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

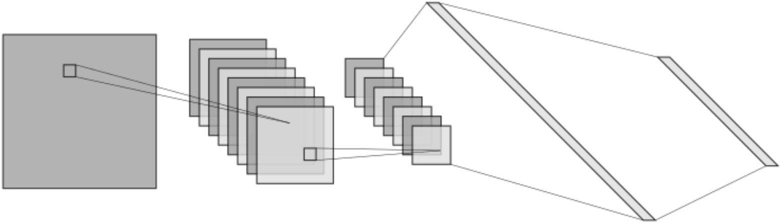
1. Introduction
2. SW Implementation
3. High Level Synthesis
4. Results and Verification
5. Conclusions

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-recongnition problems;
- highly-parallelizable algorithm.



Basic concepts

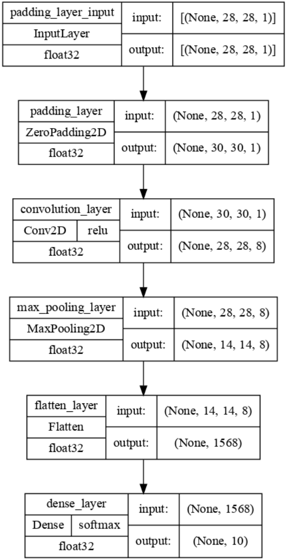
A single and abitious objective

Overtake C performances through HW parallelism!

Workflow

1. Python:
   1. model definition, training and evaluation;
   2. export of (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 | InputLayer | float32 | input: [(None, 28, 28, 1)] | output: [(None, 28, 28, 1)]

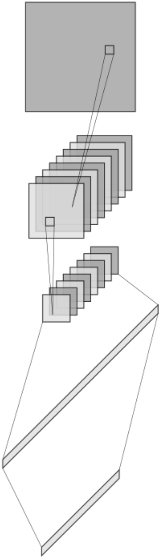
padding_layer | ZeroPadding2D | float32 | input: (None, 28, 28, 1) | output: (None, 30, 30, 1)

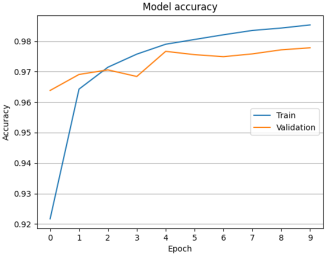
convolution_layer | Conv2D | relu | float32 | input: (None, 30, 30, 1) | output: (None, 28, 28, 8)

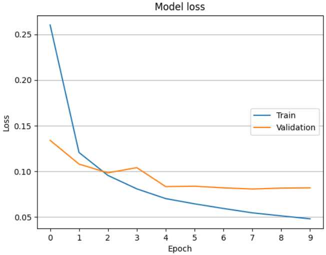
max_pooling_layer | MaxPooling2D | float32 | input: (None, 28, 28, 8) | output: (None, 14, 14, 8)

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

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



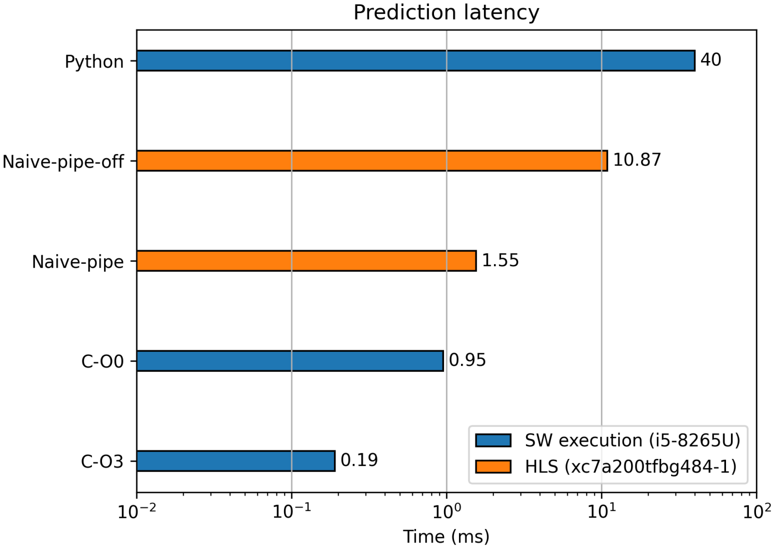




Network replication in C

```
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 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

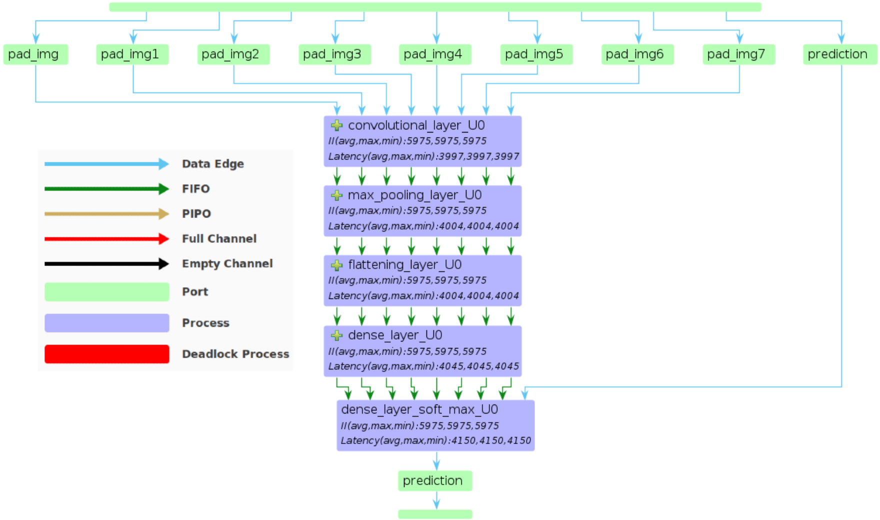
- Array must be only read or written, thus allowing a point-to-point connection.
- Program must follow a FIFO semantics (no random accesses).
- If a stream is used to transfer data between tasks, consider a dataflow region where data streams from one task to the next.

[1] Vitis-HLS User Guide

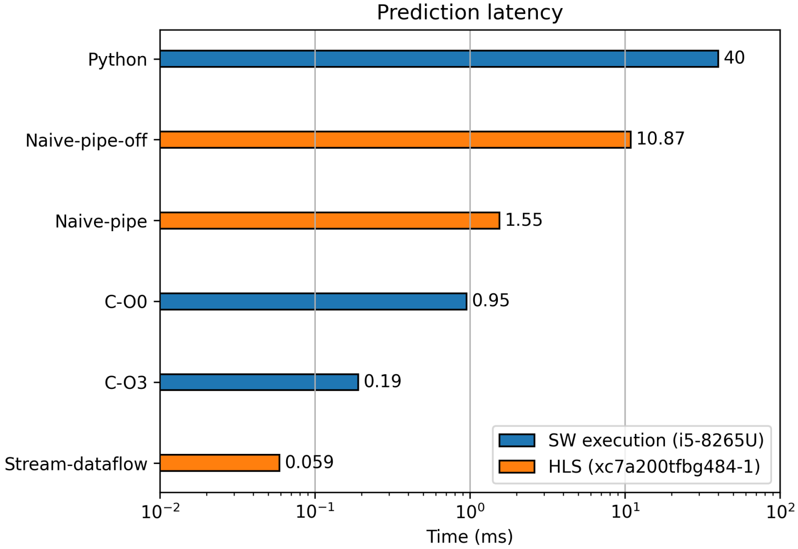
Stream-dataflow approach: code structure

```
#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. ********/

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

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

  /*  Dataflow section with streams used to transfer data between tasks:
   *  -convolution_layer;
   *  -max_pooling_layer;
   *  -flattening_layer;
   *  -dense_layer;
   *  -dense_layer_softmax.
   */
}
```

Dataflow view


pad_img
pad_img1
pad_img2
pad_img3
pad_img4
pad_img5
pad_img6
pad_img7
prediction
convolutional_layer_U0
II(avg,max,min):5975,5975,5975
Latency(avg,max,min):3997,3997,3997
max_pooling_layer_U0
II(avg,max,min):5975,5975,5975
Latency(avg,max,min):4004,4004,4004
flattening_layer_U0
II(avg,max,min):5975,5975,5975
Latency(avg,max,min):4004,4004,4004
dense_layer_U0
II(avg,max,min):5975,5975,5975
Latency(avg,max,min):4045,4045,4045
dense_layer_soft_max_U0
II(avg,max,min):5975,5975,5975
Latency(avg,max,min):4150,4150,4150
prediction
Data Edge
FIFO
PIPO
Full Channel
Empty Channel
Port
Process
Deadlock Process

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 synsthesis 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 invistigation 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.