АППАРАТНАЯ РЕАЛИЗАЦИЯ СВЕРТОЧНОЙ НЕЙРОСЕТИ НА ПЛИС С ИСПОЛЬЗОВАНИЕМ МОДЕЛЬНО-ОРИЕНТИРОВАННОГО ПРОЕКТИРОВАНИЯ

инж. Воробьев А.Н; инж., к.т.н. Шидловский Д.Ю.; инж., к.т.н. Багров А. А.

Центр инженерных технологий и моделирования «Экспонента»

В последнее время во многих областях нашли применение сверточные нейронные сети – от задач распознавания образов до задач удаления шума и повышения разрешения изображений. Однако запуск сверточных нейронных сетей на ПЛИС имеет ряд трудностей – это отсутствие готовых библиотек, высокая сложность разработки и отладки алгоритма на встраиваемой платформе. В этой работе предлагается обойти описанные выше проблемы, используя модельно-ориентированное проектирование: вначале создать модель алгоритма нейронной сети в среде MATLAB/Simulink, а после из модели по средствам автоматической генерации кода получить HDL-описание алгоритма нейронной сети для ПЛИС. В качестве примера рассматривается задача распознавания рукописных цифр.

Введение

В последнее время во многих областях нашли применение сверточные нейронные сети – от задач удаления шума и повышения разрешения изображений до задач построения современных систем содействия водителю (Advanced Driver-Assistance Systems – ADAS) способных распознавать окружающие его объекты: автомобили, дорожные знаки, пешеходов, линии разметки. Однако к этим системам предъявляется ряд дополнительных требований – обработка видео в режиме реального времени и возможность запуска алгоритма на встраиваемых платформах.

Однако запуск сверточных нейронных сетей на ПЛИС и на других встраиваемых системах имеет ряд трудностей [1] – это отсутствие готовых библиотек, ограниченные вычислительные ресурсы аппаратной платформы, высокая сложность разработки и отладки алгоритма на встраиваемой платформе. В этой работе предлагается обойти описанные выше проблемы используя модельно-ориентированное проектирование и среду разработки MATLAB/Simulink.

Структурно процесс разработки с использованием модельно-ориентированного проектирования можно разделить на три основных этапа: создание модели алгоритма и уточнение его параметров, подготовка алгоритма к аппаратной реализации и автоматическая генерация кода (рисунок 1).

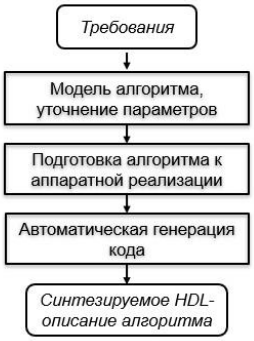


Рисунок 1. Этапы разработки при модельно-ориентированном проектировании.

На первом шаге создается структура нейронной сети в MATLAB, сеть обучается на тестовых данных и при необходимости ее структура уточняется. На следующем шаге реализуется потоковая модель алгоритма нейронной сети в среде Simulink для работы с потоковым видео, пригодная для аппаратной реализации на ПЛИС. И на третьем шаге, используя автоматическую генерацию кода из Simulink-моделей, получаем HDL-код с описанием алгоритма нейронной сети, пригодного для синтеза и запуска его на ПЛИС.

В этой работе в качестве примера по аппаратной реализации сверточной нейронной сети выбрана сеть для распознавания рукописных цифр (рисунок 2). Впоследствии созданную нейронную сеть можно обучить на другом наборе данных – дорожных знаках, автомобильных номерах – и использовать в системах содействия водителю или в системах фотовидеофиксации для распознавания автомобильных номеров.

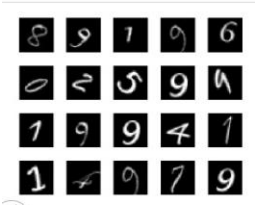
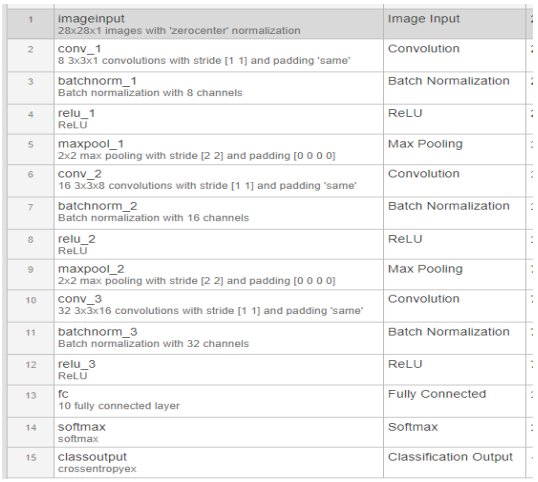


Рисунок 2. Пример изображений рукописных цифр.

Создание архитектуры нейронной сети

На первом шаге для разработки архитектуры сети или модели алгоритма согласно концепции модельно-ориентированного проектирования, использовался набор изображений, содержащий по 1000 изображений разрешением 28×28 пикселей для каждой цифры. Обучение проводилось по 750 изображений для каждой цифры, а на остальных данных проверялась точность обученной сети.

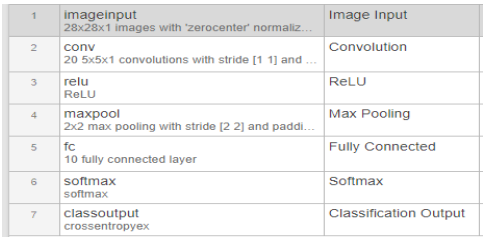
Изначально сеть имела 15 слоев и структуру, показанную на рисунке 3. Данная архитектура сети на тестовом наборе изображений показывала точность распознавания 99,8%.



1	imageinput 28x28x1 images with 'zerocenter' normalization	Image Input
2	conv_1 8 3x3x1 convolutions with stride [1 1] and padding 'same'	Convolution
3	batchnorm_1 Batch normalization with 8 channels	Batch Normalization
4	relu_1 ReLU	ReLU
5	maxpool_1 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling
6	conv_2 16 3x3x8 convolutions with stride [1 1] and padding 'same'	Convolution
7	batchnorm_2 Batch normalization with 16 channels	Batch Normalization
8	relu_2 ReLU	ReLU
9	maxpool_2 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling
10	conv_3 32 3x3x16 convolutions with stride [1 1] and padding 'same'	Convolution
11	batchnorm_3 Batch normalization with 32 channels	Batch Normalization
12	relu_3 ReLU	ReLU
13	fc 10 fully connected layer	Fully Connected
14	softmax softmax	Softmax
15	classoutput crossentropyex	Classification Output

Рисунок 3. Структура слоев в исходной нейронной сети.

Для создания нейронной сети и ее обучения использовались функции из библиотеки Deep Learning Toolbox среды MATLAB. Дальнейшие действия были направленны на оптимизацию архитектуры сети с точки зрения уменьшения вычислительной сложности сети (количества математических операций на классификацию одного изображения) для повышения ее быстродействия при небольшой потере предсказательной точности, и как следствие, последующей экономии ресурсов на ПЛИС (уменьшения количества требуемых аппаратных умножителей и сокращения памяти для хранения коэффициентов (весов) сети). В ходе работы была получена архитектура нейронной сети, состоящая из 7 слоев с точностью распознавания 98,8% (рисунок 4).



1	imageinput 28x28x1 images with 'zerocenter' normaliz...	Image Input
2	conv 20 5x5x1 convolutions with stride [1 1] and ...	Convolution
3	relu ReLU	ReLU
4	maxpool 2x2 max pooling with stride [2 2] and paddi...	Max Pooling
5	fc 10 fully connected layer	Fully Connected
6	softmax softmax	Softmax
7	classoutput crossentropyex	Classification Output

Рисунок 4. Структура слоев в оптимизированной нейронной сети.

Таким образом, на первом этапе удалось упростить архитектуру нейронной сети и получить сеть с меньшим количеством слоев. Количество слоев удалось сократить более чем в два раза (с 15 до 7), при этом точность распознавания снизилась всего на 1% (с 99,8% до 98,8%), что не является критичным для нашей задачи, так как акцент в нашей работе направлен на реализацию сети на аппаратной платформе и сокращение используемых ресурсов.

Подготовка алгоритма к аппаратной реализации на ПЛИС

На этом этапе требовалось подготовить алгоритм для аппаратной реализации на ПЛИС. Стояла задача перевести алгоритм на работу с данными с фиксированной точкой и с потоковой архитектурой. В отличие от микропроцессорной архитектуры, где данные, как правило, обрабатываются кадрами, на ПЛИС данные поступают и обрабатываются, как правило, потоково – попиксельно и в фиксированной точке [2].

Перед потоковой реализацией алгоритма нейронной сети для наглядности происходящих вычислений в ней была создана функциональная модель сети с использованием базового функционала MATLAB и без использования функций Deep Learning Toolbox. Каждый

слой сети был описан в виде отдельной функции, принимающей на вход данные и обученные веса сети.

На основе функциональной модели сети в MATLAB была создана Simulink-модель алгоритма нейронной сети с потоковой архитектурой, работающая с входным изображением попиксельно и пригодная для аппаратной реализации на ПЛИС (рисунок 5).

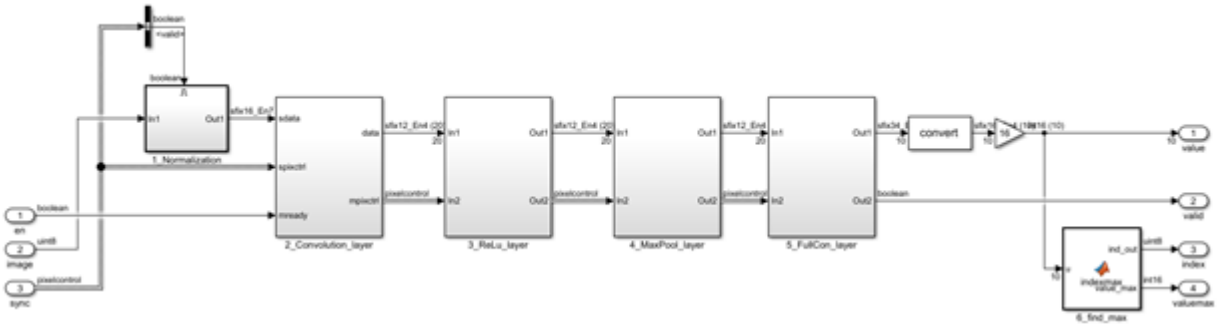


Рисунок 5. Модель алгоритма нейронной сети в среде Simulink.

В Simulink были реализованы первые 5 слоев нейронной сети за исключением 2-х последних слоев. Задача предпоследнего слоя «softmax» и последнего слоя «classification» – это нормировка вероятностей в диапазоне [0…1] и вывод метки для класса с максимальной вероятностью, но в нашей задаче не было принципиальным определение нормированного значения вероятности класса, поэтому для сокращения вычислений эти блоки были заменены на один простой блок нахождения максимума сигнала; номер проводника с максимальным сигналом и есть число с максимальной вероятностью.

Обученные коэффициенты сети, полученные на первом этапе, сохраняются в mat-файле и подгружаются при инициализации Simulink-модели. При переводе модели алгоритма в фиксированную точку, требуемая разрядность данных для распознанного числа оценивается по изменению значения вероятности. Разрядность данных подбиралась так, чтобы изменение вероятности происходило не ранее чем в третьем знаке числа, в результате для данных выбрана разрядность в 16 бит, включая один бит под знак числа.

Оптимизация ресурсов и автоматическая генерация кода

В качестве аппаратной платформы для запуска нейронной сети была выбрана отладочная плата SoCKit Development Kit с ПЛИС Cyclone V SoC (5CSXFC6D6F31) от компании Altera. Для ПЛИС наиболее важными аппаратными ресурсами являются: количество логических ячеек, количество умножителей и объем встроенной памяти [3, 4] при условии отсутствия внешней памяти. Для нашей ПЛИС основные параметры представлены в таблице 1.

Таблица 1. Основные параметры ПЛИС Cyclone V SoC (5CSXFC6D6F31)

Параметр	Значение
Количество логических ячеек	110K LE (41,9K ALM)
Количество умножителей	112
Объем встроенной памяти	5,140 Кбит

При генерации Verilog-кода из Simulink-модели алгоритма нейронной сети, используя инструмент HDL-Coder с настройками по умолчанию, были задействованы следующие ресурсы ПЛИС, которые отображены на левой части рисунка 7.

Из рисунка видно, что по количеству задействованных умножителей (600) алгоритм не укладывается в выбранную ПЛИС (112 умножителей). В отчете при детальном рассмотрении используемых ресурсов каждым блоком выяснилось, что наибольшее количество умножителей используется во втором слое сети («Convolution_layer»), в свертке изображения – 500 умножителей. Для сокращения используемых умножителей в модуле была установлена оптимизация – совместное использование ресурсов – и выставлено значение 20, то есть один и тот же умножитель используется для выполнения 20 различных операций умножения в том же блоке и заменяет 19 дополнительных умножителей. Из минусов данного подхода следует отметить, что в таком случае умножитель должен работать на частоте в 20 раз выше частоты основного дизайна схемы, при этом возрастает число используемых мультиплексоров.

Multipliers	600
Adders/Subtractors	1038
Registers	29557
Total 1-Bit Registers	480163
RAMs	109
Multiplexers	1107
I/O Bits	232
Static Shift operators	0
Dynamic Shift operators	0

Multipliers	125
Adders/Subtractors	613
Registers	4306
Total 1-Bit Registers	77343
RAMs	133
Multiplexers	1210
I/O Bits	232
Static Shift operators	0
Dynamic Shift operators	0

Рисунок 7. Используемые ресурсы ПЛИС до (слева) и после (справа) оптимизации.

Таким образом, применяя совместное использование ресурсов внутри ПЛИС, удалось существенно сократить число используемых умножителей с 600 до 125. Отчет генератора кода представлен на правой части рисунка 7. Из отчета также видно, что существенно удалось сократить количество используемых регистров с 29 тыс. до 4 тыс., а число однобитовых регистров с 480 тыс. до 77 тыс., при этом количество мультиплексоров выросло незначительно – с 1107 до 1210 единиц. Но пока алгоритму сети требуется большее количество умножителей (125 шт.), чем доступно аппаратно на ПЛИС (112 шт.). Однако синтезаторы кода для ПЛИС способны создавать умножители из базовых логических ячеек, что будет продемонстрированно в следующем разделе.

Тестирование алгоритма на отладочной плате

Для тестирования алгоритма нейронной сети использовалась отладочная плата Altera SoC Kit Development Kit. Перед тестированием алгоритма на отладочной плате алгоритм нейронной сети был протестирован на численную эквивалентность в режиме «ПЛИС в контуре». При тестировании «ПЛИС в контуре» из алгоритма генерируется HDL-код, компилируется и прошивается на ПЛИС на отладочной плате. После этого алгоритм на ПЛИС запускается на выполнение совместно с моделью Simulink, входные данные генерируются в Simulink-модели, выходные сигналы поступают обратно в Simulink, где результаты численно сравниваются с результатами «эталонной» Simulink-модели алгоритма. Тестирование показало численную эквивалентность кода на ПЛИС с Simulink-

моделью алгоритма нейронной сети. На следующем шаге сгенерированный HDL-код (Verilog) алгоритма сети был включен в проект среды Quartus II (рисунок 8).

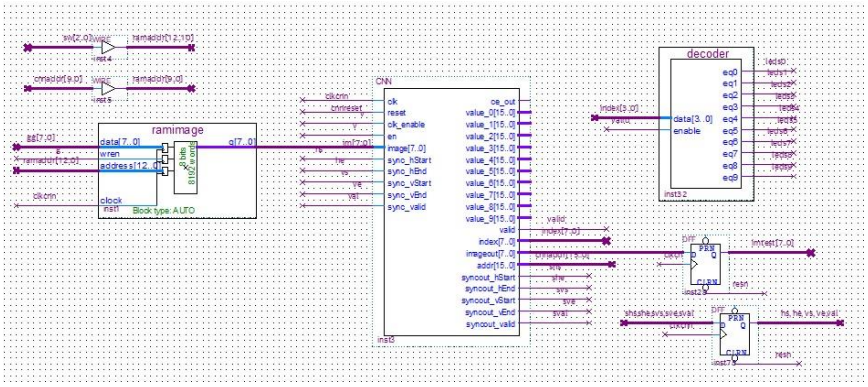


Рисунок 8. Подключение модуля нейронной сети в САПР Quartus II.

Для тестирования кода в Simulink-модели был добавлен дополнительный модуль генерации синхроимпульсов начала, конца кадра и адреса текущего пикселя изображения. Также для тестирования алгоритма в проект был добавлен модуль памяти 8192x8 бит для хранения 8 тестовых изображений цифр. Старшие адреса памяти задаются переключателями SW0-SW2, это позволяет вручную переключать изображения цифр для распознавания; память инициализируется mif-файлом, генерируемым из MATLAB. Тактирование модуля нейронной сети осуществляется частотой 50 МГц. Результат распознавания изображения цифры нейронной сетью выводится на 10 светодиодов. Отчет об используемых ресурсах на ПЛИС в среде Quartus II представлен на рисунке 9.

Flow Status	Successful - Tue Jan 29 13:49:20 2019
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	SocKit_golden_top
Top-level Entity Name	SoCKit_CNN_TOP
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	29,074 / 41,910 (69 %)
Total registers	17227
Total pins	57 / 499 (11 %)
Total virtual pins	0
Total block memory bits	804,483 / 5,662,720 (14 %)
Total DSP Blocks	112 / 112 (100 %)

Рисунок 9. Используемые ресурсы ПЛИС в САПР Quartus II.

После успешного тестирования сети на ПЛИС на тестовой выборке изображений тестирование происходило на реальных видео, получаемых с камеры Terasic D8M-GPIO, подключенной к отладочной плате (рисунок 10). Для этого проект в Quartus II был доработан соответствующим образом: вместо данных из памяти к блоку сети были подключены линии с камеры, данные с камеры инвертировались и из исходного кадра вырезалась только центральная область изображения размером 28×28 пикселей, обозначенная на рисунке 10 зеленой рамкой.

Тестирование на реальных данных с камеры показало незначительное падение точности, так как нейронная сеть обучалась на другом наборе изображений, чем те, которые приходили с камеры. Как следствие требуется набрать базу изображений непосредственно с этой камеры и дополнительно обучить сеть на новых данных. Видео с тестированием нейронной сети на ПЛИС доступно по ссылке [5].

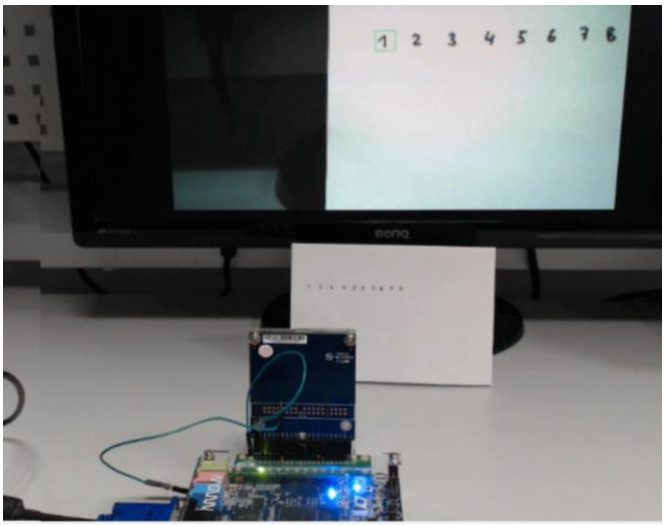


Рисунок 10. Тестирование распознавания цифр нейронной сетью на ПЛИС на данных с камеры.

Результаты

Применение модельно-ориентированного проектирования и среды MATLAB на первом этапе позволило более чем в два раза упростить архитектуру нейронной сети, быстро проверяя альтернативные варианты архитектуры сети и выбирая оптимальную по соотношению точности к вычислительной сложности для данной задачи. На втором этапе, имея «эталонную» модель сети в MATLAB, удалось верифицировать Simulink-модель сети с потоковой архитектурой и оптимально подобрать разрядность данных при переходе на фиксированную точку. На третьем этапе, используя возможность автоматической генерации кода из Simulink-моделей и гибкие настройки по оптимизации HDL-кода, получилось существенно сократить требуемые аппаратные ресурсы (количество умножителей – в 4,8 раза, а число регистров – более чем в 6 раз) и разместить алгоритм на выбранной ПЛИС.

Таким образом, применение модельно-ориентированного проектирования и среды разработки MATLAB/Simulink позволило наиболее оптимально и с минимальными временными затратами пройти процесс от создания архитектуры нейронной сети до ее аппаратной реализации на ПЛИС, выполняя сложные этапы разработки, на более высоком уровне абстракции – на уровне моделей MATLAB/Simulink.

Литература:

1. Qiu J. et al. Going deeper with embedded fpga platform for convolutional neural network // Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.
2. Solovyev R. A., Kustov A. G., Ruhlov, V. S. Schelokov A. N., Puzyrkov D. V. // Hardware implementation of a convolutional neural network in fpga based on fixed point calculations. Izvestiya SFedU. Engineering Sciences, July 2017, in Russian.
   FPGA ’16. New York, NY, USA: ACM, 2016, P. 26–35.
3. Zhang C., et al., Optimizing fpga-based accelerator design for deep convolutional neural networks // Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.
   FPGA ’15. New York, NY, USA: ACM, 2015, P. 161–170.
4. Suda N., et al. Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural network // Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate
   FPGA ’16. New York, NY, USA: ACM, 2016, P. 16–25.
5. Ссылка на видео-демонстрацию работы алгоритма на ПЛИС: https://youtu.be/A7B3cA5z96k

Контакты

exponenta.ru

E-mail: **info@exponenta.ru**

Тел.: +7 (495) 009 65 85

Адрес: **115088 г. Москва,**
2-й Южнопортовый проезд, д. 31, стр. 4

mathworks.com