

High Level Synthesis of a trained CNN for handwritten digit recognition

Federico Serafini

`federico.serafini@studenti.unipr.it`

Embedded Systems

Università degli Studi di Parma

12/07/2022

Outline

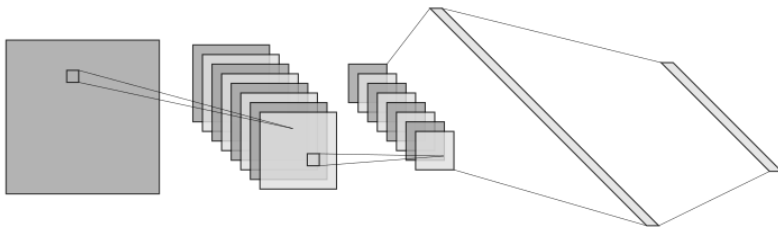
- 1 Introduction
- 2 SW Implementation
- 3 High Level Synthesis
- 4 Results and Verification
- 5 Conclusions

Basic concepts

Convolutional Neural Network (CNN)

Neural network with a special architecture that is able to detect spatial structures (features) of the input:

- widely used in image-recognition problems;
- highly-parallelizable algorithm.



Basic concepts

A single and ambitious objective

Overtake C performances through HW parallelism!

Workflow

- ① Python:
 - ① model definition, training and evaluation;
 - ② export of (trained) network weights and architecture.
- ② C: replication of the network.
- ③ Vitis HLS:
 - ① naive approach (high level synthesis of basic C);
 - ② stream and dataflow approach.
- ④ Verification.

Model definition and evaluation in Python

padding_layer_input	input:	[(None, 28, 28, 1)]
InputLayer		
float32	output:	[(None, 28, 28, 1)]

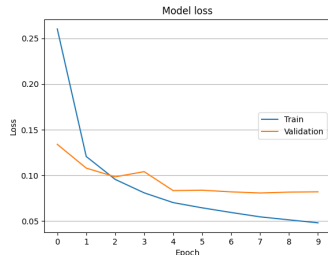
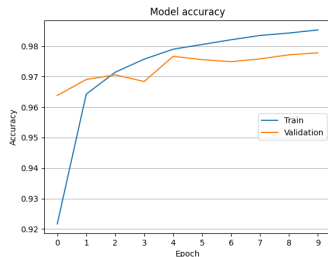
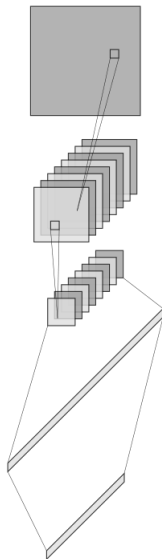
padding_layer	input:	(None, 28, 28, 1)
ZeroPadding2D		
float32	output:	(None, 30, 30, 1)

convolution_layer	input:	(None, 30, 30, 1)
Conv2D	relu	
float32	output:	(None, 28, 28, 8)

max_pooling_layer	input:	(None, 28, 28, 8)
MaxPooling2D		
float32	output:	(None, 14, 14, 8)

flatten_layer	input:	(None, 14, 14, 8)
Flatten		
float32	output:	(None, 1568)

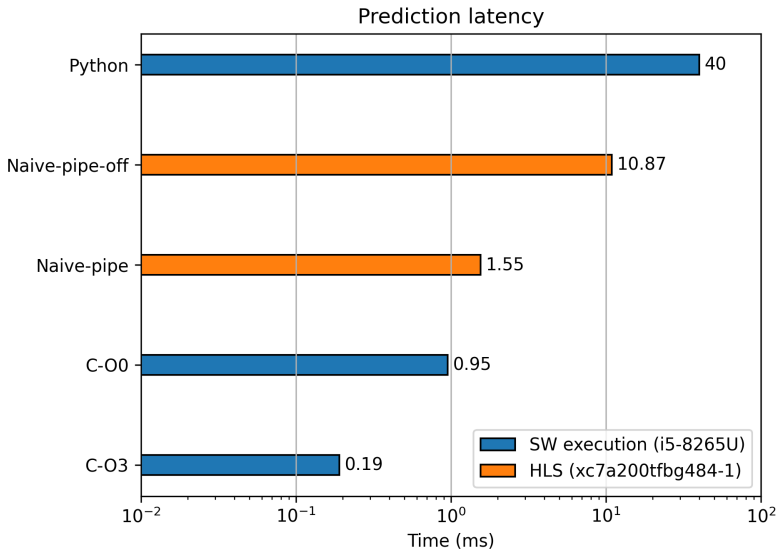
dense_layer	input:	(None, 1568)
Dense	softmax	
float32	output:	(None, 10)



Network replication in C

```
1 void cnn
2 (
3     float img_in      [IMG_ROWS][IMG_COLS],
4     float prediction [DIGITS]
5 )
6 {
7     /***** Normalization and padding. *****/
8     float pad_img [PAD_IMG_ROWS][PAD_IMG_COLS] = { 0 };
9     normalization_and_padding(img_in, pad_img);
10
11     /***** Convolution layer. *****/
12     float features [FILTERS][IMG_ROWS][IMG_COLS] = { 0 };
13     // Convolution with relu as activation function.
14     convolutional_layer(pad_img, features);
15
16     /***** Max-pooling layer. *****/
17     float pool_features [FILTERS][POOL_IMG_ROWS][POOL_IMG_COLS] = { 0 };
18     max_pooling_layer(features, pool_features);
19
20     /***** Flatten layer. *****/
21     float flat_array [FLAT_SIZE] = { 0 };
22     flattening_layer(pool_features, flat_array);
23
24     /***** Dense layer. *****/
25     dense_layer(flat_array, prediction);
26 }
```

Naive approach: latency estimation



Vitis HLS Stream ¹

Streaming data transfer

- A FIFO queue (*ap_fifo* interface).
- Data samples are sent in sequential order starting from the first (no address management is required).

Array implemented as FIFO interface

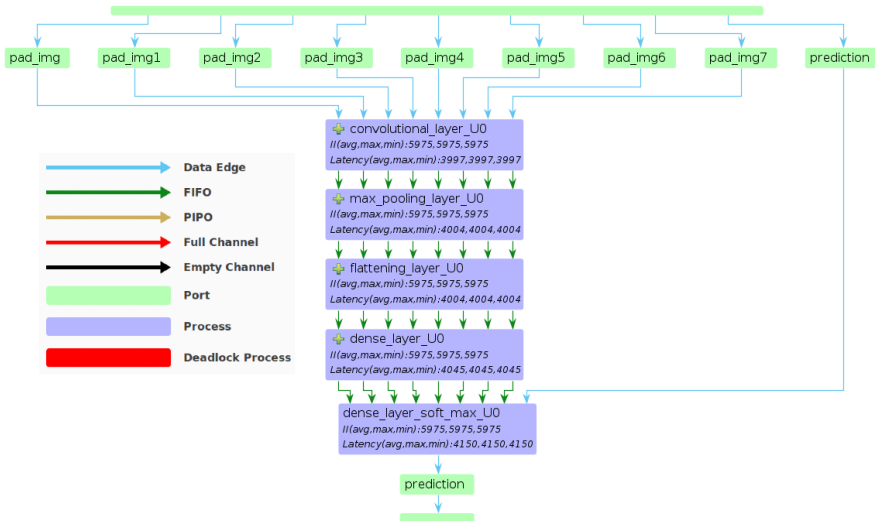
- Array must be only read or written, thus allowing a point-to-point connection.
- Program must follow a FIFO semantics (no random accesses).
- If a stream is used to transfer data between tasks, consider a dataflow region where data streams from one task to the next.

¹Vitis-HLS User Guide

Stream-dataflow approach: code structure

```
1  #define FILTERS 8
2
3  void cnn
4  (
5      float img_in      [IMG_ROWS][IMG_COLS],
6      float prediction [DIGITS]
7  )
8  {
9      /***** Pre-processing the img_in. *****/
10
11      // Normalization and padding.
12
13      /***** Clone the normalized and padded image. *****/
14
15      /*
16       * Clone the normalized and padded image in order to
17       * have an image for each parallel execution (for each filter).
18       */
19
20      /***** Parallel executions start here. *****/
21
22      /* Dataflow section with streams used to transfer data between tasks:
23       * -convolution_layer;
24       * -max_pooling_layer;
25       * -flattening_layer;
26       * -dense_layer;
27       * -dense_layer_softmax.
28       */
29  }
```

Dataflow view



Stream-dataflow approach: latency estimation



Verification

Co-simulation

Total predictions	10
Correct predictions	100%

	Min	Avg	Max
Latency (cycles)	5975	5975	5975

Export RTL with Vivado synthesis and place and route

	BRAM	DSP	FF	LUT
Vitis HLS	384	143	47201	37585
Vivado	224	143	38791	26753

	Required	Post-synth	Post-impl
Clock period (ns)	10	8.123	9.157

Conclusions and future work

Future work

- Smarter SW algorithm → smarter HW accelerator.
- Fixed-point arithmetic → reduced area.
- Vitis HLS syntax constructs and libraries → better synthesis.

Hardware or software implementation?

Further investigation is needed:

- ① apply improvements listed above;
- ② consider application domain requirements (for example the available HW and timing constraints);
- ③ consider also a co-design (HW and SW);
- ④ choose the cheapest solution satisfying the requirements.

Thank you for your attention.