ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки

каф. ЕОМ

Гетерогенна модель для глибокої нейронної мережі

Виконав ст.гр СПзм-20-1
Доценко О.А.

Керівник
ст.викладач Знайдюк В.Г.

Актуальність

- Штучний інтелект – одна з найперспективніших технологій, що базується на алгоритмах машинного навчання. У цій роботі ми пропонуємо робочий процес для реалізації глибоких нейронних мереж.
- Цей робочий процес намагається поєднати гнучкість мереж на основі компіляторів високого рівня (HLS – high-level compilers) з архітектурними можливостями управління потоками на основі мов опису апаратного забезпечення (HDL – hardware description languages).
- Архітектура складається зі згорткової нейронної мережі SqueezeNet v1.1 та твердотільної процесорної системи (HPS – hard processor system), яка співіснує з апаратними засобами прискорення, що розробляються.
- Ця методологія дозволяє порівнювати рішення, засновані виключно на програмному забезпеченні (PyTorch 1.13.1), і пропонувати гетерогенні рішення для виведення, використовуючи найкращі варіанти в рамках програмно-апаратного потоку.

Мета та задачі

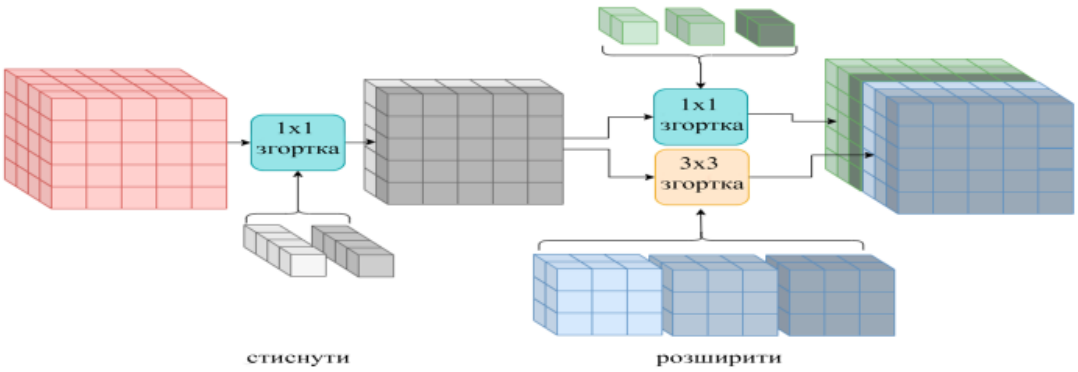
- **Метою кваліфікаційної роботи є** розробка гетерогенної моделі глибокої нейронної мережі

Задачі:

- Розглянути структуру SqueezeNet
- Проаналізувати різні алгоритми роботи згортки
- Запропонувати та побудувати модель функціонування нейронної мережі

3

SqueezeNet

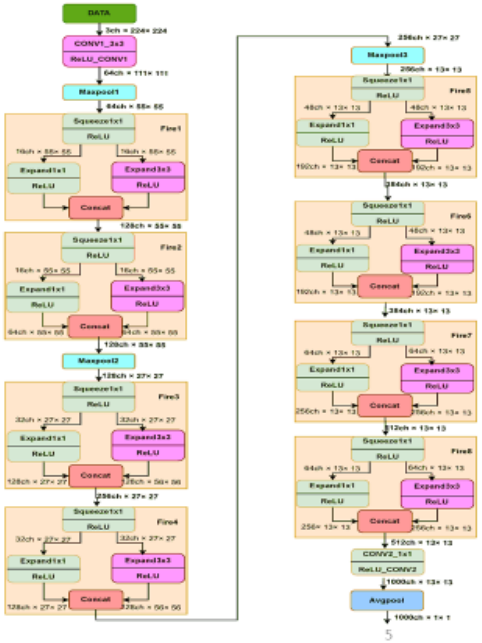


Структура Fire модуля складається з різних фільтрів згортки

4

Структура SqueezeNet

Назва/тип шару	Вихідний розмір	Розмір/крок фільтра (без пожежних модулів)	$S_{1\times1}$	$E_{1\times1}$	$E_{3\times3}$
Input image	224 × 224 × 3				
conv1	111 × 111 × 64	3 × 3/2 (× 64)			
maxpool1	55 × 55 × 64	3 × 3/2			
fire1	55 × 55 × 128		16	64	64
fire2	55 × 55 × 128		16	64	64
maxpool2	27 × 27 × 128	3 × 3/2			
fire3	27 × 27 × 256		32	128	128
fire4	27 × 27 × 256		32	128	128
maxpool3	13 × 13 × 256	3 × 3/2			
fire5	13 × 13 × 384		48	192	192
fire6	13 × 13 × 384		48	192	192
fire7	13 × 13 × 512		64	256	256
fire8	13 × 13 × 512		64	256	256
conv2	13 × 13 × 1000	1 × 1/1 (×1000)			
avgpool1	1 × 1 × 1000	13 × 13/1			



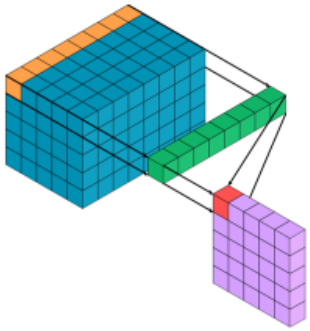
5

Згортка 1×1

Input: *in _ channels* (parameter)
Input: *in _ size* (parameter)
Input: *filte _ size* (parameter)
Input: *in _ img* (buffer read only)
Input: *filter _ weight* (buffer read only)
Input: *filter _ bias* (buffer read only)
Output: *out _ img* (buffer)

1: **for** *filter _ index* ← 0 To *filter _ size* **do**
2: *bias* ← *filter _ bias*[*filter _ index*]
3: **for** *j* ← 0 To (*in _ size* × *in _ size*) **do**
4: *tmp* ← *bias*
5: **for** *k* ← 0 To (*in _ channels*) **do**
6: *tmp* ← *in _ img*[*k* × *in _ size* × *in _ size* + *j*] × *filter _ weight*[*k* + *filter _ index* × *in _ channels*] + *tmp*;
7: **end for**
8: **if** *tmp* > 0 **then**
9: *out _ img*[*ij* + *in _ size* × *in _ size* × *filter _ index*] ← *tmp*
10: **else**
11: *out _ img*[*ij* + *in _ size* × *in _ size* × *filter _ index*] ← 0
12: **end if**
13: **end for**
14: **end for**

Згортка 1×1



Деталь згортки 1×1 з CH_{in}=8, CH_{out}=1 та in_{size}=5.

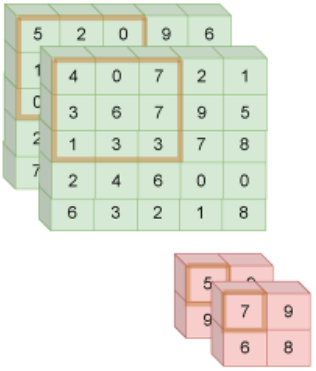
6

Максимальний пул

Input: *in_size* (parameter)
Input: *out_size* (parameter)
Input: *channel_size* (parameter)
Input: *in_img* (buffer read only)
Output: *out_img* (buffer)

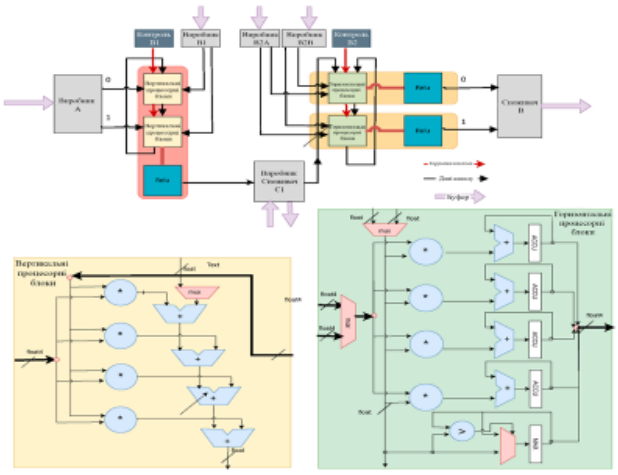
```
for channel_index ← 0 To channel_size do
    for i ← 0 To (out_size) do
        for j ← 0 To (out_size) do
            tmp ← 0
            for l ← 0 To 3 do
                for m ← 0 To 3 do
                    value ← in_img[(i × 2 + l) × in_size + j × 2 + m]
                    if value > tmp then
                        tmp ← value
                    end if
                end for
            end for
            out_img[i × out_size + j] ← tmp
        end for
    end for
    in_img ← in_img × in_img + in_img
    out_img ← out_size × out_size + out_img
end for
```

Max-pool



Деталь шару max-pool з CH_{size}=2 , in_{size}=5 та out_{size}=2 pad = 0 та stride = 2

Архітектура реалізації

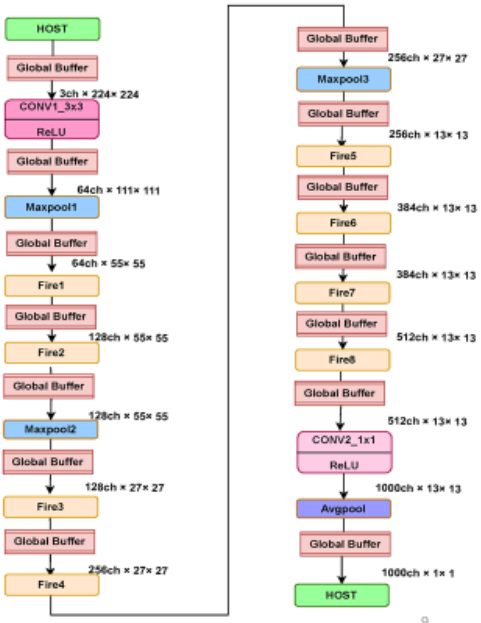


Архітектура реалізації показана на рисунку. Вертикальні процесорні блоки в основному відповідають за стадію стиснення, а горизонтальні процесорні блоки – за стадію розширення.

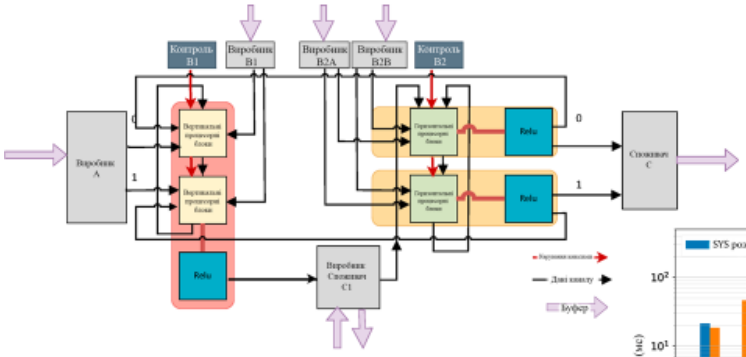
Однак останній має вищу складність і може реалізовувати шари з середнім та максимальним пулом.

Організація повторного використання архітектури розгортання для SqueezeNetv1.1.

- Ця структура повторно використовується 13 разів для повної реалізації мережі (рисунок).
- Буфери (червоні блоки), які взаємодіють з блоками виробника та споживача, показані на попередньому слайді, є фундаментальними.
- Ефективність зв'язку між ними та ядрами пристроїв і глобальною оперативною пам'яттю DE10-nano є критично важливою.

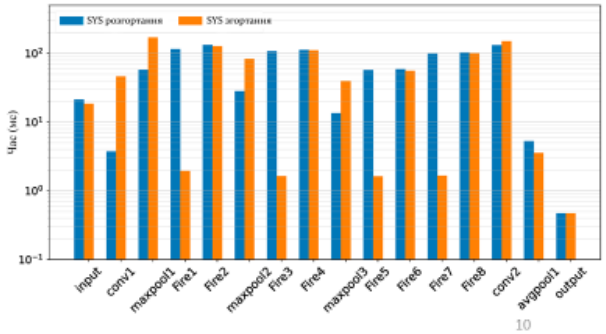


Архітектура систолічного розгортання

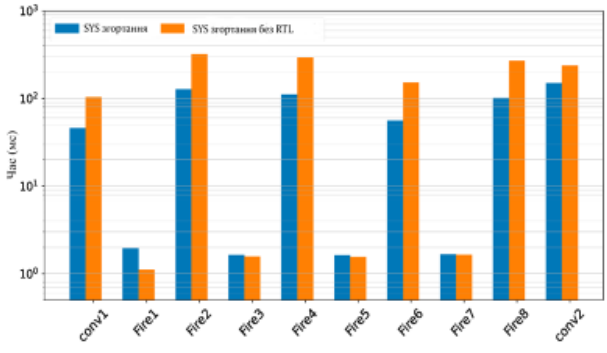


Архітектура

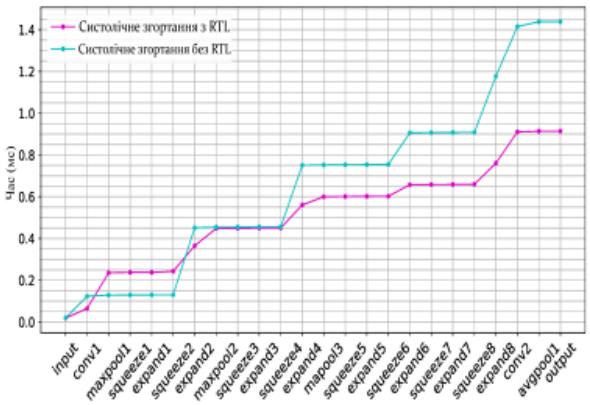
Порівняння рівня Fire між версіями згортання та розгортання



Оптимізація: використання бібліотек RTL



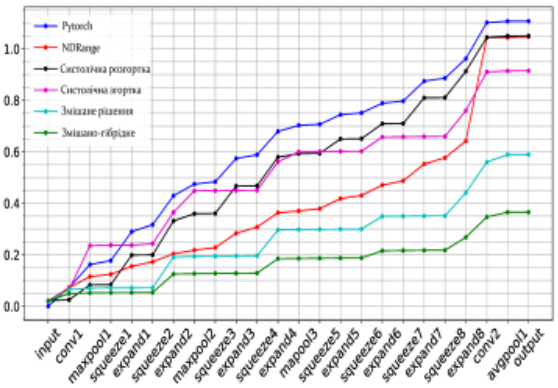
Порівняння рівня Fire між версією згортання без RTL та версією згортання з RTL



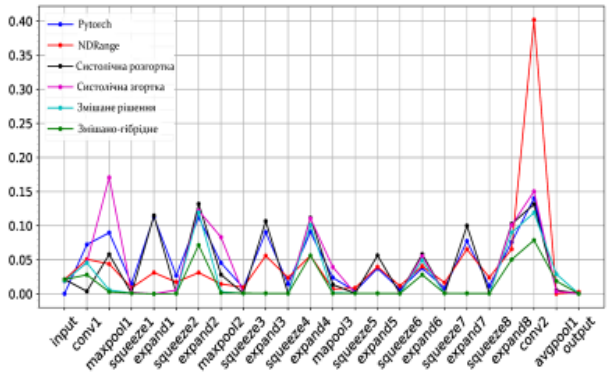
Порівняння затримок при використанні RTL бібліотек та без використання RTL бібліотек

11

Порівняння повного мережевого виводу



Порівняння повного мережевого виводу з накопиченим часом



Порівняння повного мережевого виводу з фазовим часом

12

Висновки

- У цій роботі представлено робочий процес для реалізації глибоких нейронних мереж, який поєднує гнучкість мереж на основі HLS з архітектурними можливостями управління потоками на основі HDL. OpenCL був основним інструментом, який використовувався в цьому робочому процесі, оскільки він забезпечує структурний підхід і високий рівень апаратного контролю, що особливо важливо при роботі з систолічними архітектурами.

Вирішені наступні задачі:

- Розглянуто структуру SqueezeNet
- Проаналізовано різні алгоритми роботи згортки
- Запропоновано та побудовано модель функціонування нейронної мережі