fpgaConvNet: A Toolflow for Mapping Convolutional Neural Networks on Embedded FPGAs

Dr. Christos-Savvas Bouganis

Marionet UK Many-core Research Network
11th of September, Bristol University, UK

Intelligent Digital Systems Lab
Dept. of Electrical and Electronic Engineering
www.imperial.ac.uk/idsl
The team

Stylianos I. Venieris
Machine Learning

Alexandros Kouris
Machine Learning, Robotics

Konstantinos Boikos
Computer Vision, SLAM

Manolis Vasileiadis
Computer Vision

Mudhar Bin Rabieah
Machine Learning

Nur Ahmadi
Brain-Machine Interface

Christos-Savvas Bouganis
iDSL Lab Director
Imperial College London
DNNs in the Embedded Space – Variability in Performance Requirements

- Smart homes/cities
- Aerial Monitoring
- Autonomous Driving
- High-Throughput Applications
- Multiobjective Applications
- Low-Latency Applications

Surveillance

Aerial Monitoring

Scene Understanding

Smart homes/cities

Autonomous Driving
DNNs in the Embedded Space – Variability in Performance Requirements

Power constraints

- Absolute power consumption
- Performance-per-Watt
Conventional and Unconventional Embedded Platforms for Neural Networks

**GPUs** – Tegra K1, X1 and X2
**DSPs** – Qualcomm Hexagon, Apple Neural Engine, ...

- ✓ High throughput
- ✓ Low latency
- ✓ Low power
- ✓ Tools

**FPGAs**
- ✓ Custom datapath
- ✓ Custom memory subsystem
- ✓ Programmable interconnections
- ✓ Reconfigurability

**Challenge**: Huge design space

**Our Approach**: Automated toolflows

Customisation

- TPU
- Myriad X
- GraphCore

- ✓ High throughput
- ✓ Low latency
- ✓ Low power
- ✓ Tools
Research Areas / Challenges

- Mapping Automation
- Multiple CNN Mapping
Challenge #1: Mapping Automation
Challenges:
- Learn to design hardware
- High-dimensional design space
- Diverse application-level needs
- High utilization of the FPGA resources
- Design automation (→ change of requirements)

Would like to:
- Target FPGAs
- Optimise for high performance
Challenge #1: Automated CNN-to-FPGA Toolflow

Network Description

Performance Requirements

FPGA Target Platform Specifications

Automated Design Space Exploration

Network Hardware Mapping

fpgaConvNet

Supplied by Deep Learning Expert
Key Characteristics

- Differentiation factors:
  - Streaming architecture
  - Hardware design tailored to the target CNN
  - No limit on #weights, or size of CNN

- Synchronous Dataflow Modelling for CNNs
  - CNN as a data-driven graph
  - Workload is represented as a matrix
  - Each layer mapped to a tunable set of hardware building blocks

- Design space exploration based on transformations
  - Coarse-grained folding
  - Fine-grained folding
  - Graph partitioning with reconfiguration
  - Weight Reloading

Max Throughput or Min Latency

\[
t_{\text{total}}(B, N_P, \Gamma) = \sum_{i=1}^{N_P} t_i(B, \Gamma_i) + (N_P - 1) \cdot t_{\text{reconfig}}.
\]
Under the hood: Convolutional Neural Networks (ConvNets)

- ConvNet Inference
  - Tailored to images and data with spatial patterns
  - Built as a sequence of layers (Convolutional, Nonlinearity and Pooling Layer)
  - Feedforward operation
  - Inherently streaming

fpgaConvNet – Streaming Architecture for CNNs

- **Src**
  - Sliding Window
  - Fork
  - Conv Unit
    - Nonlin Unit
    - Sliding Window
    - Pool Unit
  - Conv Unit
    - Nonlin Unit
    - Sliding Window
    - Pool Unit
  - Conv Unit
    - Nonlin Unit
    - Sliding Window
    - Pool Unit
  - Conv Unit
    - Nonlin Unit
    - Sliding Window
    - Pool Unit

- Memory Interface
- Convolutional Layer with 4 filters
- Nonlin Layer
- Pooling Layer
- Convolutional Layer
- Fork

Intelligent Digital Systems Lab
fpgaConvNet – Streaming Architecture for CNNs

CNN Hardware SDF Graph

Complex Model ➞ Bottlenecks:
- Limited compute resources
- Limited on-chip memory capacity for model parameters
- Limited off-chip memory bandwidth

Define a set of graph transformations to traverse the design space in **fast and principled** way
Transformation 1: Coarse-grained Folding

1) Exceeding the available compute resources
2) Not enough off-chip memory bandwidth
Transformation 1: Coarse-grained Folding

2 Convolutions/cycle

Src

Sliding Window

Fork

Conv Unit

Nonlin Unit

Sliding Window

Pool Unit

Sliding Window

Fork

Compute Resources

Required Bandwidth

Fine-grained Folding

Transformation 2
Transformation 3: Graph Partitioning with Reconfiguration

Input Data

1) Exceeding the available compute resources
2) Not enough on-chip memory capacity

FPGA Reconfiguration
Transformation 3: Graph Partitioning with Reconfiguration

1) Exceeding the available compute resources
2) Not enough on-chip memory capacity

FPGA Reconfiguration

Conv Layer 1
- K: 7x7
- S: 1
- Nout: 16
Nonlin Layer 1
- Type: ReLU
- Nout: 16
Pool Layer 1
- P: 2x2
- S: 2
- Nout: 16

Conv Layer 2
- K: 5x5
- S: 1
- Nout: 64
Nonlin Layer 2
- Type: ReLU
- Nout: 64
Pool Layer 2
- P: 2x2
- S: 2
- Nout: 64

Conv Layer 3
- K: 3x3
- S: 1
- Nout: 256
Nonlin Layer 3
- Type: ReLU
- Nout: 256
Transformation 3: Graph Partitioning with Reconfiguration

**Architecture 1**
- Conv Layer 1: K: 7x7, S: 1, Nout: 16
- Nonlin Layer 1: Type: ReLU, Nout: 16
- Pool Layer 1: P: 2x2, S: 2, Nout: 16

**Architecture 2**
- Conv Layer 2: K: 5x5, S: 1, Nout: 64
- Nonlin Layer 2: Type: ReLU, Nout: 64
- Pool Layer 2: P: 2x2, S: 2, Nout: 64

**Architecture 3**
- Conv Layer 3: K: 3x3, S: 1, Nout: 245
- Nonlin Layer 3: Type: ReLU, Nout: 256

- Reconfigure FPGA
- Run network over batch
- Write-back to off-chip memory
Transformation 3: Graph Partitioning with Reconfiguration

- Reconfigure FPGA
- Run network over batch
- Write-back to off-chip memory
Transformation 3: Graph Partitioning with Reconfiguration

- Reconfigure FPGA
- Run network over batch
- Write-back to off-chip memory

Architecture 1:
- Conv Layer 1: K: 7x7, S: 1, Nout: 16
- Nonlin Layer 1: Type: ReLU, Nout: 16
- Pool Layer 1: P: 2x2, S: 2, Nout: 16

Architecture 2:
- Conv Layer 2: K: 5x5, S: 1, Nout: 64
- Nonlin Layer 2: Type: ReLU, Nout: 64
- Pool Layer 2: P: 2x2, S: 2, Nout: 64

Architecture 3:
- Conv Layer 3: K: 3x3, S: 1, Nout: 245
- Nonlin Layer 3: Type: ReLU, Nout: 256
Transformation 3: Graph Partitioning with Reconfiguration

**Architecture 1**
- Conv Layer 1: K: 7x7, S: 1, Nout: 16
- Nonlin Layer 1: Type: ReLU, Nout: 16
- Pool Layer 1: P: 2x2, S: 2, Nout: 16

**Architecture 2**
- Conv Layer 2: K: 5x5, S: 1, Nout: 64
- Nonlin Layer 2: Type: ReLU, Nout: 64
- Pool Layer 2: P: 2x2, S: 2, Nout: 64

**Architecture 3**
- Conv Layer 3: K: 3x3, S: 1, Nout: 245
- Nonlin Layer 3: Type: ReLU, Nout: 256

- Reconfigure FPGA
- Run network over batch
- Write-back to off-chip memory

- Batch processing amortises reconfiguration cost → high throughput
- Latency-sensitive applications?
Transformation 4: Weights Reloading

Input Data

- 7x7 Conv, 16
- ReLU
- 2x2 Max Pool
- 5x5 Conv, 64
- ReLU
- 2x2 Max Pool
- 3x3 Conv, 256
- ReLU

Run-time vs bitstream-level reconfiguration to explore the latency-throughput trade-off

- Conv Layer 1
  - K: 7x7
  - S: 1
  - Nout: 16
- Nonlin Layer 1
  - Type: ReLU
  - Nout: 16
- Pool Layer 1
  - P: 2x2
  - S: 2
  - Nout: 16
- Load Conv Layer 2 Weights
- Conv Layer 2
  - K: 5x5
  - S: 1
  - Nout: 64
- Nonlin Layer 2
  - Type: ReLU
  - Nout: 64
- Pool Layer 2
  - P: 2x2
  - S: 2
  - Nout: 64
- Load Conv Layer 3 Weights
- Conv Layer 3
  - K: 3x3
  - S: 1
  - Nout: 245
- Nonlin Layer 3
  - Type: ReLU
  - Nout: 256
Transformation 4: Weights Reloading

**Workload 1**
- Conv Layer 1
  - K: 7x7
  - S: 1
  - Nout: 16
- Nonlin Layer 1
  - Type: ReLU
  - Nout: 16
- Pool Layer 1
  - P: 2x2
  - S: 2
  - Nout: 16

**Workload 2**
- Conv Layer 2
  - K: 5x5
  - S: 1
  - Nout: 64
- Nonlin Layer 2
  - Type: ReLU
  - Nout: 64
- Pool Layer 2
  - P: 2x2
  - S: 2
  - Nout: 64

**Workload 3**
- Conv Layer 3
  - K: 3x3
  - S: 1
  - Nout: 245
- Nonlin Layer 3
  - Type: ReLU
  - Nout: 256

**Generated Reference Architecture**
- Input Data
- Convolution Bank
  - max K: 7x7
- Nonlinear Bank
  - Type: ReLU
- Pooling Bank
  - max P: 2x2

- Reload weights from off-chip memory and reconfigure datapath
- Run network over batch
- Write-back to off-chip memory
• SDF-based Framework
  – Capture hardware mappings as matrices
  – Transformations as algebraic operations
  – Any local transformation propagates through the network
  – Static Scheduling
  – Analytical performance model
  – Cast design space exploration as a multiobjective optimization problem

\[ t_{\text{total}}(B, N_P, \Gamma) = \sum_{i=1}^{N_P} t_i(B, \Gamma_i) + (N_P - 1) \cdot t_{\text{reconfig}}. \]
Meeting the performance requirements
Comparison with Embedded GPUs: Same absolute power constraints (5W)

- Latency-driven scenario → batch size of 1
- Up to 6.65× speedup with an average of 3.95× (3.43× geo. mean)

- Throughput-driven scenario → favourable batch size
- Up to 5.53× speedup with an average of 3.32× (3.07× geo. mean)
Comparison with Embedded GPUs: Performance-per-Watt

fpgaConvNet vs Embedded GPU (GOp/s/W)

- Latency-driven scenario → batch size of 1
- Average of 1.70× (1.36× geo. mean) in GOp/s/W

- Throughput-driven scenario → favourable batch size
- Average of 1.17× (1.12× geo. mean) in GOp/s/W
Results: Comparison with existed FPGA frameworks

AlexNet and VGG-16 on Zynq 7020 and 7045

- AlexNet (Z020)
- VGG-16 (Z020)
- AlexNet (Z045)
- VGG-16 (Z045)

Normalized speed-up
The application scope of a framework determines the range and type of applications it can target. Focused on the more specific domain of Convolutional Neural Networks (CNNs), frameworks like fpgaConvNet, DeepBurning, and Angel-Eye, also provide back ends to Torch. FP-DNN has selected TensorFlow as its front end. With Theano supporting RNNs and Long Short-Term Memory (LSTM) networks, irregular CNN building blocks, residual blocks, and fpgaConvNet requires all the weights to be stored on-chip and therefore the supported hardware design and customising with respect to the target model only at the compiler level.

## Toolflow Characteristics

<table>
<thead>
<tr>
<th>Toolflow Name</th>
<th>Interface</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>fpgaConvNet [86][87][88][85]</td>
<td>Caffe &amp; Torch</td>
<td>May 2016</td>
</tr>
<tr>
<td>DeepBurning [90]</td>
<td>Caffe</td>
<td>June 2016</td>
</tr>
<tr>
<td>ALAMO [58][56][57][55][59]</td>
<td>Caffe</td>
<td>August 2016</td>
</tr>
<tr>
<td>HADDOC2 [1][2]</td>
<td>Caffe</td>
<td>September 2016</td>
</tr>
<tr>
<td>DNNWeaver [75][76]</td>
<td>Caffe</td>
<td>October 2016</td>
</tr>
<tr>
<td>Caffeine [98]</td>
<td>Caffe</td>
<td>November 2016</td>
</tr>
<tr>
<td>AutoCodeGen [54]</td>
<td>Proprietary Input Format</td>
<td>December 2016</td>
</tr>
<tr>
<td>FINN [84][19]</td>
<td>Theano</td>
<td>February 2017</td>
</tr>
<tr>
<td>FP-DNN [22]</td>
<td>TensorFlow</td>
<td>May 2017</td>
</tr>
<tr>
<td>Snowflake [21][10]</td>
<td>Torch</td>
<td>May 2017</td>
</tr>
<tr>
<td>SysArrayAccel [91]</td>
<td>C Program</td>
<td>June 2017</td>
</tr>
<tr>
<td>FFTCodeGen [100][97][96][95]</td>
<td>Proprietary Input Format</td>
<td>December 2017</td>
</tr>
</tbody>
</table>

**Stylianos I. Venieris, Alex Kouris and Christos-Savvas Bouganis**, “Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions”, ACM Computing Surveys, 2018
Challenge #2: Multi-CNN Systems
Challenge #2: Multi-CNN Systems – Autonomous Drones

Set of CNNs
- Object Detection
- Semantic Segmentation
- Navigation
- Monitoring
- Domain Task

Target Platform
- FPGA
- GPU
- DSP

Mapping?
Given a number of CNNs:
\[ \text{CNN}_1, \text{CNN}_2, \ldots, \text{CNN}_N \]
find a mapping to an FPGA device that meets user requirements such as Latency and Throughput per CNN

(Extra) Challenges:
- Resource allocation per CNN
- Memory Bandwidth allocation per CNN
- Scalability
Challenge #2: Multi-DNN System

Challenges:
- Resource allocation among CNNs
- Design automation
- Models with different performance constraints, e.g. required throughput and latency
- Competing for the same pool of resources
- High-dimensional design space
Multi-CNN Hardware Architecture

Key characteristics

- One hardware engine per CNN – highly customisable
- Hardware scheduler to control memory access schedule
Key characteristics

- One hardware engine per CNN – highly customisable
- Hardware scheduler to control memory access schedule

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline structure</td>
<td>$\Gamma_i$</td>
</tr>
</tbody>
</table>
Multi-CNN Hardware Architecture

Key characteristics

• One hardware engine per CNN – highly customisable
• Hardware scheduler to control memory access schedule

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline structure</td>
<td>$\Gamma_i$</td>
</tr>
<tr>
<td>No. of PEs in each stage</td>
<td>$N_{PE,i,j}$</td>
</tr>
<tr>
<td>No of MAC operators within each PE</td>
<td>$N_{op,i,j}$</td>
</tr>
</tbody>
</table>
Key characteristics

- One hardware engine per CNN – highly customisable
- Hardware scheduler to control memory access schedule

Parameter | Symbol
---|---
Pipeline structure | $\Gamma_i$
No. of PEs in each stage | $N_{PE,i,j}$
No of MAC operators within each PE | $N_{op,i,j}$
Schedule | $S$
Proposed Design Space Exploration Method

- CNN Hardware SDF Model
- Individual DSE
- Individual Pareto Curves
- Joint Feasible Space
- Scheduler
- Multi-CNN Hardware Mapping
- Code Generator
- HLS Files
- Target Platform Model
- Valid Design Space Explored

Target set of CNNs
Proposed Design Space Exploration Method

On-chip Resource Feasibility

\[ \sum_{i=1}^{N} rsc(\sigma_i) \leq rsc_{\text{Avail}}. \]
For each CNN
   – A set of subgraphs
   – Bandwidth requirements

Possible memory contention
Proposed Design Space Exploration Method

- Memory contention
  - Problem 1: Performance model ≠ Actual performance (scheduler)
  - Problem 2: Not full utilization of the memory bandwidth

- CNN inference over a stream of inputs
  - Cast to a cyclic scheduling problem
  - Search for a periodic solution

- Optimal ILP scheduler has very high runtimes for large-sized problems
- We propose a heuristic Resource Constrained List Scheduler (RCLS).
Slow-down Scheduler

- Increase the latency and decrease the bandwidth proportionally
- One slow-down factor per subgraph

\[
L'(s_{i,j}) = \frac{1}{s_{l_{i,j}}} \times L(s_{i,j}) \quad \text{Latency Increase}
\]
\[
b'(s_{i,j}) = s_{l_{i,j}} \times b(s_{i,j}) \quad \text{Bandwidth Decrease}
\]
The effect of slow-downs

Scheduler

Available Memory Bandwidth: 2 GB/s

Scheduler + slow downs

Bandwidth Requirement: 1.5 GB/s
CONV 7x7 → ReLU → MAX POOL
CNN1 - Subgraph 1 Exec Time: 0.05 ms
Slowdown1_1: 0.8x

Bandwidth Requirement: 0.25 GB/s
CONV 5x5 → ReLU → MAX POOL
CNN2 - Subgraph 1 Exec Time: 0.025 ms
Slowdown2_1: 0.8x

Bandwidth Requirement: 0.75 GB/s
CONV 5x5 → ReLU
CNN3 - Subgraph 1 Exec Time: 0.02 ms
Slowdown3_1: 0.75x

Bandwidth Requirement: 1.2 GB/s
CONV 7x7 → ReLU → MAX POOL
CNN1 - Subgraph 1 Exec Time: 0.062 ms

Bandwidth Requirement: 0.2 GB/s
CONV 5x5 → ReLU → MAX POOL
CNN2 - Subgraph 1 Exec Time: 0.031 ms

Bandwidth Requirement: 0.56 GB/s
CONV 5x5 → ReLU
CNN3 - Subgraph 1 Exec Time: 0.026 ms

Available Memory Bandwidth: 2 GB/s

Exec Time: 0.05 ms
Exec Time: 0.062 ms
Exec Time: 0.031 ms
Exec Time: 0.025 ms
Exec Time: 0.02 ms
Exec Time: 0.026 ms

Slowdown1_1: 0.8x
Slowdown2_1: 0.8x
Slowdown3_1: 0.75x

1
2
3
2
3
2

0.07 ms
0.0625 ms

time

1
2
3

1
• 3-CNN benchmark on ZC706

• Explored joint design points appear in triplets
  – Blue → peak platform-supported performance per CNN
  – Red → contention-unaware design
  – Yellow → memory-aware design
Comparison with Embedded GPUs

- Latency-driven scenario → batch size of 1
- Up to 19.09× speedup with an average of 6.85× (geo. mean)

- Latency-driven scenario → batch size of 1
- Up to 9.61× speedup with an average of 2.76× (geo. mean)
Conclusions

• Performance (efficiency) comes from customisation

• ML applications:
  • Fast moving area => new computational blocks appear frequently
  • Diverse application areas (ADAS, drones, Video analytics)

• To improve hardware’s efficiency
  => highly customisable architecture
  => large design space

• Need for Tools
Research topics

- Mapping Automation
- Multiple CNN Mapping
- Time-constrained Inference
- Privacy-aware Deep Learning


www.imperial.ac.uk/idsl


