DeepBurning: Automatic Generation of FPGA-based Learning Accelerators for the Neural Network Family

Conference Paper · June 2016

DOI: 10.1145/2897937.2898003

5 authors, including:

Ying Wang
Chinese Academy of Sciences
77 PUBLICATIONS **342** CITATIONS
SEE PROFILE

Huawei Li
Chinese Academy of Sciences
196 PUBLICATIONS **1,130** CITATIONS
SEE PROFILE

Xiao-Wei Li
Chinese Academy of Sciences
271 PUBLICATIONS **1,984** CITATIONS
SEE PROFILE

DeepBurning: Automatic Generation of FPGA-based Learning Accelerators for the Neural Network Family

Ying Wang, Jie Xu, Yinhe Han, Huawei Li, and Xiaowei Li
State Key Laboratory of Computer Architecture
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, P.R. China
{wangying2009, xujie, yinhes, lihuawei, lxw}@ict.ac.cn

Abstract

Recent advances in Neural Networks (NN) are enabling more and more innovative applications. As an energy-efficient hardware solution, machine learning accelerators for CNNs or traditional ANNs are also gaining popularity in the area of embedded vision, robotics and cyberphysics. However, the design parameters of NN models vary significantly from application to application. Hence, it's hard to provide one general and highly-efficient hardware solution to accommodate all of them, and it is also impractical for the domain-specific developers to customize their own hardware targeting on a specific NN model. To deal with this dilemma, this study proposes a design automation tool, DeepBurning, allowing the application developers to build from scratch learning accelerators that targets their specific NN models with custom configurations and optimized performance. DeepBurning includes a RTL-level accelerator generator and a coordinated compiler that generates the control flow and data layout under the user-specified constraints. The results can be used to implement FPGA-based NN accelerator or help generate chip design for early design stage. In general, DeepBurning supports a large family of NN models, and greatly simplifies the design flow of NN accelerators for the machine learning or AI application developers. The evaluation shows that the generated learning accelerators burnt to our FPGA board exhibit great power efficiency compared to state-of-the-art FPGA-based solutions.

1. INTRODUCTION

With decades of development, neural network, initially inspired from brain biology, is becoming a powerful class of computing model widely adopted in a variety of applications such as machine vision, cognitive and approximate computing [1]. In a broad sense, the family of neural networks (NNs) includes many diverse models from the basic multi-level perceptron (MLP), recurrent Neural Network (RNN) to complex Convolutional Neural Network (CNN). A certain NN model is thought suitable for some special applications or application domains. For example, CMAC and Hopfield networks are often used in robot control while LSTM models show fascinating accuracy in text or stream recognition. As deep learning rises, the emerging CNNs and DNNs are showing great potential in image and speech recognitions [2]. More aggressively, neural models are even introduced to accelerate general purpose computing though application approximation [1].

As the neural network family and machine learning thrive, it is expected to provide real-time NN invocation performance to address the problem of vision, control and speech processing in an energy-efficient way for embedded devices or even cost-sensitive datacenters. Therefore, hardware acceleration instead of software is thought as a more efficient solution. A fast and low-power NN accelerator that supports a wide range of NN models are enabling innovative applications in different fields. Generally, there are multiple mainstream design routes to accelerate the NN-based applications:

DAC '16, June 05-09, 2016, Austin, TX, USA
 ISBN 978-1-4503-4236-0/16/06 $15.00
DOI: http://dx.doi.org/10.1145/2897937.2898003

GPGPU Due to the high capabilities of float point matrix processing, DSPs and GPGPU are often used to achieve neural network acceleration by exploiting the data-level parallelism inside the matrix multiplication operations [2] [3]. Especially for current large-scale deep CNNs, GPU-supported learning frameworks like Caffe and Convnet are proved to deliver high throughput in network inference and backward training [3]. However, high-throughput GPGPU are thought rather cumbersome in terms of power and area overhead. They cannot meet the requirement of embedded systems and even some commercial datacenters, of which the Total Cost of Ownership (TCO) depends directly on power consumption.

ASICs In contrast, **ASIC**s possess the advantages of high performance and energy efficiency. However, the weakness of application-specific hardware is a lack of ability to support different models. As an improvement, general purpose neural accelerators are proposed to enable many diverse NN models in a unified hardware platform [1] [4]. Existing neural accelerators are thought belonging to the category of application-specific instruction processor (**ASIP**), and they are still not flexible enough to handle a myriad of complex NN models from different areas, which differs greatly in network parameters and the induced data-flows in computation. Especially, since deep learning is developing rapidly in recent years, researchers are also proposing new network topologies and optimization techniques like new activation functions, 3D convolution or normalization layers. For these designs [4] [1], it is non-trivial to extend the ISA and change the hardware specifications to enable new techniques. Even if they can support a certain NN model, they cannot offer good computation efficiency due to the mis-match between the fixed hardware structure and the network parameters.

Comparatively, **FPGA** strike a perfect balance between efficiency and versatility of enabling new network models through reconfiguration, and is thought as proper hardware solution for the application developers. Prior work has comprehensively studied the opportunities to accelerate certain neural network models with FPGA from 3-layered MLPs, Lenet-5 to Alexnet [5] [6] [7], which are classic instantiations of CNN. FPGA is able to achieve good performance by mimicking one network topology at a time through off-line reconfiguration. If the designers understand the network topologies and the data flow patterns, they could implement the specific network in FPGA. However, for the developers who focus high level learning models or neural network architectures, the design flow of FPGA still takes considerable design efforts and adds to complexity of application development.

In this work, DeepBurning, as a holistic framework of general purpose neural network acceleration, is proposed to simplify the procedure of mapping diverse neural networks into FPGA architectures or ASIC designs. DeepBurning offers an unified hardware RTL generator and an integrated complier that generate the on-line control flow for any user-specified network topologies, so that it only takes almost "one-click" effort to map a complex neural network model, including basic MLPs, RNNs, CNN and DNNs, to a high-performance and energy-efficient hardware description ready for FPGA burning or ASIC implementation. DeepBurning not only implements the model in **hardware**, but also analyzes the data reusing patterns, the scale and depth of neural networks, and the design constraints such as area and power budget, so that it could adaptively generate the **data layout** and dynamic **control flow** to best accelerate the NN propagation through "temporal and spatial folding".

The opportunities and Challenges

Why FPGA? We have explained the obstacles to change the ASICs or ISAs of neural accelerators to support different network models. Comparatively, the reconfigurability of FPGA can enable diverse NN models for different applications, and also resolves the thorny problem of **design space explosion** in NN model selection and training. The reason is because there are so many parameters and topologies even in one type of neural network, that the **model selection and training** for a certain application is hard and tedious. For example, some state-of-the-art image classification models rely on a recombination of NN layers or techniques for many times to find a really highly-accurate implementation [8] [9]. The rationale behind this design philosophy is controversial: brute force is thought as one negligible aspect of NN-based solutions [10]. To do brute-force solution search takes considerable time in model selection and training. For each candidate model under evaluation, it has to be trained with a large set of input data to determine the weight of this NN model using backward propagation. The tedious training procedure has to repeat for each candidate model under selection. In this sense, FPGAs are fast and power-efficient enough to accelerate the time-consuming NN training, at the same time it possess the reconfigubility to enable the designers to explore the space of NN models and topologies for the given application. Comparatively, neither ISA nor ASIPs can be easily changed to support the new neural layers proposed by designers or adapt to NN parameters for performance improvement.

Why DeepBurning? First, FPGA is not friendly to the high-level application developers seeking for solutions of hardware acceleration. The goal of DeepBurning is to let the architecture of neural network acceleration become totally transparent to those users. With DeepBurning, the developers can start from their familiar software framework, i.e. Caffe [3], to begin with the model design. Then if they need a hardware implementation for the application, or hardware acceleration for design space exploration and model training, they can pass the simple descriptive script of neural net from Caffe to DeepBurning to generate the final hardware. **Second**, neural network accelerator is not pure hardware implementation from design perspective, it has the software part including on-line reconfiguration and complex data accessing patterns. For example, the layers of a large scale CNN has to be mapped to the FPGA in a multiplexing way due to resource constraint, which is defined as "folding". Therefore, available automation tools like High Level Synthesizer (HLS) cannot easily orchestrate the hardware partitioning, control path and data layout, and make them work in synchronization. Zhang et. al. only use commercial HLS to ease a minor part of the component design in their learning accelerator for Alexnet [7]. Last of all, Designers without architecture expertise, or others without profound knowledge of neural networks can hardly design an efficient enough FPGA-based NN accelerator. The key to effectively accelerate NN-based learning models is to harness their data locality and memory access pattern [4], which are all considered and covered in DeepBurning.

In general, DeepBurning offers an energy-efficient and easy-to-use platform to implement the family of neural networks in FPGA or chip design. More specifically, the contributions of this work go as:

-First, we analyze and decompose many classic neural network models, and build DeepBurning RTL generator that uses the Caffe-compatible script to generate the RTL code of neural accelerators by determining the best hardware configurations for the network and resource constraint.

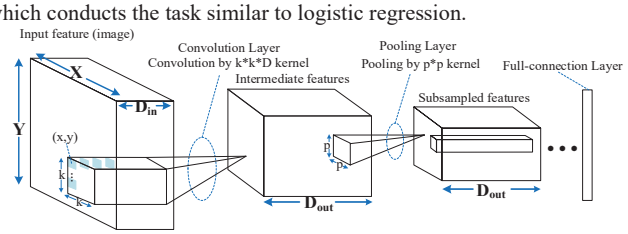
-Second, the integrated DeepBurning compiler achieves automatic software/hardware co-design by generating the on-line control flows, which is indispensable to "temporally" and "spatially" fold the neural network into FPGA within user-specified constraint.

-Last of all, DeepBurning is able to generate an optimized input data and NN weight layout in memory and the associated Address Generation Units to exploit the memory locality in NN models.

2. Preliminaries and Observation

2.1 The family of Neural Networks and learning accelerators

A typical NN consists of multiple interconnected neural processing elements that perform weighted accumulation of input feature and weight, and then input the convolved values into activation functions. With different topologies and free combinations of layers, NNs are developing into different types of variants. In respect to propagation direction, there are feed-forward NN and recurrent NN (RNN). In respect to layer count and learning method, there multi-level NNs and deep NNs. Researchers are proposing diverse NN models to deal with applications in different scope by introducing new optimization techniques. Taking a typical CNN illustrated in Fig. 1 for example, a CNN includes repetitive layers of feature-and-kernel convolution/pooling and other kernel-level operations. The first convolution layer walks through each of the *Din* input images with *k*×*k* convolution kernels at a constant stride and generates *Dout* intermediate feature maps. Then, the second pooling layer averages each *p*×*p* sub-regions into one pixel for all the *Dout* feature maps, and afterwards sends the subsampled results to the final full connection layer, which conducts the task similar to logistic regression.



X,Y = the size of the input maps, $\mathbf{D_{in}}$ = # input feature maps, $\mathbf{D_{out}}$ = # output feature maps, k = convolutional kernel size, p = pooling kernel

Fig. 1. A 4-layer CNN containing multiple layers from a high level view

A Neural Network accelerator pursues higher performance and energy-efficiency by mapping the topology into logical gates. For simple illustration, Fig. 2 shows a fully expanded 3-layer MLP circuit. It is composed of a hidden layer and an output layer. Each synapse in a layer indicates a multiplication operation of the weight, and carries the neuron output from the previous layer. The synapse-connected neuron in the next layer first accumulates the results of all its synapses and then applies a sigmoid transform to the sum as the activation function, which is implemented in logics or lookup table. In overall, a hardware MLP is a computation-intensive circuit where multiplication and addition operations dominate the propagation process. However, the performance of NN accelerator depends heavily on the memory bandwidth and locality preservation. In Fig. 2, the on-chip memory part include two buffers. One stores the input features and the other stores the weights of the network.

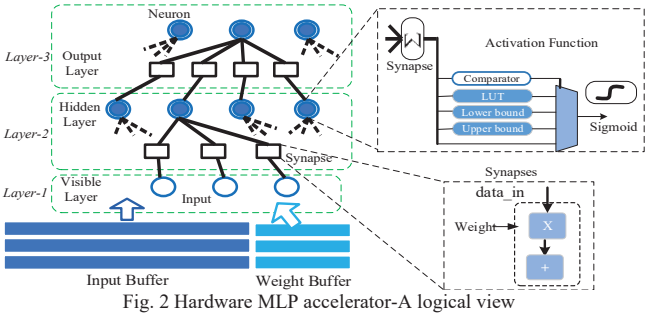


Fig. 2 Hardware MLP accelerator-A logical view

2.2 The commonality of different Neural Networks

As we can see from the basic MLPs to more complicated CNNs, they vary in topology and layer count, but they share some of the common core operations such as inter-convolution of features and weights, activation functions, inter-layer connection. With the limited hardware layer types, one can create a large space of NN models including MLP, RNNs, CNNs and etc. By analogy, it is possible to use an automatic high-level tool to generate different NN accelerators by combining the basic hardware layers of neural processing units.

3. Architecture of DeepBurning

3.1 The framework of DeepBurning

Fig. 3 depicts the general design flow of employing DeepBurning to automate the hardware acceleration of NN models. First, the DeepBurning Neural Network Generator (NN-Gen) accepts the input of model descriptive script that describes a brief macro-view of network topology and layer definition, so that it could parser developer's description into realizable hardware descriptive language (HDL) by matching the NN layers to the demanded hardware components. At the same time, the overhead constraint specified by the developer is also passed to NN-Gen. The constraint allows NN-Gen to generate a properly-scaled hardware structure for the given logic resources constraint (i.e. gates or Programmable-Logic resources). With the input, the hardware generator will explore the pre-constructed NN component library according to the specified NN topology, and connect the reconfigurable RTL modules from the library into a top-view of hardware NN structure. The RTL modules are like the bricks while the NN model descriptive script is like the architectural print. The descriptive file is easy to use and compatible with *Caffe* [3] as shown in Fig. 4 that describe three different layers. The type of layers is redefinable to support more classes of layer or operation than that in original *Caffe*. Currently DeepBurning supports typical convolutional layer, pooling layer, full-connection layer, recurrent layer, associative layer and other common CNN or ANN operations.

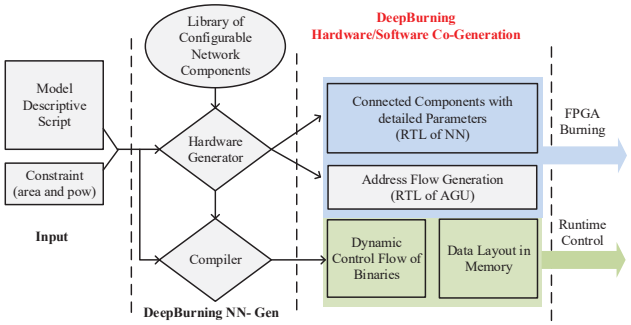


Fig 3. Framework of Learning Accelerator Generation via DeepBurning NN-Gen

```
layers {
  name: "conv1"
  type: CONVOLUTION
  bottom: "data"
  top: "conv1"
param{
    num_output: 20
    kernel_size: 5
    stride: 1}
connect {
  name: "c2p1"
  direction: forward
  type: full per channel }
}
```

```
layers {
  name: "pool1"
  type: POOLING
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
      stride: 2
    }
}
```

```
layers {
  name: "relu1"
  type: RELU
  bottom: "ip1"
  top: "ip1"
connect {
  name: "p2f2"
  direction: recurrent
  type: file_specified
  }
}
```

Fig. 4 An example of descriptive script (*.prototxt)

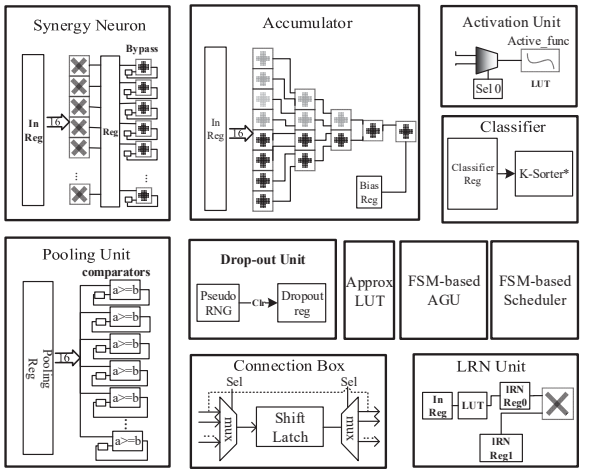
Because of resource constraint, a neural network model sometimes cannot not be fully expanded and then mapped into hardware by the hardware generator as it does in Fig. 2. Hence, the model will be "folded" into a limited set of neurons (lanes) and use the hardware in a time-multiplexing way, so the generated hardware needs a run-time control flow to synchronize the segmented "lanes" of NN in forward propagation, which is also created simultaneously by the DeepBurning *compiler* in cooperation with the hardware generator. Meanwhile, the complier will also be responsible for pre-processing the NN feature data and weight layout into proper "tiles" by analyzing the compute throughput and on-chip memory size of the NN accelerator. In this way, the hardware part (RTL) and software part (control flow and data layout) are generated by the NN-Gen at the same time. Another important component lies in between hardware and software is the address flow that is used to fetch on-chip and off-chip memory data automatically. In DeepBurning, the memory address flow is generated deterministically by the DeepBurning compiler and automatically generalized into multiple access patterns by a built-in analyzer. After that, the memory access patterns are sent to the hardware generator to tailor the Address Flow Generators (AGUs) that support these patterns in the NN accelerator.

3.2 Building the library of basic neural components

As illustrated in Fig. 3, NN-Gen relies on a pre-constructed module library to build the target machine learning accelerator for the specific neural network model. First, we examine the common operations or layers in a wide range of NN-based models to decide the basic component to prepare. Table. 1 shows several examples of NN models we built for different applications from number recognition, image classification to robot arm control. Though the operational layers are often adjusted and change into variants with different parameters. For example, the full connection (FC) layers can be partially connected while the width, depth and size of convolution layers can also be different. However, the layers from these models in Table. 1 share most of the basic operations that could be mapped into the smallest common set of hardware components.

Table. 1 A Decomposition of the typical neural networks

	MLP	Hopfield	CMAC	Alexnet	Minist	GoogleNet
Conv, Layer	×	×	×	√	√	√
FC Layer	√	√	√	√	√	√
Act-Func	√	√	√	√	√	√
Drop-Out	×	×	×	√	×	√
LRN	×	×	×	×	√	√
Pooling	×	×	×	√	√	√
Associative	×	×	√	×	×	×



*K-Sorter is implemented according to [11]

Fig. 5 Major building blocks in NN component library

By investigating typical RNN, MLP, CNN and DNN models, we designed the first batch of basic reconfigurable components, stored their hardware descriptive files and the associated configuration scripts into the library. The most important components are depicted in Fig. 5. The library is still open for extension and can be upgraded to include more components for new NN techniques. The basic components are not hardwired in RTL library but leave out multiple reconfigurable parameters, i.e. the input bit-width, the neuron-level parallelism, and disablable ports or functions, for the DeepBurning hardware generator to decide the detailed module property according to the target model and constraint.

The major building blocks could be combined to realize the decomposed layers of most neural network models. The propagation of convolutional layer can be mapped to the synergy neurons and the accumulators. The hidden layer of MLP can be mapped to multiple

synergy neurons. The connection box in Fig. 5 handles the exchange of intermediate values between layers, and reconnects them as a crossbar. Connection box also include a shifting latch used to achieve approximate division operation with the inter-layer values. Besides these, the mapping between certain high-level neural network layers and the RTL building blocks are illustrated as follows:

-Full connection layer: synergy-neurons + accumulators
-Recurrent Layer: synergy-neurons + connection_box
-Memory Layer: connection-box
-Convolution Layer: synergy-neuron +accumulator
-Pooling Layer: pooling unit/accumulator
-LRN/LCN layer: LRN unit
-Drop-out inserter: drop-out unit
-Classification Layer: classifier or classifier + synergy neuron
-Activation Function: activation-unit or activation-unit +synergy neuron
-Inception layer: pooling-unit+synergy neuron+accumulators

Besides these arithmetic and logical components, another important component is the Approximate Look-Up Table (Approx LUT) used to approximate functions or operations that cannot be efficiently mapped into logical gates. It can also be used to extend new functions that are not supported in current version of library. NN compiler also generate the content stored in the LUT to approximate some user-specified functions.

The building blocks introduced above are used to mimic the functions of NN layers, and they are defined as *functional building block*s. There are also two other important building blocks: scheduling coordinator and Address Generation Unit (AGU) and, which automatically fetch feature and weight data to drive the accelerator data-path and dynamically map the "folded" NN layers onto the data-path of any width. AGUs will be described in detail in later sections.

3.3 Mapping the target network into a data-driven accelerator

Co-design with hardware generator and compiler Hardware generator maps the target network described by input script into the hardware accelerator by connecting the generator-configured building blocks up. However, on one hand, the neural network models vary significantly in scale and complexity. On the other hand, the designer-specified area constraint or resource provision of the target FPGA board might not afford a fully-expanded neural network in hardware, i.e. mapping every neurons and every layers loyally into the circuits or logics. Therefore, NN-Gen leverages two methods to compress the NN model into hardware to the greatest extent: ***temporal folding*** that maps different layers into the common set of building blocks, and ***spatial folding*** that splits a single neural layer and let the segments share the building blocks at different time slots. Therefore, it relies on run-time control to use the hardware in a multiplexing way. In order to simplify the scheduling, the learning accelerator adopts a data-driven architecture to operate the network forward-propagation. With the correctly-directed data flow, the *functional building blocks* only start computation after the input data arrives, and pass the results to the connected consumer building blocks on completion. In this way, the coordinator only needs to link the producer blocks to the consumer blocks at pre-determined beats since the AGUs know how to load, store and forward the data sets including input feature data, NN weight data and intermediate data.

AGU and **Data Flow Direction** AGU is the core control unit for the whole data-driven architecture by correctly fetching and storing the data sets according to pre-determined patterns. There are three types of AGUs in the generated accelerator: *main* AGU, *data* AGU and *weight* AGU. Main AGU is responsible for the data exchange between on-chip buffers and the off-chip memory. Data AGU is responsible for fetching input and intermediate feature data from on-chip buffers to the accelerator data-path, whilst weight AGU handles the weight fetch and store for the on-chip buffer and the data-path.

Each of the AGUs supports multiple memory access patterns and can automatically generate data streams to drive the data-path. In NN-Gen, it needs the co-operation between hardware generator and compiler to realize the AGUs. First, after the data-path is fixed by NN-Gen according to the designer-specified script and constraint, the resource sharing and partitioning of layers are also fixed according to "temporal folding" and "spatial folding". Hence, the compiler can generate the memory (including on-chip and off-chip) access stream of addresses according to the reported hardware configuration and the target network specification. The address patterns will be described by a Finite-State-Machine using the reasoning component of the compiler, and then the FSMs will be passed to the hardware generator to create the RTL description of AGUs. In addition, a context buffer sends pattern triggering signals to AGUs in pre-determined phases marked by pre-defined events as *layer0-fold0*.

The access patterns of AGUs have several key fields: *starting address, footprint (size), x_length, y_length, stride, off-set* and *etc*. The parameters ensure that the AGUs support powerful enough accessing mode and patterns to generate the correct read and write address streams in different stages of neural network propagation. Event is used to trigger a round of pattern generation. Fig. 6 shows a unified view of the AGU template in building block library. The final AGU generated for the target network is reduced from this template AGU to provide the demanded on-chip and off-chip memory access patterns. Because DeepBurning accelerator uses a data-driven execution architecture, AGUs not only provide data but also triggers the execution of different neural layers and layer segments (fold).

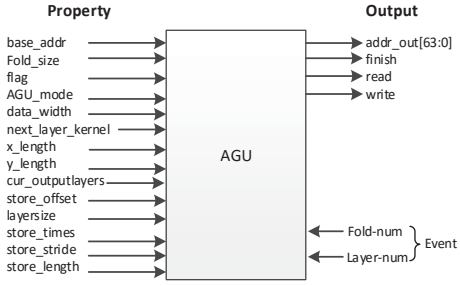


Fig. 6 Configurable property and output of template AGU

Approx LUT Generation Approx LUT is an important building block used to implement complex functions. Similar to AGU, the hardware description is pre-determined by hardware generator, but its stored content is decided by the compiler. In network forward propagation, the input of accelerated functions such as sigmoid activation are sent to index the LUT entries with the correct output result pre-stored there. Therefore, LUT can be used to approximate a continuous function by storing a limited set of sample points in table. In Approx LUT, the input keys that hit the table can directly read out the correct result, while for those input keys not sampled by the table of limited entries, the upper and lower adjacent keys will be checked and their associated values are used to generate the approximate result for these input keys through super-linear interpolation. Because NN-based algorithm are belonging to approximate computing domain where 100% arithmetic accuracy is not necessary or impractical, the computation results of NN propagation are not sensitive to the minor inaccuracy introduced by Approx LUT [1], which will be verified in our evaluation.

The size (depending on accuracy requirement) and content of Approx LUT, including the keys and values, are generated a priori by NN-Gen compiler and fed to the LUT for approximate computing. Therefore, compiler will firstly parse the complex functions, choose the necessary sampling points and then calculate the values to be filled in Approx LUTs.

Dynamic Control flow: For the scenario when the logic resources are inadequate, the connected building blocks are shared by the folded network model. The data-driven execution mode simplifies the scheduling and synchronization via AGUs, it still requires simple dynamic producer-consumer reconnection to configure the data-path from a central coordinator. For example, the synergy neuron set used by

one layer of weight-data-product operation, need to be reconnected to accumulators afterwards to walk through the next average pooling layer. The configuration signals are generated in time by the FSM-based coordinator. The FSMs are also created by NN-Gen compiler.

3.4 Hardware-aware Data Layouting

NNs typically have two important work-sets: input feature data and weight trained for applications. The layout of the two data sets have great impacts on the bandwidth and memory space utility of the generated accelerator. It is key to the effect of acceleration by preserving the memory locality [6] [4]. Considering a 2D 57x57 pixel input in Fig. 7 for example, originally the image is continuously aligned in memory row by row from $pixel_{0,0}$, $pixel_{0,1}$ to $pixel_{0,57}$. Suppose a 12x12 convolutional window is to scan through the image from the beginning 12x12 sub-region to the end at a stride 4, the continuous mapping leads to a poor bandwidth utilization because only the first 12 pixels are used if the whole first row is fetched. Therefore, the 2-D image is decomposed into 12x12 tiles, and pixels in each tile are continuously put in memory, so that the memory locality is preserved. This is also called automatic data *tiling*.

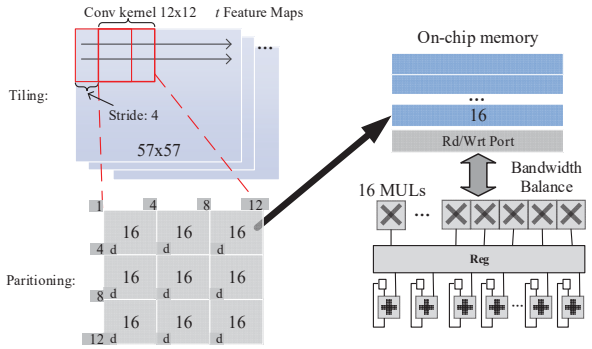


Fig. 7 Data tiling and partitioning

Except data tiling that increases the utility of buffer space, another step of data *partitioning* is still needed to improve locality and balance PE and memory throughput. For example in Fig. 7, a 12x12 tile is further partitioned into 16-pixel regions for better data reusability, because a 4x4 sub-region will not be re-accessed when the kernel moves at a stride of 4, reducing the times of data re-fetch. This step of data partitioning also makes sure that the fetched feature blocks are aligned in the on-chip memory of particular output-width, simplifying the data fetching for AGUs. The reason is because the width of read/write port is set to match the throughput of the accelerator data-path. With *partitioning*, the 12x12 kernel-size window are so divided into continuous 4x4 sub-blocks that can perfectly fit the memory row that is the minimum size to activate in buffer and the data-path as well. For instance in Fig. 7, if the on-chip buffer offers 1 sub-block (16 pixels) every one cycle, and the 32 synergy neurons complete the data-weight-product every two cycles, the accelerator is perfectly balanced. Fig. 7 only shows a realistic example of data layout creation in NN-Gen. However, since kernel and map sizes are always different for NN models, some maps or kernels can never be perfectly partitioned to fit the hardware, in that case, the width of memory port and data-path will be reshaped to make it easy to achieve data alignment by the hardware generator. The overall data tiling and partitioning policy is described in Method-1.

Method-1

Input: kernel is kxk, stride is s, memory port width is d, map count is t

1: IF $k^2 = d^2$, reorganize the data layout into kxk tiles, align the tiles of one map continuously in memory, and then the next map;

2: IF $k^2 != d^2$ && (s is the common divisor of k and d), partition the data layout into sxs tiles, align the tiles within one map continuously;

3: ELSE, parition the layout into fxf tiles for f is the common divisor of k, d and s, then interleaves the tiles of t maps one by one in memory

Meanwhile, the layout of network weight is partitioned accordingly to accompany the layout of feature data for computation.

4. Evaluation

4.1 Experimental Setup

We established the whole DeepBurning framework including NN-Gen hardware generator, library and compiler. NN-Gen is running at the Intel Xeon 2.4Ghz CPU with 8MB Last level cache. The generated learning accelerator is synthesized and implemented onto a Xilinx board with Zynq-7045 SoC device. The operating frequency is 100MHz. The system software that manages the generated accelerator as a peripheral device is run at an ARM Cortex-A9 core. The ARM core reorganizes the input data and weight data of neural networks into an optimized layout as directed by NN-Gen compiler, and then stores them into 2GB on-board DDR3 memory. The generated learning accelerator accesses the DRAM memory through AXI switches.

Application Eight NN models are testified with NN-Gen: three 4-layer ANNs, 2-layer Hopfield, 2-layer CMAC, 5-layer MNIST, Alexnet, NiN and Cifar. Their functions are depicted in Table. 2. Alexnet and NiN are advanced deep CNN models and also show good performance ILSVRC [5], which is the most popular object detection and image classification challenges globally

Three ANNs are designed and trained respectively to implement three AxBench benchmarks for general purpose approximate computing [1]. The training of neural network models are conducted with Matlab except that Alexnet, NiN, Cifar and MNIST are trained in Caffe with the imageNet, Cifar and MNIST training set. The trained weights are preprocessed by ARM core and stored into the memory of Zynq board. The RTL-level simulation of forward-propagation is conducted with Vivado to verify the timing and function of the generated accelerators.

TABLE. 2 BENCHMARKS

	Conv	FC.	Rec.	Application
ANN-0, 1, 3	×	√	×	*fft, jpeg, kmeans*
Alexnet, NiN	√	√	×	Image recognition
Cifar	√	√	×	Image classification
CMAC	×	√	√	Robot arm control
Hopfield	×	√	√	TSP solver
MNIST	√	√	×	Number recognition

4.2 Experimental Results

Since prior work mostly aims to accelerate one type of NN model and also differ in FPGA platforms, it is hard to have a direct comparison between DeepBurning and them. To provide a fair comparison, a fourth-year graduate student with sufficient experience on deep learning and FPGA manually designed the customized NN accelerators for every application before NN-Gen is used to generate the corresponding DeepBurning accelerators.

Performance We map the well-trained network models onto the accelerators under comparison, and records the running time it takes for them to complete a round of network forward-propagation with the input set. The forward propagation time is a direct metric to measure the performance of NN accelerator, and also a critical metric to evaluate the model training speed with the accelerator due to the repetitive network inference in training. In experiments, we are going to compare the performance and power consumption of different platforms with DeepBurning accelerators for the same application.

Custom are our manually-coded accelerators for different applications. DB indicates the performance of NN-Gen generated accelerator with mediate resource budget, whilst DB-L shows the performance of NN-Gen generated accelerators with a high resource budget for Z-7045 device. DB-S shows the performance of NN-Gen generated accelerators with a low resource budget for Z-7020 device. As shown in Fig. 8, Custom mostly beats DB in performance. When compared to CPU (Xeon 2.4Ghz), DB achieves up to 4.7x speed-up. However, DB-L is 3.5x faster than DB

on average. [7] shows a customized Alexnet accelerator also implemented with FPGA at operating frequency of 100MHz [7]. As to Alexnet for image classification, the implementation in [7] is much faster than DB. However, when we change the constraint file to that of DB-L, DeepBurning shows comparable performance to that of Custom and [7] (~20ms) for Alexnet.

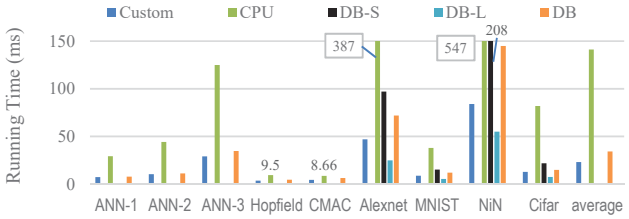


Fig. 8 Performance comparison

Power At the same time, the energy consumed in different schemes are compared in Fig. 9. DB consumes 1.8x more energy than Custom, while DB-L and DB-S dissipate almost the same amount of energy to Custom on average. CPU consumes about 58x more energy than DB on average. Though DB-L has a higher power consumption rate than DB due to higher area and power budget, it completes the tasks faster, and so eventually dissipates less energy than DB. [7] (~0.5J) consumes more energy than both DB-L and DB-S for it uses a much larger-scale FPGA devices.

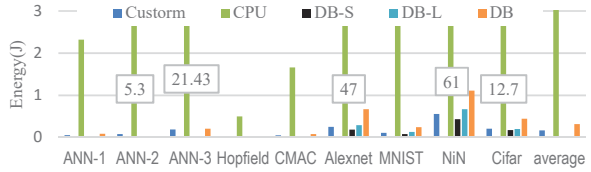


Fig. 9 Energy comparison

Accuracy In this experiment, test sets are also sent to the accelerators to evaluate the function correctness and output quality of the generated accelerators. Because we have some approximation techniques like Approx LUT and accuracy loss due to the fixed-point operation, there might be possible impacts on the output. The baseline is the original software neural networks executed on CPU. Alexnet, NiN, MNIST and Cifar are tested with Caffe, while others are tested with Matlab. The measured accuracy indicates the percentages of correctly-classified images in the whole input image set.

For other NN models like ANNs or CMAC not used for classification, the accuracy is the calculated by using the relative distances between the output of accelerator and the 100%-accurate results of golden reference.

$$\text{accuracy} = \left(1 - \frac{(A-B)^2}{B^2}\right) \times 100\% \quad (1)$$

B is the result of golden-reference application implemented with orthodox program of accurate modeling, and A is the result given by NN used to approximate the same application. For example, when we refer to [1] and use ANN-1 to approximate *jpeg* decoding. A is the result of MLP-1 with the same input image. B is the baseline result of standard software Jpeg-decoder. Both ANN running in CPU and DeepBurning accelerators will have accuracy loss compared to the golden reference implemented with traditional programming model other than artificial NN. As shown in Fig. 10, the accuracy loss of DeepBurning is comparable to that of the NNs running on CPU. For some models, it is even more accurate than software NN on CPU since the approximation techniques sometimes randomly eliminates the noises and suppresses over-fitting. In general, the DeepBurning accuracy shows only 1.5% variation over that of CPU-based NNs on average.

Hardware Overhead The implementations generated by DeepBurning have varied hardware resource utility rate. Table. 3 decomposes the resource usage of DeepBurning accelerators implemented with Custom (CU) and DeepBurning (DB). Alexnet-L is the result of DB-L with higher resource budget. In general, the implementation of DeepBurning consumes more resources than Custom on average. However, it possesses the flexibility to vary in performance provision and resource consumption.

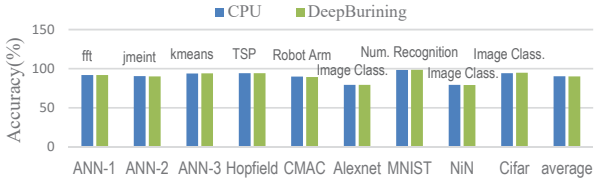


Fig. 10 Accuracy comparison

TABLE. 3 HARDWARE RESOURCE OCCUPATION

	DSP		LUT		FF	
	CU	DB	CU	DB	CU	DB
ANN-0	2	2	56	64	48	48
ANN-1	2	2	56	64	48	48
ANN-2	2	2	56	64	48	48
Alexnet	9	9	24413	25442	14336	14553
Alexnet-L	-	144	-	37832	-	33427
Cifar	12	12	5834	6342	4431	4842
NiN	42	42	31768	52356	46134	61724
CMAC	1	1	34	42	44	44
Hopfield	2	2	56	58	44	46
MNIST	12	12	3312	3621	2474	2532

5. CONCLUSION

DeepBurning is proposed to simplify the design flow of NN-based accelerators for ML or AI applications. As we proved, it enables an instant generation of the hardware and software solution for different NN models and instantiations, and makes it easy for the software developers to conduct NN model searching and training or implement their applications. With the proposals of NN-Gen hardware generator, compiler-directed network "folding", function approximation and automatic data tiling, DeepBurning offers optimized NN accelerator solutions on user's demand, which exhibits comparable performance and cost-effectiveness to the manually customized designs. Meanwhile, for the evaluated cases, it shows a significant 4.7x performance speedup and over 90% energy saving on average in contrast to the software solutions on CPU.

6. ACKNOWLEDGMENTS

This work was supported in part by National Natural Science Foundation of China under Grant No. 61432017, 61176040, 61504153, 61402146 and 61521092. The corresponding author is Huawei Li.

7. REFERENCES

[1] Esmaeilzadeh et al., "Neural acceleration for general-purposeapproximate programs,". In Proc. *MICRO*, 2012.
[2] G. Hinton et al., "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," in IEEE Signal Processing Magazine, , vol. 29, no. 6, pp. 82–97, 2012.
[3] Y Jia et al., "Caffe: Convolutional architecture for fast feature embedding," in ACM Proc. *Multimedia*, 2014.
[4] T. Chen et al., "DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning,"in Proc. *ASPLOS*, 2014.
[5] A. Krizhevsky et al., "Imagenet classification with deep convolutional neural networks," in Proc. *NIPS*, 2012.
[6] M. Peemen et al., "Memory-centric accelerator design for convolutional neural networks," in Proc. *ICCD*, 2013.
[7] C. Zhang et al., "Optimizing fpga-based accelerator design for deep convolutional neural networks,"in Proc. *FPGA*, 2015.
[8] W. Ouyang et al,. "DeepID-Net: Deformable Deep Convolutional Neural Networks for Object Detection," in Proc. *CVPR*, 2015.
[9] Y. Pan et. al., "Jointly Modeling Embedding and Translation to Bridge Video and Language," in arXiv, 2015.
[10] V. Vapnik, "Does Deep Learning Come from the Devil?," Yandex conference on machine learning prospects and applications, Berlin, 2015
[11] R. Beigel et al., "Sorting n Objects With a k-Sorter,"IEEE Trans. on Computers, 1990.