УДК 004.052.32 DOI 10.31114/2078-7707-2018-3-126-131

Методика реализации нейронной сети для распознания рукописных цифр в FPGA на основе вычислений с фиксированной точкой

Р.А. Соловьев, А.Г. Кустов, В.С. Рухлов

Институт проблем проектирования в микроэлектронике РАН, г. Москва, turbo@ippm.ru

***Аннотация* — В работе исследуются нейронные сети для обработки видеопотока в реальном времени на оборудовании с ограниченными вычислительными ресурсами. Для решения проблем, связанных с производительностью аппаратуры, в работе предлагается перейти от вычислений на программном уровне к аппаратной реализации, а также перейти от вычислений с плавающей точкой к фиксированной точке. Предлагается набор методов для проектирования нейросети, которая будет наиболее быстрой и точной в случае использования фиксированной точки. Приводится пример проектирования устройства для решения конкретной практической задачи. Показывается, как можно приспособить существующие наборы данных (датасеты) для использования в другой задаче, где данные отличаются. Приводится пример разработки проекта для прототипирования в ПЛИС с использованием камеры и устройства вывода. Финальный проект, реализованный на базе предложенных методик, успешно работает в реальном времени на студенческой плате De0Nano на базе ПЛИС Cyclone IV.**

***Ключевые слова* — свёрточные нейронные сети, ПЛИС, фиксированная точка, двухмерная свёртка.**

I. Введение

Последние исследования в области нейронных сетей показали, что они хорошо справляются с множеством задач, связанных с классификацией и обработкой изображений, аудио и видео данных, причем в некоторых случаях даже лучше человека [1, 2]. Большинство современных архитектур имеют в своем составе convolution (свёрточные) блоки (например, VGG [1], Inception [3], ResNet [4], U-Net [5]). Такие сети называются convolutional neural nets (CNN). А размерность и 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, как следствие, дорогие GPU (видеокарты) [6]. Особенно это актуально для обработки видеоинформации в реальном времени.

Некоторые структуры нейросетей при очень высокой точности классификации изображений обладают свойствами, которые легко переносятся на аппаратную платформу:

1) Высокая регулярность – все слои имеют похожую структуру (Conv3x3, Conv1x1, MaxPooling, Fully-Connected, GlobalAvgPooling).

2) Маленькая размерность блока свёртки: 3x3.

3) Активация Relu (сравнение значения с 0), в отличие от популярных ранее функций активации Sigmoid и Tanh, обладающих большей вычислительной сложностью.

4) Из-за высокой регулярности размер нейронной сети может гибко регулироваться. Например, за счёт разного числа блоков свёртки (чем их больше, тем быстрее будет работать нейронная сеть). В случае ПЛИС или СБИС, например, это позволит прошивать одну и ту же нейронную сеть на разные кристаллы, с разной скоростью работы.

Существует также направление по проектированию нейронных сетей для использования в мобильных устройствах: MobileNets [7], SqueezeNet [8]. Они отличаются малым числом весов и относительно небольшим числом арифметических операций. Однако они тоже выполняются на программном уровне и используют для работы вычисления с плавающей точкой. В устройствах, где требуется высокая производительность, например, обработка видео в реальном времени (скорость обработки 30 кадров в секунду), использование даже мобильных сетей может быть недостаточно, без существенной оптимизации.

Для использования нейросетей на этапе когда у нас есть обученная модель в реальном устройстве, применение набора оптимизаций может ускорить выполнение вычислений в несколько раз. Для этого уже существует 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 счёт:

1) Аппаратный блок для вычисления свёртки, работает быстрее, чем свертка, выполненная на программном уровне.

2) Переход от вычислений с плавающей запятой к фиксированной запятой.

3) Уменьшение размерности вычислений с сохранением приемлемой точности.

4) Сокращение части нейронной сети с сохранением точности классификации.

5) Модификации структуры нейронной сети с незначительным уменьшением точности (или даже без неё) с увеличением скорости работы и уменьшением размера аппаратной реализации и хранимых весов.

6) Использование нескольких свёрточных блоков, работающих параллельно.

II. Подготовка набора данных для обучения

Для задачи распознания цифр существует известный набор данных MNIST [12]. Однако он сильно отличается от изображений, полученных с камеры (рис 1). По этой причине в MNIST потребуется внести изменения.

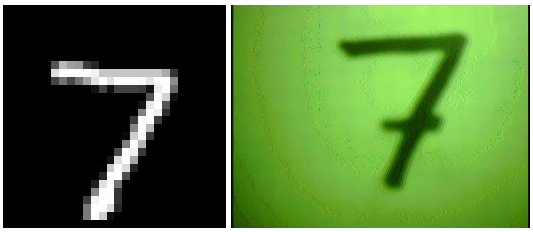

Рис 1. Изображение из набора MNIST (слева) и изображение, полученное с камеры (справа)

Главные отличия:

1) Изображение с MNIST светлые цифры на тёмном фоне, тогда как на камере наоборот, тёмные цифры на светлом фоне.

2) Изображение с камеры приходит цветным.

3) Размер изображений MNIST 28 на 28 пикселов, а изображение с камеры 320x240 пикселов.

4) На изображении с камеры присутствуют шумы и артефакты, а цифры могут быть сдвинуты и немного повёрнуты, тогда как в цифры в MNIST всегда находятся строго по центру, а фон всегда однородный и идеально чёрный.

5) В отличие от MNIST нам также потребуется добавить отдельный одиннадцатый класс «отсутствие цифры на изображении».

Максимальная точность распознания для набора MNIST крайне высокая - современные сети распознают цифры с точностью более 99.5% [13]. Было принято решение уменьшать изображения с камеры до 28x28 пикселов и переводить изображения в градации серого. Тем самым решаются следующие проблемы:

1) Мы существенно не теряем в точности, так как даже на небольших изображениях цифры всё равно легко распознаются человеком.

2) Для однородных цифр цвет не имеет значения и не влияет на их распознание на изображении.

3) Шумное изображение с камеры при уменьшении и усреднении соседних пикселов становится более чётким. Часть шумов нейтрализуется.

Поскольку преобразование изображения тоже будет выполняться на аппаратном уровне, то требуется заранее продумать минимальный набор арифметических функций, которые эффективно приведут изображение к нужному нам виду.

Алгоритм для перевода изображений в нужный формат следующий:

1) Из изображения 320x240 берётся только центральная часть размером 224x224 пиксела. Выбор числа 224 обусловлен тем фактом, что 224 = 28*8, что позволит впоследствии легко перейти к нужному размеру изображения.

2) Далее преобразуем эту часть изображения в оттенки серого. В связи с особенностями человеческого зрения [14] для перевода в оттенки серого используется не среднее, а взвешенное среднее с коэффициентами порядка 60% на зелёный, 30% на красный и 10% на синий. Для облегчения перевода на аппаратном уровне мы использовали следующую формулу:

$$BW = (8*G + 5*R + 3*B) / 16$$

Умножение на 8 и, самое главное, деление на 16 позволяет обойтись просто сдвигами.

3) Далее изображение 224x224 разбивается на блоки 8x8. В каждом из таких блоков ищется среднее значение, формируя соответствующий пиксель в изображении 28x28.

Полученный алгоритм работает очень быстро на аппаратном уровне.

Чтобы использовать изображения из базы данных MNIST для обучения нейросети использовался метод генерации аугментированных изображений на лету. В этом методе при формировании следующего набора (mini-batch) для обучения к каждому изображению применяется набор различных фильтров. Эта техника используется для увеличения размера набора данных без существенных усилий, а так же для приведения этих данных к требуемому виду как в нашем случае.

Применялся следующий набор фильтров:

1) Инверсия цвета.

2) Случайный поворот на 10 градусов в обе стороны.

3) Случайное увеличение или уменьшение изображения на 4 пикселя.

4) Случайное изменение интенсивности изображения (от 0 до 80).

5) Добавление случайного шума от 0 до 10 процентов.

Дополнительно в наборы данных для обучения подмешивались реальные изображения, полученные с камеры.

III. Разработка нейронной сети с повышенной скоростью работы

Типовая структура свёрточной нейронной сети приведена на рис. 2. Позже появились некоторые усовершенствованные структуры с разветвлениями [3-4], но суть осталась той же: размер изображения от слоя к слою уменьшается, а количество фильтров растёт. В конце свёрточной сети образуется набор признаков, которые подаются на классификационный слой (или слои) и выходные нейроны сигнализируют вероятность принадлежности изображения к конкретному классу.

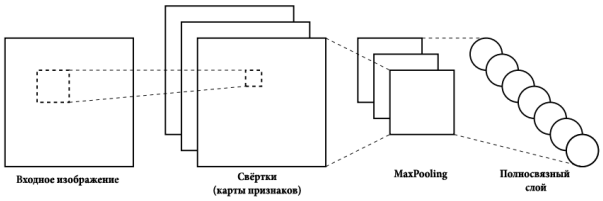

Рис. 2. Структура свёрточной нейросети

Мы предлагаем следующий набор правил для конструирования нейронной сети для минимизации общего числа хранимых весов (что критично для мобильных систем) и облегчения перехода к вычислениям с фиксированной точкой.

1) Требуется минимизировать число полносвязных слоёв, которые являются основным потребителем памяти для хранения весов. В идеале полносвязный слой должен быть только один на последнем слое нейронной сети.

2) По возможности сократить число фильтров на каждом свёрточном слое. Но так как каждое такое сокращение может привести к ухудшению свойств сети, то важно найти хороший баланс между точностью и количеством весов.

3) Отказаться от использования смещения (bias). Это важно при переходе от плавающей точки к фиксированной точке. Из-за того что наличие сложения с константой мешает контролировать диапазон значений и ошибка от округления смещения на каждом слое имеет свойство накапливаться.

4) Использовать активации простого типа. Наиболее подходящая активация RELU. Следует избегать активаций sigmoid, tahn и т.д., которые содержат деление и возведение в степень, и другие сложные с точки зрения аппаратной реализации функции.

5) Минимизировать число разнородных слоёв, чтобы один аппаратный блок мог выполнять вычисления на большом числе участков маршрута.

Перед переводом нейронной сети в аппаратный вид её требуется обучить на подготовленном наборе данных и получить её математическую модель для тестирования. Математическая модель готовилась на базе языка Python и модуля для работы с нейронными сетями Keras с бекэндом Tensorflow. В своей прошлой публикации [15] мы предложили архитектуру нейронной сети VGG Simple, основанной на принципах эффективной нейронной сети VGG16.

Несмотря на высокую скорость работы главной проблемой этой модели стало количество весов, которые не помещались во внутреннюю память, а обмен с внешней памятью работает не так быстро, как хотелось бы. Дополнительно в этой модели присутствует «bias», который тоже надо хранить и который требует дополнительных блоков для обработки и имеет тенденцию к накоплению ошибки при переходе к фиксированной точке. Поэтому мы видоизменили сеть, избавившись от больших полносвязных слоёв и от смещения. Добавили слой GlobalMaxPooling, вместо GlobalAvgPooling, который обычно используют, например в ResNet50. Эффективность у них примерно одинаковая, но найти максимум с вычислительной точки зрения проще, чем среднее.

Далее убедились, что это не привело к снижению точности работы нейросети. Новая структура приведена на рис. 3.

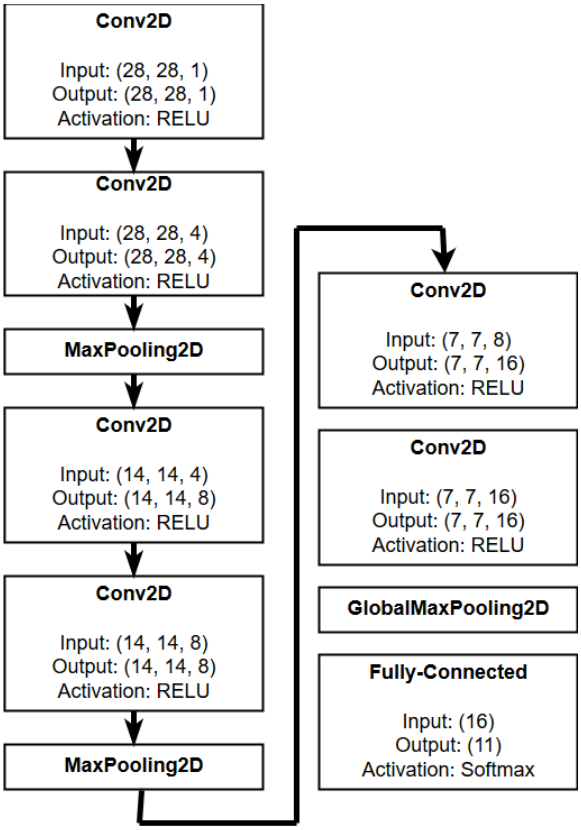

Рис. 3. Структура нейронной сети Low Weights Digit Detector

Изменение структуры нейросети позволило снизить количество весов с 25 тысяч до примерно 4.5 тысяч. Чего вполне достаточно для хранения весов во внутренней памяти ПЛИС.

На модифицированном наборе MNIST с аугментациями нейронная сеть довольно быстро тренируется до ~96% точности.

IV. Перевод вычислений от плавающей точки к фиксированной точке

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. 4) – в большинстве свёрточных нейросетей этот блок является основным.

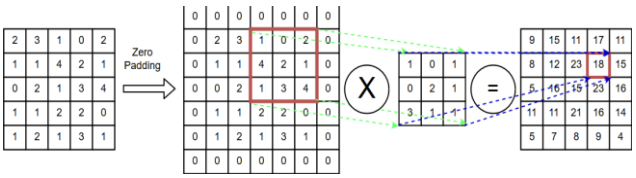

Рис. 4. Схематическое изображение операции Convolution

На входе слоя находится двухмерная матрица (исходная картинка) 28x28, значения которой находятся на интервале [0; 1).

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

Формула для расчета конкретного пикселя в позиции (i, j) второго слоя с учетом того, что используется Convolution с размером 3x3, имеет следующий вид:

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

Поскольку веса $w_{i,j}$ и смещение b известны, то можно рассчитать потенциальный минимум mn и максимум mx на втором слое. Пусть $M = \max(|mx|, |mn|)$. Если разделить $w_{i,j}$ и b на значение M, то мы можем гарантировать, что при любой конфигурации входных данных значение на втором слое не превысит 1. Назовём M коэффициентом редукции слоя. На втором слое получается такая же ситуация, как и на первом, а именно, на входе слоя значение из промежутка [0; 1) и рассуждения можно повторить.

Можно легко показать, что для предложенной нейронной сети на последнем слое после всех редукций весов позиция максимума на последнем нейроне не изменится, то есть 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 точкой 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 $z' = xb + yb = \lfloor x \cdot 2^N \rfloor + \lfloor y \cdot 2^N \rfloor = \lfloor (x + y) * 2^N \rfloor = \lfloor z * 2^N \rfloor = \lfloor zb \rfloor$,

умножение $z' = xb \cdot yb = \lfloor x \cdot 2^N \rfloor \cdot \lfloor y \cdot 2^N \rfloor = \lfloor (x \cdot y) \cdot 2^N \cdot 2^N \rfloor = \lfloor z \cdot 2^N \cdot 2^N \rfloor = \lfloor zb \cdot 2^N \rfloor$,

то есть 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, либо ориентируясь на значение 3 sigma или же увеличивая максимум 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 свёртки.

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. Чтобы проверить корректность работы требуется прогнать все тестовые изображения на математической модели с плавающей точкой и на математической модели (или в тестовой среде на языке Verilog HDL) с фиксированной точкой

и затем сравнить ответы на выходе нейронной сети. Отношение количества несовпадений к общему числу тестов и будет мерой ошибки для заданной разрядности весов и промежуточных вычислений. При выборе разрядности можно ориентироваться на разрядность, при которой число ошибок равно 0.

При вычислении на свёрточных блоках возможны две различных стратегии:

- округление после каждой элементарной операции сложения и умножения,

- вычисление с полной точностью и округление в самом конце операции свёртки.

Было проведено два эксперимента чтобы определить, что выгоднее (см. таблицу 1). В случае с округлением в начале для нулевого расхождения с математической моделью требуется 17 бит на хранение числа, для округления в конце всего 12 бит.

Таблица 1

Ошибка по сравнению с математической моделью при различной стратегии округления

Разрядность весов	**Округление в начале (%)**	**Округление в конце (%)**
10	32.03	1.96
11	15.03	0.65
12	11.76	0
13	9.15	0
14	3.27	0
15	1.96	0
16	0.65	0
17	0	0
18	0	0

Округление после каждой операции немного увеличивает скорость, но существенно увеличивает накладные расходы на память. Поэтому в общем случае выгодно выполнять операцию округления после свёрточного блока.

V. Экспериментальные результаты

Проект, был успешно синтезирован в ПЛИС Cyclone IV на базе студенческой отладочной платы De0Nano. Подробности по занимаемым логическим ячейкам и памяти приведены в таблице 2.

Обработка изображений происходит в реальном времени и исходное изображение вместе с полученным результатом выводится на дисплей. На классификацию одного изображения требуется порядка 230 тысяч тактов и общая скорость обработки получилась с большим запасом в 150 кадров/сек.

При нехватке производительности и в случае наличия свободных логических ячеек ускорить вычисления можно за счёт добавления дополнительных свёрточных блоков, которые выполняют вычисления в параллельном режиме, как это было показано в [15].

Таблица 2

Информация по ресурсам, занимаемым устройством в ПЛИС после этапа Place & Route

	Лог. ячейки (доступно 22320)	9-bit elements (132)	Внутр. память (доступно: 608256 бит)	PLLs (4)
Преобр. входного изображения	964 (4%)	0 (0%)	0 (0%)	0 (0%)
Нейронная сеть	4014 (18%)	23 (17%)	285428 (47%)	0 (0%)
База данных с весами	0 (0%)	0 (0%)	70980 (12%)	0 (0%)
Хранение промежут. результатов	1 (<1%)	0 (0%)	214448 (35%)	0 (0%)
Всего занято	5947 (27%)	23 (17%)	371444 (61%)	2 (50%)

Демонстрационное видео с результатом работы доступно на Youtube [16]. Verilog код для устройства и схема подключения компонентов доступны на GitHub [17].

Поддержка

Работа выполнена при поддержке гранта РФФИ 17-07-00409.

Литература

[1] Very Deep Convolutional Networks for Large-Scale Image Recognition https://arxiv.org/pdf/1409.1556v6.pdf (дата обращения: 16.05.2018)

[2] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification http://arxiv.org/abs/1502.01852. (дата обращения: 16.05.2018)

[3] Going Deeper with Convolutions http://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf (дата обращения: 16.05.2018)

[4] Deep Residual Learning for Image Recognition http://arxiv.org/abs/1512.03385 (дата обращения: 16.05.2018)

[5] U-Net: Convolutional Networks for Biomedical Image Segmentation http://arxiv.org/abs/1505.04597 (дата обращения: 16.05.2018)

[6] Abadi M. et al. TensorFlow: A System for Large-Scale Machine Learning // OSDI. – 2016. – Т. 16. – С. 265-283.

[7] Howard A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications // arXiv preprint arXiv:1704.04861. – 2017.

[8] Iandola F. N. et al. 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. – T. 18.
[11] URL: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=593&PartNo=2 (дата обращения: 16.05.2018)
[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998
[13] Regularization of Neural Networks using DropConnect, 2013, https://cs.nyu.edu/~wanli/dropc/dropc.pdf (дата обращения: 16.05.2018)
[14] Wilhelm Burger, Mark J. Burge (2010). Principles of Digital Image Processing Core Algorithms. Springer Science & Business Media. pp. 110–111. ISBN 978-1-84800-195-4.
[15] Соловьев Р.А., Кустов А.Г., Рухлов В.С., Щелоков А.Н., Пузырьков Д.В. Аппаратная реализация свёрточной нейронной сети в ПЛИС на базе вычислений с фиксированной точкой // Известия ЮФУ. Технические науки. 2017. № 7 (192). С. 186-197.
[16] URL: https://www.youtube.com/watch?v=Lhnf596o0cc (дата обращения: 16.05.2018)
[17] URL: https://github.com/ZFTurbo/Verilog-Generator-of-Neural-Net-Digit-Detector-for-FPGA (дата обращения: 16.05.2018)

A Technique for Implementing a Neural Network for Recognizing Handwritten Digits in FPGAs Based on Fixed Point Calculations

R.A. Solovyev, D.V. Telpukhov, A.G. Kustov, V.S. Rukhlov

Institute for Design Problems in Microelectronics of RAS, Moscow, turbo@ippm.ru

***Abstract* — The article proposes the implementation of a convolutional neural network in FPGA for the solution of a practical problem, namely the recognition of handwritten digits from video stream. The first part of the article is devoted to the adaptation of the MNIST dataset to the problem of recognition of digits from a real camera. A new proposed training method allowed to avoid the long process of collecting data and to confine only to a small set in addition to the modified MNIST. The second part of the article is devoted directly to the method of implementing a neural network in a hardware device. Requirements for real-time video processing on a neural network imposed greater restrictions on the device performance.**

Thanks to the proposed approaches, it became possible to achieve the required performance characteristics and even exceed them by several times. Such indicators were achieved due to a number of optimizations. A neural network with a small number of weights and architectural features aimed at reducing consumed resources was designed. Calculations by a special algorithm are transferred from floating point to fixed point. It turned out that only 12 bits are sufficient to store the weights, so that there are no differences from the mathematical model. The proposed device was implemented and tested on a De0Nano debug card based on the Cyclone IV FPGA. The processing speed of the video stream exceeded 150 frames per second. The code for training the neural network and for generating the Verilog code of the target device was published in Open Source.

***Keywords* — convolutional neural nets, FPGA, fixed point calculations, 2D convolution**

REFERENCES

[1] Very Deep Convolutional Networks for Large-Scale Image Recognition https://arxiv.org/pdf/1409.1556v6.pdf (access date: 16.05.2018).
[2] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification http://arxiv.org/abs/1502.01852 (access date: 16.05.2018).
[3] Going Deeper with Convolutions http://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf (access date: 16.05.2018).
[4] Deep Residual Learning for Image Recognition http://arxiv.org/abs/1512.03385 (access date: 16.05.2018).
[5] U-Net: Convolutional Networks for Biomedical Image Segmentation http://arxiv.org/abs/1505.04597 (access date: 16.05.2018).
[6] Abadi M. et al. TensorFlow: A System for Large-Scale Machine Learning //OSDI. – 2016. – T. 16. – С. 265-283.
[7] Howard A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications //arXiv preprint arXiv:1704.04861. – 2017.
[8] Iandola F. N. et al. 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. – T. 18.
[11] http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=593&PartNo=2 (access date: 15.02.2018)
[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998
[13] Regularization of Neural Networks using DropConnect, 2013, https://cs.nyu.edu/~wanli/dropc/dropc.pdf (access date: 16.05.2018).
[14] Wilhelm Burger, Mark J. Burge (2010). Principles of Digital Image Processing Core Algorithms. Springer Science & Business Media. pp. 110–111. ISBN 978-1-84800-195-4.
[15] Solovyev R.A., Kustov A.G., Rukhlov V.S., Shelokov A.N., Puzyrkov D.V. Apparatnaya realizaciya svyortochnoj nejronnoj seti v PLIS na baze vychislenij s fiksirovannoj tochkoj (Hardware implementation of a convolutional neural network in FPGAs based on fixed point calculations) // Izvestiya YUFU. Tekhnicheskie nauki. 2017. № 7 (192). pp. 186-197.
[16] URL: https://www.youtube.com/watch?v=Lhnf596o0cc (access date: 15.02.2018)
[17] URL: https://github.com/ZFTurbo/Verilog-Generator-of-Neural-Net-Digit-Detector-for-FPG (access date: 16.05.2018.