Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions

STYLIANOS I. VENIERIS, ALEXANDROS KOURIS, and CHRISTOS-SAVVAS BOUGANIS,
Imperial College London

In the past decade, Convolutional Neural Networks (CNNs) have demonstrated state-of-the-art performance in various Artificial Intelligence tasks. To accelerate the experimentation and development of CNNs, several software frameworks have been released, primarily targeting power-hungry CPUs and GPUs. In this context, reconfigurable hardware in the form of FPGAs constitutes a potential alternative platform that can be integrated in the existing deep-learning ecosystem to provide a tunable balance between performance, power consumption, and programmability. In this article, a survey of the existing CNN-to-FPGA toolflows is presented, comprising a comparative study of their key characteristics, which include the supported applications, architectural choices, design space exploration methods, and achieved performance. Moreover, major challenges and objectives introduced by the latest trends in CNN algorithmic research are identified and presented. Finally, a uniform evaluation methodology is proposed, aiming at the comprehensive, complete, and in-depth evaluation of CNN-to-FPGA toolflows.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computing methodologies** → **Neural networks**; • **Hardware** → **Reconfigurable logic and FPGAs**; **Electronic design automation**;

Additional Key Words and Phrases: Convolutional neural networks, FPGA toolflows, deep learning

ACM Reference format:
Stylianos I. Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. 2018. Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions. *ACM Comput. Surv.* 51, 3, Article 56 (June 2018), 39 pages.
https://doi.org/10.1145/3186332

1 INTRODUCTION

Convolutional Neural Networks (CNNs) [47] have demonstrated remarkable performance in Artificial Intelligence (AI) tasks. Being able to achieve high accuracy and frequently outperform traditional AI approaches, CNNs have been employed in a vast range of applications over the past decade, from object detection [53, 72] and classification [78, 82] to drone navigation [20] and autonomous driving [7, 11]. While becoming the state-of-the-art algorithm in AI fields such as machine vision, CNNs are challenged to deal with tasks of continuously increasing complexity. This

The support of the EPSRC Centre for Doctoral Training in High Performance Embedded and Distributed Systems (HiPEDS, Grant Reference EP/L016796/1) is gratefully acknowledged.
Authors' addresses: S. I. Venieris, A. Kouris, and C.-S. Bouganis, Department of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ, London, UK; emails: {stylianos.venieris10, a.kouris16, christos-savvas.bouganis}@imperial.ac.uk.

ACM 0360-0300/2018/06-ART56 $15.00
https://doi.org/10.1145/3186332

leads to the design of deeper, more expressive networks at the expense of an increase in computational and memory requirements.

Several software libraries and frameworks have been developed to facilitate the deep learning community with the fast development and high-performance execution of CNNs. Toolflows, such as Caffe,[1] Torch,[2] and Theano,[3] and more recently Caffe2,[4] PyTorch,[5] TensorFlow,[6] MXNet,[7] CoreML,[8] CNTK,[9] and TensorRT,[10] aim to increase the productivity of CNN developers by providing high-level APIs together with high-performance execution of models on power-costly multi-core CPUs, GPUs, and DSPs, or on specialized ASICs [42]. In this context, FPGAs stand as a promising alternative target platform that can bridge the gap between power-hungry programmable architectures and fixed-function power-efficient ASICs. The reconfiguration capabilities of FPGAs could allow the generation of high-performance, low-power hardware mappings of CNNs that can be configured to meet system-level requirements such as throughput, latency, and power in diverse environments, from embedded systems to data centres.

In the past few years, High-Level Synthesis (HLS) tools have demonstrated considerable progress in generating FPGA-based hardware designs from a high level of abstraction [39]. Existing tools such as Xilinx's Vivado HLS, Intel FPGA OpenCL SDK, Maxeler's MaxCompiler, and LegUp [8] employ commonly used programming languages such as C, C++, OpenCL, and Java to facilitate the development of functionally correct hardware designs. Nevertheless, the existing HLS tools aim to yield an efficient design based on the mapping and scheduling of low-level primitive operations, leading to a large design space that does not take into account the inherent structure of the application domain. CNN workloads comprise a well-defined structure consisting of layers, with each layer having a predefined parameterization. The highly structured nature of CNN workloads enables the development of automated domain-specific frameworks that are tailored to CNNs. Such design tools could represent design points along the most important dimensions of CNNs, by capturing crucial application-level parameters, such as the topology of the CNN and the types and configurations of the layers, and map them to architectural parameters.

Currently, various systematic approaches toward the direction of automated mapping of CNNs to FPGAs have been presented. Table 1 lists the published CNN-to-FPGA toolflows in chronological order. Using the proposed frameworks, an optimized FPGA-based accelerator can be generated, given a CNN-FPGA pair. The integration of this class of accelerator generators in the existing deep-learning software frameworks would enable the user community to obtain customized hardware implementations of CNNs, without requiring any hardware design expertise, and thus would enhance the integrability of FPGAs within the deep learning ecosystem.

In this article, a survey of the various CNN-to-FPGA toolflows is presented. For this work, we consider as a toolflow any developed software that performs direct mapping of any input high-level description of a CNN to a hardware architecture that implements the inference computations of the network, under input-specified resource constraints for a target FPGA platform. The article presents a comparison between these frameworks in terms of supported neural network models,

[1] http://caffe.berkeleyvision.org/.
[2] http://torch.ch/.
[3] http://deeplearning.net/software/theano/.
[4] https://caffe2.ai/.
[5] http://pytorch.org/.
[6] https://www.tensorflow.org/.
[7] https://mxnet.apache.org/.
[8] https://developer.apple.com/documentation/coreml.
[9] https://www.microsoft.com/en-us/cognitive-toolkit/.
[10] https://developer.nvidia.com/tensorrt.

Table 1. CNN-to-FPGA Toolflows

Toolflow Name	Interface	Year
fpgaConvNet [85–88]	Caffe & Torch	May 2016
DeepBurning [90]	Caffe	June 2016
Angel-Eye [23, 24, 68]	Caffe	July 2016
ALAMO [55–59]	Caffe	August 2016
Haddoc2 [1, 2]	Caffe	September 2016
DnnWeaver [75, 76]	Caffe	October 2016
Caffeine [98]	Caffe	November 2016
AutoCodeGen [54]	Proprietary Input Format	December 2016
Finn [19, 84]	Theano	February 2017
FP-DNN [22]	TensorFlow	May 2017
Snowflake [10, 21]	Torch	May 2017
SysArrayAccel [91]	C Program	June 2017
FFTCodeGen [95–97, 100]	Proprietary Input Format	December 2017

interface, generated hardware architecture, methods used to explore the design space, supported arithmetic precision, and performance. Moreover, major challenges introduced by the latest trends in deep learning are identified and possible research directions for automated frameworks are presented. Finally, a benchmark suite together with a uniform evaluation methodology are proposed, aiming at the thorough and in-depth evaluation of CNN-to-FPGA toolflows.

2 CNN-TO-FPGA TOOLFLOW CHARACTERISTICS

In this section, existing toolflows are analysed with respect to their applicability, design methodology, and performance. The applicability to an end user is investigated based on the supported neural network models, the input interface, and the portability. The design methodology is examined based on the hardware architecture, the design space exploration approach, and the arithmetic precision choices. Finally, the performance is analysed based on the reported results of each toolflow.

2.1 Supported Neural Network Models

The application scope of a framework determines the range and type of applications it can target. The majority of the existing toolflows limit their focus on the automated mapping of CNN inference, with Finn focusing on the more specific field of Binarized Neural Networks (BNNs) [37]. The most common types of layers in a CNN are the convolutional (CONV), nonlinear (NONLIN), pooling (POOL), and fully connected (FC) layers [47]. All existing frameworks support these layers, with ALAMO, DeepBurning, DnnWeaver, and AutoCodeGen also supporting Local Response Normalization (NORM) layers [46]. Moreover, fpgaConvNet, ALAMO, and Snowflake focus mostly on the feature extractor part of CNNs, including CONV, NONLIN, and POOL layers, and offer unoptimized support for FC layers by casting them as CONV layers with 1×1 kernels. With respect to compound, irregular CNN building blocks, residual blocks [33] are supported by fpgaConvNet, ALAMO, and Snowflake, Inception modules [82, 83] by fpgaConvNet and Snowflake, and dense blocks [36] by fpgaConvNet. Haddoc2 requires all the weights to be stored on-chip, and therefore the supported model size is constrained by the storage resources of the target device. Currently, DeepBurning and FP-DNN demonstrate the widest range of supported applications by also supporting Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks [34].

2.2 Interface

2.2.1 Input. The input interface of an FPGA framework plays a decisive role in its ease-of-use and accessibility to CNN developers. Caffe constitutes the most widely supported front end with support from seven of the FPGA frameworks, including fpgaConvNet, DeepBurning, Angel-Eye, ALAMO, Haddoc2, DnnWeaver, and Caffeine, due to its structured, protobuf-based[11] syntax, the vast availability of pretrained models,[12] and the large user community. fpgaConvNet and Snowflake also provide back ends to Torch, and FP-DNN has selected TensorFlow as its front end. With Theano being the first framework to support BNNs, Finn supports Theano-defined BNNs as its input.

SysArrayAccel, AutoCodeGen, and FFTCodeGen have so far adopted custom front ends. SysArrayAccel uses C programs with embedded pragma directives as its front end and exploits the open-source ROSE[13] compiler to capture them. Similarly, AutoCodeGen uses its own proprietary network descriptor, resembling the Caffe syntax. FFTCodeGen employs a custom interface that is based on the YAML[14] serialization framework to specify the CNN model, packaged in a Python 3 wrapper. The design choice of using custom front ends makes it more difficult to integrate with the existing deep-learning toolchains and requires additional infrastructure to make it easily accessible to deep-learning practitioners.

2.2.2 Portability. A primary characteristic of a CNN-to-FPGA toolflow is the range of supported FPGAs. This feature entails the property of design portability. Portability is defined as the degree to which a toolflow can target FPGA platforms with different specifications. A toolflow with high portability would be able to target (1) devices by multiple vendors and families, (2) different setups such as System-on-Chips (SoCs), host-FPGA servers, and standalone FPGA devices, as well as (3) FPGAs of different sizes. Moreover, the choice of development tools and level of design, e.g., RTL, vendor-specific HLS, or open-source HLS, can affect a toolflow's portability.

Currently, the highest degree of portability has been demonstrated by DnnWeaver. DnnWeaver generates portable RTL in Verilog and has been reported to target both SoCs and server-grade FPGAs from both Xilinx and Intel, including the Xilinx Zynq XC7Z020 SoC and the larger Intel Stratix V GSD5 and Arria 10 GX115. In a similar manner, Haddoc2 generates RTL, which targets both Intel and Xilinx devices, while AutoCodeGen restricts its scope to RTL targeting Xilinx devices. fpgaConvNet generates its accelerators in Vivado HLS by Xilinx, while DeepBurning and Angel-Eye use RTL-level design optimized for Xilinx devices. All three toolflows currently support Xilinx SoCs with results reported on Zynq XC7Z020 and XC7Z045. In a similar manner, Snowflake targets Xilinx SoCs, such as Zynq XC7Z045. Caffeine is also developed in Vivado HLS and supports server-grade FPGAs with reported results on Kintex UltraScale KU060 and projected results on the larger Virtex 7 VX690T. At the moment, Caffeine's fully automated components target Xilinx devices that support a runnable SDAccel[15] environment and a PCIe interface between the FPGA and a host. FFTCodeGen generates RTL designs in Verilog and targets the Intel Heterogeneous Research Platform (HARP), consisting of tightly coupled CPU and FPGA with shared memory between them. The target FPGA device is Stratix V GXA7, with a 10-core Intel Xeon E5-2600 v2 CPU as a host.

FP-DNN employs both RTL-level design for its computation engine and Intel OpenCL for interfacing and control logic. In the same direction as Caffeine, FP-DNN targets Intel server-grade

[11] https://developers.google.com/protocol-buffers/.
[12] http://caffe.berkeleyvision.org/model_zoo.html.
[13] http://rosecompiler.org/.
[14] http://yaml.org/.
[15] https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html.

FPGAs, with results reported on a Catapult system [9] hosting a Stratix V GSD5 FPGA. Similarly to FP-DNN, SysArrayAccel's hardware is developed in Intel OpenCL with results reported on Arria 10 GT115. FINN generates synthesizable Vivado HLS accelerators and has demonstrated support for the Zynq XC7Z020 and XC7Z045 SoCs as well as the server-grade UltraScale KU115 device in a host-FPGA server setup. Finally, ALAMO's generated RTL designs have demonstrated support for Intel standalone and SoC platforms by targeting the standalone, high-bandwidth Stratix V GXA7 and the Arria 10 GX115 SoC.

2.3 Hardware Architecture

The architectures generated by the tools can be taxonomized in two main categories:

Streaming architectures. A streaming architecture typically consists of one distinct hardware block for each layer of the target CNN, where each block is optimized separately to exploit the parallelism of its layer. All the heterogeneous blocks are chained to form a pipeline as depicted in Figure 1. The data proceed through the different parts of the neural network as they are streamed through the architecture. As a result, this design approach exploits the parallelism between layers by means of pipelining and enables their concurrent execution. Nevertheless, the increased efficiency comes with long compilation times, since a new bitstream has to be generated for each CNN.

1) fpgaConvNet. fpgaConvNet employs a streaming architecture that assigns one processing stage per layer. Given a CNN, each layer is mapped to a series of building blocks that are chained together as a coarse pipeline. fpgaConvNet's building blocks include the most commonly utilized components of CNNs, such as convolution and pooling units as well as sliding window structures that provide line-buffering functionality. Moreover, fpgaConvNet employs specialized hardware blocks to map networks with irregular dataflow [33, 36, 82, 83], including Inception, residual, and dense hardware blocks. The performance-resource trade-off of each instantiated block is tuned separately to meet the needs of each layer in the design space exploration phase. fpgaConvNet supports multi-bitstream designs, where different hardware architectures are responsible for executing different parts of the CNN. Currently, this feature requires the full reconfiguration of the FPGA when data have to enter a new architecture, with the potential for multi-FPGA mappings.

fpgaConvNet employs a set of strategies to tailor the generated design to the input CNN while respecting the FPGA resources. For latency-sensitive applications, where the time cost of bitstream-level reconfiguration is prohibitive and batch processing cannot be used to amortise it, fpgaConvNet generates a flexible, latency-optimized architecture, which is time-shared to execute different parts of the network by means of soft, run-time reconfiguration of its datapath. Although this latency-driven design approaches the time-shared, single computation engine paradigm, the hardware stages that comprise the architecture are derived and customized based on the structure of the target network and still operate in a streaming manner. Internally, fpgaConvNet utilises a Synchronous Dataflow (SDF) model [48] to represent architectures. With SDF, the processing rates of all blocks in the system are known a priori and therefore a static schedule is generated to drive the datapath.

2) DeepBurning. In a similar approach to fpgaConvNet, DeepBurning's core consists of a library of building blocks that follow the functionality of common neural network components. Currently, the library combines conventional hardware elements, such as nonlinear and pooling operators, with more exotic components, such as dropout units [80]. Given a network structure, the framework's hardware generator builds the neural network architecture by selecting and instantiating blocks from the library, with the appropriate interconnections between them. To meet the target FPGA resource constraints, each block is parameterized so that it can be time-shared both across layers and across parts of a single layer.

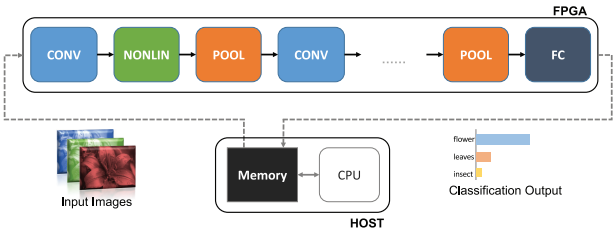

Fig. 1. Example of a streaming accelerator architecture.

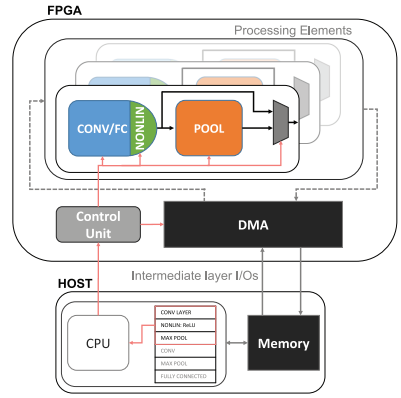

Fig. 2. Example of a single computation engine accelerator.

The architecture adopts a run-time, data-driven mechanism where each block executes whenever data are present at its inputs and largely depends on the time-sharing pattern of each block. After the datapath structure and the memory transactions schedule have been determined, the hardware generator creates a centralized control unit, which is responsible for the data movement between the off- and the on-chip memory. Moreover, a dynamic, run-time control approach is adopted by means of dedicated finite state machines that dynamically control the operation of each time-shared block. DeepBurning's dynamic dataflow approach differs from fpgaConvNet's synchronous dataflow scheme in that DeepBurning does not model the data rates of all blocks and thus requires dynamic control logic, rather than generating a static schedule at compile time.

3) Haddoc2. Haddoc2 generates its architecture by modeling the target CNN as a dataflow graph of actors and directly mapping each actor to a dedicated compute unit. This approach results in the mapping of each layer to a hardware stage, similarly to fpgaConvNet and DeepBurning, with layers executing in parallel in a pipelined manner. The hardware mapping of each layer exploits the full unrolling of its input and output feature maps, and the dot products of convolutions. Unrolling along the three aforementioned dimensions increases the required number of multipliers and on-chip storage, rapidly making the available DSPs and memory of the target FPGA device the limiting factors with respect to the size of CNN that can be mapped. To alleviate the excessive requirement for DSPs, Haddoc2 implements all its multipliers solely with logic. Furthermore, since all trained weights are required to be stored on-chip, with off-chip transactions being limited to only the input and output of the network, the weights constitute constant operands for the multipliers. As a result, during synthesis, multiplications with weight values of 0, 1, or powers of 2 are either removed or mapped to direct connections or shift operators, respectively.

With respect to scheduling, Haddoc2's architecture follows a data-driven approach with the schedule generated statically at compile time. This scheduling method is similar to fpgaConvNet's approach and differs from the dynamic control mechanism of DeepBurning. Nevertheless, in contrast to fpgaConvNet and DeepBurning, which support the time-sharing of their resources by means of folding, Haddoc2 does not support partial unrolling and, therefore, given a target device, the maximum model size can be quickly bounded either by the available logic or on-chip storage.

4) AutoCodeGen. AutoCodeGen includes parameterized hardware blocks at the layer level, supporting CONV, POOL, NORM, and FC layers. CONV blocks consist of convolvers that perform dot-product operations in a fully unrolled manner. The instantiated convolvers are

further organized in a tunable number of groups, with input feature maps being shared across all groups. Each convolver group processes the input feature maps with a different set of weights to compute independent output feature maps. Within a group, the inputs are parallelized across the convolvers, followed by an adder tree for the reduction of the partial results. FC layers are mapped to compute units, named FCcores, that tunably exploit the input neurons parallelism and can be time-multiplexed. Similarly, POOL blocks exploit the parallelism of output feature maps to a tunable degree. NORM layers are mapped to a fixed hardware block, which employs a piecewise linear approximation scheme for exponential operations and single-precision floating-point arithmetic to minimise precision loss. In contrast to the data-driven control mechanisms of the rest of the toolflows that generate streaming architectures, AutoCodeGen performs the scheduling and control of each hardware block in a distributed manner, with dedicated, local FSMs coordinating the operation of each block.

5) FINN. FINN adopts the data-driven paradigm and generates a custom streaming architecture based on a BNN's structure. Given a target BNN, each layer is mapped to a dedicated computation engine and all engines are connected in a pipelined manner. With this design, each computation engine can be configured to meet the requirements of the associated layer and match the processing rate of neighbouring engines. In this manner, the overall architecture is tailored to the particular network. With emphasis placed on BNNs, the computation engines differ from conventional CNN hardware designs and are optimized for the efficient mapping of binarized layers, including dedicated hardware for binarized convolutions, max pooling, and batch normalization [40]. FINN expresses binarized convolutions as matrix-vector operations followed by thresholding. To this end, the integral block of the architecture is the Matrix-Vector-Threshold Unit (MVTU), which is optimized to perform the majority of the core binarized operations. In terms of scheduling, FINN's approach lies closer to fpgaConvNet's synchronous dataflow scheme and farther from DeepBurning's dynamic dataflow, with static schedules generated at compile time. Finally, in contrast to fpgaConvNet and DeepBurning and similarly to HADDOC2, all the binarized weights are required to be stored on-chip, with the external memory transfers focusing only on the input and output of the network, imposing a hard limit to the size of networks that can be addressed.

Single computation engines. This design approach favors flexibility over customization. Such an architecture comprises a single computation engine, typically in the form of a systolic array of processing elements or a matrix multiplication unit, that executes the CNN layers sequentially. The control of the hardware and the scheduling of operations is performed by software (Figure 2). This design paradigm consists of a fixed architectural template that can be scaled based on the input CNN and the available FPGA resources. With this scheme, each CNN corresponds to a different sequence of microinstructions that are executable by the hardware. By taking this approach to the extreme, the architecture can be configured and scaled based only on the resources of the target FPGA without targeting a specific CNN and, as a result, after a single compilation, the same bitstream can target many CNNs without the overhead of bitstream-level reconfiguration. Despite the flexibility gains, inefficiencies are introduced due to control mechanisms that resemble those of a processor [27]. Moreover, the one-size-fits-all approach can lead to high variability in the achieved performance across CNNs with different workload characteristics.

1) Angel-Eye. The design principle behind the Angel-Eye framework is based on having a single flexible computation engine that can be programmed and controlled by software. The main computational component is an array of Processing Elements (PEs) with each PE containing a bank of convolvers, an adder tree and an optional pooling path. The input feature maps of a CONV layer are shared across all PEs and each PE processes its inputs with a different set of kernels to produce independent output feature maps. Within a PE, the inputs are parallelized across the convolvers,

followed by the adder tree that combines partial results to produce the output. Overall, the organization of Angel-Eye's and AutoCodeGen's hardware for CONV layers are following the same strategy by organizing convolvers into groups and tunably unrolling with respect to input and output feature maps.

The framework's compiler translates the input CNN to a sequence of instructions from Angel-Eye's custom instruction set and the computation engine executes the instructions. This process corresponds to the sequential execution of the layers in a time-sharing manner. With different CNNs mapped to different instruction sequences, the architecture can be reused to execute various models without recompilation or reconfiguration. In this respect, the hardware design is configured and scaled based only on the available resources of the target device and hence is CNN-independent.

2) ALAMO. In contrast to Angel-Eye, ALAMO customises the generated computation engine to the input CNN. The architecture comprises hardware blocks for POOL, ReLU, and NORM layers, together with a 2D array of compute units that is shared between CONV and FC layers. In CONV layers, the array exploits the parallelism within one input feature map and across multiple output feature maps. At each time instant, each row of the array is responsible for one output feature map, with its columns processing different windows of the same input feature map and combining their partial results synergistically. FC layers are mapped on the same hardware block, by casting them as 1×1 CONV layers. Moreover, ALAMO includes a batch normalization block and an elementwise adder. These components are employed as complementary to the main blocks, with the elementwise adder used to implement models with irregular dataflow, including residual networks [33].

Overall, ALAMO's compiler considers the layers that are present in the target CNN and instantiates only the necessary hardware blocks. After the architecture has been generated, the layers are scheduled in a sequential manner. This approach alleviates the problem of allocating resources among different layers of the same type and simplifies the design space to include only the scaling of each hardware block and the scheduling of the layers. The control of the generated accelerator is statically determined at compile time and is encoded as configurations that are loaded sequentially on the accelerator as different parts of the network are executed.

3) DnnWeaver. DnnWeaver's hardware is based on a parameterized architectural template. The template comprises an array of coarse Processing Units (PUs). Each PU contains a datapath that includes an array of Processing Elements (PEs) that execute CONV and FC layers, followed by dedicated units for NORM, POOL, and NONLIN layers. Within a PU, the CONV and POOL layers are pipelined and their execution is overlapped to exploit the parallelism across layers. The computation of output feature maps for CONV and POOL layers and output neurons for FC layers are scheduled across PUs, with PEs exploiting the parallelism between different elements of each output feature map. Generating a specific instance of the template requires trading between the number of PUs and PEs per PU, which resemble the tunable parameters of Angel-Eye's and AutoCodeGen's architectures. However, in contrast to Angel-Eye, which considers only the available resources of the target device, in DnnWeaver this tuning is performed at the design space exploration stage and is tailored to the input CNN and constrained by the resources of the target FPGA, as in the case of ALAMO.

4) Caffeine. Caffeine's hardware consists of a systolic array of PEs that perform multiplication operations. The array offers scalability in implementing convolution operations by exploiting different levels of parallelism, with optional connections between the output of each PE and dedicated blocks for ReLU and POOL layers. Moreover, support for FC layers is achieved by transforming the matrix-vector multiplications of FC layers into batched convolutions and mapping them to the

existing convolution structure, which allows the reuse of the exact same hardware for both layers. Given a CNN-FPGA pair, the number of parallel PEs is set after the design space exploration phase, so that the hardware will be tailored to the target CNN.

5) FP-DNN. Drawing from the fact that CONV and FC layers as well as recurrent connections in RNNs and the gate blocks in LSTMs can be converted to matrix multiplications, FP-DNN generates an architecture with a single generic Matrix Multiplication (*MM*) engine as its core. To balance the computational resources with the external memory bandwidth, tiling is applied on the input matrices, with the tiles processed in a pipelined manner. The *MM* engine processes the tiles in a vector by vector basis by means of a dot-product unit. The dot-product unit consists of an array of multipliers, which fully unrolls all the multiplications of the dot product, followed by an adder tree. To sustain a high utilization of the computational resources and hide the latency of the off-chip memory, FP-DNN employs double buffering for the transfer of matrix tiles. The *MM* engine is time-shared between layers, with nonlinearities and pooling operations applied by separate hardware prior to writing back intermediate results to the off-chip memory. The on-chip memory is organized as a pool of buffers that can be reused by different data at run time to sustain a high utilization, with the allocation schedule for each buffer handled as part of the design space exploration. Finally, the layer-specific control logic and the interface with the external memory and the host CPU are implemented with OpenCL-based modules.

6) Snowflake. Snowflake's hardware design employs a hierarchical structure that is designed to be controlled by software. At the top level, the architecture comprises a number of hardware compute clusters, organized as an array of tunable size. Each compute cluster contains four parallel compute units (CUs) with a shared buffer for storing feature maps of the current layer and with each CU consisting of four vector MACC (vMAC) units. Internally, each vMAC includes 16 MACC operators, that process 16-bit operands, together with a private buffer for storing weights of the current layer. In a vMAC, the MACC operators can be configured in two modes, based on the type of parallelism to be exploited. The two modes include either assigning the computation of one output feature map to each MACC operator, exploiting in this way the parallelism with respect to the output feature maps, or assigning the computation of one input feature map to each MAC operator, where the MACC operators collaborate to produce each output feature map by computing partial results. Moreover, each CU also contains a vector max pooling operator (vMAX). Similarly to FP-DNN, double buffering is employed to overlap computation and communication and hide the latency of the external memory transfers.

From an operational perspective, Snowflake is similar to Angel-Eye's programming flow. The target CNN is translated by a custom compiler, named Snowball, into a series of instructions from Snowflake's instruction set and the generated architecture executes the instructions. This process yields the execution of layers in a sequential manner. Moreover, instead of generating a different hardware design for each target CNN, different models are mapped to their own stream of instructions and the architecture can be reused without bitstream-level reconfiguration. In a similar manner to Angel-Eye, the generated hardware is CNN-independent and is scaled based only on the available resources of the target device.

7) SysArrayAccel. SysArrayAccel follows Caffeine's approach and adopts a 2D systolic array of PEs to execute all the CONV layers of the target CNN. The main differentiating factor from Caffeine's hardware is that SysArrayAccel's architecture has been designed so that each PE is only connected locally to its neighbouring PEs. With this approach, SysArrayAccel avoids the need for large multiplexers at the output of each PE, simplifying the routing and achieving high clock frequencies. Each of the two dimensions of the array corresponds to one loop in the CONV layer and each PE performs a configurable number of parallel MACC operations between inputs

and weights. The shape of the systolic array can be configured at compile time, so that different degrees of parallelism can be exploited based on the workload characteristics of the target CNN and the available FPGA resources. For the rest of the layers, dedicated hardware blocks are instantiated, with FC layers mapped to a 1D array. Given a CNN-FPGA pair, the selection of loops to be mapped on the systolic array and the shape of the array are selected in the design space exploration phase, to optimize the structure of the systolic array for the target CNN.

8) FFTCodeGen. FFTCodeGen differentiates from the rest of the existing toolflows in two main ways. First, FFTCodeGen is optimized to target the heterogeneous Intel HARP platform. In this manner, the framework partitions the CNN workload between the CPU and the FPGA, so that the CONV layers time-share the FPGA device and the rest of the layers are executed in software by the CPU. Second, in contrast to the rest of the existing frameworks, FFTCodeGen performs convolutions in the frequency domain by means of an FFT-based algorithm. With this approach, the convolution operations in the space domain are mapped to Hadamard element-by-element products in the frequency domain with decreased computational complexity.

The generated architecture consists of three main components. These comprise 2D FFT and Inverse FFT (IFFT) blocks for transforming feature maps between the space and frequency domains, and a Hadamard-Accumulation (HAC) unit. To perform FFT, FFTCodeGen organizes the input feature maps and the kernels as matrices. To support the flexible and tiled FFT-based processing of CONV layers without the need for hardware reconfiguration, FFTCodeGen combines the conventional Overlap-and-Add (OaD) method with the custom Concatenate-and-Pad (CaP) technique. OaD enables the partitioning of the input matrices into tiles of tunable size. The CaP method adds further flexibility by treating the batch size of the network as another dimension of the input feature maps matrix and introduces a tunable folding factor for the batch. The combination of OaD and CaP enable the derivation of a fixed computation engine that can be time-shared among CONV layers with different input feature map sizes, while sustaining high utilization. In this respect, all tiles of a CONV layer are sequentially fed into the generated accelerator, with double buffering used to hide memory latency, and their partial results are accumulated to produce the output feature maps matrix. Overall, FFTCodeGen uses batch processing to amortize the costs of FFT and IFFT and to enable the CaP method to sustain a high utilization of the generated accelerator by replacing ineffectual zero-padded operations with useful computations.

The 2D FFT and IFFT blocks perform N-point FFT and IFFT, respectively, by applying N-point 1D FFT on the rows of the input feature maps matrix, followed by an N-point 1D FFT on the columns of the transpose of the resulted matrix. The two blocks contain N 1D pipelines each, and share common, tunable folding factors for their rows and columns pipelines. The HAC unit performs elementwise multiplication-accumulation and comprises an array of MACC operators, which is parameterized with respect to its size. FFTCodeGen also comprises software modules for the execution of NONLIN, POOL, and FC layers by the CPU. Overall, the processing of CONV layers by the FPGA and the rest of the operations by the CPU are executed in a pipelined manner.

2.4 Design Space Exploration

Based on the parameterization and organization of its hardware, a toolflow defines a particular architectural design space. Each design point in the design space can be characterized by its performance, including latency and throughput, resource consumption and power efficiency. Typically, a framework would employ a mathematical model of the hardware with the aim to predict how a particular design point performs and investigate how to influence its performance. Design Space Exploration (DSE) refers to the task of traversing the design space and selecting one among the alternative design points based on an application-specific objective. This enables a trade-off

between attainable performance and resource distribution and utilization across the multiple tunable parameters of the architecture, under the resource constraints of the target platform for any given CNN model.

Parameter space. The proposed architecture of each framework provides different degrees of freedom for customization, expressed in terms of a set of parameters. fpgaConvNet employs a Synchronous Dataflow (SDF) model [48] to capture both the workload and the hardware mapping of CNNs and express them as SDF graphs. Each layer of the input CNN is mapped to a series of coarse hardware blocks, with each block represented as a node of the graph. The architectural space is traversed by applying a set of transformations over the SDF graph representation of the CNN hardware, such as (1) coarse-grained and (2) fine-grained folding of blocks, (3) graph partitioning with full FPGA reconfiguration, and (4) weights reloading. The folding transformations are used to control the degree of time-multiplexing of each block and influence its performance and resource consumption. The FPGA reconfiguration is used to partition the CNN into several subgraphs and effectively change the hardware as the data flow through the CNN, with one optimized hardware design (and bitstream) per subgraph. In this case, batch processing is used to amortize the reconfiguration overhead, with (5) the batch size being a configurable parameter. The weights reloading transformation includes the generation of a single flexible architecture that can be configured at run time to execute different parts of the CNN, by loading different weights from the memory and changing the datapath.

Similarly, Finn's strategy to maximize performance entails the tailoring of each hardware block along the generated streaming architecture to its layer's workload. To achieve the required performance, the processing rate between the blocks has to be balanced, since the slowest block determines the overall throughput of the system. CONV and FC layers are converted to a matrix multiplication between the trained weights and the layer's inputs. With the MVTU being the core computation engine for these operations (Section 2.3), Finn contains a mechanism to fold and time-multiplex the MVTU. Each MVTU in the architecture is compile-time configurable with respect to two parameters: (1) the number of PEs per MVTU and (2) the number of SIMD lanes per PE, which correspond to the *neuron* and *synapse folds*, respectively, following Finn's terminology.

DeepBurning's accelerator generation is performed by a hardware generator and a compiler in two steps. As a first step, the hardware generator processes the description of a neural network and creates a baseline architecture. This is achieved by selecting appropriate blocks from DeepBurning's library of neural network components and connecting them as necessary, to create a streaming architecture, as happens in fpgaConvNet and Finn. In the second step, the compiler tunes each block in the architecture so that the accelerator complies with the target FPGA resource constraints. Each block can be configured using (1) temporal folding, where several layers share the same hardware block, and (2) spatial folding, where a single layer is partitioned and all parts are processed by the hardware block in a time-multiplexed manner.

AutoCodeGen instantiates one hardware block per CNN layer. Similarly to Finn, the rate of processing between blocks has to be balanced by tuning the parallelism degree of each hardware block. Each CONV block is compile-time configurable with respect to (1) the number of convolver groups and (2) the number of convolvers per group. Accordingly, each FCcore (Section 2.3) is configurable, with respect to (3) the size of the multiplier array and the corresponding adder tree.

In DnnWeaver, the input CNN is mapped to a dataflow-based intermediate representation, similar to fpgaConvNet. Each node represents an instruction from DnnWeaver's custom instruction set, with one instruction associated with each layer. The adopted dataflow representation differs from fpgaConvNet's SDF model in that it is utilized to obtain a high-level model of the CNN's workload while fpgaConvNet employs SDF to model both the CNN workload and its hardware mapping. The architectural template is parameterized and tunable with respect to (1) the number

of PUs and (2) the number of PEs per PU as described in Section 2.3, as well as with respect to (3) the scheduling of operations. The scheduling is controlled via the tiling factors for each layer's output feature maps, which is processed by each PU, and influences the amount of communication with the off-chip memory.

Caffeine adopts a uniform representation for both the CONV and FC layers, which allows the reuse of the same hardware for both layers, as happens in Finn. The design parameters to be optimized include (1) the tiling factors along the three dimensions of the input and output feature maps, (2) the tiling factors of the kernels in CONV layers, and (3) the batch size.

SysArrayAccel interprets CONV layers as nested loops. Analytical performance and resource consumption models have been constructed for the systolic array hardware, which are parameterized with respect to (1) the data reuse patterns of the nested loops and (2) the shape of the array. Given a target CNN, different data reuse strategies yield different degrees of parallelism and correspond to selecting two of the nested loops to be mapped on the two dimensions of the systolic array and one loop on the parallel MACC resources of each PE. The shape of the array consists of three parameters that determine the size of each of the two dimensions in the array and the number of parallel MACC units in each PE. SysArrayAccel's tunable parameters enable the exploration along different data reuse strategies and the shaping of the computation engine with three degrees of freedom, to traverse the throughput-resource cost space.

ALAMO generates an accelerator by integrating a set of parameterized modules. Depending on the amount of resources of the target device and the distribution of computational workload in the target CNN, different degrees of parallelism (1) within an input feature map and (2) across output feature maps are exploited. FFTCodeGen instantiates N-point FFT and IFFT hardware blocks for converting feature maps between the space and frequency domains (1) with N being a design parameter. The two blocks are individually parameterized with respect to (2) the folding factor of each pipeline. The HAC unit (Section 2.3) is also parameterized with respect to (3) the number of MACC operators. Finally, (4) the buffer sizes for feature maps and weights are also tunable.

In contrast to the previously described approaches, Angel-Eye's, FP-DNN's, and Snowflake's design principle dictates that the hardware architecture should be independent of the CNN workload. In accordance to this approach, Angel-Eye's generated architecture is parameterized with respect to (1) the number of PEs and (2) the number of convolvers per PE as described in Section 2.3 and their values are selected so that the resource utilization of the target platform is maximized. Similarly, FP-DNN configures its main computation block based only on the available resources of the target platform. As a result, FP-DNN's Matrix Multiplication (*MM*) engine is compile-time configurable with respect to (1) tile size, which is set so that *MM*'s throughput matches the off-chip memory bandwidth of the target platform. Moreover, FP-DNN adopts a resource-sharing strategy for the available on-chip memory resources by organizing the on-chip memory as a pool of buffers, with (2) the allocation schedule of each buffer left as a parameter for the DSE. Finally, Snowflake can be scaled at compile time only with respect to (1) the number of compute clusters based on the available resources of the target device, while the number of compute units (CUs) and MACC operators per CU are fixed.

In a different approach to the rest of the toolflows, Haddoc2 captures the input CNN as a dataflow graph of actors and maps each actor to a physical dedicated hardware block, via a process named Direct Hardware Mapping (DHM). With this approach, the architecture is generated deterministically following the exact topology of the network, without configurable parameters.

Design space formulation and search. The existing FPGA frameworks adopt different levels of analysis for design space exploration, which leads to different DSE methods. fpgaConvNet and DnnWeaver cast the DSE as a formal constrained optimization problem subject to the resource

budget of the target FPGA. In each case, the objective function is a mathematical performance model of the hardware, with fpgaConvNet offering either throughput maximization [86], latency minimization [88] or multiobjective criteria [85] (such as latency-constrained throughput maximization) based on the user's needs, while DnnWeaver focuses on throughput maximization and employs batch processing. Due to the large parameter space that would make a brute-force enumeration intractable, both frameworks employ heuristic search methods to obtain a solution to the optimization problem. DnnWeaver employs a proprietary search algorithm while fpgaConvNet utilizes a custom global optimizer based on the Simulated Annealing algorithm [71].

Following a different approach, Caffeine bases its DSE on an enhanced version of the roofline model [93, 99]. The refined roofline model yields a better estimate of the effective off-chip memory bandwidth by making it dependent on the burst length of each transfer. In contrast to DnnWeaver and fpgaConvNet, Caffeine's adoption of the higher-level roofline-based modeling leads to a relatively small design space, which enables exhaustive enumeration, with the roofline model used to select the design point with the highest throughput subject to the target platform's memory bandwidth and FPGA resources. To limit the latency overhead caused by batch processing, Caffeine converts FC layers to CONV with a method that allows even small batches to reach high throughput.

SysArrayAccel's DSE formulation lies closer to fpgaConvNet's analytical high-level modeling. Emphasis is placed on constructing accurate performance and resource models of the hardware given the selected data reuse patterns and the shape of the systolic array, and casting the DSE as an optimization problem that aims to maximize throughput. The analytical approach of SysArrayAccel leads to a high-dimensional design space, which makes DSE a difficult task. While fpgaConvNet and DnnWeaver employed a global optimizer and a heuristic search algorithm, respectively, to address this issue, SysArrayAccel applies a number of pruning strategies, including only the consideration of design points that demonstrate high consumption of the FPGA resources, to reduce the design space and make an exhaustive enumerative search feasible. As a result, although SysArrayAccel substitutes Caffeine's roofline-based modeling with analytical models, both frameworks employ exhaustive enumeration for the selection of the highest-throughput design point subject to the target off-chip memory bandwidth and FPGA resource constraints.

FFTCodeGen formulates DSE in a manner that combines the analytical approaches of fpgaConvNet, DnnWeaver and SysArrayAccel, with the roofline model of Caffeine. A roofline model is developed as a function of the number of points (N) of the FFT, to obtain the value of N that balances the computation-to-communication ratio for the input CNN on the target platform. FFTCodeGen's DSE expresses the computational roof as a function of N and captures the computation-to-communication bound of the target device by means of a single custom metric, named *device coefficient*. This formulation enables the efficient traversal of the high-dimensional design space and differs to the strategies of fpgaConvNet, DnnWeaver and SysArrayAccel to handle large design spaces. After the highest performing N for the target CNN-FPGA pair has been determined, the analytical models are used to obtain the rest of the tunable parameters in a closed form. FFTCodeGen optimizes for high throughput, with customisable constraints on the number of points of the FFT and the batch size to also support latency-driven applications.

Finn's objective is to reach a user-defined throughput. The framework's synthesizer module is responsible for determining the values for the folding parameters, using the balancing of the processing rates of all Matrix-Vector-Threshold Units as a heuristic. Besides throughput maximization, Finn's generated hardware design is also optimized with respect to latency, since no batching of inputs is required. With an approach close to Finn's, DeepBurning's compiler performs a heuristic search to set the folding parameters of the generated hardware to comply with the resource constraints. Similarly to Finn, the generated design runs with a batch size of 1 and hence both

throughput and latency are optimized simultaneously. In resemblance to SysArrayAccel and fpgaConvNet, AutoCodeGen employs high-level analytical performance and resource models to set the tunable parameters of each instantiated hardware block, with balancing the processing rates of all hardware blocks as a heuristic, in a similar approach to Finn.

FP-DNN's mapping strategy focuses on reusing the FPGA resources across layers. With respect to computational resources, the tile size of the single Matrix Multiplication engine is heuristically selected to match the off-chip memory bandwidth of the target platform. With respect to memory resources, the allocation schedule of the pool of on-chip buffers is cast as a graph coloring problem that is solved algorithmically, by taking into account the time slots during which the data of each buffer have to remain intact and aiming to find a feasible reuse schedule that maximizes buffer utilization.

ALAMO's DSE focuses on the instantiation of the appropriate hardware blocks, the scaling of each block and the scheduling of layers. The structure of the compute engine is derived based on the topology and layers of the input CNN. After the necessary modules have been instantiated, the compiler's heuristic considers the resource budget of the target FPGA device and determines the unroll factors within an input feature map and across the output feature maps of each layer, to scale the 2D array of MACC operators and the POOL block (Section 2.3). ALAMO is designed to operate with a batch size of 1 and therefore throughput and latency are co-optimized.

In contrast to the rest of the toolflows, Angel-Eye's and Snowflake's hardware generation are CNN-independent and rely only on the available resources. Each of the two frameworks has a compiler that translates the input CNN to a series of instructions for the accelerator in a heuristic manner. The DSE process includes the CNN-to-instructions mapping, with throughput maximization as an objective. When several mappings with equal performance are possible, Angel-Eye's compiler prioritizes mappings that minimize the off-chip memory accesses to reduce the bandwidth requirements and power consumption. Moreover, Snowflake's compiler performs optimizations based on the structure of the target CNN, including loop removal, unrolling and rearrangement, and includes a communication load balancing technique to sustain a high utilization of the compute resources. Similarly to Finn and DeepBurning, Angel-Eye and Snowflake are designed to operate with batch size of 1 and hence throughput and latency are co-optimized.

Haddoc2's DHM approach performs a one-to-one mapping between the target network and the generated hardware, without considering the specifications of the target platform. As a result, given an input CNN, the toolflow deterministically generates a hardware design independently of the available resources, and the resulting design is feasible only if it fits within the resource budget of the target device. Moreover, the generated architecture operates with a batch size of 1 and hence is optimized for both throughput and latency.

2.5 Arithmetic Precision

In FPGA-based CNN implementations, data quantization with few bits has been widely employed. Low-precision fixed-point data representation has been studied to achieve comparable accuracy with high-precision floating-point due to the significant redundancy of the models, while demonstrating a drastic increase in performance [81]. The benefits of employing custom-precision arithmetic are manifold, including reducing the external memory bandwidth requirements (and thus decreasing power consumption due to off-chip memory data transfers), minimizing the on-chip memory footprint, reducing the resource utilization by implementing fixed-point arithmetic units, and thus leading to better hardware efficiency.

Based on the observation that significant variation is demonstrated between the dynamic range of data in different layers of the same network, Angel-Eye employs an automated quantization method to perform dynamic quantization across layers. Given a predefined wordlength for the

whole network, different scaling, which determines the radix point position, is selected for each layer. Determining the scaling for each layer is formulated as an optimization problem, solved by a greedy method that minimizes the residual error between the network's outputs when fixed-point and floating-point representations are used. After the scaling of each layer has been selected, the quantized, fixed-point weights are fine-tuned by means of a retraining step, to compensate for the accuracy loss due to quantization.

ALAMO and AutoCodeGen allow the wordlength and scaling of each unit to be adjusted at compile time, so that in layers with narrow dynamic range a longer fractional part will be allocated and vice versa. Similarly, DnnWeaver features dedicated bits within each instruction (generated by the toolflow's translator module) that dictate whether floating- or fixed-point results should be generated, together with the number of fractional bits and the total bitwidth. Also, in DeepBurning, all components in the hardware library support parameterizable input bitwidth, the value of which is determined by the hardware generator of the toolflow based on the resource constraints.

Caffeine, fpgaConvNet, FP-DNN, SysArrayAccel, and FFTCodeGen provide support for both floating- and fixed-point representations of feature maps and weights. However, a uniform quantization is applied to all layers in all cases, with fixed wordlength and scaling across them. The same approach is followed by Haddoc2, except that only fixed-point representation is supported. Snowflake also employs uniform quantization across all layers but with a fixed bitwidth of 16 bits. Finally, Finn consists almost entirely of binary operations as it focuses on BNNs.

2.6 Performance

The most critical characteristic of a CNN-to-FPGA toolflow is the achieved performance of the generated system given a CNN-FPGA pair. An accelerator's primary performance metrics of interest are throughput and latency. A tool's Quality of Results (QoRs) can be evaluated with respect to two factors: (1) comparison with other toolflows for the same CNN-FPGA pair and (2) comparison with hand-tuned accelerators for the same CNN-FPGA pair. Meaningful and fair comparisons across all toolflows would require each toolflow to generate an accelerator for the same CNN targeting the same FPGA device. Nevertheless, the majority of the existing toolflows have not yet been publicly released, which does not allow us to obtain results for the same CNN-FPGA benchmarks. At the moment of writing, DnnWeaver has an open-source version[16] that provides limited support for the Zynq XC7Z020 platform, Haddoc2 has been open-sourced,[17] Finn has been released in a lightweight version[18] that targets Xilinx's PYNQ-Z1 board and a set of specific BNNs, fpgaConvNet has a dedicated webpage[19] that presents up-to-date benchmarking results on several networks, and Angel-Eye is internally used by DeePhi. Due to this fact, the sole feasible method to evaluate each toolflow's achieved performance is by referring to the reported results either in the corresponding publications or by direct communication with the authors. In this study, we combined both approaches to collect the presented results.

In this section, a performance comparison is presented with the aim to depict an as much as possible well-rounded view of the strengths and weaknesses of each toolflow and draw conclusions about the different mapping strategies. Our evaluation methodology consists of two components: (1) to conduct a fair and meaningful evaluation, we perform direct comparisons only between tools that have mapped the same CNN model on the same FPGA device, and (2) we assess the quality of

[16] http://act-lab.org/artifacts/dnnweaver/.
[17] https://github.com/KamelAbdelouahab/haddoc2.
[18] https://github.com/Xilinx/BNN-PYNQ.
[19] http://cas.ee.ic.ac.uk/people/sv1310/fpgaConvNet.html.

the automatically generated designs by comparing with the current state-of-the-art, hand-tuned designs for the same CNN-FPGA pairs.

So far, results have been reported on a variety of CNN models, with different tools selecting different benchmarks and devices. Our evaluation is focused on the most commonly mapped AlexNet and VGG16 networks, with a number of additional comparisons on LeNet-5, CIFAR-10, GoogLeNet, and ResNet-152. Detailed results for both the feature extractors (CONV) and the feature extractors followed by classifiers (OVRL)[20] are listed in Tables 2 and 3 for AlexNet and VGG16, respectively, on Zynq and UltraScale platforms and in Tables 4 and 5 for AlexNet and VGG16, respectively, on Stratix V and Arria 10 platforms. Resource-normalizd metrics[21] are also included, since, despite their limitations, which are discussed in Section 3.1, they constitute the current literature standard metric for CNN accelerator comparisons across different devices. For platforms from the same FPGA family and vendor, normalization with respect to LUTs and ALMs can be used. For heterogeneous platforms, normalization with DSPs is employed.

Comparison between toolflows. Figure 4(a) and 4(b) present comparisons of toolflows for the mapping of AlexNet and VGG16 on Zynq platforms. fpgaConvNet, DeepBurning, and DnnWeaver mapped AlexNet on the resource-limited Zynq XC7Z020 platform (Figure 4(a)). With respect to the feature extractor, fpgaConvNet achieves a throughput of 38.30GOp/s and outperforms DeepBurning and DnnWeaver by 2.06× and 1.9×, respectively, while for the whole AlexNet, DnnWeaver reaches 1.34× higher throughput than DeepBurning. With respect to latency, fpgaConvNet's latency-driven methodology yields a 1.37× lower latency than DeepBurning for AlexNet's feature extractor. DnnWeaver has been optimized for high-throughput applications and requires batch processing to achieve high performance. Therefore, DnnWeaver's latency has not been considered. When targeting the resource-richer Zynq XC7Z045, fpgaConvNet achieves 1.82× higher throughput and 1.49× lower latency compared to DeepBurning, demonstrating a similar trend to AlexNet on Zynq XC7Z020. Compared to Snowflake, fpgaConvNet reaches 1.64× higher throughput and 1.21× lower latency, with Snowflake achieving 1.11× higher throughput than DeepBurning. With respect to mapping VGG16 on Zynq XC7Z020 (Figure 4(b)), fpgaConvNet achieves 1.22× higher throughput than DnnWeaver. The gap between the two toolflows is small and possibly due to the finer exploration method of fpgaConvNet.

Both fpgaConvNet and Angel-Eye have mapped VGG16 on Zynq XC7Z045 (Figure 4(b)). Angel-Eye has achieved the current state-of-the-art performance of VGG16 on Zynq XC7Z045 with 1.20× higher throughput than fpgaConvNet for the feature extractor and 136.97 GOp/s for the whole network. Moreover, Angel-Eye achieves 1.52× lower latency than fpgaConvNet.

fpgaConvNet and Haddoc2 have both generated accelerators for the low-end LeNet-5 and CIFAR-10 on Zynq XC7Z045 (Figure 3). In these two cases, Haddoc2 achieves 1.71× on LeNet-5 and 2.63× on CIFAR-10 higher throughput than fpgaConvNet, while using 3-bit and 6-bit bitwidth, respectively, for the two networks compared to the 16-bit representation of fpgaConvNet.

DnnWeaver and FP-DNN mapped VGG16 and VGG19, respectively, on Stratix V GSD5 (Figure 5). VGG19 has a larger workload compared to VGG16 by having three additional CONV layers. A larger number of CONV compared to FC layers can facilitate an accelerator's performance, since CONV layers are computation bounded. Noting this difference in the VGG19 and VGG16 workloads, we use FP-DNN's performance on VGG19 as an indicator of its throughput on VGG16. In this respect, FP-DNN achieves a throughput of 364.36 GOp/s and outperforms DnnWeaver by 2.31×.

[20] The complete performance results were obtained by contacting the authors.
[21] Results are normalized over the available resources of the target device.

Table 2. Performance Comparison of AlexNet on Zynq and UltraScale Platforms

	fpgaConvNet		DeepBurning		DnnWeaver	Snowflake	Caffeine**
FPGA Platform	Zynq XC7Z020	Zynq XC7Z045	Zynq XC7Z020	Zynq XC7Z045	Zynq XC7Z020	Zynq XC7Z045	UltraScale KU060
Frequency	125 MHz	125 MHz	100 MHz	100 MHz	150 MHz	250 MHz	200 MHz
Logic Capacity	53.20 kLUTs	218.60 kLUTs	53.20 kLUTs	218.60 kLUTs	53.20 kLUTs	218.60 kLUTs	331.68 kLUTs
DSPs*	220	900	220	900	220	900	2760
On-chip Memory	0.6 MB	2.4 MB	0.6 MB	2.4 MB	0.6 MB	2.4 MB	4.7 MB
Arithmetic Precision	Q8.8 16-bit fixed-point	Q8.8 16-bit fixed-point	16-bit fixed-point	16-bit fixed-point	Q3.13 16-bit fixed-point	Q8.8 16-bit fixed-point	Q8.8 16-bit fixed-point
Performance	38.30 (CONV)	197.40 (CONV)	18.53 (CONV)	108.25 (CONV)	20.16 (CONV)	120.30 (CONV)	163.00 (CONV)
(GOp/s)	-	-	15.29 (OVRL)	57.31 (OVRL)	20.51 (OVRL)	-	165.00 (OVRL)
Performance Density	0.72 (CONV)	0.90 (CONV)	0.35 (CONV)	0.49 (CONV)	0.38 (CONV)	0.55 (CONV)	0.81 (CONV)
(GOp/s/kLUT)	-	-	0.29 (OVRL)	0.26 (OVRL)	0.38 (OVRL)	-	0.83 (OVRL)
Performance Density	0.1709 (CONV)	0.2193 (CONV)	0.0842 (CONV)	0.1203 (CONV)	0.0916 (CONV)	0.1336 (CONV)	0.0983 (CONV)
(GOp/s/DSP)	-	-	0.0695 (OVRL)	0.0637 (OVRL)	0.0932 (OVRL)	-	0.0996 (OVRL)
Latency (batch size = 1)	12.70 ms (CONV)	8.22 ms (CONV)	71.75 ms (CONV)	12.30 ms (CONV)	-	9.95 ms (CONV)	-
	-	-	95.48 ms (OVRL)	25.47 ms (OVRL)	-	-	-

* 25×18 DSP configurations.

** Caffeine's use of SDAccel is reported to have an up-limit of 60% of the available resources. Therefore, the 60% is used for the resource-normalized metrics.

Table 3. Performance Comparison of VGG16 on Zynq and UltraScale Platforms

	fpgaConvNet		DnnWeaver	Angel-Eye	Caffeine**
FPGA Platform	Zynq XC7Z020	Zynq XC7Z045	Zynq XC7Z020	Zynq XC7Z045	UltraScale KU060
Frequency	125 MHz	125 MHz	150 MHz	150 MHz	200 MHz
Logic Capacity	53.20 kLUTs	218.60 kLUTs	53.20 kLUTs	218.60 kLUTs	331.68 kLUTs
DSPs*	220	900	220	900	2760
On-chip Memory	0.6 MB	2.4 MB	0.6 MB	2.4 MB	4.7 MB
Arithmetic Precision	Q8.8 16-bit fixed-point	Q8.8 16-bit fixed-point	Q3.13 16-bit fixed-point	Q8.8 16-bit fixed-point	Q8.8 16-bit fixed-point
Performance	48.53 (CONV)	155.81 (CONV)	31.35 (CONV)	187.80 (CONV)	310.00 (CONV)
(GOp/s)	-	-	31.38 (OVRL)	136.97 (OVRL)	266.00 (OVRL)
Performance Density	0.91 (CONV)	0.71 (CONV)	0.59 (CONV)	0.86 (CONV)	1.55 (CONV)
(GOp/s/kLUT)	-	-	0.59 (OVRL)	0.62 (OVRL)	1.33 (OVRL)
Performance Density	0.2206 (CONV)	0.1731 (CONV)	0.1425 (CONV)	0.2086 (CONV)	0.1871 (CONV)
(GOp/s/DSP)	-	-	0.1426 (OVRL)	0.1522 (OVRL)	0.1606 (OVRL)
Latency (batch size = 1)	633.01 ms (CONV)	249.50 ms (CONV)	-	163.42 ms (CONV)	-
	-	-	-	224.60 ms (OVRL)	-

* 25×18 DSP configurations.

** Caffeine's use of SDAccel is reported to have an up-limit of 60% of the available resources. Therefore, the 60% is used for the resource-normalized metrics.

Table 4. Performance Comparison of AlexNet on Stratix V and Arria 10

	DnnWeaver		ALAMO	SysArrayAccel	FFTCodeGen
FPGA Platform	Stratix V SGSD5	Arria 10 GX115	Stratix V GXA7	Arria 10 GT115	Stratix V GXA7
Frequency	200MHz	200MHz	100MHz	239.62MHz	200MHz
Logic Capacity	172.60 kALMs	427.20 kALMs	234.72 kALMs	427.20 kALMs	234.72 kALMs
DSPs*	3180	3036	512	3036	512
On-chip Memory	4.9MB	6.6MB	6.25MB	6.6MB	6.25MB
Arithmetic Precision	Q3.13 16-bit fixed-point	Q3.13 16-bit fixed-point	Q8.8 16-bit fixed-point	32-bit floating-point	16-bit fixed-point
Performance	97.10 (CONV)	265.36 (CONV)	134.10 (CONV)	406.1 (CONV)	-
(GOp/s)	97.56 (OVRL)	184.33 (OVRL)	114.50 (OVRL)	360.4 (OVRL)	780.60 (OVRL)
Performance Density	0.56 (CONV)	0.62 (CONV)	0.57 (CONV)	0.95 (CONV)	-
(GOp/s/kALM)	0.56 (OVRL)	0.43 (OVRL)	0.49 (OVRL)	0.84 (OVRL)	3.32 (OVRL)
Performance Density	0.0305 (CONV)	0.0874 (CONV)	0.2619 (CONV)	0.1337 (CONV)	-
(GOp/s/DSP)	0.0307 (OVRL)	0.0607 (OVRL)	0.2236 (OVRL)	0.1187 (OVRL)	1.5246 (OVRL)
Latency (batch size = 1)	-	-	9.92 ms (CONV)	-	-
	-	-	12.75 ms (OVRL)	4.05 ms (OVRL)	-

* 18×18 DSP configurations.

ALAMO maps AlexNet, VGG16, and ResNet-152 on Stratix V GXA7, which differs from the GSD5 device used by DnnWeaver and FP-DNN, with FFTCodeGen mapping AlexNet and VGG16 on the same device that is present on the Intel HARP platform. Despite belonging to the same FPGA family, the two devices have been designed and optimized for applications with different characteristics. Stratix V GSD5 has been designed for algorithms with a large number of multiply-accumulate operations, while Stratix V GXA7 is optimized for high-bandwidth applications. In this respect, FP-DNN, DnnWeaver, ALAMO, and FFTCodeGen are compared with respect to DSP-normalized throughput (Figure 5). ALAMO demonstrates 6× and 13.89× higher normalized throughput than FP-DNN and DnnWeaver on VGG16. Furthermore, ALAMO achieves 7.28× higher normalized throughput than DnnWeaver for the mapping of AlexNet on Stratix V and 7.64× higher normalized throughput than FP-DNN on ResNet-152 on Stratix V. Similarly, FFTCodeGen's VGG16 accelerator achieves 11.41×, 26.4×, and 1.9× higher GOp/s/DSP than FP-DNN, DnnWeaver and ALAMO, with FFTCodeGen's AlexNet design demonstrating 49.66× and 5.82× higher GOp/s/DSP over DnnWeaver and ALAMO. However, this metric does not capture the bandwidth difference of the two devices. Despite having fewer DSP blocks, the high bandwidth of Stratix V GXA7 enables ALAMO and FFTCodeGen to sustain a higher utilization of their DSP resources. As a result, DSP-normalized throughput does not reflect the intrinsic strengths and weaknesses of the four designs, since it does not take into account essential device-specific characteristics. Moreover, ALAMO is employing both DSPs and ALMs to implement its compute units, which enables the toolflow to reach higher performance than the majority of its DSP-based counterparts, while FFTCodeGen performs convolutions in the frequency domain with lower computational complexity and outperforms the competing toolflows.

DnnWeaver and SysArrayAccel mapped AlexNet on Arria 10 GX115 and GT115 (Figure 6). On AlexNet, SysArrayAccel achieves a throughput of 360.4 GOp/s and outperforms DnnWeaver by 1.95×. By targeting VGG16 on the same device, SysArrayAccel reaches 1.27× (with FP precision) and 3.24× (with 16-bit FXP precision) higher throughput than DnnWeaver, while outperforming ALAMO by 1.62× (with 16-bit FXP precision) and with ALAMO overpassing by 1.56× (with FP precision). AutoCodeGen and Caffeine are the only toolflows to target Virtex 7 VX690T and UltraScale KU060, respectively, and therefore no meaningful comparison can be conducted with the rest of the toolflows.

Table 5. Performance Comparison of VGG16 on Stratix V and Arria 10

	DnnWeaver		ALAMO		FP-DNN**	SysArrayAccel	FFTCodeGen
FPGA Platform	Stratix V SGSD5	Arria 10 GX115	Stratix V GXA7	Arria 10 GX115	Stratix V SGSMD5	Arria 10 GT115	Stratix V GXA7
Frequency	200MHz	200MHz	150MHz	200MHz	150MHz	231.85MHz	200MHz
Logic Capacity	172.60 kALMs	427.20 kALMs	234.72 kALMs	427.20 kALMs	172.60 kALMs	427.20 kALMs	234.72 kALMs
DSPs*	3180	3036	512	3036	3180	3036	512
On-chip Memory	4.9MB	6.6MB	6.25MB	6.6MB	4.9MB	6.6MB	6.25MB
Arithmetic Precision	Q3.13 16-bit fixed-point	Q3.13 16-bit fixed-point	Q8.8 16-bit fixed-point	Q8.8 16-bit fixed-point	16-bit fixed-point	16-bit fixed-point	16-bit fixed-point
Performance	157.39 (CONV)	390.02 (CONV)	-	-	-	-	-
(GOp/s)	157.51 (OVRL)	361.55 (OVRL)	352.24 (OVRL)	720.15 (OVRL)	364.36 (OVRL)	1171.30 (OVRL)	669.10 (OVRL)
Performance Density	0.91 (CONV)	0.91 (CONV)	-	-	-	-	-
(GOp/s/kALM)	0.91 (OVRL)	0.84 (OVRL)	1.50 (OVRL)	2.74 (OVRL)	2.11 (OVRL)	2.74 (OVRL)	2.85 (OVRL)
Performance Density	0.0495 (CONV)	0.1284 (CONV)	-	-	-	-	-
(GOp/s/DSP)	0.0495 (OVRL)	0.1191 (OVRL)	0.6879 (OVRL)	0.2372 (OVRL)	0.1145 (OVRL)	0.3858 (OVRL)	1.3068 (OVRL)
Latency (batch size = 1)	-	-	-	-	-	-	-
	-	-	87.87 ms	42.98 ms (OVRL)	-	26.85 ms (OVRL)	-

* 18×18 DSP configurations.

** FP-DNN maps VGG19 on Stratix V.

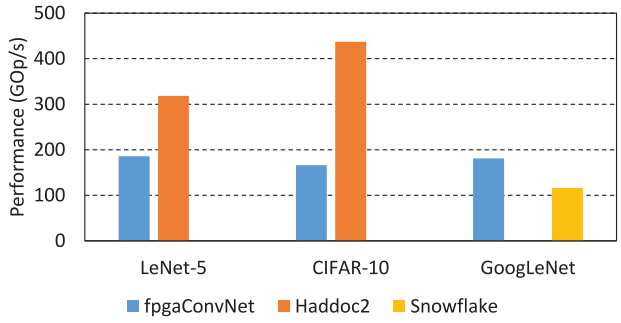

Fig. 3. Comparison on mapping LeNet-5, CIFAR-10, and GoogLeNet on Zynq XC7Z045.

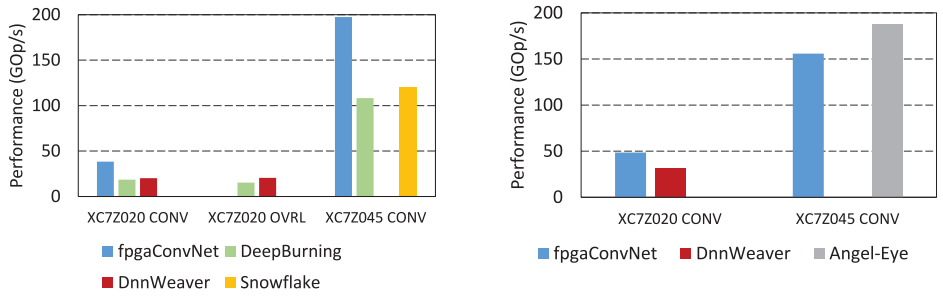

Fig. 4. Comparison targeting Zynq platforms.

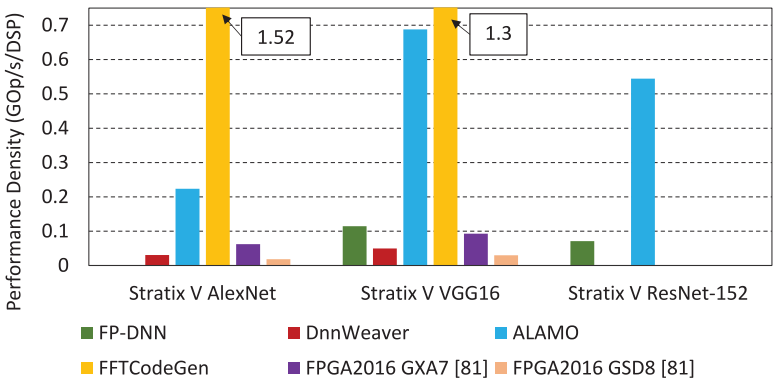

Fig. 5. DSP-normalized comparison on mapping AlexNet, VGG16, and ResNet-152 on Stratix V.

Comparison between toolflows discussion. Among fpgaConvNet, DnnWeaver, DeepBurning, and Snowflake, fpgaConvNet's higher throughput comes potentially as a result of its SDF-based design methodology that allows for a finer exploration of the design space. However, DnnWeaver's heuristic mapping and scheduling algorithm aims to optimally configure the templates of its accelerator, which leads to higher performance than DeepBurning, but still with less room for finer-grained customization over a given CNN-FPGA pair than fpgaConvNet. With respect to latency, although DeepBurning and Snowflake are designed to co-optimize latency and throughput by operating with a batch size of 1, fpgaConvNet's latency-driven methodology [88] is explicitly used

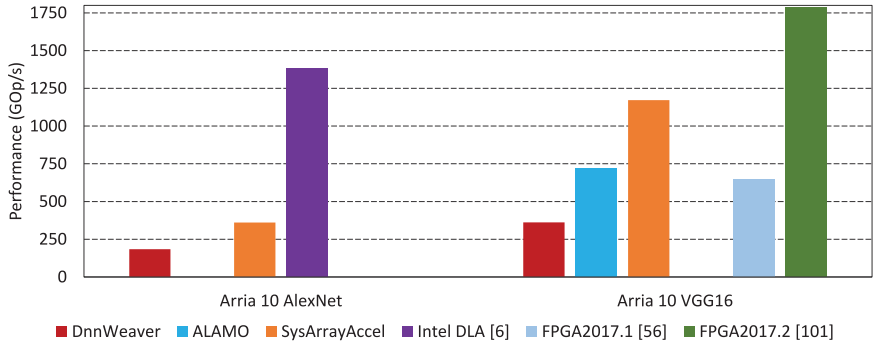

Fig. 6. Comparison on mapping AlexNet and VGG16 on Arria 10.

for the generation of latency-optimized accelerators leading to a lower latency for AlexNet's feature extractor on both Zynq XC7Z020 and XC7Z045. Snowflake has been optimized from both an architectural and a compiler level to sustain close to peak utilization of its compute resources and operate at the high clock frequency of 250MHz. Despite utilizing 3.5× fewer DSPs than DeepBurning, Snowflake demonstrates a 11% higher throughput on AlexNet by reaching a computational efficiency of 94%. Overall, the analytical design space exploration methods of fpgaConvNet and DnnWeaver, which provide a finer optimization of the hardware, proved to give a slight advantage over the brute-force mapping approach of DeepBurning.

By mapping VGG16 on Zynq XC7Z045, Angel-Eye outperforms the fpgaConvNet-generated design. By taking into account the clock frequency difference between the two designs, the clock frequency-normalized throughput ratio becomes 1×. Nevertheless, fpgaConvNet employs batch processing to achieve high throughput while Angel-Eye has been designed to co-optimize throughput and latency by operating with a batch size of 1. In this context, Angel-Eye achieves 1.52× lower raw latency and 1.27× lower clock frequency-normalized latency. The latency gap between the two frameworks comes potentially from the fact that Angel-Eye's parameter space, as presented in Section 2.4, enables the tool to exploit the parallelism across both the input and output feature maps of each CONV layer, by partially unrolling both dimensions to minimize latency. However, fpgaConvNet's architecture tunably unrolls only the output feature maps and applies pipelining to the input feature maps, which sets a higher bound on the latency.

For the mapping of LeNet-5 and CIFAR-10, Haddoc2 employs 3-bit and 6-bit representations for both weights and feature maps. The toolflow implements all its compute units in logic instead of DSPs, and hence such low-precision operands enable the instantiation of a large number of units and the extraction of high throughput from the target device, without being limited by the available number of DSPs. However, fpgaConvNet employs 16-bit representation and implements its operators solely using DSPs. Consequently, Haddoc2 outperforms fpgaConvNet in the particular set of benchmarks. Nevertheless, the requirement of Haddoc2 for all weights to be stored on-chip and the mapping of all CNN computations to dedicated units limits the maximum size of CNNs that can be mapped to a particular device and, in this respect, Haddoc2 can mainly support aggressively quantized networks.

FP-DNN has combined OpenCL-based control circuitry with a hand-crafted RTL computation engine to overcome the limitations of OpenCL-generated compute units and exploit its advantages in the handling of control and interfacing with the host and the external memory. FP-DNN's scope is limited to data-centre setups, which allows for server-specific assumptions and optimizations, such as PCIe-based communication. FP-DNN's highly optimized RTL Matrix Multiplication engine

together with the sophisticated on-chip buffer allocation scheduling method have demonstrated the ability to target large-scale CNNs, such as VGG19 and ResNet-152 with a limitation on the supported FPGA setup. However, DnnWeaver demonstrates a wider scope, including both embedded and server-based FPGAs across different FPGA vendors and hence generality is a higher priority. ALAMO maps its compute units to both DSPs and logic to extract higher throughput from the target device and employs RTL-level design to perform low-level optimizations. These properties have enabled ALAMO to outperform FP-DNN and DnnWeaver on the same CNN models. However, FFTCodeGen performs convolutions in the frequency domain and achieves higher throughput than other toolflows on the same CNN-FPGA pairs by exploiting the lower computational complexity of this approach. Nevertheless, batch processing of the inputs is required by FFTCodeGen to amortize the overhead of converting between the space and frequency domains, which sets a limit to the lowest attainable latency of the framework. Finally, SysArrayAccel employs its analytical performance and resource models to efficiently traverse the design space of systolic arrays and automatically configure tunable parameters at a finer grain than DnnWeaver. This leads to SysArrayAccel's higher performance on Arria 10.

Comparison with hand-tuned FPGA designs. ALAMO and FFTCodeGen map AlexNet and VGG16 on Stratix V GXA7. We compare it with the design by Suda et al. [81], which targets the same CNN-FPGA pairs (Figure 5). The design in Reference [81] employs a formal optimization formulation to configure an OpenCL-based accelerator that achieves 31.8GOp/s on AlexNet and 47.5GOp/s on VGG16. Compared to these designs, ALAMO achieves a speed-up of 3.6× on AlexNet and 7.41× on VGG16, with FFTCodeGen outperforming by 38.07× and 21.37×.

Despite the analytical design space exploration approach, Suda et al. use an off-the-shelf OpenCL matrix multiplication kernel for the CONV layers. ALAMO's RTL designs avoid the inefficiencies introduced by OpenCL and leave space for hardware optimizations at a lower level by mapping MACC units to both DSPs and logic. As a result, ALAMO trades off sophisticated design space exploration for highly optimized, RTL-based compute units to reach high performance. From a different perspective, the lower computational complexity of performing convolutions in the frequency domain and the generation of a highly optimized computation engine enable FFTCodeGen to outperform the OpenCL-based accelerator of Reference [81].

Figure 5 also presents a comparison between FP-DNN and DnnWeaver with the design of Suda et al. on Stratix V GSD8. With the particular device belonging to the same class as Stratix V GSD5, DSP-normalized metrics provide a meaningful comparison. Both FP-DNN and DnnWeaver demonstrate higher performance than Suda et al. due to their highly optimized RTL designs.

Figure 6 presents a throughput comparison of the mapping of AlexNet and VGG16 on Arria 10 between toolflows and state-of-the-art, hand-crafted designs. For DnnWeaver's and SysArrayAccel's AlexNet accelerators on Arria 10 GX115 and GT115, we compare with the state-of-the-art Deep Learning Accelerator (DLA) by Intel [6]. DLA achieves 1.382TFLOp/s on AlexNet, with DnnWeaver and SysArrayAccel reaching 13.26% and 26% of DLA's performance. DLA employs a series of strategies to increase the DSP utilization of Arria 10. These include the use of the Winograd transform to reduce the number of operations in convolutions and the design of a high-throughput 1D systolic array that operates at the high frequency of 303MHz. Although DLA outperforms both DnnWeaver and SysArrayAccel with respect to AlexNet's mapping on Arria 10, it does not include an automated design flow and operates under the assumption that all intermediate feature maps can be cached on-chip. This is an assumption that is valid for networks of comparable size to AlexNet, but it does not hold for larger-scale models, such as VGG16 and ResNet-152.

For the VGG16 designs of DnnWeaver and SysArrayAccel on Arria 10, we compare with the accelerators presented in References [56] and [101]. DnnWeaver achieves 56.03% and 20.20% of the

throughput of References [56] and [101], respectively. SysArrayAccel outperforms the throughput of References [56] by 1.81× with a 1.78× improved latency and reaches 65.43% of the throughput of Reference [101] with 1.56× degraded latency. The higher performance of Zhang et al. [101] comes due to the fact that the authors embedded RTL in OpenCL kernels to introduce register-level optimizations for particular networks, which enabled them to reach the high frequency of 385MHz and sustain high utilization of the FPGA's on-chip RAM. Despite achieving higher performance, such optimizations are hand-crafted and hence hard to exploit in other models in an automated manner.

For the AlexNet design of AutoCodeGen on Virtex 7 VX690T, we compare with the 565.94GOp/s accelerator presented in Reference [49]. AutoCodeGen achieves 222.1GOp/s and reaches 39% of the throughput of the highly optimized accelerator. The design in Reference [49] has achieved the current state-of-the-art performance of AlexNet on Virtex 7 VX690T, with manual optimizations for the particular CNN-FPGA pair. In this respect, despite achieving a lower raw throughput, AutoCodeGen is able to target a wider range of networks than Reference [49] by means of its automated flow.

At the time of writing, Angel-Eye's mapping of VGG16 on Zynq XC7Z045 and fpgaConvNet's mappings of AlexNet on Zynq XC7Z020 and XC7Z045 and VGG16 on Zynq XC7020 are the state-of-the-art designs for the particular CNN-FPGA pairs. Similarly, Caffeine is currently the highest performing design to target Xilinx UltraScale KU060 and therefore no meaningful comparison can be made with hand-tuned designs at the moment.

2.7 Discussion: Quality of Results

The limitations of the ad hoc benchmarking methodology of each toolflow do not allow us to draw conclusive and meaningful results on the comparative QoR of the generated accelerators. A uniform evaluation methodology that aims at surpassing the drawbacks of the current evaluation procedures for CNN-to-FPGA toolflows is proposed in Section 3.1. For the toolflows that target the same CNN-FPGA pairs, the following observations are made.

Toolflows that generate highly optimized RTL-based designs tend to outperform their HLS counterparts. This property can be observed in the comparison of ALAMO with the accelerator by Suda et al. [81]. Although Suda et al. employed a more sophisticated DSE method, ALAMO trades off a complex DSE to a detailed, RTL-level optimization of its hardware design and outperforms by 3.6×. A similar trend is observed between FP-DNN with DnnWeaver. Despite the fact that both toolflows generate RTL designs, FP-DNN focuses more on the low-level RTL optimization of its computation engine and manages to achieve higher throughput, despite not performing the extensive design space exploration of DnnWeaver. The highest performing VGG16 accelerator on Arria 10 by Zhang et al. [101] provides additional evidence. The design in Reference [101] embeds custom RTL-level optimizations in OpenCL kernels to boost the performance and avoid the current limitations of the OpenCL programming model. In this way, Reference [101] achieves higher performance than the CNN-to-FPGA toolflows. In spite of this advantage, the manual, hand-crafted RTL design that is required to achieve this level of performance prohibits the automation that is essential for a toolflow and therefore a trade-off between RTL performance and HLS productivity is necessary.

Design space exploration methods that allow for finer customization tend to offer an advantage in terms of QoRs. fpgaConvNet's analytical methodology outperformed DnnWeaver's slightly more restricted design space, which in turn outperformed DeepBurning's heuristic mapping. Moreover, SysArrayAccel's detailed design-space-exploration method enabled the traversal of a larger design space than DnnWeaver, which led to higher performing designs. Similarly, the CaP

technique (Section 2.3) introduced by FFTCodeGen added another level of customization and enabled to full exploitation of the FFT-based convolution by sustaining a high utilization of the generated accelerator across CONV layers of different sizes.

Single computation engine architectures tend to reach high performance on CNNs with a uniform structure. This property can be observed in the case of the increase in the throughput of DnnWeaver and SysArrayAccel when mapping VGG16 compared to AlexNet. The single computation engine of both toolflows manages to sustain a very high utilization across the layers of VGG16 due to the uniform kernel size of the CONV layers and the power of 2 number of input and output feature maps after the first CONV layer. In contrast, the variable kernel sizes of AlexNet, including 11×11, 5×5, and 3×3, lead to an underutilization of the shared computation engine. Moreover, Angel-Eye's mapping of VGG16 on Zynq XC7Z045 is also benefited by the uniformity of VGG16 and reaches the highest reported raw performance for VGG16 on the particular device. Nevertheless, FFTCodeGen is not affected by the irregularity in the kernel sizes of AlexNet due to its tiled FFT-based algorithm and hence its throughput is not deteriorated.

2.8 Discussion: Suitability for Deep-Learning Application Challenges

CNNs have been successfully employed in a variety of problem domains, including video surveillance [74], healthcare [17], and autonomous transportation [11]. Depending on the nature of the domain, CNNs have to be deployed on processing platforms with different constraints and compute capabilities, spanning from server-grade setups in a data centre [9] to low-power devices on the edge [79]. Moreover, the variability of applications requires from the CNN implementations to comply with diverse performance requirements, from the high-throughput needs of large-scale cloud-based services to the critical low-latency requirements of autonomous drones and cars, with low power consumption standing as a ubiquitous requirement. In this context, the different design approaches of the existing CNN-to-FPGA toolflows determine their suitability to particular use cases in the deep-learning application landscape.

Table 6 summarizes the features of each toolflow and the chart in Figure 7 depicts the effect of each toolflow's strategic design decisions on a number of aspects. Finn trades off very high throughput and low latency by restricting its focus on the fine niche of binarized neural networks. The toolflow's Vivado HLS-based accelerators can target only Xilinx devices, but with enough infrastructure to target both embedded SoCs and standalone devices. These properties make Finn-generated designs to be lightweight and applicable to both throughput-driven and latency-sensitive applications, that also exhibit high error tolerance, due to the potential impact of binarization on the accuracy.

DnnWeaver places FPGA compatibility and portability as a priority and bases its internal design on device-independent, RTL-based templates. The toolflow's infrastructure and template-level parameterization allow for variable precision along the CNN layers and has demonstrated the widest support for SoCs, standalone, and server-grade FPGAs from different vendors. Its application scope is restricted to high-throughput applications with large batch sizes, without special consideration for low-latency requirements, which restricts the toolflow's supported optimization objectives.

FP-DNN places emphasis on the low-level optimization of a computation engine that would support different types of NN models. The toolflow restricts its scope to cloud-based environments with Intel FPGAs and is optimized for the high-throughput workloads of data centres. The toolflow adopts uniform quantization across the CNN layers as specified by the user.

Caffeine and SysArrayAccel concentrated on the optimization of systolic array structures for high-throughput CNNs. The two toolflows are restricted to Xilinx and Intel FPGAs due to their

Table 6. Summary of Toolflow Characteristics

Framework Name	Interface	NN models	Devices	Architecture	Precision*	DSE
fpgaConvNet [85–88]	Caffe & Torch	CNN,Res,Incep,Dense	Xilinx SoC	Streaming	FXP (Uniform) & FP	Global Optimizer (Simulated Annealing)
DeepBurning [90]	Caffe	CNN,RNN,DNN	Xilinx SoC	Streaming	FXP (Dynamic)	Heuristic
Angel-Eye [23, 24, 68]	Caffe	CNN,DNN	Xilinx SoC	Single-Engine	FXP (Dynamic)	Heuristic with Analytical Model
ALAMO [55–59]	Caffe	CNN,DNN	Intel SoC & Standalone	Single-Engine	FXP (Dynamic)	Heuristic
Haddoc2 [1, 2]	Caffe	CNN,DNN	Xilinx & Intel Standalone	Streaming	FXP (Uniform)	Deterministic
DnnWeaver [75, 76]	Caffe	CNN,DNN	Xilinx & Intel	Single-Engine	FXP (Dynamic)	Custom Search Algorithm
Caffeine [98]	Caffe	CNN,DNN	Xilinx Standalone	Single-Engine	FXP (Uniform) & FP	Exhaustive over Roofline Model
AutoCodeGen [54]	Proprietary Input	CNN,DNN	Xilinx Standalone	Streaming	FXP (Dynamic)	Heuristic with Analytical Model
Finn [19, 84]	Theano	BNN	Xilinx SoC & Standalone	Streaming	Binary	Heuristic
FP-DNN [22]	TensorFlow	CNN,RNN,DNN,Res	Intel Standalone	Single-Engine	FXP (Uniform) & FP	Algorithmic
Snowflake [10, 21]	Torch	CNN,Res,Incep	Xilinx SoC	Single-Engine	16-bit FXP (Uniform)	Heuristic
SysArrayAccel [91]	C Program	CNN,DNN	Intel Standalone	Single-Engine	FXP (Uniform) & FP	Exhaustive over Analytical Model
FFTCodeGen [95–97, 100]	Proprietary Input	CNN,DNN	Intel HARP	Single-Engine	FXP (Uniform) & FP	Roofline and Analytical Models

* FXP: Fixed-Point, FP: Floating-Point.

use of Vivado HLS and OpenCL, respectively. SysArrayAccel draws from the lessons learned from Caffeine's design and its finer design space exploration method tends to yield higher performance. However, the different target devices of the two toolflows do not allow for a meaningful performance comparison.

ALAMO is designed to combine the high throughput and low latency of RTL designs with high precision flexibility across layers. Nevertheless, precision quantization is performed manually and is not part of the automated flow. The toolflow uses Intel's off-the-shelf IPs, including the NIOS soft processor and the scatter-gather DMA block, and hence the generated designs are restricted and tailored for Intel FPGAs, which affects its portability across vendors.

Angel-Eye bases its competitive advantage on its automatic dynamic quantization scheme. The selected parameter space allows for the unrolling of both the input and output feature maps and hence latency and throughput are co-optimized.

fpgaConvNet prioritizes the support of various optimization objectives based on the application-level performance needs to target diverse workloads. In this context, distinct methodologies are used for high-throughput, low-latency or multiobjective applications. Similarly to Caffeine and Angel-Eye, the use of Vivado HLS currently restricts fpgaConvNet to Xilinx devices.

DeepBurning's design principle entails modularity and support of a wide range of NN models. In this respect, the toolflow's RTL building blocks can target various types of NNs, including the emerging RNNs and LSTMs, and offer flexibility with respect to precision quantization across the layers. By design, the generated accelerators are optimized to operate with a batch size of

Fig. 7. Overview of toolflow characteristics.

1, and hence their optimization objectives are simultaneously high throughput and low latency. Similarly, AutoCodeGen also places focus on the modular design of RTL hardware blocks for high throughput, with a more restricted scope than DeepBurning by supporting only CNN models and with the addition of high-level performance and resource modeling.

HADDOC2 follows a direct mapping approach, where all layers and neurons in a network are mapped deterministically to dedicated hardware resources. This approach enables achieving both high throughput and low latency in the cases where all weights of the target CNN can be accommodated by the on-chip memory resources and enough logic is available to map all operators. This assumption holds in the case of small-scale networks, such as LeNet-5 and CIFAR-10, and aggressively quantized models, such as the BNNs targeted by FINN, but HADDOC2 cannot currently handle the state-of-the-art large-scale models, due to the lack of tunable time-sharing mechanisms.

Snowflake's design principle places programmability and high utilization of the computational resources at the forefront. In this respect, both Snowflake's architecture and compiler are tailored to removing inefficiencies and extracting close to peak performance from the allocated resources. Overall, Snowflake favors programmability over hardware specialization, by employing a fixed hardware design and customizing with respect to the target model only at the compiler level.

Finally, FFTCodeGen addresses CNN acceleration from both an algorithmic and an architectural level. In contrast to the rest of the toolflows, convolutions are performed in the frequency domain with a significantly lower computational complexity. Moreover, the free parameters of the algorithm and the architecture enable the generated compute engine to sustain high throughput across convolutional layers of different sizes and fully exploit the computational complexity gains. Furthermore, the use of the powerful, server-grade CPU of the target Intel HARP platform alleviates the complexities of mapping the memory-bounded fully-connected layers to hardware and further contributes to FFTCodeGen's throughput gains, making it suitable for throughput-driven cloud-based applications.

Table 7. Benchmark Suite: CNN Models and Their Computational Challenges

Model Name		Year	Depth	Design Principles	Challenges
AlexNet	[46]	2012	8	(1) Increased depth (2) Increased layer width	(1) Non-uniform filter sizes (2) Grouped convolutions
ZFNet	[94]	2013	8	(1) Wider layers	(1) Computational load (2) Memory footprint
VGG16	[78]	2014	16	(1) Increased layer depth (2) Uniform filter size	(1) Computational load (2) Memory footprint
GoogLeNet	[83]	2014	22	(1) Inception module	(1) Irregular computations (2) Irregular layer connectivity
ResNet-152	[33]	2015	152	(1) Residual block	(1) Irregular computations (2) Irregular layer connectivity
Inception-v4	[82]	2016	72	(1) Residual Inception block	(1) Irregular computations (2) Irregular layer connectivity
DenseNet-161	[36]	2017	161	(1) Dense block	(1) Irregular computations (2) Irregular layer connectivity

2.9 Other Related Work

Apart from the presented toolflows, several FPGA-based designs for CNNs have been proposed by the FPGA community. These include highly optimized, hand-tuned accelerators for particular CNN-FPGA pairs in RTL [16, 18, 49], HLS [6, 44], and mixed RTL-HLS [101], together with designs that focus on optimizing the external memory bandwidth utilization [5, 77]. A number of existing works lie close to the presented CNN-to-FPGA toolflows, but lack essential components that would form a complete automated flow. These include References [15, 61, 62, 81], with References [61, 62, 81] focusing on the design space exploration task and Reference [15] presenting an FPGA back end to Caffe, for the execution of 3×3 convolutional layers by means of the Winograd transform.

3 THE FUTURE OF CNN-TO-FPGA TOOLFLOWS

3.1 Toward a Uniform Evaluation Methodology

The existing FPGA toolflows have employed ad hoc evaluation methodologies, by targeting different CNN models and reporting the achieved performance in a non-uniform manner. A uniform evaluation methodology is proposed here to enable the thorough and comparative evaluation of CNN-to-FPGA toolflows. The proposed methodology comprises a benchmark suite and guidelines for evaluation metrics.

Benchmark Suite.[22] A comprehensive benchmark suite should include CNNs that are widely used and whose accuracy has been extensively studied by the deep-learning community. Each CNN should pose a unique hardware mapping challenge and stress the FPGA toolflow from a different aspect. The main challenges to be addressed include CNNs that are (1) computation bounded, (2) off-chip memory bandwidth bounded, (3) on-chip memory capacity bounded, (4) with high layer dependency, and (5) irregular and sparse layer connectivity that challenges scheduling.

To this end, we propose a benchmark suite with the following CNN models (Table 7): AlexNet, ZFNet, VGG16, GoogLeNet, ResNet-152, Inception-v4, and DenseNet-161. AlexNet was the winner of the 2012 ILSVRC competition and its pretrained feature extractor is widely used as a starting point for new applications [70], making it a fundamental CNN model in the deep-learning community. AlexNet's mapping challenge lies in the non-uniform filter and stride sizes across its

[22]The proposed benchmark suite of representative CNNs can be found at http://www.imperial.ac.uk/intelligent-digital-systems/cnn-benchmark-suite/.

convolutional layers, including 11×11, 5×5, and 3×3 kernels, which can stress the utilization of convolution engines and assess a toolflow's mapping capabilities. As an example, the Angel-Eye computation engine is currently tailored to 3×3 kernels and optimized for the VGG16 network, which has a uniform filter size. In this case, AlexNet could evaluate Angel-Eye's efficiency of mapping CONV layers that do not have 3×3 kernels. Moreover, fpgaConvNet derives a single streaming architecture for low-latency designs. The mapping of AlexNet would examine the quality of the derived streaming architecture of fpgaConvNet. In a similar manner, ZFNet also has non-uniform filter sizes, consisting of 7×7, 5×5, and 3×3 kernels, and wider CONV layers than AlexNet, while being used as a starting template for novel applications [72]. Both networks can lead to the exposure of finer strengths and weaknesses and indicate potential room for improvements in the toolflows.

VGG16 is a substantially deep model with high computation and memory requirements and constitutes one of the most widely employed pretrained models for new applications [7]. Due to its large computational load, number of weights and layers, it is proposed as a representative neural network that poses challenges (1), (2), (3), and (4). Because of these challenges, the majority of existing tools have already evaluated their design flows on VGG16.

To challenge the CNN-to-FPGA frameworks with irregular and sparse computations, the benchmark suite includes GoogLeNet, ResNet-152, Inception-v4, and DenseNet-161. GoogLeNet introduced the Inception module that made the CNN topology more complex than conventional CNNs by breaking the uniform layer connectivity. ResNet-152 introduced a residual block that allows for a forward connection that bypasses intermediate layers and enabled the construction of a 152-layered network. Inception-v4 increases the CNN complexity by combining both concepts from GoogLeNet and ResNet-152 to achieve higher performance. Despite the higher performance of Inception-v4, pretrained versions of GoogLeNet and ResNet-152 are still widely used. Finally, DenseNet-161 presented a dense block that enables the output of each layer to be directly connected to the input of every following layer. This type of networks would provide a thorough evaluation of the CNN-to-FPGA toolflows. FP-DNN, ALAMO, and fpgaConvNet have already demonstrated their performance when targeting ResNet-152, with Snowflake targeting ResNet-50 and GoogLeNet. Since FP-DNN consists of a single Matrix Multiplication engine, the ResNet-152 mapping reduced to the problem of scheduling the different layers given their irregular connectivity. In a similar manner, Snowflake's compiler breaks down residual blocks and Inception modules into MACC operations and schedules them over Snowflake's accelerator. However, ALAMO and fpgaConvNet followed a different approach by enhancing their architectures with specialized blocks to handle irregular networks. ALAMO designed and integrated a dedicated elementwise addition block in its computation engine to support the residual connections of ResNets. fpgaConvNet has introduced three specialized streaming hardware blocks, tailored for the Inception module and the residual and dense blocks. Overall, toolflows that generate streaming architectures would have to cope with not breaking the streaming principle of operation and demonstrate how their mapping and scheduling methods can compare with the more flexible, single computation engine designs.

Evaluation metrics. Evaluation metrics aim to characterize the quality of a toolflow's generated results and highlight the various strengths and weaknesses. These metrics should include essential attributes such as performance including throughput and latency, resource consumption, power efficiency and application-level accuracy. Reporting all these criteria play an important role in determining the strategic trade-offs made by a toolflow.

In terms of *performance*, the most commonly reported metrics are currently affected by two limitations: (1) normalized quantities such as GOp/s/Logic and GOp/s/DSP attempt to indicate the quality of the generated design solely as a measure of computational resource utilization. This

approach does not capture the available bandwidth and capacity of the off- and on-chip memory, which can have a decisive effect on performance; (2) normalizing with a resource that is FPGA family-specific, such as Xilinx's LUTs or Intel's ALMs, does not enable the fair comparison across different vendors and across FPGA families from the same vendor. As a characteristic example, we point to the Stratix V devices targeted by FP-DNN and ALAMO, namely Stratix V GSD5 and GXA7. Despite the fact that both devices belong to the Stratix V family, GSD5 belongs to the GS FPGAs, which are optimized for DSP-focused applications with an abundance of MACC operations and hence contains 6.2× more DSPs, smaller on-chip memory and fewer ALMs, while GXA7 belongs to the GX FPGAs, which are optimized for high-bandwidth applications and hence offers a higher bandwidth interface to the off-chip memory, 1.14× larger on-chip memory and 1.36× more ALMs. As a result, a single resource-normalized metric such as DSP- or logic-normalized performance is not able to capture the quality of the generated hardware across devices that are optimized for different application domains.

Throughput is the primary performance metric of interest in throughput-driven applications such as high-throughput image recognition and large-scale, multi-user analytics services over large amounts of data. Throughput is measured in GOp/s and is often achieved by processing large batches of inputs. *Latency*, or *response time*, becomes the primary critical factor in latency-sensitive applications, such as self-driving cars and autonomous systems, but also in particular real-time cloud-based services. Measured in seconds, latency is the time between when an input enters the computing system and when the output is produced. In such scenarios, batch processing adds a prohibitive latency overhead and is often not an option. Different toolflows choose to either optimize for one of the two metrics, co-optimize them simultaneously by using a batch size of 1 or selectively optimize for one of the two based on the application's performance requirements.

Resource consumption is an indicator of the efficiency of the utilization of the available resources on the target platform by the designs generated by a toolflow, including the DSPs, on-chip RAM, logic, and FFs. *Application-level accuracy* is a crucial metric when approximation techniques are employed by a toolflow for the efficient mapping of CNNs. Such techniques may include precision optimization, such as the dynamic precision quantization scheme by Angel-Eye, or lossy compression methods, such as the SVD-based compression applied on the weights of the FC layers by Angel-Eye, and can have an impact on the application-level accuracy of the CNN. Potential performance-accuracy trade-offs have to be quantified and reported in terms of accuracy degradation.

To measure the quality of a CNN-to-FPGA toolflow, we propose the following methodology. Throughput in GOp/s with explicitly specified GOp/network, amount of weights and batch sizes, and latency in seconds/input with batch size of 1, to present the throughput-latency relationship, should be included in the evaluation reports. Resource-normalized metrics are meaningful when comparing designs that target devices from the same FPGA family optimized for the same application domain. In this scenario, performance normalized with respect to logic and DSPs would allow the comparison of hardware designs for the same network on FPGAs of the same family. Power-normalized throughput and latency should also be reported for comparison with other parallel architectures such as CPUs, GPUs, and DSPs. Since resource-normalized performance does not capture the effect of off- and on-chip memory bandwidth and capacity despite being critical for achieving high performance, target FPGA platform details should be included that explicitly indicate the off- and on-chip memory specifications. All measurements should be made for various CNNs, with emphasis on the proposed benchmark suite of the previous section, to demonstrate the QoRs subject to the different mapping challenges posed by each benchmark model.

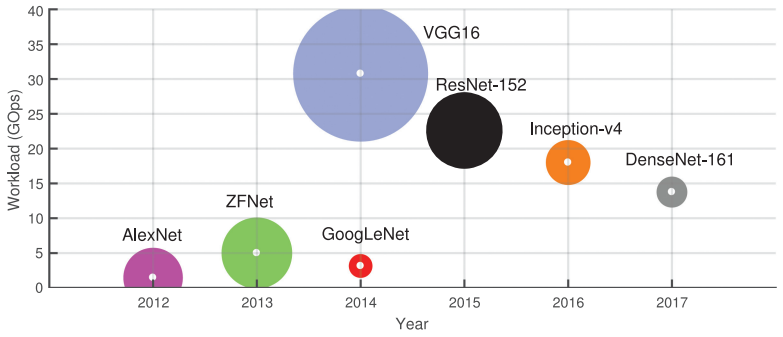

Fig. 8. CNN computation and memory evolution (circle size is relative to the amount of weights).

3.2 Objectives of a CNN-to-FPGA Toolflow

Recent developments in deep learning have led to new challenges in the deployment of CNNs. This section presents a set of objectives for CNN-to-FPGA toolflows based on recent research trends.

Objective 1. Targeting next-generation CNN models. Since AlexNet's win in the 2012 ILSVRC competition, a number of CNN models [33, 36, 78, 82, 83, 94] paved the way for the state-of-the-art accuracy in visual tasks. Typically, improvements in accuracy have been achieved at the expense of increased complexity in the structure of the CNN. In Table 7, which lists a number of representative models together with the source of challenge in their design (also visualized in Figure 8), three main trends are identified in CNN design: (1) the increase in the depth of the models, from the 8-layer AlexNet up to the 152-layer ResNet-152 and the 161-layer DenseNet-161, (2) the increased inference workload, with an increase of 20× from AlexNet to VGG16 in GOps/input, and (3) the introduction of novel compound components. With respect to trend (3), networks such as GoogLeNet, ResNet-152, Inception-v4, and DenseNet-161 enhanced the CNN layers with the introduction of complex blocks, as indicated in Table 7. This type of complex blocks break the uniform connectivity and computation pattern of conventional CNNs with irregular layer connections and challenge the automation of their mapping to hardware. Currently, ALAMO, Snowflake, FP-DNN, and fpgaConvNet provide optimized support for residual blocks in networks. Moreover, Snowflake and fpgaConvNet also target networks that use Inception modules, such as GoogLeNet. Finally, fpgaConvNet supports dense blocks by means of a specialized hardware building block, tailored to dense block structures. Nevertheless, with such compound components becoming mainstream in the deep-learning literature, CNN-to-FPGA toolflows ought to investigate further the optimization opportunities of their mapping to optimized hardware.

Beyond the spatial pattern recognition abilities of CNNs, Recurrent Neural Networks (RNNs) enhance NNs with the ability to learn data sequences by retaining memory [60]. While this broadens the application field of neural networks, additional recurrent connections between layers introduce further challenges in the parallelization of computations. RNN models, with the prominence of LSTM networks [34], demonstrate state-of-the art accuracy in applications that require capturing long-range dependencies by processing information from past inputs [89]. Thus, designing optimized hardware units that support network architectures with such recurrence in connections between layers is becoming an increasingly important feature for FPGA toolflows.

In contrast to the computation-bounded CNNs, RNNs and LSTMs comprise inherently memory-bounded workloads due to the large number of matrix-vector multiplications. This property necessitates a different design approach for their optimized mapping to hardware. At the moment, DeepBurning and FP-DNN offer support for RNNs, with FP-DNN also targeting LSTMs. In the

same direction, the authors of Angel-Eye together with DeePhi[23] have proposed an LSTM accelerator [28], but it has not been integrated into Angel-Eye. The FPGA community has further proposed designs targeting from high-throughput data-centre services [63] to latency-critical embedded setups [73]. Moreover, industrial companies such as Google [42] and Microsoft[24] report that a large fraction of their data-centre workloads are LSTM-based and have focused their efforts on optimizing the execution of LSTMs with customized ASIC and FPGA designs, respectively. In this context, along with the automated mapping of CNNs, end-to-end frameworks that would focus on the high-performance deployment of RNNs and LSTMs on FPGAs emerges as an essential objective.

Objective 2. Support of compressed and sparse CNNs. Recent work from the deep-learning community has proposed techniques of reducing the inference time of CNNs, by exploiting the redundancy across its trainable parameters. The existing approaches can be divided into (1) post-training and (2) training-time methods. Post-training methods assume fully trained CNNs and add a preprocessing step prior to deployment. Works such as References [14, 41] focus on decreasing the computational cost of the computation-bounded CONV layers by means of the low-rank decomposition of filters. However, works such as References [30, 50] focus on minimizing the excessive memory footprint of the memory-bounded FC layers, by projecting the weights matrix to a lower-dimensional space. Training-time methods attempt to create sparse CNNs by means of pruning or sparsity regularization techniques during the training phase [31, 52]. Although the sparsification of CNNs can reduce the theoretical computational and memory costs, the elimination of weights and connections between layers breaks the uniformity of computation and memory accesses, and hence requires a rethinking of the hardware mapping. At the moment, a few ASIC designs have been proposed to tackle the challenges of compressed and sparse networks [29, 65, 102]. In this context, there is an emerging need for the CNN-to-FPGA tools to support compressed and sparse networks to offer competitive high-performance, low-power alternatives to the existing CPU, GPU, and DSP platforms.

Objective 3. Support of low-precision and extremely low-precision CNNs. The robustness of CNNs to low-precision quantization of weights and feature maps at the inference stage has been widely studied [25, 26, 32, 35, 105]. At the moment, the majority of CNN-to-FPGA toolflows support either uniform or dynamic quantization across layers, depending on whether the wordlengths and scaling at each layer are the same. Angel-Eye, ALAMO, DnnWeaver, DeepBurning, and AutoCodeGen support dynamic quantization, with a fixed, uniform wordlength and different scaling across layers. The focus on quantization has been taken a step further by Angel-Eye, which employs an automated quantization method to automatically determine the scaling for each layer of the target network, given a fixed wordlength. Nevertheless, existing works have been investigating more irregular quantization schemes, with the ASIC designs of References [3, 43] varying both the wordlength and scaling of each layer of the network and mapping the variable-wordlength computations on optimized bit-serial arithmetic units, and Intel Nervana proposing a custom floating-point variant format [45]. The robustness of CNNs to quantization offers CNN-to-FPGA toolflows the opportunity to explore and integrate automatic quantization methodologies as part of their design flows. Adding precision quantization as a design dimension can offer more room for customization in the architectural design space, provide a closer coupling of network and hardware design, and offer more room for improvement over CPU, GPU, and DSP counterparts that cannot benefit from this type of fine-grained data representation optimizations.

[23] http://www.deephi.com/.
[24] https://www.microsoft.com/en-us/research/blog/microsoft-unveils-project-brainwave/.

In the same context, Ternary [4, 107] and Binary [37, 69] CNNs form extreme—but widely studied—cases of low-precision CNNs. It has been demonstrated that the accuracy degradation introduced by training popular CNN models on that level of precision (using 1- or 2-bit weights and feature maps) does not have a considerable effect for several real-life applications, while their reduced memory footprint offers significant space for acceleration with customized hardware. In this context, FINN was the first FPGA framework to undertake the challenge of optimizing hardware units for BNNs. With more studies on the optimization of FPGA-based BNNs [51, 103] and ternary networks [67], the automated optimized mapping of binary and ternary operations can offer FPGAs a competitive advantage over competing platforms that cannot be customized to efficiently support such operations.

Objective 4. Integration with the existing deep-learning infrastructure. So far, Caffe has been the best-supported framework by CNN-to-FPGA automated tools, as discussed in Section 2.2.1. However, other interfaces such as TensorFlow by Google seem to gain attention by the academic and industrial communities because of the wide variety of supported machine-learning models and the provided flexibility for deployment across different heterogeneous systems. While FP-DNN provides back-end support for TensorFlow, building the necessary infrastructure for newer deep-learning frameworks, such as MXNet, PyTorch, Caffe2, CoreML, and CNTK, and developing methodologies that can efficiently process the Intermediate Representation (IR) of each framework to yield optimized FPGA mappings can constitute a critical factor in exposing the deployment of CNNs on FPGAs to the wide community of deep-learning researchers and practitioners.

As a recent example toward this direction, Xilinx introduced reVISION,[25] a resource suite that allows rapid development of highly responsive embedded-vision reconfigurable systems, through a software-level design flow. The reVISION stack enables the combination of machine-learning and computer vision algorithms with reconfigurable devices, while allowing sensor fusion and high-level connectivity and supporting standard frameworks such as Caffe for application development.

Objective 5. FPGA-based CNN training. In both academic and industrial work, GPUs constitute the main computing platform for the acceleration of the training task. Big industrial companies such as Facebook and Baidu typically employ GPU-powered clusters, situated in data centres, to handle model training. For data centres, the power and cooling infrastructure costs constitute one of the most critical factors of the operational expenses. Since GPUs provide high performance at the expense of high power consumption, they become costly platforms to maintain. This fact has led Google to design and deploy the Tensor Processing Unit (TPU) ASIC [42] in its servers for the training and inference stage of machine-learning models. With next-generation FPGAs achieving promising performance and power efficiency [64], FPGAs can provide a high-performance, low-power alternative back end for the training task. To this end, big industrial companies such as Microsoft [9] and Amazon[26] have modified their data centre facilities to host FPGAs and offer opportunities for the training of neural network models using FPGAs. Moreover, recent advances in low-precision neural network training [12, 13, 38, 92, 106] offer room for customization and variable-precision arithmetic that suits FPGA-based computing and cannot be efficiently exploited by conventional programmable platforms. At the moment, FPGA-based CNN training has only slightly been explored [66, 104] with a lot of space for further investigation.

Objective 6. Hardware-Network co-design. Ideally, a fully automated CNN framework would provide an end-to-end toolchain. Starting from a user-specified dataset and a target application, the

[25] https://www.xilinx.com/revision.
[26] https://aws.amazon.com/ec2/instance-types/f1/.

tool would start by analysing the data and proposing an initial neural network model. By including the hardware performance and power consumption as metrics in the training phase, the hardware tunable parameters and the model weights and topology would be jointly modified during the optimization process to co-optimize both the application-level accuracy and the required inference execution time and power consumption. Such a methodology would encompass the algorithmic model design together with the generation of efficient hardware under a holistic view that could potentially close the loop between CNN design and implementation. We envision frameworks that would provide this functionality as a long-term objective for the community to make steps toward the efficient hardware execution of high-performing neural networks.

4 CONCLUSION

This article presents a survey of CNN-to-FPGA toolflows. A comparative analysis of their main characteristics and features indicates the strengths and weaknesses of each toolflow together with its mapping techniques. The non-uniform evaluation methodologies that have been employed so far introduce limitations in the comparison between the toolflows and fall short of taking into account both the computational and memory resources of the target FPGA platform. To this end, a comprehensive benchmark suite and thorough evaluation metrics are proposed to lead to further and more rapid developments in CNN-to-FPGA toolflows. Moreover, based on recent developments in deep learning, promising research directions and future objectives are identified to address the emerging challenges of the field, exploit FPGA-specific performance optimizations, and enhance the accessibility of FPGAs to the wide community of deep learning.

REFERENCES

[1] Kamel Abdelouahab, Cédric Bourrasset, Maxime Pelcat, François Berry, Jean-Charles Quinton, and Jocelyn Serot. 2016. A holistic approach for optimizing DSP block utilization of a CNN implementation on FPGA. In *Proceedings of the 10th International Conference on Distributed Smart Camera (ICDSC'16)*. ACM, New York, NY, 69–75.

[2] K. Abdelouahab, M. Pelcat, J. Sérot, C. Bourrasset, and F. Berry. 2017. Tactics to directly map CNN graphs on embedded FPGAs. *IEEE Embed. Syst. Lett.* 9, 4, 113–116.

[3] Jorge Albericio et al. 2017. Bit-pragmatic deep neural network computing. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'17)*. ACM, New York, NY, 382–394.

[4] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot. 2017. Ternary neural networks for resource-efficient AI applications. In *Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN'17)*. 2547–2554.

[5] M. Alwani, H. Chen, M. Ferdman, and P. Milder. 2016. Fused-layer CNN accelerators. In *Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*. 1–12.

[6] Utku Aydonat, Shane O'Connell, Davor Capalija, Andrew C. Ling, and Gordon R. Chiu. 2017. An OpenCL deep learning accelerator on Arria 10. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, New York, NY, 55–64.

[7] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. 2017. SegNet: A deep convolutional encoder-decoder architecture for scene segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 12, 2481–2495.

[8] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Tomasz Czajkowski, Stephen D. Brown, and Jason H. Anderson. 2013. LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems. *ACM Trans. Embed. Comput. Syst.* 13, 2, Article 24, 27 pages.

[9] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J. Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger. 2016. A cloud-scale acceleration architecture. In *Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*. 1–13.

[10] Andre Xian Ming Chang, Aliasger Zaidy, Vinayak Gokhale, and Eugenio Culurciello. 2017. Compiling deep learning models for custom hardware accelerators. arXiv:1708.00117.

[11] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. 2015. DeepDriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV'15)*. IEEE, Los Alamitos, CA, 2722–2730.

[12] X. Chen, X. Hu, H. Zhou, and N. Xu. 2017. FxpNet: Training a deep convolutional neural network in fixed-point representation. In *Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN'17)*. 2494–2501.

[13] Christopher De Sa, Matthew Feldman, Christopher Ré, and Kunle Olukotun. 2017. Understanding and optimizing asynchronous low-precision stochastic gradient descent. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA'17)*. ACM, New York, NY, 561–574.

[14] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS'14)*. 1269–1277.

[15] R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor, and S. Areibi. 2016. Caffeinated FPGAs: FPGA framework for convolutional neural networks. In *Proceedings of the 2016 International Conference on Field-Programmable Technology (FPT'16)*. 265–268.

[16] Aysegul Dundar, Jonghoon Jin, Berin Martini, and Eugenio Culurciello. 2017. Embedded streaming deep neural networks accelerator with applications. *IEEE Trans. Neural Netw. Learn. Syst.* 28, 7, 1572–1583.

[17] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. 2017. Dermatologist-level classification of skin cancer with deep neural networks. *Nature* 542, 115–118.

[18] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello. 2010. Hardware accelerated convolutional neural networks for synthetic vision systems. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. 257–260.

[19] Nicholas J. Fraser, Yaman Umuroglu, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. Scaling binarized neural networks on reconfigurable logic. In *Proceedings of the 8th Workshop and the 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM'17)*. ACM, New York, NY, 25–30.

[20] D. Gandhi, L. Pinto, and A. Gupta. 2017. Learning to fly by crashing. In *Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'17)*. 3948–3955.

[21] Vinayak Gokhale, Aliasger Zaidy, Andre Xian Ming Chang, and Eugenio Culurciello. 2017. Snowflake: An efficient hardware accelerator for convolutional neural networks. In *Proceedings of the 2017 IEEE International Symposium on Circuits and Systems (ISCAS'17)*.

[22] Yijin Guan, Hao Liang, Ningyi Xu, Wenqiang Wang, Shaoshuai Shi, Xi Chen, Guangyu Sun, Wei Zhang, and Jason Cong. 2017. FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates. In *Proceedings of the 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'17)*. 152–159.

[23] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Song Yao, Song Han, Yu Wang, and Huazhong Yang. 2016. Angel-Eye: A complete design flow for mapping CNN onto customized hardware. In *Proceedings of the 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI'16)*. 24–29.

[24] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang. 2018. Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 37, 1, 35–47.

[25] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*. 1737–1746.

[26] Philipp Gysel, Mohammad Motamedi, and Soheil Ghiasi. 2016. Hardware-oriented approximation of convolutional neural networks. In *Proceedings of the Workshop Contribution at International Conference on Learning Representations (ICLR'16)*.

[27] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. 2010. Understanding sources of inefficiency in general-purpose chips. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*. ACM, New York, NY, 37–47.

[28] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, Huazhong Yang, and William (Bill) J. Dally. 2017. ESE: Efficient speech recognition engine with sparse LSTM on FPGA. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, New York, NY, 75–84.

[29] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA'16)*. IEEE, Los Alamitos, CA, 243–254.

[30] Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. In *Proceedings of the International Conference on Learning Representations (ICLR'16)*.

[31] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS'15)*. 1135–1143.

[32] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda. 2017. Understanding the impact of precision quantization on the accuracy and energy of neural networks. In *Proceedings of the Design, Automation, and Test in Europe Conference Exhibition (DATE'17)*. 1474–1479.

[33] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*. 770–778.

[34] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9, 8, 1735–1780.

[35] J. L. Holi and J. N. Hwang. 1993. Finite precision error analysis of neural network hardware implementations. *IEEE Trans. Comput.* 42, 3, 281–290.

[36] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*. 2261–2269.

[37] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in Neural Information Processing Systems 29*. 4107–4115.

[38] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Quantized neural networks: Training neural networks with low precision weights and activations. arXiv:1609.07061.

[39] G. Inggs, S. Fleming, D. Thomas, and W. Luk. 2014. Is high level synthesis ready for business? A computational finance case study. In *Proceedings of the 2014 International Conference on Field-Programmable Technology (FPT'14)*. 12–19.

[40] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*. 448–456.

[41] M. Jaderberg, A. Vedaldi, and A. Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference (BMVC'14)*.

[42] Norman P. Jouppi et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA'17)*. ACM, New York, NY, 1–12.

[43] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos. 2016. Stripes: Bit-serial deep neural network computing. In *Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*. 1–12.

[44] J. H. Kim, B. Grady, R. Lian, J. Brothers, and J. H. Anderson. 2017. FPGA-based CNN inference accelerator synthesized from multi-threaded C software. In *Proceedings of the 2017 30th IEEE International System-on-Chip Conference (SOCC'17)*. 268–273.

[45] Urs Köster, Tristan Webb, Xin Wang, Marcel Nassar, Arjun K. Bansal, William Constable, Oguz Elibol, Stewart Hall, Luke Hornof, Amir Khosrowshahi, Carey Kloss, Ruby J. Pai, and Naveen Rao. 2017. Flexpoint: An adaptive numerical format for efficient training of deep neural networks. In *Advances in Neural Information Processing Systems 30*. 1740–1750.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*. 1097–1105.

[47] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* (Nov. 1998), 2278–2324.

[48] E. A. Lee and D. G. Messerschmitt. 1987. Synchronous data flow. *Proc. IEEE* 8, 11, 1235–1245.

[49] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou, and Lingli Wang. 2016. A high performance FPGA-based accelerator for large-scale convolutional neural networks. In *Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications (FPL'16)*. 1–9.

[50] Jinyu Li, Jian Xue, and Yifan Gong. 2013. Restructuring of deep neural network acoustic models with singular value decomposition. In *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH'13)*.

[51] Shuang Liang, Shouyi Yin, Leibo Liu, Wayne Luk, and Shaojun Wei. 2018. FP-BNN: Binarized neural network on FPGA. *Neurocomputing* 275, C, 1072–1086.

[52] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Penksy. 2015. Sparse convolutional neural networks. In *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*. 806–814.

[53] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision (ECCV'16)*. 21–37.

[54] Zhiqiang Liu, Yong Dou, Jingfei Jiang, and Jinwei Xu. 2016. Automatic code generation of convolutional neural networks in FPGA implementation. In *Proceedings of the 2016 International Conference on Field-Programmable Technology (FPT'16)*. 61–68.

[55] Y. Ma, Y. Cao, S. Vrudhula, and J. s. Seo. 2017. An automatic RTL compiler for high-throughput FPGA implementation of diverse convolutional neural networks. In *Proceedings of the 2017 27th International Conference on Field Programmable Logic and Applications (FPL'17)*. 1–8.

[56] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae sun Seo. 2017. Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, New York, NY, 45–54.

[57] Yufei Ma, Minkyu Kim, Yu Cao, Sarma Vrudhula, and Jae Sun Seo. 2017. End-to-end scalable FPGA accelerator for deep residual networks. In *Proceedings of the 2017 IEEE International Symposium on Circuits and Systems (ISCAS'17)*. IEEE, Los Alamitos, CA, 1–4. DOI: http://dx.doi.org/10.1109/iscas.2017.8050344

[58] Yufei Ma, Naveen Suda, Yu Cao, Jae Sun Seo, and Sarma Vrudhula. 2016. Scalable and modularized RTL compilation of convolutional neural networks onto FPGA. In *Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications (FPL'16)*. 1–8.

[59] Yufei Ma, Naveen Suda, Yu Cao, Sarma Vrudhula, and Jae Sun Seo. 2018. ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler. Integration, the VLSI Journal. DOI: 10.1016/j.vlsi.2017.12.009

[60] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH'10)*.

[61] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi. 2016. Design space exploration of FPGA-based deep convolutional neural networks. In *Proceedings of the 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC'16)*. 575–580.

[62] Mohammad Motamedi, Philipp Gysel, and Soheil Ghiasi. 2017. PLACID: A platform for FPGA-based accelerator creation for DCNNs. *ACM Trans. Multimedia Comput. Commun. Appl.* 13, 4, Article 62, 21 pages.

[63] E. Nurvitadhi, Jaewoong Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr. 2016. Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC. In *Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications (FPL'16)*. 1–4.

[64] Eriko Nurvitadhi, Suchit Subhaschandra, Guy Boudoukh, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason Ong Gee Hock, Yeong Tat Liew, Krishnan Srivatsan, and Duncan Moss. 2017. Can FPGAs Beat GPUs in accelerating next-generation deep neural networks? In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, Los Alamitos, CA, 5–14.

[65] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA'17)*. ACM, New York, NY, 27–40.

[66] Jongse Park, Hardik Sharma, Divya Mahajan, Joon Kyung Kim, Preston Olds, and Hadi Esmaeilzadeh. 2017. Scale-out acceleration for machine learning. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'17)*. ACM, New York, NY, 367–381.

[67] A. Prost-Boucle, A. Bourge, F. Pétrot, H. Alemdar, N. Caldwell, and V. Leroy. 2017. Scalable high-performance architecture for convolutional ternary neural networks on FPGA. In *Proceedings of the 2017 27th International Conference on Field Programmable Logic and Applications (FPL'17)*. 1–7.

[68] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going deeper with embedded FPGA platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'16)*. ACM, New York, NY, 26–35.

[69] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV'16)*.

[70] Joseph Redmon and Anelia Angelova. 2015. Real-time grasp detection using convolutional neural networks. In *Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA'15)*. 1316–1322.

[71] Colin R. Reeves (Ed.). 1993. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, New York, NY.

[72] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2017. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 6, 1137–1149.

[73] Michalis Rizakis, Stylianos I. Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. 2018. Approximate FPGA-based LSTMs under computation time constraints. In *Proceedings of the 14th International Symposium on Applied Reconfigurable Computing (ARC'18)*.

[74] J. Shao, C. C. Loy, K. Kang, and X. Wang. 2017. Crowded scene understanding by deeply learned volumetric slices. *IEEE Trans. Circ. Syst. Video Technol.* 27, 3, 613–623.

[75] Hardik Sharma, Jongse Park, Emmanuel Amaro, Bradley Thwaites, Praneetha Kotha, Anmol Gupta, Joon Kyung Kim, Asit Mishra, and Hadi Esmaeilzadeh. 2016. DnnWeaver: From high-level deep network models to FPGA acceleration. In *Proceedings of the Workshop on Cognitive Architectures*.

[76] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. 2016. From high-level deep neural models to FPGAs. In *Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*. 1–12.

[77] Yongming Shen, Michael Ferdman, and Peter Milder. 2017. Escher: A CNN accelerator with flexible buffering to minimize off-chip transfer. In *Proceedings of the 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'17)*. 93–100.

[78] K. Simonyan and A. Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR'15)*.

[79] Nikolai Smolyanskiy, Alexey Kamenev, Jeffrey Smith, and Stan Birchfield. 2017. Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness. In *Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'17)*. 4241–4247.

[80] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1929–1958.

[81] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae Sun Seo, and Yu Cao. 2016. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'16)*. ACM, New York, NY, 16–25.

[82] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. 2017. Inception-v4, inception-ResNet and the impact of residual connections on learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[83] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*. 1–9.

[84] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, New York, NY, 65–74.

[85] Stylianos I. Venieris and Christos-Savvas Bouganis. 2017. fpgaConvNet: A toolflow for mapping diverse convolutional neural networks on embedded FPGAs. In *Proceedings of the Workshop on Machine Learning on the Phone and Other Consumer Devices (MLPCD'17)*.

[86] Stylianos I. Venieris and Christos-Savvas Bouganis. 2016. fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs. In *Proceedings of the 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'16)*. 40–47.

[87] Stylianos I. Venieris and Christos-Savvas Bouganis. 2017. fpgaConvNet: Automated mapping of convolutional neural networks on FPGAs (abstract only). In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, New York, NY, 291–292.

[88] Stylianos I. Venieris and Christos-Savvas Bouganis. 2017. Latency-driven design for FPGA-based convolutional neural networks. In *Proceedings of the 2017 27th International Conference on Field Programmable Logic and Applications (FPL'17)*. 1–8.

[89] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. 2017. Show and tell: Lessons learned from the 2015 MSCOCO image captioning challenge. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 4, 652–663.

[90] Ying Wang, Jie Xu, Yinhe Han, Huawei Li, and Xiaowei Li. 2016. DeepBurning: Automatic generation of FPGA-based learning accelerators for the neural network family. In *Proceedings of the 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC'16)*. Article 110, 6 pages.

[91] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *Proceedings of the 54th Annual Design Automation Conference (DAC'17)*. ACM, New York, NY, Article 29, 6 pages.

[92] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. TernGrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in Neural Information Processing Systems 30*. 1508–1518.

[93] Samuel Williams et al. 2009. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4, 65–76.

[94] Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV'14)*.

[95] Hanqing Zeng, Ren Chen, Chi Zhang, and Viktor Prasanna. 2018. A framework for generating high throughput CNN implementations on FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA'18)*.

[96] Hanqing Zeng, Chi Zhang, and Viktor Prasanna. 2017. Fast generation of high throughput customized deep learning accelerators on FPGAs. In *Proceedings of the 2017 International Conference on Reconfigurable Computing and FPGAs (ReConFig'17)*.

[97] Hanqing Zeng, Chi Zhang, and Viktor Prasanna. 2017. *Optimizing Frequency Domain Implementation of CNNs on FPGAs*. Technical Report. University of Southern California.

[98] Chen Zhang, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. 2016. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. In *Proceedings of the 35th International Conference on Computer-Aided Design (ICCAD'16)*. ACM, New York, NY, Article 12, 8 pages.

[99] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'15)*. ACM, New York, NY, 161–170.

[100] Chi Zhang and Viktor Prasanna. 2017. Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, New York, NY, 35–44.

[101] Jialiang Zhang and Jing Li. 2017. Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, New York, NY, 25–34.

[102] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen. 2016. Cambricon-X: An accelerator for sparse neural networks. In *Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*. 1–12.

[103] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. 2017. Accelerating binarized convolutional neural networks with software-programmable FPGAs. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, New York, NY, 15–24.

[104] Wenlai Zhao, Haohuan Fu, Wayne Luk, Teng Yu, Shaojun Wang, Bo Feng, Yuchun Ma, and Guangwen Yang. 2016. F-CNN: An FPGA-based framework for training convolutional neural networks. In *Proceedings of the 2016 IEEE 27th International Conference on Application-Specific Systems, Architectures, and Processors (ASAP'16)*. 107–114.

[105] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental network quantization: Towards lossless CNNs with low-precision weights. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*.

[106] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. 2016. DoReFa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv:1601.06160.

[107] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. 2017. Trained ternary quantization. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*.

Received July 2017; revised February 2018; accepted February 2018