QS-NAS: Optimally Quantized Scaled Architecture Search to Enable Efficient On-Device Micro-AI

Morteza Hosseini, *Member, IEEE*, and Tinoosh Mohsenin, *Member, IEEE*

***Abstract*— Because of their simple hardware requirements, low bitwidth neural networks (NN) have gained significant attention over the recent years, and have been extensively employed in the state-of-the-art devices that seek efficiency and performance. Research has shown that scaled-up low bitwidth NNs can have accuracy levels on par with their full-precision counterparts. As a result, there is a trade-off between quantization (q) and scaling (s) of NNs to maintain the accuracy. To capture that trade-off, in this paper, we propose QS-NAS which is a systematic approach to explore the best quantization and scaling factors for a NN architecture that satisfies a targeted accuracy level and results in the least energy consumption per inference when deployed to a hardware–FPGA in this work. We first approximate the accuracy of a NN using a polynomial regression based on experiencing over a span of q and s. Then, we design a hardware that is scalable with P processing engines (PE) and M multipliers per PE, and infer that the configuration of the most energy-efficient hardware as well as its energy per inference for a NN $\langle q, s\rangle$ are, in turn, a function of q and s. Experiencing the NNs with various q and s over our hardware, we approximate the energy consumption using another polynomial regression. Given the two approximators, we obtain a pair of q and s that minimizes the energy for a given targeted accuracy. The method was evaluated on SVHN, CIFAR-10, and ImageNet datasets trained on VGG-like and MobileNet-192 architectures, and the optimized models were deployed to Xilinx FPGAs for fully on-chip processing. The implementation results outperform the related work in terms of energy-efficiency and/or power consumption, yet having similar or higher accuracy. The proposed optimization method is fast, simple, and scalable to emerging technologies. Moreover, it can be used on top of other AutoML frameworks to maximize efficiency of running artificial intelligence on edge devices.**

***Index Terms*— Quantized scaled neural networks, neural architecture search, energy optimization, hardware, accelerator.**

I. Introduction

With the rapid growth of the computational ability of processors, convolutional neural networks (CNNs) have evolved to the point where they can surpass human-level accuracy in many applications such as physiological data processing, speech recognition and computer vision [1]–[6]. With all the advancements in their accuracy out-performance, nevertheless, the energy consumption of CNNs on electronic devices is still far away from any levels comparable to their biological paradigms [7].

Manuscript received May 16, 2021; revised August 4, 2021 and October 4, 2021; accepted October 18, 2021. Date of publication November 15, 2021; date of current version December 13, 2021. This article was recommended by Guest Editor X. Zhang. *(Corresponding author: Morteza Hosseini.)*

The authors are with the Computer Science and Electrical Engineering, University of Maryland, Baltimore County, Baltimore, MD 21250 USA (e-mail: hs10@umbc.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JETCAS.2021.3127932.

Digital Object Identifier 10.1109/JETCAS.2021.3127932

General-purpose CPUs and GPUs can efficiently test CNNs with precisions ranging from 8-bit fixed-point to 64-bit floating-point for inference [8]. Even though they can handle SIMD (Single Instruction Multiple Data) instructions such as XNOR and population-count to implement extremely quantized NNs such as binarized neural networks (BNNs) [9], they are not yet as versatile as FPGAs to efficiently implement CNNs with arbitrary quantization [10]. On the contrary, FPGAs are commercial off-the-shelf devices that allow the implementation of any arithmetic with arbitrary precision using any desired implementation styles from serial to fully-parallel. Additionally, they have built-in algorithms that can concatenate and activate as many required block RAM (BRAM) primitives and hardware components as opted, and in many cases, are used for prototyping and proof-of-concept development.

When the hardware efficiency is a subject of matter, metrics such as inference/J for the same application seem more fair of a comparison than metrics such as TOPJ that are subjective to quantization level and much to the advantage of low bitwidth CNNs. Thus, commitment to a specific quantization level with the sole aim of maximizing the TOPJ, rather inference/J, might be a misleading avenue of research particularly when there are multiple options for quantized arithmetic implementation. *In this paper, we raise a problem as: given a choice of CNN that has a few degrees of freedom implemented on a hardware that also has a few degrees of freedom, what is the best selection of the independent variables that meets an implementation goal?* A grid-search as depicted in Fig. 1 over a span of variously quantized scaled CNNs experienced on hardware can provide options to trade off between accuracy and efficiency, but not necessarily the most optimal options. We present QS-NAS which is a systematic approach to explore the best quantization and scaling factors for a neural network architecture that results in the least energy consumption per inference and meets a targeted accuracy when deployed to a hardware. The main contributions of this paper include:

- A systematic methodology referred to as QS-NAS that relies on both experimental and analytical methods to optimize energy consumption of CNNs on FPGAs.
- Precise energy measurement of fundamental components in an accelerator hardware designed for q-bit arithmetic and targeting Xilinx FPGAs (28nm).
- Two polynomial regression methods that rely on characteristics of CNNs, to predict the accuracy of quantized scaled CNNs, and their energy consumption on hardware.
- A scalable hardware that, if properly engineered, implements the same image classification task with comparable

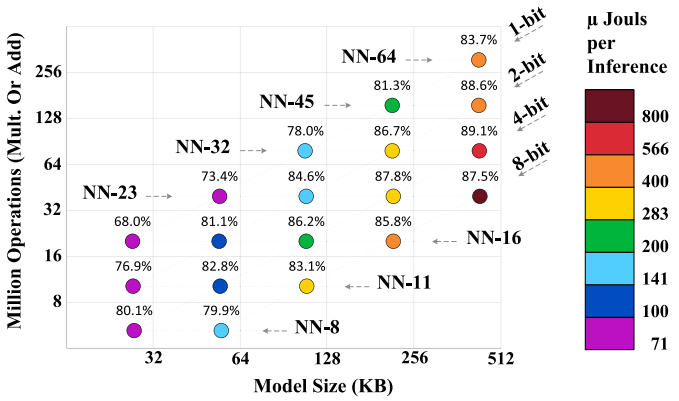

Fig. 1. A grid-search to explore an efficient implementation over VGG-like CNNs with various q and s, highlighted with model size, computation and accuracy for CIFAR-10, and color-coded with their energy consumption per inference when deployed to a tiny low-power Xilinx FPGA from ZedBoard.

accuracy on a tiny low-power low-cost FPGA that yields the highest efficiency as compared to the literature.

The rest of this paper is organized into the following sections: Section II underlines the background and motivations of this paper. Section III overviews the related works. Section IV introduces and formulates the problem statement of the QS-NAS. In Section V, the neural network design space of the QS-NAS is proposed. In Section VI a hardware design to implement each of the proposed neural network architectures for inference is detailed. Section VII formulates the characteristics of the hardware design under the topic hardware space. Section VIII navigates through the two neural network and hardware spaces to propose an optimal solution that meets the accuracy of the neural network and the minimum energy of the hardware. Section IX compares our work to the related work. Finally, Section X presents concluding remarks.

II. Motivation and Background

In this Section, we experiment with a toy problem of training a multilayer perceptron (MLP) on the MNIST [11], which is a dataset of 60,000 28×28 pixel (8-bit) grayscale images of handwritten digits of 0 to 9. Through this experiment we show that a grid-search within an exploration space can pinpoint desirable options for an implementation goal, yet hinting at a question as to which direction of the grid should one navigate to explore even better options? We also study the hardware characteristics of fundamental components when synthesized on FPGAs to infer relations between their energy consumption and their precision that will be used in our regression-based model later described in Section VII. Our MLP for the MNIST has a scalable architecture as:

$$(in:784)-(392s\ FC)-(392s\ FC)-(10\ FC)-(out:10), \tag{1}$$

which includes three FC (fully-connected) layers with $784\times 392s$, $392s\times 392s$, and $392s\times 10$ synapses respectively. In addition to the parameter s, we add another parameter quantization (q) to all weights, activations, as well as the input pixels. We intentionally quantize MNIST pixels in this Section to unify all operations in its MLP and to precisely measure energy of q-bit operations. We won't quantize input pixels of CIFAR and ImageNet datasets, as it causes severe information loss. We will show their differently quantized input layers will be implemented differently in hardware. To quantize the input pixels of the MNIST dataset, each 8-bit pixel is truncated to its q MSBs (most significant bits). We train this MLP for 4 experiments given by a cross-production of subsets $q \in \{2,4\}$ and $s \in \{0.5,1\}$, and measure the accuracy of each experiment. To measure implementation energy of this MLP on hardware, we design a simple hardware consisting of a $784q$ input buffer for input fmap, another $392qs$ output buffer for output fmap, one large block of memory with width $392qs$ and depth $(784+2\times 392s+10)$ to store the weight values, one array of $392s$ q-bit multipliers, and a tree of $392s$ adders whose input widths progressively increase from $2q$ through the stages of the adder tree. We implement this design on a tiny FPGA from the ZedBoard, deploy the workloads for the 4 experiments, and measure the power, delay, and consequently the energy per inference of the FPGA chip using corresponding test-benches that provide precise toggle rate and timing of the implementation. Table I shows the accuracy, energy per inference, and resource utilization of the 4 experiments implemented on the FPGA 7Z020 from the ZedBoard at 143MHz, that captures the trade-off between scaling and quantization of the MLP for accuracy and energy consumption. The Table implies that if, for instance, an MLP$\langle q,\ s\rangle$ and a corresponding hardware is provided and an MNIST accuracy of nearly 97.8% with minimum energy consumption is of interest, it is not easy to decide which pairs of $\langle q,\ s\rangle$ to select unless a grid space of $\langle q,\ s\rangle$ is methodically navigated through both neural network and hardware spaces.

Through this experiment, we also measure the energy consumption of fundamental modules in an accelerator hardware designed for q-bit arithmetic, including q-bit multipliers, q-bit adders, average adders in a large tree of q-bit adders, and q-bit read/write from FPGA BRAMs. Fig. 2 shows the aforementioned components and demonstrates their energy consumption and resource utilization per operations. Fig. 2 also discloses that for $1\le q\le 8$ both resource utilization and energy per operation of LUT-based combinational multipliers and adders in the Xilinx FPGAs are proportional to q^2 and q respectively. More interestingly, it experimentally shows that the average energy per addition in a large adder tree with q-bit inputs is proportional to $(q+1)$. This can be analytically proven as well: consider an adder tree that sums up $2M$ q-bit numbers in $log_2 M$ stages as depicted in Fig. 2-(C). The tree has $(2M-1)$ adders in total. The first stage consists of M adders with q-bit inputs, the next stage consists of $M/2$ adders with $(q+1)$-bit inputs, ... and the last stage consists of one adder with two $(q+log_2 M)$-bit inputs and one $(q+1+log_2 M)$-bit output. Given that the power of each adder is proportional to its input width, the total power of the tree is proportional to $qM+(q+1)M/2+(q+2)M/4+\ldots+(q+log_2 M)$ that approaches $(2q+2)M$ for large values of M. Thus, by averaging the calculated total energy over the $(2M-1)$ adders, the average energy per operation is proportional to $(q+1)$.

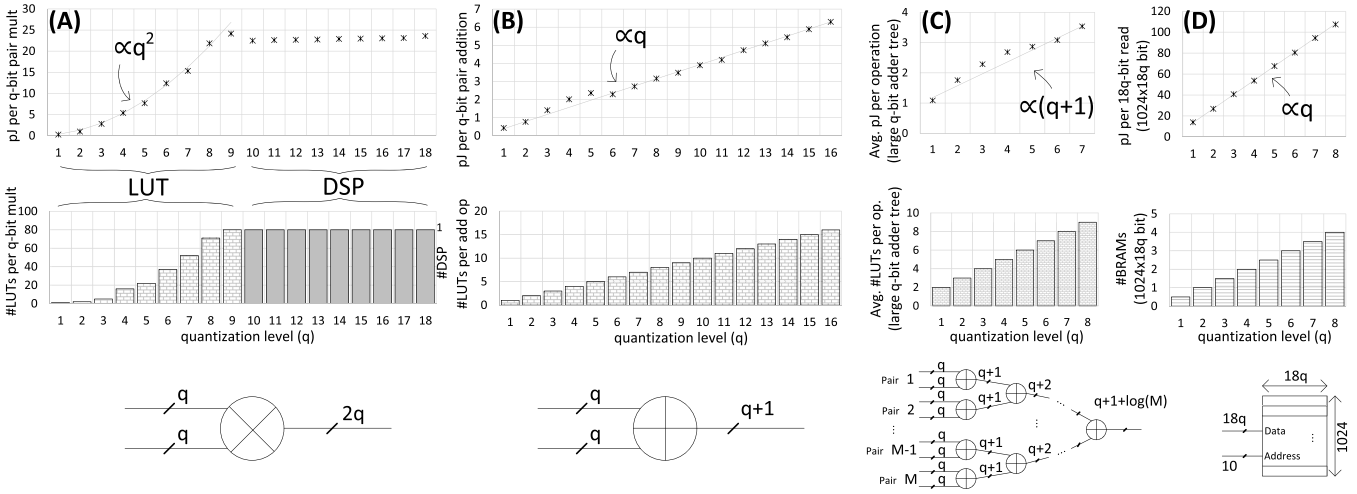

Fig. 2. Energy consumption and resource utilization per q-bit operations on Xilinx FPGA 7VX690T (28nm) including (A) a q-bit multiplier, (B) a q-bit adder, (C) an average adder in a large tree of q-bit adders, and (D) an $1024\times18q$ memory instantiated with 36Kb BRAMs. Exception: for q=1 the multiplier is an XNOR gate with a 1-bit output.

TABLE I

IMPLEMENTING MLP:784-392s-392s-10 FOR MNIST USING 392s Q-BIT MULTIPLIERS AND A 392s-ADDER TREE ON FPGA 7Z020 AT 143MHz

q	s	LUT	BRAM	Delay	Power (W)		Energy (uJ)		FPS/W	Acc.
		#	#	(us)	dyn.	total	dyn.	total	(inf/J)	(%)
2	0.5	1,717	22	8.30	0.19	0.32	1.55	2.65	377k	96.53
4	0.5	4,976	44	8.30	0.51	0.63	4.21	5.25	191k	97.71
2	1.0	3,406	44	11.05	0.38	0.51	4.16	5.59	179k	97.90
4	1.0	9,934	87.5	11.05	1.02	1.15	11.28	12.69	78k	98.23

TABLE II

ENERGY CONSUMPTION (PJ) PER q-BIT OPERATION ON XILINX FPGA (28nm) FROM 7Z020. THE MEMORY IS ONE SINGLE 18K-BIT BRAM

Quantization (q)	1	2	3	4	5	6	7	8
Addition	0.43	0.77	1.41	2.02	2.37	2.30	2.73	3.16
Multiplication	0.27	0.98	2.80	5.39	7.70	12.39	15.33	21.84
Mem. Read	0.75	1.50	2.25	3.00	3.76	4.51	5.26	6.01
Mem. Write	0.76	1.53	2.29	3.06	3.82	4.59	5.35	6.12

Table II tabulates the numerical results of the Fig. 2 that can be used as a reference for a back-of-envelope estimation of on-chip implementation of q-bit arithmetic on Xilinx FPGAs.

III. RELATED WORK

In light of minimizing energy consumption and/or maximizing performance of CNNs on hardware, model complexity reduction methods such as pruning [12], [13], quantization [8], [14], and compact CNN design [6], [15] have been proposed over the recent years, in which the mathematical functions of CNNs are altered to simpler and less compute-intensive forms. Many of quantization efforts take the parameter quantization as a variable to optimize to the point at which accuracy is maintained [16], [17]. Also, many hardware-oriented methods take quantization fixed as a priori. For instance, Yang *et al.* [18] adopted a 4-bit quantization in advance to start off with proposing an algorithm-hardware co-design for efficient CNN accelerators. Recently, with the advent of BNNs [9] and ternary neural networks (TNNs) [19], an enormous attention of the research community has been dedicated to the design of energy-efficient or high-performance accelerators for low bitwidth CNNs on hardware [10], [19]–[22]. Zhao *et al.* [10] introduced a BNN accelerator architecture with variable-width line buffers to exploit binary arithmetic and explored the potentiality of reduced storage requirements belonging to BNN feature maps (fmaps). Also, a flexible framework called FINN is brought forth by the work of in Umuroglu *et al.* [21] that fully exploits the FPGA on-chip memory resources to implement fast inference of BNNs. The goal of achieving the sweet-spot between resource utilization and precision has been analyzed in detail in the work of Prost-Bouscle *et al.* [22] where the authors suggested that TNNs, when properly trained, can obtain the performance of state of the art implementations. Most recently, the QKeras library [23] is introduced as an extension of the existing Keras library [24] that facilitates design and deployment of heterogeneously quantized versions of CNN models onto FPGAs.

Moreover, neural architecture search (NAS) has emerged as a recent methodology that relies on search strategies to manually or automatically explore CNNs with a defined goal–commonly increasing the accuracy. HotNAS [25] is a fast method that integrates a compression space during its co-search, and takes advantage of a set of existing pre-trained models to reduce the typical search time from 200 GPU hours to less than 3 GPU hours, and achieves up to 3.97% Top-5 accuracy gain on ImageNet dataset implemented on a Xilinx FPGA within the timing constraint of 5 mS. HAO (Hardware-aware Neural Architecture Optimization) [26] is another NAS algorithm that incorporates integer programming into its search algorithm to prune the design space and achieves a solution with 72.5% top-1 accuracy on ImageNet at frame rate of 50 FPS (frames per second), which is more than 60% faster than comparable related works. A recent approach involving NAS proposed Codesign-NAS [27] that investigates

different reinforcement learning based strategies to automatically navigate the search space of CNNs and hardware architectures to simultaneously improve accuracy and efficiency of image classification tasks (e.g. CIFAR-10 and CIFAR-100 datasets) on FPGAs. In [28], a simultaneous FPGA/CNN co-design methodology is proposed whose implementation results on a PYNQ-Z1 FPGA for an object detection task outperform similar works in terms of Intersection-over-Union, FPS, power consumption, and energy efficiency. Given all the benefits that these approaches grant when targeting FPGAs, they often disregard that a scaled-up lower-quantized (or vice-versa) CNN might enjoy higher efficiency on hardware while maintaining the same accuracy–A goal we attempt to address in this paper.

IV. Problem Statement

To simplify and unify formulations in this work, we consider a CNN block composed of a 2D convolution with weights and input quantized to q bits, followed by a batch-normalization layer, and a quantizing ReLU activation function as instructed in [23]. We define this block in whole as a Conv layer i with the function $\mathcal{Y}_i = \mathcal{F}_i(\mathcal{X}_i)$, where $\mathcal{X}_i$ and $\mathcal{Y}_i$ are input and output tensors with shapes $\langle \hat{Q}_{\mathcal{X}_i}, \hat{H}_{\mathcal{X}_i}, \hat{W}_{\mathcal{X}_i}, \hat{C}_{\mathcal{X}_i} \rangle$ and $\langle \hat{Q}_{\mathcal{Y}_i}, \hat{H}_{\mathcal{Y}_i}, \hat{W}_{\mathcal{Y}_i}, \hat{C}_{\mathcal{Y}_i} \rangle$ respectively, and $\mathcal{F}$ has a 2D convolution with shape $\langle \hat{Q}_{\mathcal{F}_i}, \hat{N}_{\mathcal{F}_i}, \hat{H}_{\mathcal{F}_i}, \hat{W}_{\mathcal{F}_i}, \hat{C}_{\mathcal{F}_i} \rangle$, and a batch normalization layer in the form of $\mathcal{X} \leftarrow \zeta_c(\mathcal{X} - \alpha_c)$ that has 2 compressed parameters (α_c and ζ_c) per channel of the convolution layer it follows. Note that we account for the quantization level as another dimension of every tensor shape. Therefore, $\mathcal{X}_i \in 2^{\hat{Q}_{\mathcal{X}_i} \times \hat{H}_{\mathcal{X}_i} \times \hat{W}_{\mathcal{X}_i} \times \hat{C}_{\mathcal{X}_i}}$ and $\mathcal{F}_i \in 2^{\hat{Q}_{\mathcal{F}_i} \times \hat{N}_{\mathcal{F}_i} \times \hat{H}_{\mathcal{F}_i} \times \hat{W}_{\mathcal{F}_i} \times \hat{C}_{\mathcal{F}_i}}$. We then consider a CNN is made by stacking L of these blocks and, to further unify, the quantization level of all layers and fmap, except the input data and the output classification labels, is equal to $\hat{Q}$. Thus, with a little abuse of notation that disregards the two differently quantized layers, we define a baseline CNN as follows:

$$\mathcal{NN}\langle \hat{Q},\, 1\rangle = \bigodot_{i=1\ldots L} \mathcal{F}_i^{\langle \hat{Q},\, \hat{N}_i,\, \hat{H}_{F_i},\, \hat{W}_{F_i},\, \hat{C}_i\rangle}(\mathcal{X}_i^{\langle \hat{Q},\, \hat{H}_{X_i},\, \hat{W}_{X_i},\, \hat{C}_i\rangle}) \quad (2)$$

where $\odot$ denotes that block $\mathcal{F}_i$ with different shapes is repeated L times while sequentially passing and reshaping the fmap $\mathcal{X}_i$ from the first layer to the next layers. To compensate the accuracy loss resulted by low quantization, we impose one more scaling variable (s) to each 2D convolution layer of the CNN, as illustrated in Fig. 3, and reformulate it to $\mathcal{NN}\langle q,\, s\rangle$. For simplicity in formulations, the scaling factor in this work pertains to only the width of a CNN where it scales only the number of all filters per layers, and correspondingly the number of channels per filter and fmap. In a more sophisticated scenario that we don't practice in this paper, scaling is recommended to be applied to the depth and input resolution of a CNN in tandem with its channels (width) [5].

Given an adequate choice of FPGA, there are many implementation styles for the CNN that may meet an application requisite such as execution time, power, performance, and efficiency. We consider a commonly-practiced hardware design consisting of a 2D grid of $P \times M$ multiply-accumulate (MAC) units, and define the following optimization problem:

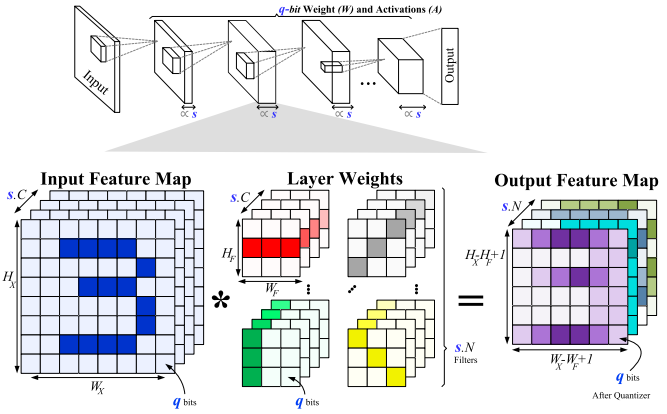

Fig. 3. (top) A neural network architecture scaled to s and quantized to q, and (bottom) a convolution layer with filters and channels scaled by s, and q-bit precision per weight and fmap values. The fmap is quantized after a quantizer.

TABLE III
Parameters to Explore for Optimization

Parameter	Description
q	quantization level of both weights and feature-map
s	scaling factor for number of filters and channels per layer
P	number of processing engines in hardware
M	number of multipliers (and adders) per processing engine

$$\begin{aligned} \min_{q,\, s,\, P,\, M} \quad & Energy(\,\mathcal{HW}\langle P,\, M\rangle \mid \mathcal{NN}\langle q,\, s\rangle\,) \\ s.t. \quad & \mathcal{NN}\langle q,\, s\rangle = \bigodot_{i=1\ldots L} \mathcal{F}_i^{\langle q,\, s\,\hat{N}_i,\, \hat{H}_{F_i},\, \hat{W}_{F_i},\, s\,\hat{C}_i\rangle} \\ & (X^{\langle q,\, \hat{H}_{X_i},\, \hat{W}_{X_i},\, s\,\hat{C}_i\rangle}) \\ & Accuracy(\mathcal{NN}) \geq \text{target\_accuracy} \end{aligned} \quad (3)$$

that reads: minimize the energy per inference of CNN deployed to hardware w.r.t. q, s, P, and M, provided that a target_accuracy is satisfied. The solution to this problem lies in understanding the behaviour of both the $Energy$ and the $Accuracy$ functions. Based on the evidence and experiments we use approximators to predict the behaviour of the two functions, and solve the problem through the following steps:

- (1) Experience with CNNs on a limited span of q and s.
- (2) Adopt, fit, and evaluate an approximator for step (1).
- (3) Design a hardware for an implementation goal & metric.
- (4) Measure the metric for deployed CNNs from step (1).
- (5) Adopt, fit, and evaluate another approximator that predicts behaviour of the hardware for the measured metrics.
- (6) Optimize the problem w.r.t. the two approximators.

Table III summarizes the optimization parameters and Fig. 4 shows a top view of our proposed QS-NAS methodology.

To further simplify our problem statement, we assume: NNs are deep. For $q = 1$, we consider BNNs, where multipliers are swapped with XNOR gates, and 0/1 bit-values represent $-1/+1$ weight and fmap values [9]. The only layers that do not follow the uniform quantization and scaling are the input data and the classification labels from the first/last layers. Thus,

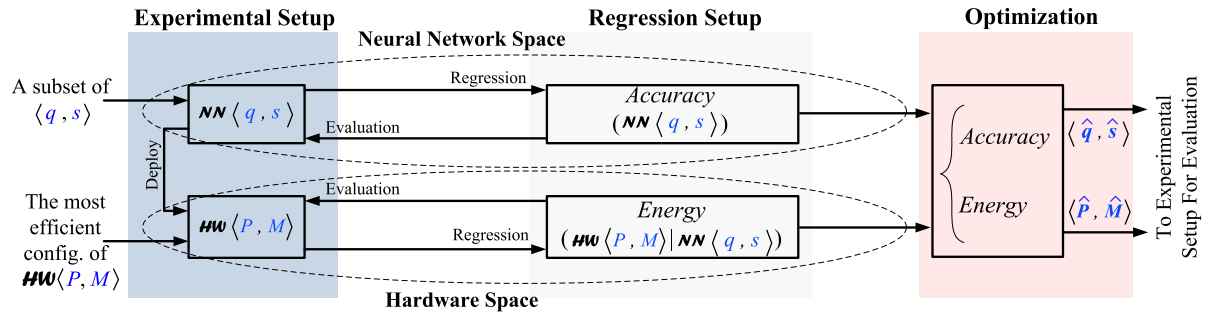

Fig. 4. A high-level diagram of the proposed QS-NAS experimental-analytical methodology, highlighting the major steps.

TABLE IV
CHARACTERISTICS OF WORKLOADS VGG AND MOBILENETV1-192

	Size (million bits)		Million Ops per Layer Type		
	Model	Largest Fmap	1st Layer	DW ($K \times K$)	Vanilla Layers (CONV or FC)
VGG $\langle q, s \rangle$	$3.7qs^2$	$0.06qs$	$3.5s$	-	$298s^2$
MobileNet-192 $\langle q, s \rangle$	$4.2qs^2$	$0.59qs$	$3.2s$	$26s$	$794s^2$

neglecting the impact of the first/last layers for a $\mathcal{NN}\langle q, s\rangle$, we conclude:

$$\begin{aligned} Model\_Size(\mathcal{NN}\langle q, s\rangle) &\propto q \,.\, s^2 \\ Largest\_Fmap\_Size(\mathcal{NN}\langle q, s\rangle) &\propto q \,.\, s \\ Operations(\mathcal{NN}\langle q, s\rangle) &\propto s^2 \\ Mult\_Op\_Cost(\mathcal{NN}\langle q, s\rangle) &\propto q^2.\, s^2 \\ Add\_Op\_Cost(\mathcal{NN}\langle q, s\rangle) &\propto q \,.\, s^2 \end{aligned} \tag{4}$$

The sizes of the model and the largest fmap of our CNN govern the sizes of the weight memory and fmap memory of our scalable hardware respectively, that will be explained in Section VI. The computation is also proportional to s^2. Since $q \leq 8$ in this paper, as detailed in Section II, we observed that look-up table (LUT)-based MAC components are more energy-efficient than FPGA's built-in digital signal processor (DSP) units, and note that the power and utilization costs of LUT-based combinational multipliers and adders in Xilinx FPGAs are proportional to q^2 and q respectively. We put aside the limited amount of DSPs of the FPGA to implement the batch-normalization operations.

V. NEURAL NETWORK SPACE

A. Experiments: Training CNNs With Different q & s

We perform our experiments on three datasets, CIFAR-10 [29] and SVHN [30], that contain 32×32 RGB images, as well as ImageNet [31] that contain large-sized RGB images and are used for benchmarking methods. CIFAR-10 includes 50K and 10K training and testing images respectively, that contain 10 classes of animals and vehicles. SVHN consists of 604K and 26K images of digits cropped from street view images for training and testing respectively. ImageNet consists of approximately 1 million and 50,000 natural images for training and testing categorized in 1000 classes. For CIFAR-10 and SVHN datasets, we selected a VGG-like architecture [32], and for the ImageNet, we used MobileNetV1 [6]. To borrow simplicity in proposing our method and formulations, we purposefully adopted these architectures as they easily match our scalable setup. The exact VGG-like architecture has been extensively used in [9], [10], [19], [21], [22] with various q and s. For MobileNet, all publicly available models are already standardized with channel-scaled variants that make them ideal candidates for our approach, for each of which there exists 8-bit [33] and floating-point [6] models.

The VGG-like networks has the following architecture:

$$\begin{aligned} &(in) - 2 \times (64s\, C_{3\times3}) - Pool_{2\times2} - 2 \times (128s\, C_{3\times3}) - Pool_{2\times2} \\ &- 2 \times (256s\, C_{3\times3}) - Pool_{2\times2} \\ &- 2 \times (512s\, FC) - 10FC - (out), \end{aligned} \tag{5}$$

where $C_{3\times3}$ and FC represent convolution (CONV) and fully-connected layers. The parameters q (quantization) is implicitly included for weight and activation of all layers (in/out data excluded). Also, the parameter s is clearly shown to have been used to scale the number of filters per layer. Throughout this work, we denote this VGG-like CNN with either $\mathcal{NN}\langle q, s\rangle$ or with q-bit NN-64s. This CNN has approximately $2 \times 149s^2$ million multiply or add operations and $3.7s^2$ million parameters, thereby having a model size $3.7qs^2$ Mb. The largest fmap size is also contributed by its second layer of size $64qs$ Kb.

For ImageNet, we used MobileNetV1 [6] with architecture:

$$\begin{aligned} &(in) - 32sC_{3\times3} - 32sDW_{3\times3} - 64sDS_{3\times3} - 2 \times (128sDS_{3\times3}) \\ &- 2 \times (256sDS_{3\times3}) - 6 \times (512sDS_{3\times3}) - 1 \times (1024sDS_{3\times3}) \\ &- 1 \times (1024sC_{1\times1}) - AvgPool - 1000FC - (out) \end{aligned} \tag{6}$$

where $DW_{3\times3}$ are depthwise convolution layers, and $DS_{3\times3}$ are depthwise separable layers, each composed of one $C_{1\times1}$ followed by a $DW_{3\times3}$ layer. Table IV summarizes the $Largest\_Fmap\_Size$ and $Model\_Size$, as well as the $Operations$ breakdown of the two CNN architectures.

We train all our models using QKeras [23] libraries in 120 epochs, using Adam optimizer [34], general data augmentation techniques, and with an initial learning rate 0.001 that decays by 0.1 every 40 epochs. For the VGG-like architecture We down-scaled the CNN from 1 to 1/8th, and up-scaled the quantization from 1, which is a BNN, up to 8 bits, and jot down the accuracy for the 16-point experiments. For

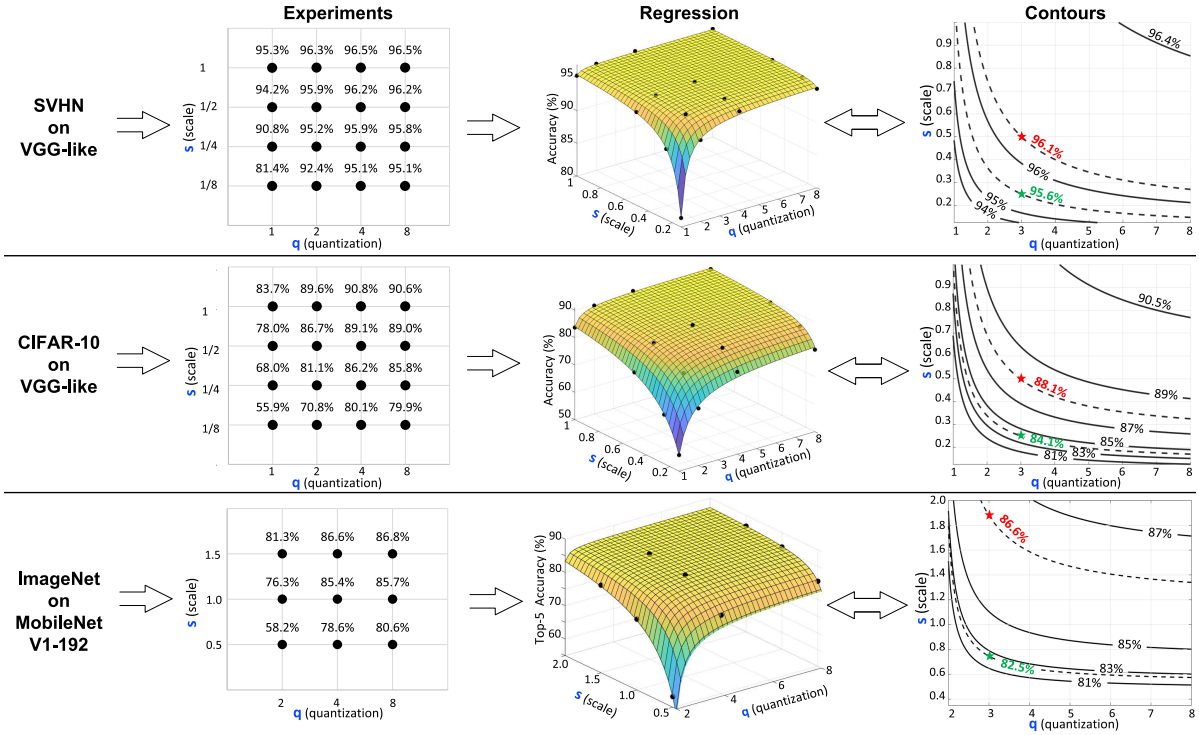

Fig. 5. Modeling the accuracy for SVHN on VGG, CIFAR-10 on VGG, and ImageNet on MobileNet from experiments, by experiencing $\mathcal{NN}\langle q, s\rangle$ for a subset of q and s, then using a regression method to fit to the experiments and from which concluding same-accuracy contours.

the MobileNet architecture, we selected $q \in \{2, 4, 8\}$ and $s \in \{0.5, 1.0, 1.5\}$, and jot down the Top-5 accuracy for the 9-point experiments. To unify and expedite our experiments, we trained all of the neural networks in this paper under the same training setup, and didn't employ extra fine-tuning methods. Consequently, the Top-5 accuracy of our 8-bit MobileNets are within a margin of 2.6% (see [33]) and 3.5% (see [6]) loss as compared to their corresponding Google TFlite (8-bit) and floating-point models respectively. Figs. 5-(left) shows the results of the cross-product experiments for the three datasets.

B. Regression on the Accuracy

It is well-studied that scaling CNNs increases their accuracy [5], [6], and that the accuracy of quantized models degrades to their least when the quantization is lowered to BNNs [10], [21]. Clearly, the accuracy does not exceed a certain level and will saturate to a level no more than 100%. Based on the evidence and our experiments, we postulate that a rational polynomial, as follows, can approximate the accuracy function with respect to q and s, as it satisfies all the mentioned characteristics.

$$Accuracy(\mathcal{NN}\langle q,\ s\rangle) \approx \frac{\hat{A}_6.q.s\ +\ \hat{A}_5.s\ +\ \hat{A}_4.q\ +\ \hat{A}_3}{q.s\ +\ \hat{A}_2.s\ +\ \hat{A}_1.q\ +\ \hat{A}_0} \tag{7}$$

where $\hat{A}_i$ are learnable constant parameters. Using the MATLAB cftool, we got a least-square-error method to fit this function to our cross-product (16-point and 9-point) experiments. Fig. 5 (middle) shows how the fitted surfs look like. For the CIFAR-10 and the SVHN, the root mean square error (RMSE) is 0.74% and 0.18%, and the mean absolute error (MAE) is 0.56% and 0.13% respectively, which indicate a good fit. We evaluate this regression on two new testing data, i.e. $\mathcal{NN}\langle q = 3, s = 1/4\rangle$ and $\mathcal{NN}\langle q = 3, s = 1/2\rangle$, that predicts 84.1% / 95.7% and 88.1% / 96.1%, whilst actual values are 84.2% / 95.9% and 88.7% / 96.2% for CIFAR-10 / SVHN datasets respectively. Table VI summarizes the quality factors of the regressions for the three *Accuracy* experiments, and Table VII evaluate each regression on 2 new data points by comparing their predicted and actual values.

As an interesting conclusion from the regressions, Fig. 5-(right) demonstrates that by plotting the accuracy contours for the fitted surfs, a better visualization is obtained for how the accuracy of the three models behave with respect to q and s.

TABLE V

PERFORMANCE OF HARDWARE FOR DIFFERENT LAYER TYPES. FOR BOTH WORKLOADS VGG$\langle q, s\rangle$ AND MOBILENET$\langle q, s\rangle$, THE HARDWARE IS CONFIGURED TO $P = M = 64s$

Peak Performance of $\mathcal{HW}\langle P, M\rangle$ for Layer	Layer Type		
	1st Layer (input with C channels)	DW ($K \times K$)	Vanilla Layers (CONV or FC)
Ops/clk	$2 \times P \times C$	$2 \times P \times K$	$2 \times P \times M$

VI. HARDWARE DESIGN

All hardware designs in this paper are implemented on either ZedBoard or VC709 evaluation platforms. The former platform is equipped with the Xilinx FPGA XC7Z020-CLG484 that incorporates 4.9Mb (=140 × 36Kb)

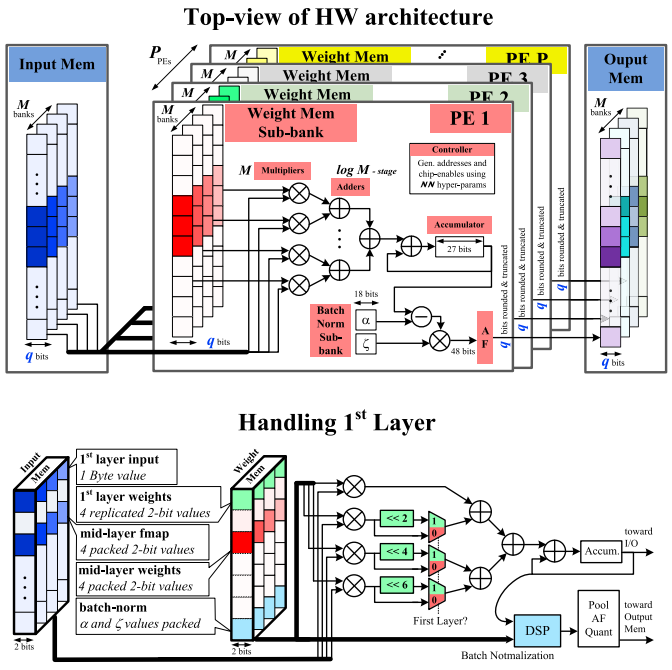

Fig. 6. (Top) QS-NAS hardware ($\mathcal{HW}\langle P, M\rangle$) configurable with P processing engines and M multipliers/adders per PE for the deployment of a $\mathcal{NN}\langle q, s\rangle$. (Bottom) handling the first heterogeneously quantized layer within the same pipeline that implements the other uniformly quantized mid-layers.

on-chip BRAMs, and the latter is equipped with the Xilinx FPGA 7VX690T that incorporates 52.9Mb (=1470 × 36Kb) on-chip BRAMs. Also, using the Xilinx Vivado Design Suite, in all our analyses corresponding test-benches were used that provide precise toggle rate and timing for precise FPGA chip power and latency measurements respectively. Throughout this work, the term 'Performance' indicates the amount of multiplication or addition operations an accelerator is able to carry out in a second and we measure it in terms of GOPS (giga operations per second). Meanwhile, the term 'Efficiency' indicates the amount of multiplication or addition operations an accelerator is able to perform per unit of energy, i.e. Joule, and we measure it in terms of GOPJ (giga operations per joule).

A. Scalable Design

Similar to works of [10], [19], [21], [22], we design a hardware accelerator for FPGAs that relies on FPGA's resources and BRAMs that store the weight and intermediate fmap of $\mathcal{NN}\langle q, s\rangle$ during run-time. Our proposed hardware, depicted in Fig. 6-(top) and described in Verilog HDL, comprises two main blocks: 1) an array of P processing engines (PEs) that each incorporates M multipliers/adders and a weight memory sub-bank that stores a partition of the CNN weights, and 2) an input and an output memory that swap turn per process termination of every CNN's layer and store the temporary fmap data. Both the partitioned CNN weights and the fmap data are packed along their channels for every layer and are stored and folded in one or multiple entries in their designated BRAMs. As a result, the CNN weights are partitioned in weight memory sub-banks each with width Mq and depth $Model\_Size/P/M/q$. Correspondingly, each of the two input/output memories has width Mq and depth $Largest\_Fmap\_Size/M/q$. For the max-pool, a 4-entry LUT-RAM buffers 2-by-2 patches and bubble sorts them in 3 clock cycles to determine the maximum value in the pool. We note that fully-connected (FC) layers from Eqn. (5) are corresponding to convolution layers operated with valid size. Thus, handling them is equivalent to handling convolution layers with a nuance in their state machine.

1) Vanilla FC/CONV Layers: The computation tiling scheme for regular FC/CONV layers is as follows: each PE concurrently computes one output channel from a layer of the CNN at a time, while internally parallel-processes its task using an input-channel tiling scheme where the packed fmap channel values directly accessed from the input memory, and the packed weight channel values from the weight sub-banks are element-wise multiplied and accumulated. The performance of the hardware over a CNN layer with N filters and C channels operating at a clock rate $freq$ is $2.min(N, P).min(C, M).freq$. Thus, the average performance of the hardware would be $2.P.M.freq$ provided all N_i and C_i from the layer i of the CNN are multiples of P and multiples of M respectively.

2) Depthwise Layers: Due to having a low arithmetic intensity, implementing depthwise convolution layers on hardware delivers poor scalability and inferior performance as opposed to the vanilla convolution layers [35]. To embed the DW layers in our hardware design, each PE adopts one DW channel to implement as follows: all $K \times K$ values of the DW channel are stored along one entry of the weight memory of the PE in charge. A $K \times K$ patch of a corresponding input fmap is buffered in a $K \times K$ array of flip flops that are updated with the next consecutive patch of the fmap fetched from the input memory after every K clock cycles. Using multiplexers, the updated buffered patch is aligned with the $K \times K$ values along the weight memory entry, and is executed through the pipeline of the multipliers and adders after every K clock cycles. As a result, the performance of the hardware to implement a DW layer with C filters of $K \times K$ that takes an input data with C channels is $2.min(C, P).K.freq$.

3) The First Layer: To handle the first layer whose input is an RGB image (8 bits per pixel), we slightly alter the array of multipliers, that perform element-wise q-bit multiplications, to the extent that an 8-bit × q-bit operation through the same pipeline is performed by summing over shifted partial multiplications between q-bit chunks of the input pixel and replications of the q-bit weight. The performance of the hardware over a CNN layer with N filters that takes an input data with C channels is $2.min(N, P).C.freq$. Fig. 6-(bottom) illustrates our hardware alteration when configured for $q = 2$. If $q = 3$, then 9 bits are allocated per input data pixel.

Table V summarizes the performance of the proposed hardware for each of the three aforementioned types of layers. The peak performance is achievable when the parameters P and M are small enough to allow full utilization of the HW pipelines. When configuring this hardware, at least three goals can be pursued: the least power (P=1 and M=1), the highest performance ($P = \infty$ and $M = \infty$), and the most

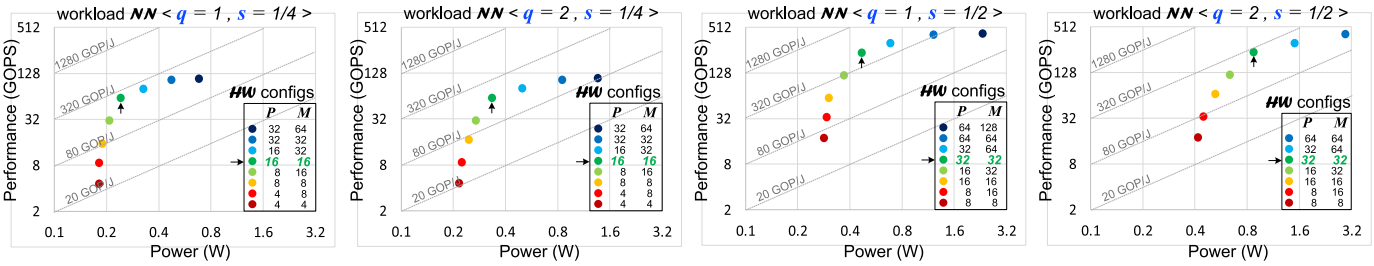

Fig. 7. Exploring the best configuration by tweaking the number of PEs (P) and the number of multipliers per PE (M) for the QS-NAS hardware design that yields the highest efficiency on FPGA (7Z020) given a deployed workload CNN (VGG). In each plot, the most efficient configuration is the green dot that is closest to the highest efficiency lines. From the four experiments, it is seen that for the most efficient hardware is given by $P = M = 64s$.

energy-efficient configuration that is given by a pair of P and M that will be explored in the next SubSection.

B. The Most Efficient Configuration

To explore the most energy-efficient hardware configuration, a roofline model is usually sought that pinpoints a ridge point where the maximum performance is achieved using the minimum hardware resource utilization [21]. The ridge point in the roofline model of our hardware is obtained per scaling of the hardware corresponding to the scaling of the CNN. Intuitively, if $\mathcal{HW}_{efficient}$ is the most energy-efficient configuration for a baseline $\mathcal{NN}$, then the most energy-efficient configuration for the hardware that implements $\mathcal{NN}\langle q, s\rangle$ is given by $P(\mathcal{HW}_{efficient}\,|\mathcal{NN}\langle q, s\rangle) \propto s$, and $M(\mathcal{HW}_{efficient}\,|\mathcal{NN}\langle q, s\rangle) \propto s$. To further support our claim and to find the $\mathcal{HW}_{efficient}$ for a baseline workload, we empirically tweak the hardware parameters for workloads VGG$\langle q, s\rangle$ within the range $1 \leq q \leq 2$ and $1/4 \leq s \leq 1/2$ as illustrated in Fig. 7. We observe that the most energy-efficient hardware configuration is obtained when P and M are equal to the CNN's smallest number of filters and channels per layer respectively, both of which are $64s$ (input layer excluded). Any other selection of P and M larger than $64s$ slightly increases the performance, but at the expense of under-loading the hardware pipelines, thereby deviating from the maximum efficiency. Comparably, for their VGG-like BNN with $q = 1$ and $s = 1$, authors in [21] selected $P = 64$ and $M = 128$, a configuration that is not under-loaded because of engineering their hardware (implemented on a large FPGA from VC706) to interleave filters and fmaps for the maximum performance. Conducting the same experiment, we observed that the most energy-efficient configuration hardware for MobileNet$\langle q, s\rangle$ is as well given by $P = M = 64s$.

For VGG implementation on the tiny FPGA from the ZedBoard, With $P = M = 64s$ and the $Model\_Size = 3.7qs^2$ Mb, the depth of every partitioned weight memory is calculated as $3.7qs^2/64s/64s/q$ Mega entries, which is a constant equal to 872 per simultaneously scaling the CNN and the hardware. Also, given the $Largest\_Fmap\_Size = 64qs$ Kb, the depth of the input/output memory units are equal to $64qs/64s/q$ Kilo entries, which is a constant 1024. Taking 1024 for the depth of all memory sub-banks in our hardware configurations, the total design requires approximately $[4096qs^2/36]$ and $2 \times [64qs/36]$ of 36Kb BRAMs for the weight memory and input/output memory allocation that justifies employing the tiny FPGA from the ZedBoard for the span of q and s in this work. Similarly, it can be shown that for fully on-chip implementation of the MobileNet, $[4096qs^2/36]$ and $24qs$ of 36Kb BRAMs are required to allocate to the weight memory and input/output memory.

VII. Hardware Space

A. Experiment: Deploying CNNs With Different q & s

The 16 VGG-like architectures (for CIFAR-10 and SVHN) with different q and s from Section V-A were implemented on the FPGA from the ZedBoard, and the power, delay, and energy per inference were measured for each deployment. Fig. 8-(top/left) shows the measurements of the experiments demonstrating that 4 of them over-utilized the resources of our selected tiny FPGA. Similarly, the 9 MobileNets for ImageNet were implemented on the FPGA from the VC709. Fig. 8-(bottom/left) highlighting the energy per inference measurements of the experiments and indicating 1 experiment over-utilizing the resources of the 7VX690T FPGA.

B. Regression on the Energy

In deriving a mathematical model for the delays and the energy of our hardware, the data transfer is a significant contributor. Particularly, when an off-chip memory is utilized, the delays caused by data marshalling between the FPGA and the off-chip memory are nontrivial. In our work and in similar related works (e.g. [19], [21], [22]) the inference is processed fully on FPGA's on-chip BRAMs. Thus, before processing a new input image, the model weights are stored on FPGA BRAMs, and, because it's a once-and-for-all data transfer, we don't account for the delay to load them onto the FPGA BRAMs. In the next Subsections, we model the $Performance$, $Power$, and the $Energy$ per inference of our hardware during the system run-time, i.e. after both the model weights and the input data are loaded and during the on-chip processing for an inference.

1) Modeling the Performance and the Execution Time: Having determined that a $\mathcal{HW}_{efficient}$ for all of our $\mathcal{NN}\langle q, s\rangle$ architectures in this work is given by $\mathcal{HW}\langle P = 64s, M = 64s\rangle$, it can be concluded:

$$Performance_{FC/CONV}(\mathcal{HW}_{efficient}\,|\mathcal{NN}\langle q, s\rangle) \propto s^2$$

TABLE VI

QUALITY OF THE REGRESSION OVER A LIMITED NUMBER OF POINTS (EXPERIMENTS) USING QUALITY FACTORS RMSE AND MAE FOR FUNCTIONS *Accuracy* AND *Energy*

Function	#Points	RMSE	MAE
Accuracy (VGG | SVHN)	16	0.2%	0.1%
Accuracy (VGG | CIFAR-10)	16	0.7%	0.6%
Accuracy (MobileNet | ImageNet)	9	0.1%	0.1%
Energy (ZedBoard | VGG)	12	9.7uJ	8.1uJ
Energy (VC709 | MobileNet)	8	1.3mJ	1.2mJ

$$Execution\_Time_{FC/CONV}(\mathcal{HW}_{efficient}\,|\mathcal{NN}\langle q,\ s\rangle) \propto \frac{s^2}{s^2} = 1 \tag{8}$$

that indicates when both the CNN and hardware are scaled by s, both the operations count of FC/CONV layers and the total number of multipliers in hardware are scaled by s^2 as well. Thus, the execution time to implement FC/CONV layer remains the same. Similarly, the operations count of the first layer as well as DW layers are proportional to s (Table IV), and so is the performance of the hardware for these layers (Table V). Thus, the execution time for implementing the 1st layer and DW layers and, consequently, the whole CNN workload remains constant. As a result, the execution time per inference and the throughput of all our VGG implementations on ZedBoard at clock rate 143 MHz is approximately 0.34 mS and 2,982 FPS respectively. Similarly, the execution time per inference and the throughput of all our MobileNet implementations on VC709 at clock rate 100 MHz is approximately 3.02 mS and 331 FPS respectively.

2) Modeling the Power: We model the power of the hardware with meticulous consideration upon full recognition of its components and their toggling rate (nearly 100% for both MAC and memory units for vanilla FC/CONV layers) during the run-time. We break down the power as follows:

$$Power_{\mathcal{HW}} \approx Power_{Memory} + Power_{Logic} + Power_{Static} \tag{9}$$

Given that $\mathcal{HW}_{efficient} = \mathcal{HW}\langle P = 64s,\ M = 64s\rangle$ for the workload $\mathcal{NN}\langle q,\ s\rangle$, and that the depth of all memory units are equal to 1024, with the scaling of the $\mathcal{HW}$, only the width of memory units scales. Consequently, each of the P weight memory sub-banks and the I/O memory widen by Mq. Therefore:

$$Power_{Memory} \approx \hat{B}_2.q\ .s^2 + \hat{B}_1.q.s \tag{10}$$

The major portion of the logic is contributed by the multipliers and adders that scale by PMq^2 and PMq respectively. Thus:

$$Power_{Logic} \approx \hat{B}_3.q^2.s^2 + \hat{B}_4.q\ .s^2 \tag{11}$$

By taking the static power into account and merging $\hat{B}_4$ and $\hat{B}_2$ from Eqns. (10) and (11), the $Power$ is modeled as follows:

$$Power(\mathcal{HW}|\mathcal{NN}\langle q,\ s\rangle) \approx \hat{B}_3.q^2.s^2 + \hat{B}_2.q.s^2 + \hat{B}_1.q.s + \hat{B}_0 \tag{12}$$

where $\hat{B}_i$ are constant parameters determined after regression.

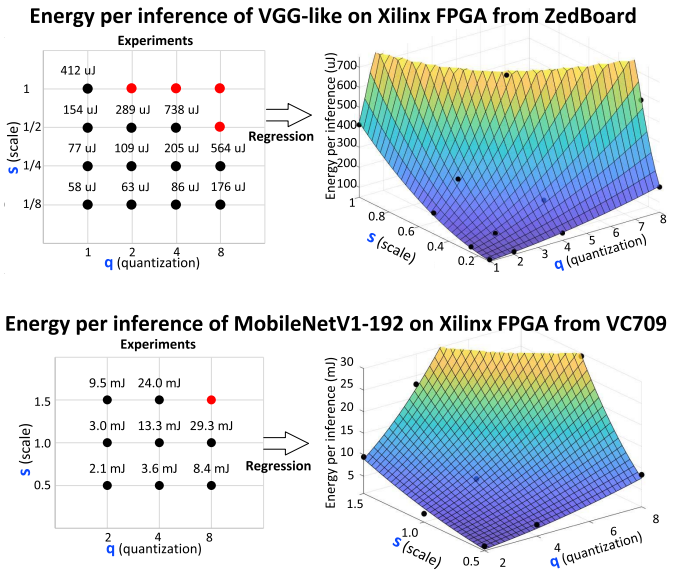

Fig. 8. (Left) energy per inference measured on hardware, and (right) a regression on function *Energy*, for (top) the 16 VGGs deployed to their efficiently-configured QS-NAS hardware on ZedBoard, and (bottom) the 9 MobileNets deployed to VC709. Four and one experiments (red dots) over-utilized the FPGAs from the latter and the former experiments respectively.

TABLE VII

EVALUATING THE REGRESSION ON FUNCTIONS *Accuracy* AND *Energy* WITH TWO NEW DATA POINTS. ALSO, THE TWO POINTS ARE THOSE PROPOSED BY THE OPTIMIZATION SETUP, AND THEIR ACTUAL VALUES ARE USED TO AMEND THE PREDICTED VALUES FROM THE PROPOSITION

New Point in Functions of $\mathcal{NN}\langle q,\ s\rangle$ | Dataset	$Accuracy(\mathcal{NN})$		$Energy(\mathcal{HW}\ \|\ \mathcal{NN})$	
	Predicted	Actual	Predicted	Actual
VGG $\langle 3,\ 1/4\rangle$ | SVHN	95.6%	95.9%	154 uJ	140 uJ
VGG $\langle 3,\ 1/2\rangle$ | SVHN	96.1%	96.2%	477 uJ	414 uJ
VGG $\langle 3,\ 1/4\rangle$ | CIFAR-10	84.1%	84.2%	154 uJ	140 uJ
VGG $\langle 3,\ 1/2\rangle$ | CIFAR-10	88.1%	88.7%	477 uJ	414 uJ
MobileNet $\langle 3,\ 0.75\rangle$ | ImageNet	82.5%	81.9%	3.95 mJ	5.13 mJ
MobileNet $\langle 3,\ 1.87\rangle$ | ImageNet	86.6%	86.5%	22.86 mJ	23.36 mJ

3) Regression on the Energy: Because the execution time remains constant, dependency of $Energy$ to q and s corresponds to that of the $Power$. Hence:

$$Energy(\mathcal{HW}|\mathcal{NN}\langle q,\ s\rangle) \approx \hat{E}_3.q^2.s^2 + \hat{E}_2.q.s^2 + \hat{E}_1.q.s + \hat{E}_0 \tag{13}$$

where $\hat{E}_i$ are constant parameters to be learned and determined from a regression. From another perspective and with reference to Eqn. (4), the first leftmost term reflects the computation energy resulted from multiplications, the second term is partially contributed by the computation from additions and partially by communication for the model weight parameters. The third term reflects the energy consumption for the fmap communication, and the last term is a result of the static power and components in hardware that are irrelevant to q and s.

We use Eqn. (13) to fit the 12-point and 9-point energy experiments from the deployed VGG and MobileNet. Fig. 8-(right) shows the fitted surfs. For the energy of the deployed VGG models, fitting the 12 points to our proposed polynomial in Eqn. (13) results in RMSE = 9.7uJ and

MAE = 8.1uJ. For evaluation, we tested the $Energy(\mathcal{HW}\langle P = 16, M = 16\rangle \mid \mathcal{NN}\langle q = 3, s = 1/4\rangle)$ and the $Energy(\mathcal{HW}\langle P = 32, M = 32\rangle \mid \mathcal{NN}\langle q = 3, s = 1/2\rangle)$ configurations, for which the polynomial predicts 154 uJ and 477 uJ, whereas their actual measurements are 140 uJ and 414 uJ respectively. Table VI summarizes the quality factors of the regressions for the two *Energy* measurements, and Table VII evaluate each regression on 2 new data points by comparing their predicted versus actual values.

VIII. Optimizing Energy per Inference

To this point, we approximated both of our unknown functions in Eqn. (3) w.r.t. to our two variables and established a system of non-linear equations:

$$\begin{cases} Energy(\mathcal{HW}|\mathcal{NN}\langle q,s\rangle) \\ \quad = \hat{E}_3.q^2.s^2 + \hat{E}_2.q.s^2 + \hat{E}_1.q.s + \hat{E}_0 \\ Accuracy(\mathcal{NN}\langle q,s\rangle) \\ \quad = \dfrac{\hat{A}_6.q.s + \hat{A}_5.s + \hat{A}_4.q + \hat{A}_3}{q.s + \hat{A}_2.s + \hat{A}_1.q + \hat{A}_0} \end{cases} \tag{14}$$

To minimize the *Energy* in this system of non-linear equations, we can take s as a function of q and *Accuracy* and plug it in the function *Energy* and solve the following equation:

$$\frac{\partial Energy}{\partial q}|_{Accuracy=target\_accuracy} = 0 \tag{15}$$

We used MATLAB optimization toolbox to plot and to solve the Eqns. (14) and (15). When plotting the function *Energy* w.r.t. to q and *Accuracy*, we obtain convex curves as depicted in Fig. 9-(right) that reveal different quantization levels result in the least energy consumption given different accuracy levels. We look for delicate pairs of q and s that result in moderately high (VGG$\langle q = 3, s = 1/2\rangle$ and MobileNet$\langle q = 3, s = 1.875\rangle$) and moderately low (VGG$\langle q = 3, s = 1/4\rangle$ and MobileNet$\langle q = 3, s = 0.75\rangle$) accuracy levels, such that q is a natural number near the minima of the convex curves, and s is such that $64s$ is a natural number (for hardware-friendly implementation), and the selection of both of which do not over-utilize the FPGA resources.

Evidently, both the $\mathcal{NN}$, given a dataset, and the $\mathcal{HW}$ are the key contributors in characterizing the two functions $Accuracy(\mathcal{NN}\langle q, s\rangle)$ and $Energy(\mathcal{HW}|\mathcal{NN}\langle q, s\rangle)$. An interesting case is observed for the VGG $\langle q, s\rangle$ given either of the SVHN or CIFAR-10 datasets, in which an identical neural network is used for two different datasets, and is implemented on the same hardware. In this case, if the two datasets have comparable distributions, the function $Energy(\mathcal{HW}|\mathcal{NN}\langle q, s\rangle)$ is the same, and the function $Accuracy(\mathcal{NN}\langle q, s\rangle)$ becomes the key contributor in proposing ideal pairs of $\langle q, s\rangle$. This scenario is clearly reflected in Fig. 9-(top) and Table VII.

A. Amending the Results

The proposed optimal pairs of q and s should again be evaluated and amended through the experimental setup, and be accepted only if the predicted values from the regression reasonably approximate the actual values from the experiments. In the end, the actual values substitute the proposed predicted values wherever it applies. Table VII evaluates the regression on the 2 proposed data points per dataset by comparing their predicted versus actual values. With the amendments applied, Table VIII summarize 5 implemented configurations on the ZedBoard, 3 from experimental setup and 2 optimal ones from optimization setup (highlighted with bold text), indicating that the optimal ones, in SVHN for instance, have comparable accuracy levels while more than 1.5× as higher inference/J as their experimentally obtained counterparts. Similarly, Table IX summarize 4 implemented configurations on the VC709 platform, 2 from experimental setup and 2 optimal ones from optimization setup, indicating that a MobileNet$\langle q = 3, s = 1.875\rangle$ obtained through the optimization setup can have approximately same accuracy and efficiency levels as a MobileNet$\langle q = 4, s = 1.5\rangle$ obtained through the experimental setup, yet with slightly more BRAM and significantly less LUT utilization.

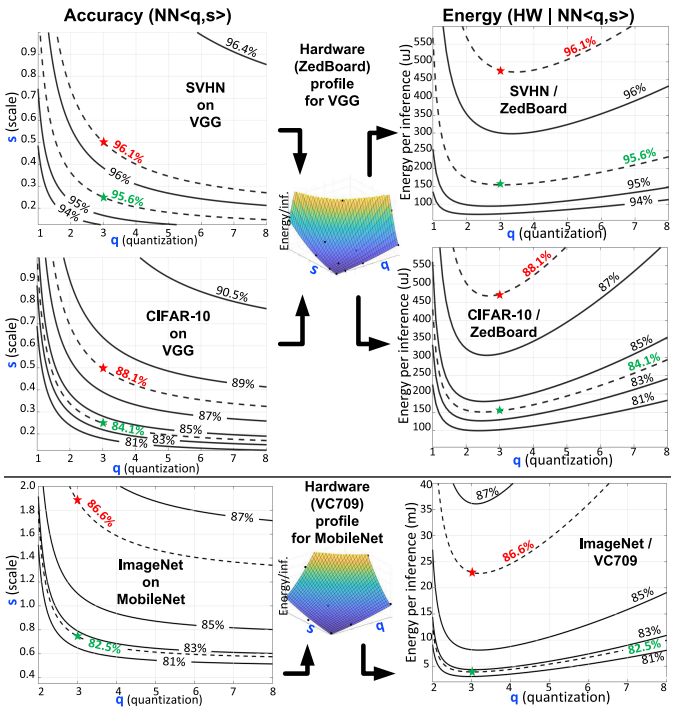

Fig. 9. Optimizing energy w.r.t. q and accuracy and selecting $\hat{q}$ (and $\hat{s}$) for a moderately high (red star) and low (green star) accuracy for (top) SVHN, (middle) CIFAR-10, and (bottom) ImageNet datasets. For ImageNet, the Top-5 accuracy is reported.

TABLE VIII

Summary of ZedBoard's FPGA Implementations for VGG. For All Implementations, FPS=2,982

VGG ⟨q, s⟩	$\mathcal{HW}$ ⟨P, M⟩	LUT #	BRAM #	Power (W)	GOPJ	FPS/W	CIFAR10 Acc.(%)	SVHN Acc.(%)
⟨2, 1/4⟩	⟨16, 16⟩	7,218	18	0.325	178	9,158	81.1	95.2
⟨3, 1/4⟩	**⟨16, 16⟩**	**8,733**	**27**	**0.417**	**139**	**7,158**	**84.2**	**95.9**
⟨4, 1/4⟩	⟨16, 16⟩	13,285	36	0.612	95	4,874	86.2	95.9
⟨3, 1/2⟩	**⟨32, 32⟩**	**26,057**	**102**	**1.233**	**188**	**2,418**	**88.7**	**96.2**
⟨4, 1/2⟩	⟨32, 32⟩	44,627	136	2.201	105	1,355	89.1	96.2
Device:	7Z020	53,200	140					

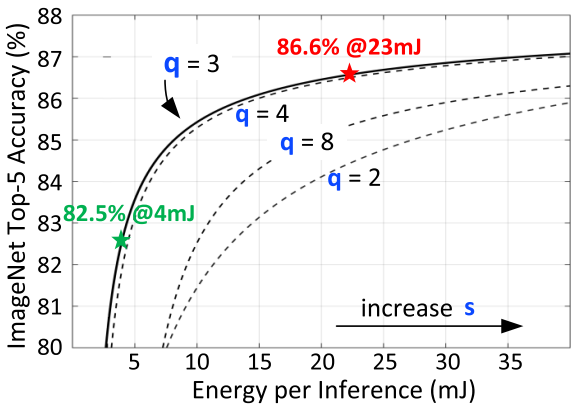

Fig. 10. Obtained from the regression and optimization setups, this plot shows ImageNet top-5 accuracy versus dedicated energy budget, that increases by increase of s, for differently quantized MobileNetV1-192, indicating the superiority of q=3 for delivering the highest efficiency.

TABLE IX

SUMMARY OF OUR VC709'S FPGA IMPLEMENTATIONS FOR MOBILENET. FOR ALL IMPLEMENTATIONS, FPS=331

MobileNet ⟨q, s⟩	$\mathcal{HW}$ ⟨P, M⟩	# LUT	# BRAM	#Mult. units	Power (W)	Avg. GOPS	GOPJ	FPS/W	ImageNet Top-5
⟨4, 1.5⟩	⟨96, 96⟩	315K	1,344	9K	7.9	698	88	41.7	86.6%
⟨3, 1.875⟩	**⟨120, 120⟩**	**263K**	**1,470**	**14K**	**7.8**	**1083**	**139**	**42.8**	**86.5%**
⟨4, 1.0⟩	⟨64, 64⟩	141K	672	4K	4.4	316	72	75.2	85.4%
⟨3, 0.75⟩	**⟨48, 48⟩**	**42K**	**266**	**2K**	**1.7**	**181**	**107**	**194.9**	**81.9%**
Device:	7VX690T	433K	1,470						

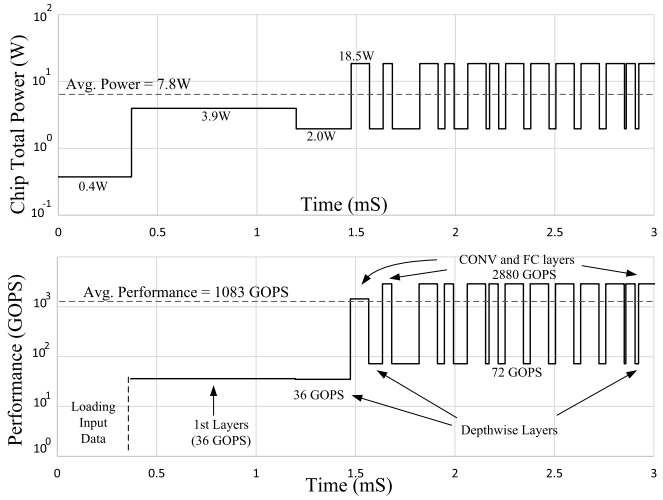

Fig. 11. The power consumption (top) and the performance (bottom) varies for different layers of MobileNetV1-192 $\langle q = 3, s = 1.875\rangle$ deployed to the FPGA 7VX690T. At 100MHz, the execution time per inference, the average power and the average performance are 3mS, 7.8W and 1083 GOPs respectively.

B. Further Analysis on Regression

One interesting aspect of the regression-based method is that it reveals how incrementally increasing the energy budget for a classification task could improve the accuracy. For instance, Fig. 10, which is another representation for Fig. 9-(bottom/right), indicates that for quantized scaled MobileNets, the $q = 3$ is superior in delivering the highest efficiency as compared to other quantization levels, yet approximately on par with $q = 4$. It also indicates that in order to achieve a Top-5 accuracy of 80% for ImageNet, more than 2× energy is required if $q = 2$ or $q = 8$ would be selected instead of $q = 3$. Lastly, it can be observed that in order to achieve 4% higher accuracy than 82.5%, the energy budget should increase more than 5×.

C. Further Analysis on Experiments

To wrap up this Section, we detail the experimental results of the MobileNet$\langle q = 3, s = 1.875\rangle$. According to Eqn. (6), this model has 27 layers whose 4^{th} layer generates the largest fmap data. Based on the Table IV, this CNN has a $Model\_Size$ and $Largest\_Fmap\_Size$ of approximately 5.5MB and 0.5MB respectively that occupy all FPGA BRAM resources of 6.6MB (=1470 × 36Kb) for full on chip-processing. As explained in SubSection VI-B, the hardware should be configured to 120 PEs while each PE includes 120 3-bit multipliers and a tree of 119 adders and one accumulator. As per provision of Table V, the performance of this hardware at 100MHz using the 2D grid of 120 × 120 multipliers doesn't exceed 72, 72, and 2880 GOPS for three layer types including the 1st layer (heterogeneously quantized), DW layers, and regular FC/CONV layers respectively. At 100MHz it takes 0.36mS to load one 192 × 192×3 Byte input data through 24 I/O pins. Considering the amount of time to load a batch size of one input data, and with different performance levels for different layers of the neural network, Fig. 11 shows that it takes 3.02 mS to thoroughly process one ImageNet sample for inference and to obtain a Top-5 accuracy of 86.5%. Fig. 11-(top) and -(bottom) depict the power consumption and performance fluctuation of the hardware as inference time proceeds through various layers, demonstrating an average power and performance of 7.8W and 1083 GOPS respectively. Given the average power and the performance over the course of 3.02 S, the average efficiency of the hardware is 139 GOPS/W and the energy consumption per inference of the ImageNet sample is 23.36 mJ. Figs. 12-(left) and -(right) show the execution time and the energy consumption breakdown for different phases of the inference, including loading the input data and processing different layers. While Table IV indicates that the DW layers in our MobileNet$\langle q = 3, s = 1.875\rangle$ contribute to approximately 2% of the computation, Fig. 13 indicate that they occupy 27% of the total inference time of hardware, yet to only 6% of the total energy consumption. Finally, Fig. 13 depicts the power breakdown per FPGA resources and a top-view of the resource utilization.

IX. COMPARISON TO THE RELATED WORK

For SVHN and CIFAR-10 datasets, compared to the literature [10], [19], [21], [22], as summarized in Table X, using the same VGG-like CNN architecture, our selected optimal CNN configurations, $\mathcal{NN}\langle q = 3, s = 1/2\rangle$ and $\mathcal{NN}\langle q = 3, s = 1/4\rangle$, deployed to the tiny low-cost low-power Xilinx FPGA from the ZedBoard results in accuracy levels higher or on par with those of the related work, while giving the least power dissipation and the highest inference/J.

TABLE X

COMPARISON WITH FPGA IMPLEMENTATIONS USING DIFFERENTLY QUANTIZED SCALED VGG-LIKE ARCHITECTURES ON CIFAR-10 AND SVHN. WE RUN OUR MODELS 5 TIMES AND REPORT THE BEST (MEAN±STD) FOR ACCURACY. P_{chip} AND P_{wall} ARE THE POWERS OF FPGA AND THE BOARD

Dataset	Authors	Platform	Price	Workload (NN-64s)	Quant. (q bit)	Accuracy (%)	Clock (MHz)	Throughput (FPS)	P_{chip} (W)	P_{wall} (W)	FPS/P_{chip} (Inference/J)	FPS/P_{wall} (Inference/J)
SVHN	**This work**	ZedBoard	**low**	NN-16	3	95.86±(0.10)	143	2,982	**0.42**	-	**7,158**	-
	This work	ZedBoard	**low**	NN-32	3	96.21±(0.08)	143	2,982	**1.23**	-	2,418	-
	Umuroglu et al. [21]	ZC706	high	NN-64	1	96.40	200	21,900	3.60	11.7	6,080	1,870
	Prost-Boucle et al. [22]	VC709	high	NN-64	2	97.60	250	27,043	7.08	-	3,820	-
	Alemdar et al. [19]	VC709	high	NN-64	2	97.27	200	3,390	4.80	-	709	-
	Prost-Boucle et al. [22]	VC709	high	NN-128	2	97.70	250	13,526	13.70	-	987	-
CIFAR-10	**This work**	ZedBoard	**low**	NN-16	3	84.16±(0.15)	143	2,982	**0.42**	-	**7,158**	-
	This work	ZedBoard	**low**	NN-32	3	88.74±(0.18)	143	2,982	**1.23**	-	2,418	-
	Umuroglu et al. [21]	ZC706	high	NN-64	1	80.10	200	21,900	3.60	11.7	6,080	1,870
	Prost-Boucle et al. [22]	VC709	high	NN-64	2	86.71	250	27,043	6.80	-	3,976	-
	Prost-Boucle et al. [22]	VC709	high	NN-128	2	89.39	250	13,526	13.64	-	992	-
	Zhao et al. [10]	ZedBoard	low	NN-128	1	88.68	143	168	-	4.7	-	35.8
	Alemdar et al. [19]	VC709	high	NN-128	2	87.89	200	1,695	9.58	-	178	-

TABLE XI

COMPARISON WITH FPGA WORKS FOR IMAGENET CLASSIFICATION. THE REPORTED POWER IN ALL WORKS INCLUDES ONLY THE P_{chip} OF THE FPGA

Related work	[36]	[18]	[37]	[38]	[39]	[40]	[41]	**This work**	
Model	0.5 MobileNet-224	DiracDeltaNet	ResNet-18	AlexNet	DoReFaNet/PF	1.0 MobileNet-224	AlexNet	1.87 MobileNet-192	0.75 MobileNet-192
Method	(Baseline)	Compact	Ternarized	Pruned	Hybrid	Redundancy-	Structurally	**QS-NAS**	**QS-NAS**
-	-	Model	-	-	Quantization	Reduced	Sparse	q=3, s=1.87	q=3, s=0.75
Precision (W/A)	16/16	4/4	2/2	16/16	1/2	8/4	16/16	3/3	3/3
Top-1 Acc. (%)	63.3	68.3	65.6	57.1	50.3	64.6	57.3	65.7	58.3
Top-5 Acc. (%)	84.9	88.1	-	80.2	-	84.5	-	86.5	81.9
Device	7Z045	ZU3EG	7Z020	ZCU102	ZU3EG	ZU9EG	7VX690T	7VX690T	7VX690T
DSP	109	360	202	1144	-	1452	1352	120	48
LUT	9K	52K	38K	552K	36K	139K	390K	263K	42K
BRAM	110	159	97	912	432	1729	1460	1470	266
Power (W)	2.1	5.5	2.6	23.6	10.2	-	15.4	**7.8**	**1.7**
Clock (MHz)	100	250	250	200	220	150	200	100	100
Inf./sec (FPS)	1.4	41	21	446	200	127	987	331	331
Inference/J	0.6	7.5	7.9	18.9	19.6	-	64.1	**42.8**	**194.7**

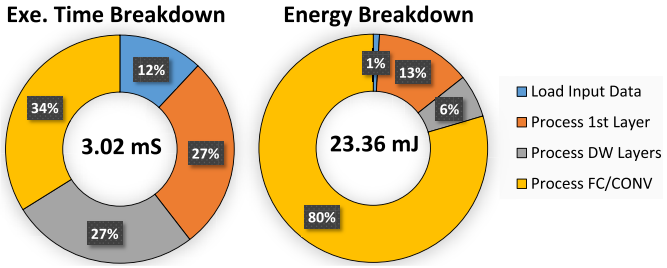

Fig. 12. The execution time and the energy consumption breakdown for different phases of the inference for MobileNetV1-192 $\langle q = 3, s = 1.875\rangle$ on 7VX690T at 100MHz.

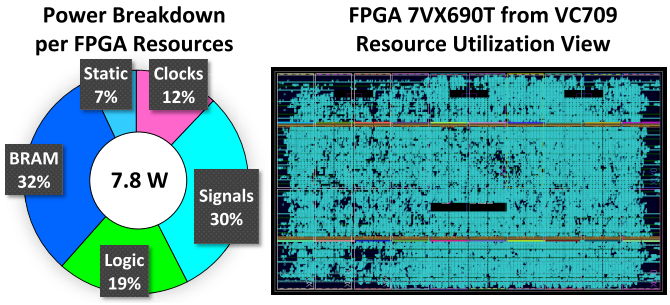

Fig. 13. (Left) the power breakdown per FPGA resources at clk rate of 100MHz, and (right) a top-view of the resource utilization for MobileNetV1-192 $\langle q = 3, s = 1.875\rangle$ on 7VX690T.

For ImageNet, we compare our hardware implementation with the related work that seek performance and/or efficiency of ImageNet classification on FPGAs. Table XI summarizes the FPGA implementation results of our quantized scaled MobileNets for the classification of the ImageNet dataset and compares it with related FPGA implementations that adopt different DCNN models for ImageNet classification, and use some sort of a redundancy-reduction technique in tandem with various quantization schemes. The Table is arranged from left to right in an efficiency (Inference/J) ascending order. While most of these works use heterogeneous computing platforms that employ FPGAs along with DRAMs and/or CPUs, our implementation is a minimal design that relies solely on the FPGA resources to implement a compact MobileNet and gain efficiency advantages. In fact, using the QS-NAS method, the MobileNet in our work was quantized and scaled to the extent that a fully on-chip processing method on a small FPGA would be justified to deliver a targeted accuracy. This implementation style is also practiced in the work of Su *et al.* [40] for a redundancy-reduced MobileNet, thereby allowing their implementation to benefit from the high bandwidth of FPGA on-chip BRAMs, and to gain a relatively high performance of 127 FPS. Compared to the MobileNet implementation in the work of Liao *et al.* [36] that adopts a baseline MobileNet-224 $\langle q = 16, s = 0.5\rangle$, our MobileNet-192 $\langle q = 3, s = 1.875\rangle$ implementation has approximately 2% more accuracy,

while gaining 236× and 70× in terms of FPS and efficiency at clock rate 100 MHz. Also, in comparison to the works of Lu *et al.* [38] and Zhu *et al.* [41] that adopt an AlexNet model sparsified to 10.80% and 11.03%, our MobileNet $\langle q = 3,\ s = 0.75\rangle$ implementation has approximately the same level of accuracy, yet more than 9× less power dissipation and more than 3× more energy efficiency.

X. Discussion and Future Work

Grid-search is a naive traditional method for hyper-parameters optimization, but it can be both exhaustive and non-comprehensive. For example, exploring a grid of $q \in \{1, 2, 3, 4, 5, 6, 7, 8\}$ and $s \in \{0.25, 0.5, 0.75, 1.0, 1.25, 1.5\}$ can be as exhaustive as conducting 48 experiments, yet the provided data may lack information about corners of the exploration space that, in cases, may contain optimal points. On the other hand, in our work, these corners can be revealed by the QS-NAS. For instance, the MobileNet $\langle q = 3,\ s = 1.875\rangle$ is an optimal point outside of the experimented grid given by $q \in \{2, 4, 8\}$ and $s \in \{0.5, 1.0, 1.5\}$, yet detected and proposed by the method. At its core, the QS-NAS relies on a small grid-search that provides some priori information for the user and is used in a regression setup, the solution of which sheds further light on the exploration space and reveals optimal corners that would otherwise have been overlooked.

In our presented methodology, for every new pair of q and s that the QS-NAS proposes, a new training procedure is required. However, the method can be used in a run-time configurable scenario in which there exists an abundant number of pre-trained models that can be quickly fine-tuned to new datasets using fast methods such as transfer learning. In such a scenario, without undergoing a completely new training procedure, the method can pinpoint a pre-trained model, out of a pool of many pre-trained models, nearest to the proposed optimal solution and can fine-tune it using a fast method for new given datasets.

Lastly, as a future work, the hyper-parameters used in QS-NAS can be extended to more aspects of both the neural network space as well as the implementation space. For example, the "quantization" can be swapped with "input resolution", and the "scaling" can be extended to both "width" and "depth" of the neural network. The criterion "Energy" can be swapped with another criterion such as "inference time", and a similar approach to QS-NAS can be pursued to explore the optimal hyper-parameters for the fastest inference on hardware.

XI. Conclusion

We proposed QS-NAS: a regression-based approach to explore the optimal quantization (q) and scaling (s) of a NN for the least energy consumption on hardware. Having designed a scalable hardware, we empirically approximate both accuracy and energy per inference of $\mathcal{NN}\langle q, s\rangle$ with polynomials. Given the two approximators, we obtain a pair of $\langle q, s\rangle$ that minimizes the energy on hardware for a targeted accuracy. In summary, our methodology: (A) has no commitment to a specific quantization, but rather takes quantization (in tandem with scaling) as a variable to optimize the energy consumption for a targeted accuracy, (B) is fast in that a limited subset of quantized scaled architectures, that can be conducted in parallel experiments, are used to model the design space exploration, (C) relies on simple, yet well-reasoned principles, and can be used on top of another AutoML framework, and (D) is scalable in that the regression-based method is applicable to other devices and other aspects of NNs. Both variables quantization and scaling can be applied and extended to other aspects of a NN and a corresponding hardware. For instance, the "scaling" can be extended to "input resolution", "number of filters" and/or "CNN layers", and the criterion "Energy" can be swapped with another criterion "inference time", and by defining the implementation design space as a regression problem, the optimal independent variables can be sought and evaluated.

Compared to implementation results from the related work [10], [19], [21], [22] that use the same VGG-like CNN for CIFAR-10 and SVHN datasets, and compared to ImageNet classification implementation results from the works of [18], [36]–[41], our optimally quantized scaled architectures deployed to the Xilinx FPGAs (28nm) have the highest inference/Joule and/or the least power consumption while delivering higher or comparable accuracy levels.

Acknowledgment

The authors would like to thank Dr. Christopher Krieger and Mohammad Ebrahimabadi for their help and valuable inputs on this project.

References

[1] M. Khatwani *et al.*, "A flexible multichannel EEG artifact identification processor using depthwise-separable convolutional neural networks," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 17, no. 2, pp. 1–21, Apr. 2021.

[2] A. Jafari, A. Ganesan, C. S. K. Thalisetty, V. Sivasubramanian, T. Oates, and T. Mohsenin, "SensorNet: A scalable and low-power deep convolutional neural network for multimodal data classification," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 1, pp. 274–287, Jan. 2019.

[3] N. Attaran, A. Puranik, J. Brooks, and T. Mohsenin, "Embedded low-power processor for personalized stress detection," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 12, pp. 2032–2036, Dec. 2018.

[4] A. N. Mazumder *et al.*, "Automatic detection of respiratory symptoms using a low power multi-input CNN processor," *IEEE Design & Test*, early access, May 11, 2021, doi: 10.1109/MDAT.2021.3079318.

[5] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.

[6] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[7] R. Kozma, C. Alippi, Y. Choe, and F. C. Morabito, *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. New York, NY, USA: Academic, 2018.

[8] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.

[9] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 4107–4115.

[10] R. Zhao *et al.*, "Accelerating binarized convolutional neural networks with software-programmable FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 15–24.

[11] Y. LeCun. (1998). *The Mnist Database of Handwritten Digits*. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[12] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.
[13] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 11264–11272.
[14] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*.
[15] M. Hosseini *et al.*, "Cyclic sparsely connected architectures for compact deep convolutional neural networks," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 10, pp. 1757–1770, Oct. 2021.
[16] S. Khoram and J. Li, "Adaptive quantization of neural networks," in *Proc. Int. Conf. Learn. Represent.*, Feb. 2018.
[17] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit quantization of neural networks for efficient inference," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Oct. 2019, pp. 3009–3018.
[18] Y. Yang *et al.*, "Synetgy: Algorithm-hardware co-design for ConvNet accelerators on embedded FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2019, pp. 23–32.
[19] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Petrot, "Ternary neural networks for resource-efficient AI applications," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 2547–2554.
[20] M. Samragh, S. Hussain, X. Zhang, K. Huang, and F. Koushanfar, "On the application of binary neural networks in oblivious inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2021, pp. 4630–4639.
[21] Y. Umuroglu *et al.*, "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 65–74.
[22] A. Prost-Boucle, A. Bourge, F. Petrot, H. Alemdar, N. Caldwell, and V. Leroy, "Scalable high-performance architecture for convolutional ternary neural networks on FPGA," in *Proc. 27th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2017, pp. 1–7.
[23] C. N. Coelho *et al.*, "Ultra low-latency, low-area inference accelerators using heterogeneous deep quantization with QKeras and hls4ml," 2020, *arXiv:2006.10159*.
[24] F. Chollet *et al.* (2015). *Keras*. [Online]. Available: https://keras.io
[25] W. Jiang, L. Yang, S. Dasgupta, J. Hu, and Y. Shi, "Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 4154–4165, Nov. 2020.
[26] Z. Dong, Y. Gao, Q. Huang, J. Wawrzynek, H. K. H. So, and K. Keutzer, "HAO: Hardware-aware neural architecture optimization for efficient inference," in *Proc. IEEE 29th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2021, pp. 50–59.
[27] M. S. Abdelfattah, L. Dudziak, T. Chau, R. Lee, H. Kim, and N. D. Lane, "Codesign-NAS: Automatic FPGA/CNN codesign using neural architecture search," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2020, p. 315.
[28] C. Hao *et al.*, "FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.
[29] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.
[30] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011.
[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.
[32] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[33] *Mobilenetv1 8-Bit Models*. Google. Accessed: Jul. 3, 2021. [Online]. Available: https://www.tensorflow.org/lite/guide/hosted_models
[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[35] P. Zhang, E. Lo, and B. Lu, "High performance depthwise and pointwise convolutions on mobile devices," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 4, 2020, pp. 6795–6802.
[36] J. Liao, L. Cai, Y. Xu, and M. He, "Design of accelerator for MobileNet convolutional neural network based on FPGA," in *Proc. IEEE 4th Adv. Inf. Technol., Electron. Autom. Control Conf. (IAEAC)*, Dec. 2019, pp. 1392–1396.
[37] Y. Chen *et al.*, "T-DLA: An open-source deep learning accelerator for ternarized DNN models on embedded FPGA," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 13–18.
[38] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on FPGAs," in *Proc. IEEE 27th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2019.
[39] M. Blott *et al.*, "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 1–23, Dec. 2018.
[40] J. Su *et al.*, "Redundancy-reduced mobilenet acceleration on reconfigurable logic for ImageNet classification," in *Proc. Int. Symp. Appl. Reconfigurable Comput.*, Springer, 2018, pp. 16–28.
[41] C. Zhu, K. Huang, S. Yang, Z. Zhu, H. Zhang, and H. Shen, "An efficient hardware accelerator for structured sparse convolutional neural networks on FPGAs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 9, pp. 1953–1965, Sep. 2020.
[42] A. Mirzaeian, S. Manoj, A. Vakil, H. Homayoun, and A. Sasan, "Conditional classification: A solution for computational energy reduction," in *Proc. 22nd Int. Symp. Quality Electron. Design (ISQED)*, Apr. 2021, pp. 325–330.
[43] A. Shiri *et al.*, "E2hrl: An energy-efficient hardware accelerator for hierarchical deep reinforcement learning," *ACM Trans. Design Autom. Electron. Syst. (TODAES)*, to be published.
[44] A. N. Mazumder *et al.*, "A survey on the optimization of neural network accelerators for micro-AI on-device inference," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, to be published.
[45] A. Shiri, A. N. Mazumder, B. Prakash, H. Homayoun, N. R. Waytowich, and T. Mohsenin, "A hardware accelerator for language guided reinforcement learning," *IEEE Design & Test*, early access, Mar. 2, 2021, doi: 10.1109/MDAT.2021.3063363.