardware are considered. The main difficulties in their implementation in the basis of modular arithmetic, namely the rounding operation and the comparison operation, are examined and analyzed. A number of methods have been proposed to solve these problems, such as using convolutions with stride greater than 1 instead of MaxPooling layers, using non-standard activations containing only addition, subtraction and multiplication operations, as well as an efficient algorithm for implementing rounding operations. Additionally, a complete route for designing and transferring a MobileNet neural network to the hardware level in a modular basis is proposed.

Keywords: *neural networks, modular arithmetic, residue number system, rounding operation, hardware implementation*

DOI: 10.17587/it.25.747-756

References

1. **Simonyan K., Zisserman A.** Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556, 2014.

2. **He K., Zhang X., Ren S., Sun J.** Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026—1034.

3. **Szegedy C., Liu W., Jia Y., Sermanet P., Reed S., Anguelov D., Rabinovich A.** Going deeper with convolutions, Cvpr, 2015.

4. **He K., Zhang X., Ren S., Sun J.** Deep residual learning for image recognition, *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770—778.

5. **Ronneberger O., Fischer P., Brox T.** U-net: Convolutional networks for biomedical image segmentation, *International Conference on Medical image computing and computer-assisted intervention*, Springer, Cham, 2015, pp. 234—241.

6. **Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., Kudlur M.** TensorFlow: A System for Large-Scale Machine Learning, *OSDI*, 2016, vol. 16, pp. 265—283.

7. **Howard A. G., Zhu M., Chen B., Kalenichenko D., Wang W., Weyand T., Adam H.** Mobilenets: Efficient convolutional neural

networks for mobile vision applications, arXiv preprint arXiv: 1704.04861, 2017.

8. **Iandola F. N., Han, S., Moskewicz M. W., Ashraf K., Dally W. J., Keutzer K.** SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size, arXiv preprint arXiv:1602.07360, 2016.

9. **Han S., Mao H., Dally W. J.** Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, arXiv preprint arXiv:1510.00149, 2015.

10. **Jouppi N.** Google supercharges machine learning tasks with TPU custom chip, *Google Blog,* May, 2016, vol. 18.

11. **Nakahara H., Sasao T.** A deep convolutional neural network based on nested residue number system, Field Programmable Logic and Applications (FPL), 2015 *25th International Conference on, IEEE*, 2015, pp. 1—6.

12. **Arrigoni V., Rossi B., Fragneto P., Desoli G.** Approximate operations in Convolutional Neural Networks with RNS data representation, *European Symposium on Artificial Neural Networks (ESANN 2017).*

13. **Chervyakov N. I., Lyakhov P. A., Kalita D. I., Valueva M. V.** Convolutional neural network architecture using computations in residue number system with specific module set, *Neirocomputery: Razrabotka, Primenenie*, 2017, no. 1, pp. 3—15 (in Russian).

14. **Solovyev R., Kustov A., Telpukhov D., Rukhlov V., Kalinin A.** Fixed-Point Convolutional Neural Network for Real-Time Video Processing in FPGA. In 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), IEEE, 2019, January, pp. 1605—1611.

15. **Solovyev R. A., Kalinin A. A., Kustov A. G., Telpukhov D. V., Ruhlov V. S.** FPGA implementation of convolutional neural networks with fixed-point calculations, 2018, arXiv preprint arXiv:1808.09945.

16. **Cong J., Xiao B.** Minimizing computation in convolutional neural networks, *International conference on artificial neural networks*, Springer, Cham, 2014, pp. 281—290.

17. **Brönnimann H., Emiris I., Pan V. Y., Pion S.** Sign determination in residue number systems, *Theoretical Computer Science*, 1999, vol.210, pp. 173—197.

18. **Van Vu T.** Efficient implementations of the Chinese remainder theorem for sign detection and residue decoding, *IEEE Transactions on Computers*, 1985, vol. 100, no. 7, pp. 646—651.

19. **Hitz M. A., Kaltofen E.** Integer division in residue number systems, *IEEE transactions on computers*, 1995, vol. 44, no. 8, pp. 983—989.

20. **Kong Y., Phillips B.** Fast scaling in the residue number system, *IEEE transactions on very large scale integration (VLSI) systems*, 2009, vol. 17, no. 3, pp. 443-447.

21. **Low J. Y. S., Chang C. H.** A VLSI Efficient Programmable Power-of-Two Scaler for {2^{n}-1, 2^{n}, 2^{n} + 1} RNS, *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2012, vol. 59, no. 12, pp. 2911—2919.

22. **Smyk R., Czyżak M., Ulman Z.** Scaling of signed residue numbers with mixed-radix conversion in FPGA with extended scaling factor selection, *Computer Applications in Electrical Engineering*, 2013, vol. 11.

23. **Barsi F., Pinotti M. C.** Fast base extension and precise scaling in RNS for look-up table implementations, *IEEE transactions on signal processing*, 1995, vol. 43, no. 10, pp. 2427—2430.

24. **Kuznetsova A., Rom H., Alldrin N., Uijlings J., Krasin I., Pont-Tuset J., Ferrari V.** The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale, 2018, arXiv preprint arXiv:1811.00982.

25. **Goncharenko A., Denisov A., Alyamkin S., Terentev E.** Fast adjustable threshold for uniform neural network quantization, 2018, arXiv preprint arXiv:1812.07872.

26. **Stempkovsky A. L., Amerbaev V. M., Solovyev R. A.** Principles of Recursive Modular Arithmetic, *Informacionnye tekhnologii*, 2013, no. 2, pp. 22—27 (in Russian).

27. **Amerbaev V. M., Solovyev R. A., Telpukhov D. V.** Library implementation of modular arithmetic operations based on logic functions minimization algorithms, *Izvestiya Yughnogo Federalnogo Universiteta. Technicheskie Nauki,* 2013, vol. 7, no. 144 (in Russian).

28. **Amerbaev V. M., Solovyev R. A., Telpukhov D. V., Balaka E. S.** Construction of residue number system reverse converters with error correction, based on mixed-number system, *Neirocomputery: Razrabotka, Primenenie*, 2014, no. 9, pp. 30—35 (in Russian).