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

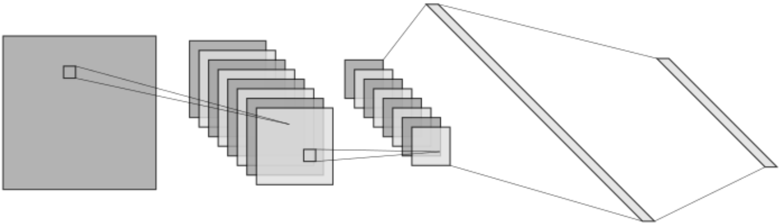
Basic concepts

Convolutional Neural Network (CNN)

Neural networks able to detect spatial structures (features) of the input through a special architecture based on:

- local receptive fields (convolution operation);
- shared weights (filters);
- local translation invariance (pooling operation).

$\Longrightarrow$ Widely used in image-recongnition problems.
$\Longrightarrow$ Highly-parallelizable problem.



Basic concepts

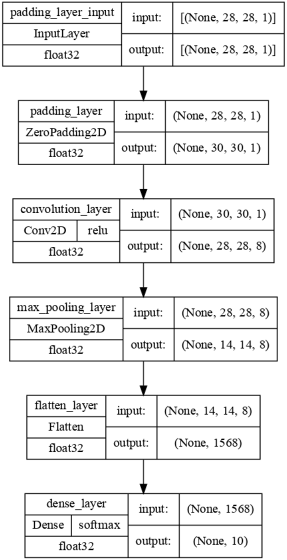
A single and abitious objective

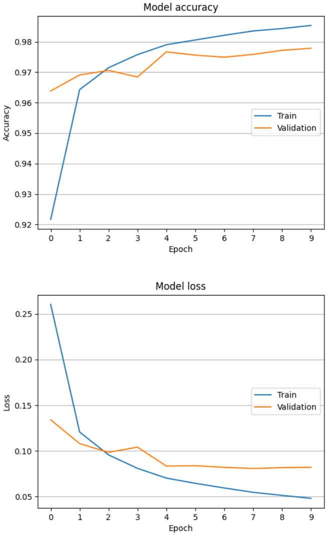
Overtake C performances through HW parallelism!

Workflow

1. Python:
    1. model definition, training and evaluation;
    2. export of network weights and architecture.
2. C: replication of the network architecture.
3. Vitis HLS:
    1. naive implementation (basic C synthesis);
    2. stream and dataflow implementation.
4. Validation.

Model definition and evaluation in Python

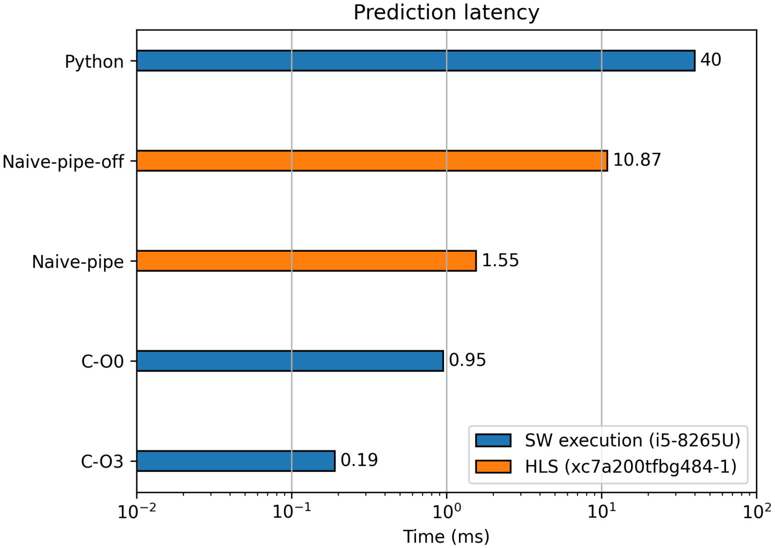




Network replication in C

```c
void cnn
(
  float img_in      [IMG_ROWS][IMG_COLS],
  float prediction [DIGITS]
)
{
  /******** Normalization and padding. ********/
  float pad_img [PAD_IMG_ROWS][PAD_IMG_COLS] = { 0 };
  normalization_and_padding(img_in, pad_img);

  /******** Convolution layer. ********/
  float features [FILTERS][IMG_ROWS][IMG_COLS] = { 0 };
  // Convolution with relu as activation function.
  convolutional_layer(pad_img, features);

  /******** Max-pooling layer. ********/
  float pool_features [FILTERS][POOL_IMG_ROWS][POOL_IMG_COLS] = { 0 };
  max_pooling_layer(features, pool_features);

  /******** Flatten layer. ********/
  float flat_array [FLAT_SIZE] = { 0 };
  flattening_layer(pool_features, flat_array);

  /******** Dense layer. ********/
  dense_layer(flat_array, prediction);
}
```

Naive implementation: latency estimation



Stream [1]

Streaming data

A type of data transfer in which data samples are sent in sequential order starting from the first sample:

- a FIFO of infinite depth (*ap_fifo*);
- no address management is required.

Array implemented as FIFO interface

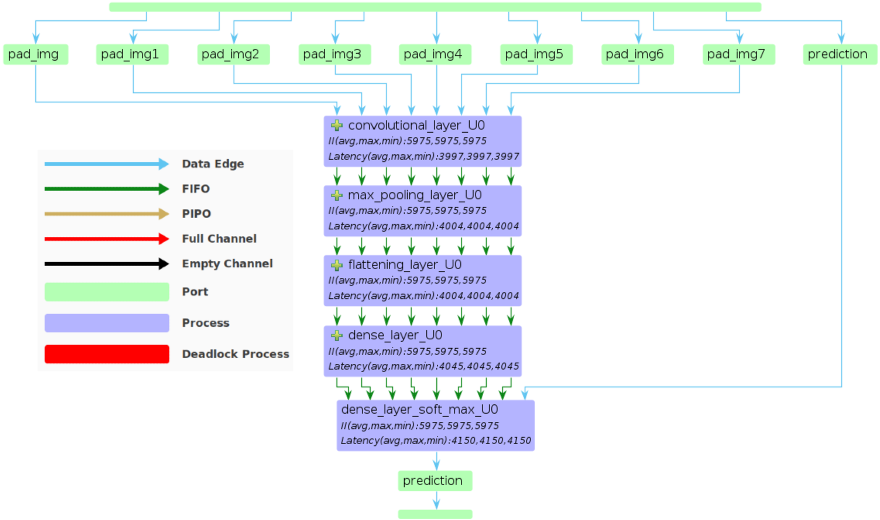
- Array must be only read or written, thus allowing a point-to-point connection.
- Program must follow first in, first out semantics (random access is not supported).
- If a stream is used to transfer data between tasks, consider a dataflow region where data streams from one task to the next.

[1] Vitis-HLS User Guide

Stream-dataflow implementation: code structure

```
#define FILTERS 8

void cnn
(
  float img_in     [IMG_ROWS][IMG_COLS],
  float prediction [DIGITS]
)
{
  /******** Pre-processing the img_in. ********/

  // Normalization and padding.


  /******** Clone the normalized and padded image for FILTERS times. ********/

  /*
   *  Clone the normalized and padded image in order to
   *  have an image for each parallel execution.
   */


  /******** Parallel executions start here. *********/

  /*
   *  Dataflow section with streams between tasks:
   *  -convolution_layer;
   *  -max_pooling_layer;
   *  -flattening_layer;
   *  -dense_layer.
   */
}
```

Dataflow view



High level synthesis details report

'cnn' report

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
5821	5821	58.210 us	58.210 us	5822	5822	none

Detail

Instance

		Latency (cycles)		Latency (absolute)		Interval (cycles)		
Instance	Module	min	max	min	max	min	max	Type
grp_dataflow_section_fu_466	dataflow_section	3997	3997	39.970 us	39.970 us	3998	3998	dataflow

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- pad_for_rows_pad_for_cols	918	918	20	1	1	900	yes
- clone_for_rows_clone_for_cols	901	901	3	1	1	900	yes

'dataflow_section' report

Latency

Summary

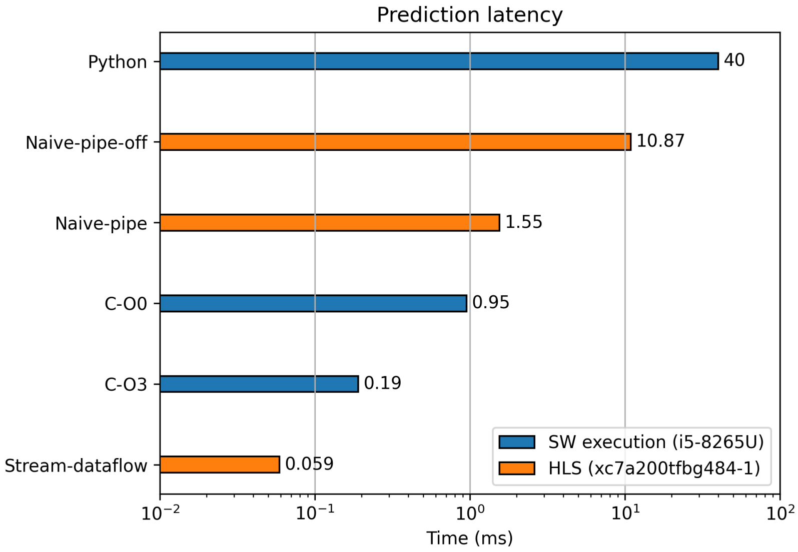
Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
3997	3997	39.970 us	39.970 us	3998	3998	dataflow

Detail

Instance

		Latency (cycles)		Latency (absolute)		Interval (cycles)		
Instance	Module	min	max	min	max	min	max	Type
convolutional_layer_U0	convolutional_layer	3997	3997	39.970 us	39.970 us	3998	3998	dataflow
dense_layer_U0	dense_layer	1998	1998	19.980 us	19.980 us	1999	1999	dataflow
dense_layer_soft_max_U0	dense_layer_soft_max	113	113	1.130 us	1.130 us	113	113	none
max_pooling_layer_U0	max_pooling_layer	793	793	7.930 us	7.930 us	794	794	dataflow
flattening_layer_U0	flattening_layer	198	198	1.980 us	1.980 us	199	199	dataflow

Stream-dataflow implementation: latency estimation



Validation

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 $\Longrightarrow$ faster HW accelerator.
- Fixed-point arithmetic $\Longrightarrow$ reduced area.
- Vitis HLS syntax constructs and libraries $\Longrightarrow$ smarter syntesis.

Hardware or software implementation?

Further invistigation is needed:

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

Thank you for your attention.