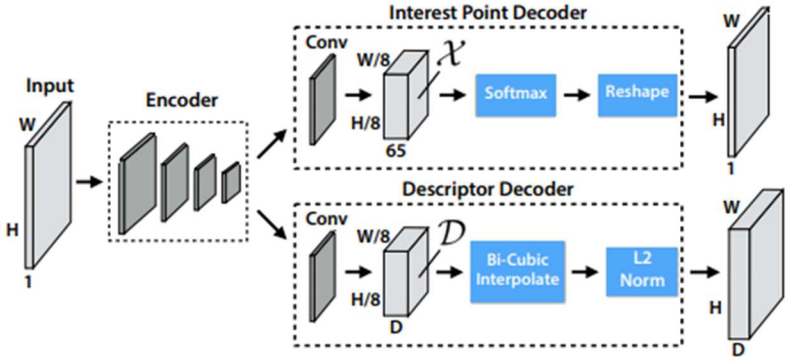
Opis działania sieci i przepływ danych SuperPoint

SuperPoint: Self-Supervised Interest Point Detection and Description



Rys1. Schemat działania sieci

1. Parametry sieci

Wejscie: szary obraz o wymiarach W, H, typ float32

Enkoder					
Warstwa	**Parametry**	**Wejście (shape)**	**Wyjście (shape)**	**Typ danych**	**Liczba Mnożenia/dodawania (takie same wartości)**
Conv2D + ReLU +BatchNorm2d	Kernel: 3×3, stride=1, pad=1, Eps=0.001	(1, W, H)	(64, W, H)	Float32	576*W*H
Conv2D + ReLU +BatchNorm2d	Kernel: 3×3, stride=1, pad=1, Eps=0.001	(64, W, H)	(64, W, H)	Float32	**36 864*W*H**
MaxPool2D	kernel=2, stride=2, padding=0, dilation=1	(64, W, H)	(64, W/2, H/2)	Float32	
Conv2D + ReLU +BatchNorm2d	Kernel: 3×3, stride=1, pad=1, Eps=0.001	(64, W/2, H/2)	(64, W/2, H/2)	Float32	9 216*W*H
Conv2D + ReLU +BatchNorm2d	Kernel: 3×3, stride=1, pad=1,	(64, W/2, H/2)	(64, W/2, H/2)	Float32	9 216*W*H

	Eps=0.001				
MaxPool2D	kernel=2, stride=2, padding=0, dilation=1	(64, W/2, H/2)	(64, W/4, H/4)	Float32	
Conv2D + ReLU +BatchNorm2d	Kernel: 3×3, stride=1, pad=1, Eps=0.001	(64, W/4, H/4)	(128, W/4, H/4)	Float32	4 608*W*H
Conv2D + ReLU +BatchNorm2d	Kernel: 3×3, stride=1, pad=1, Eps=0.001	(128, W/4, H/4)	(128, W/4, H/4)	Float32	9 216*W*H
MaxPool2D	kernel=2, stride=2, padding=0, dilation=1	(128, W/4, H/4)	(128, W/8, H/8)	Float32	

Detector					
Warstwa	**Parametry**	**Wejście (shape)**	**Wyjście (shape)**	**Typ danych**	**Liczba Mnożenia/dodawania (takie same wartości)**
Conv2D + ReLU +BatchNorm2d	Kernel: 3×3, stride=1, pad=1, Eps=0.001	(128, W/8, H/8)	(256, W/8, H/8)	Float32	4 608 *W*H
Conv2D +BatchNorm2d	Kernel: 1×1, stride=1, Eps=0.001	(256, W/8, H/8)	(65, W/8, H/8)	Float32	260*W*H

Descriptor					
Warstwa	**Parametry**	**Wejście (shape)**	**Wyjście (shape)**	**Typ danych**	**Liczba Mnożenia/dodawania (takie same wartości)**
Conv2D + ReLU +BatchNorm2d	Kernel: 3×3, stride=1, pad=1, Eps=0.001	(128, W/8, H/8)	(256, W/8, H/8)	Float32	4 608 *W*H
Conv2D +BatchNorm2d	Kernel: 1×1, stride=1, Eps=0.001	(256, W/8, H/8)	(256, W/8, H/8)	Float32	1 024*W*H

2. Opis działania elementów sieci (notatki):

2.1 Konwolucja:

Dla każdego filtra i każdego punktu wyjścia (x, y na obrazie wyjściowym):

1. **Wycięcie lokalnego okna wejściowego**:
   Wycinane jest fragment wejściowego tensora o wymiarach [3,3] dla każdego kanału wejściowego do warstwy
2. **Mnożenie elementów okna przez elementy filtra**:
   Każdy element wyciętego okna (3, 3, liczba kanałów na wejściu) jest przemnożony przez odpowiedni element filtra.

   9 * , liczba kanałów na wejściu – liczba mnożeń, dla jednego elementu wyjścia
3. **Dodanie wyników**:
   Wyniki mnożeń są sumowane. To odpowiada iloczynowi skalarnemu między filtrem a danymi wejściowymi.
4. **Dodanie biasu** :
   Stała wartość b (jeden bias na każdy filtr) może zostać dodana do wyniku.

   9 * , liczba kanałów na wejściu – liczba dodawań, dla jednego elementu wyjścia
5. Przesunięcie okna według kroku (stride) i powtórzenie procesu dla wszystkich możliwych pozycji w obrazie wejściowym, a następnie dla wszystkich filtrów.

2.2 RELU:

$$f(x) = \max(0, x)$$

Operacja dla jednego elementu:

- Porównanie liczby x z 0 oraz wybór większej wartości.
- Na poziomie hardware'u można to zrealizować jako:
  - 1 operacja porównania,
  - 1 operacja przypisania.

2.3 BatchNorm2D:

Dla każdego piksela x:

$$y = weight\left(\frac{x - mean}{\sqrt{var + eps}}\right) + bias$$

Wszystko jest już nauczone.

(spierwiastkować var + eps na sztywno i zapisać w pamięci, żeby ominąć to w obliczeniach?)

2.4 MaxPolling:

Wybranie największej liczby z okna o wymiarach 2 na 2. Zmniejszenie wymiarów.

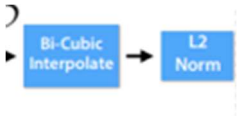
Liczba porównań: 3 * liczba kanałów * wymiary obrazu wyjściowego.

3. Opis przekształceń punktów charakterystycznych po wyjściu z sieci

Wyjście z sieci to tensor o wymiarach W/8, H/8, 65. W pomniejszonym obrazie jeden piksel odpowiada za 64 piksele oryginalnego. 65 kanałów odpowiada właśnie za te 64 piksele + kanał na „no interest point". Wyjściem ostatecznym powinna być reprezentacja prawdopodobieństwa, że punkt jest punktem charakterystycznym o wymiarach W, H.

1. Softmax
2. Usunięcie ostatniego kanału
3. Reshape do W,H (wykorzystując 64 kanały)
4. NMS (radius=4)

4. Opis przekształceń deskryptora po wyjściu z sieci



1. W znalezionej implementacji dodano jeszcze L2 na początku.
   Norma L2:

$$\|\mathbf{v}\|_2 = \sqrt{\sum_i v_i^2},$$

   Normalizacja deskryptora dla każdego piksela osobno.
2. interpolacja

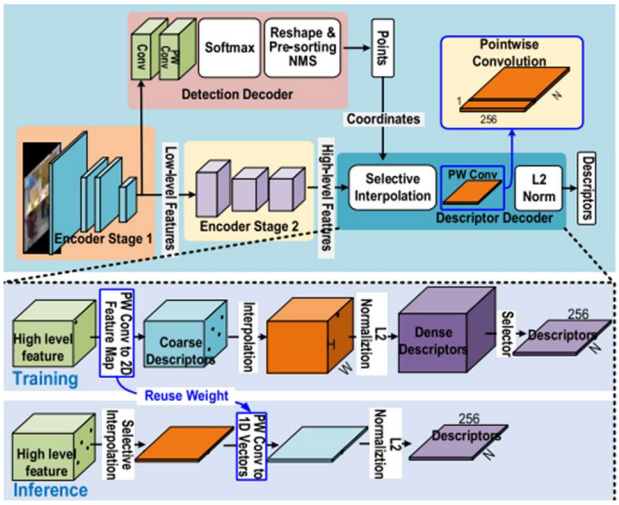
MobileSP, przeniesienie SuperPoint na FPGA, zmiany w architekturze dla przyspieszenia

MobileSP: An FPGA-Based Real-Time Keypoint Extraction Hardware Accelerator for Mobile VSLAM

Problem: zmiana architektury sieci – wymaga trenowania na nowo, na razie nie znalazłem przetrenowanych wag.

A: Poniżej 1% punktów jest punktami charakterystycznymi. SuperPoint liczy deskryptory (interpolacja, normalizacja) dla każdego punktu. Dużo niepotrzebnych obliczeń. Można by to policzyć tylko do wykrytych punktów.

W tym artykule robią coś takiego:



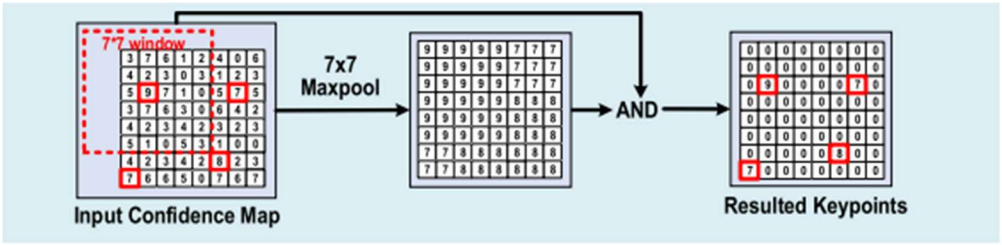
PARAMETERS OF THE MOBILESP

Layer	Kernel Size	Output Size*
Conv_1	64*[1, 3, 3]	[64, H, W]
Conv_2	64*[64, 3, 3]	[64, H/2, W/2]
Conv_3	64*[64, 3, 3]	[64, H/2, W/2]
Conv_4	128*[64, 3, 3]	[128, H/4, W/4]
Conv_Keypoint_1	256*[128, 3, 3]	[256, H/8, W/8]
Conv_Keypoint_2	65*[256, 1, 1]	[65, H/8, W/8]
Conv_Descriptor_1	128*[128, 3, 3]	[128, H/4, W/4]
Conv_Descriptor_2	128*[128, 3, 3]	[128, H/8, W/8]
Conv_Descriptor_3	256*[128, 3, 3]	[256, H/8, W/8]

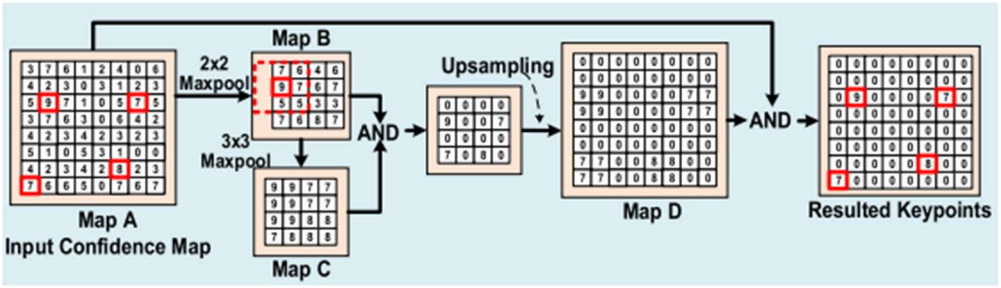
*The size of input image is H*W.

Rozdzielają enkoder na 2 części. Pierwsza wylicza low-level features i to już wystarczy do detekcji punktów. Następnie równolegle dzieje się przetwarzanie tych punktów i druga część enkodera do deskryptorów. Dzięki temu można po wyjściu z drugiej części enkodera od razu przejść do przetwarzania deskryptorów i to tylko tych, których potrzeba (bo znamy już punkty).

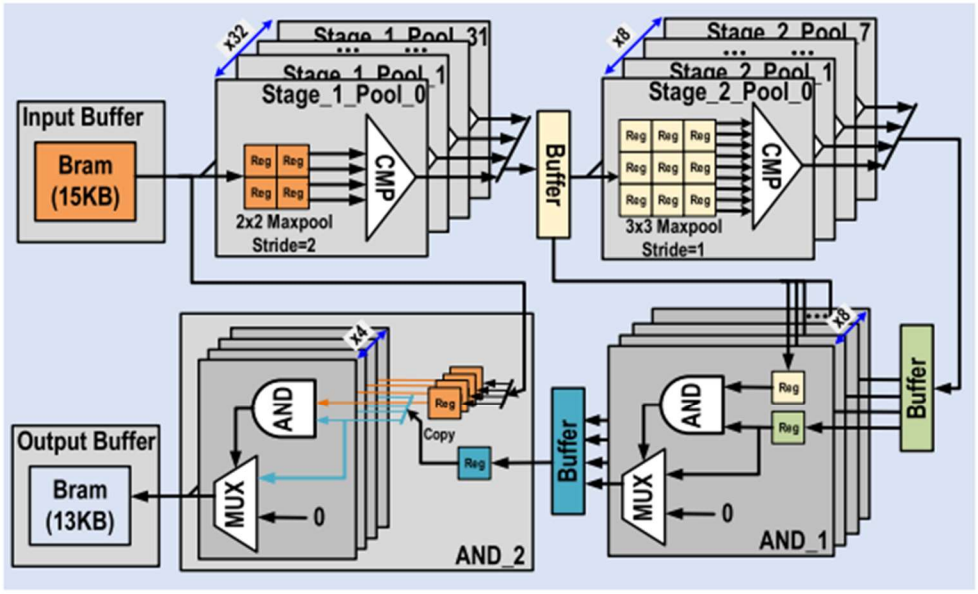
B: Przyspieszenie NMS



(a)



(b)



(c)

Fig. 4. (a) Conventional NMS (b) Proposed pre-sorting based NMS (c) Hardware architecture of the NMS engine.

Oryginalnie 49HW porównań, a w tej wersji 4HW.

Inne podejście do akceleracji

CNN-based Feature-point Extraction for Real-time Visual SLAM on Embedded FPGA

1.Optymalizacja softmax

2.Optymalizajca NMS

3.Ranking – znajdowanie k najlepszych punktów

4.Optymalizacj L2-norm

Brak zmiany architektury. Można zastosować przetrenowane wagi. Zmienić na 8-bit fixed point.

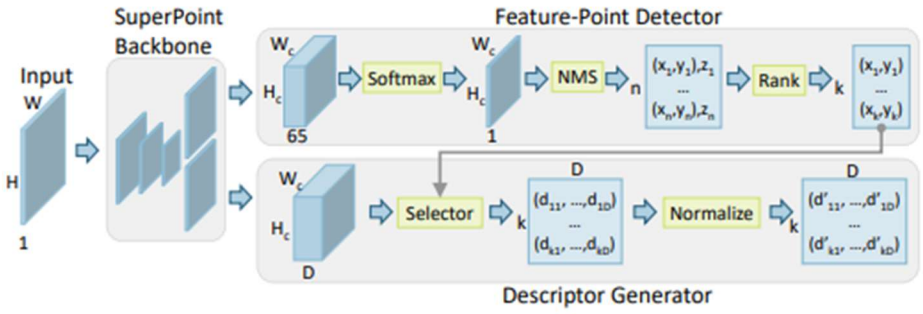


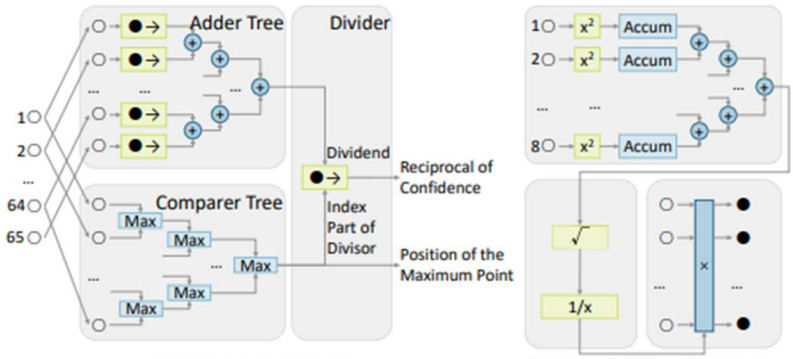
TABLE I
HARDWARE CONSUMPTION OF THE PROPOSED HARDWARE

	#DSP	#LUT	#FF	#BRAM
On-Board Resource	2520	418080	548160	912
DPU	1282	74496	171294	499
Softmax	0	4714	3205	0
Normalization	25	1389	935	0.5

TABLE II
RUNNING TIME COMPARISON OF EACH OPERATION

	CNN backbone*	Post-processing				
		Softmax	NMS	Rank	Norm	Total
CPU	24ms	31ms	27ms	0.97ms	42ms	100.97ms
Ours		1.97ms	0.7ms	0.12ms	1.44ms	4.23ms

* The CNN backbone runs on the accelerator.



(a) Softmax Accelerator

(b) Normalization Accelerator