On the Automation of High Level Synthesis of Convolutional Neural Networks

E. Del Sozzo, A. Solazzo, A. Miele, M. D. Santambrogio
emanuele.delsozzo@polimi.it
Politecnico di Milano
Rationale

• Convolutional Neural Networks (CNNs) represent the state of the art in image recognition and classification

• CNNs are applied in different fields like Big Data analysis, video surveillance and robot vision

• However…
  – due to the huge amount of data to be processed, it is crucial to find techniques to speed up the computation
  – In particular, the dataflow pattern of CNN classification algorithm results to be suitable for hardware acceleration
Proposed Solution

• A framework to automatically generate a hardware implementation of CNNs on FPGAs, based on the HLS of configurable offline-trained networks

• Main features of the framework:
  – generation of a synthesizable C++ code starting from the weights of a CNN
  – generation of scripts for Xilinx Vivado and Vivado HLS toolchains
  – CNN design customization and support for Zedboard and Zybo platforms
Motivation

• In order to generate the weights of a CNN, a software version of the CNN itself has to be built

• So, why should one use this framework?
  – HLS tools deal only with a small set of programming languages (C, C++, etc.), while machine learning frameworks use many different languages
  – Even though the CNN is implemented in C/C++, it may not be synthesizable
• A CNN is a particular type of Artificial Neural Network inspired by cells in the primary visual cortex of animals [1]
• A CNN is composed of one or more convolutional and linear layers
• In this example, the CNN is of 2 convolutional layers and 1 linear layer

• extract features from images by applying different filters (kernels)
• the more layers are used, the more complex features are extracted
• may be alternated with sub-sampling layers to reduce stored data
Linear Layers

- Implemented as a fully connected *Multi-Layer Perceptron*
- Group information collected by convolutional part
- Predict the class of the initial input image
State of the Art

• Nowadays CNNs are employed in different fields:
  – Human action recognition [2]
  – Image classification [3]
  – Natural language processing [4]

• The dataflow pattern of classification phase well suits hardware acceleration on both GPUs [5] and FPGAs [6]

• To the best of our knowledge, there are no available frameworks that ease the synthesis of CNNs on FPGAs

The Proposed Framework

We propose an easy-to-use framework that allows to design and configure a CNN.

- The front-end is designed as a web application.
- The back-end is designed in Python.

Trained Convolutional Neural Network specification

- Main structure design
- Single layer configuration
- Upload of weights file

Framework

Network generation

- GUI
- JSON file

Hardware design

- Source code generation
- C++ source code
- Scripts *.tcl generation
- Python wrappers

High Level Synthesis with Vivado Design Suite
The input are the weights of a trained CNN

The weights may be generated by means of machine learning framework like Torch [7] and TensorFlow [8]

Network Generation

Customization of:

- Convolutional part
  - Number of layers
  - Size and number of kernels
  - Presence of sub-sampling
  - Kernel size of sub-sampling

- Linear part
  - Number of layers
  - Number of neurons

Trained Convolutional Neural Network specification

Framework

- Main structure design
- Single layer configuration
- Upload of weights file

GUI

JSON file

Source code generation

C++ source code

Python wrappers

Scripts *.tcl generation

High Level Synthesis with Vivado Design Suite
Hardware Design

- Choice of target platform (Zybo or Zedboard)

- Hardware design composed of:
  - ZYNQ7 Processing System
  - AXI DMA
  - 2 AXI Interconnect
  - Processor System Reset
  - CNN IP Core

- The CNN IP Core uses *AXI4-Stream Connection* for data streaming
Block Design
• Generation of
  – CNN C++ source code
  – tcl scripts for Xilinx Vivado and Vivado HLS toolchains (2015.2 version)

• HLS and bitstream generation is (at the moment) up to the user
Experimental Results

• We synthesized different types of CNNs for Zedboard platform

• FPGA performance were compared with ARM A9 processor in terms of:
  – Prediction error
  – Execution time
  – Power/energy consumption

• We employed USPS and CIFAR-10 [9] datasets

**Test 1**

**Setup**
- 16x16 grayscale USPS Dataset
- one convolutional layer:
  - six 5x5 kernels and sub-sampling
- one linear layer:
  - 10 neurons

<table>
<thead>
<tr>
<th>Prediction Error</th>
<th>Execution Time</th>
<th>Speedup</th>
<th>Power</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>Hardware</td>
<td></td>
<td>CPU</td>
<td>Software</td>
</tr>
<tr>
<td>3.9%</td>
<td>3.9%</td>
<td>3.3s</td>
<td>2.8s</td>
<td>1.18X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPU+FPGA</td>
<td>4.19W</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Software</td>
<td>7.26J</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Percentage of FPGA Resource Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRAM</td>
</tr>
<tr>
<td>0%</td>
</tr>
</tbody>
</table>

**Software**

**Hardware**

**CPU**

**CPU+FPGA**

**Software**

**Hardware**

**CPU**

**CPU+FPGA**

**Software**

**Hardware**
Setup

- 16x16 grayscale USPS Dataset
- one convolutional layer:
  - six 5x5 kernels and sub-sampling
- one linear layer:
  - 10 neurons
- Directives:
  - DATAFLOW
  - PIPELINE

<table>
<thead>
<tr>
<th></th>
<th>Prediction Error</th>
<th>Execution Time</th>
<th>Speedup</th>
<th>Power</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Software</td>
<td>Hardware</td>
<td>Software Hardware</td>
<td>CPU</td>
<td>CPU+FPGA</td>
</tr>
<tr>
<td></td>
<td>3.9%</td>
<td>3.9%</td>
<td>3.3s</td>
<td>0.53s</td>
<td>6.23X</td>
</tr>
</tbody>
</table>
Test 3

Setup
- 16x16 grayscale USPS Dataset
- 1° convolutional layer:
  - six 5x5 kernels and sub-sampling
- 2° convolutional layer:
  - six 5x5 kernels and sub-sampling
- one linear layer:
  - 10 neurons
- Directives:
  - DATAFLOW
  - PIPELINE

<table>
<thead>
<tr>
<th>Prediction Error</th>
<th>Execution Time</th>
<th>Speedup</th>
<th>Power</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>Hardware</td>
<td>Software</td>
<td>Hardware</td>
<td></td>
</tr>
<tr>
<td>7.1%</td>
<td>7.1%</td>
<td>4.3s</td>
<td>0.48s</td>
<td>9.0X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>CPU+FPGA</th>
<th>Software</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>2.2W</td>
<td>4.24W</td>
<td>9.46J</td>
<td>2.04J</td>
</tr>
</tbody>
</table>

Diagram:
- Percentage of FPGA Resource Utilization
- BRAM
- DSP Slices
- Flip-Flops
- LUT
- Memory LUT
Setup

- 32x32 RGB CIFAR-10 Dataset
- 1° convolutional layer:
  - Twelve 5x5 kernels and sub-sampling
- 2° convolutional layer:
  - Thirty-six 5x5 kernels and sub-sampling
- 1° linear layer:
  - 36 neurons
- 2° linear layer:
  - 10 neurons
- Directives:
  - DATAFLOW
  - PIPELINE

<table>
<thead>
<tr>
<th>Prediction Error</th>
<th>Execution Time</th>
<th>Speedup</th>
<th>Power</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>Hardware</td>
<td>Software</td>
<td>Hardware</td>
<td>CPU</td>
</tr>
<tr>
<td>89.4%</td>
<td>89.4%</td>
<td>2565s</td>
<td>223s</td>
<td>11.5X</td>
</tr>
</tbody>
</table>
Experimental Results Summary

FPGA Speedup w.r.t. CPU

Energy Consumption (logarithmic scale)
Conclusions & Future Works

• We presented a preliminary framework for the automation of HLS of CNNs

• We plan to:
  – Reduce FPGA resource consumption
  – Expand the framework to support other platforms
  – Add more CNN configuration options

• The new version of the framework will be online at: http://cnn2fpga.hosting.necst.it

• Follow us on Facebook:
  CNNECST-Convolutional Neural Network
Thank You for the Attention