A Fast Network Exploration Strategy to Profile Low Energy Consumption for Keyword Spotting

Arnab Neelim Mazumder
arnabm1@umbc.edu
University of Maryland Baltimore County
Baltimore, Maryland, USA

Tinoosh Mohsenin
tinoosh@umbc.edu
University of Maryland Baltimore County
Baltimore, Maryland, USA

ABSTRACT

Keyword Spotting nowadays is an integral part of speech-oriented user interaction targeted for smart devices. To this extent, neural networks are extensively used for their flexibility and high accuracy. However, coming up with a suitable configuration for both accuracy requirements and hardware deployment is a challenge. We propose a regression-based network exploration technique that considers the scaling of the network filters (s) and quantization (q) of the network layers, leading to a friendly and energy-efficient configuration for FPGA hardware implementation. We experiment with different combinations of $\mathcal{NN}\langle q, s\rangle$ on the FPGA to profile the energy consumption of the deployed network so that the user can choose the most energy-efficient network configuration promptly. Our accelerator design is deployed on the Xilinx AC 701 platform and has at least 2.1× and 4× improvements on energy and energy efficiency results, respectively, compared to recent hardware implementations for keyword spotting.

KEYWORDS

keyword spotting, neural architecture search, MFCC, neural networks, FPGA

ACM Reference Format:
Arnab Neelim Mazumder and Tinoosh Mohsenin. 2022. A Fast Network Exploration Strategy to Profile Low Energy Consumption for Keyword Spotting. In *Proceedings of tinyML Research Symposium (tinyML Research Symposium'22).* ACM, New York, NY, USA, 6 pages.

1 INTRODUCTION

The recent advancements in deep learning have led to significant breakthroughs in computer vision, data analytics, text authentication, speech recognition, etc. The application requirements in these fields have forced researchers to concentrate on making neural networks more flexible and accurate to a degree where they can surpass human-level accuracies. Speech recognition is one such domain where the deep learning techniques have allowed the introduction and evolution of voice assistant devices like Siri, Alexa, Amazon Echo, etc. However, speech recognition requires devices to be always on, which burdens battery life. Additionally, speech recognition tasks utilize cloud-based solutions where communication and latency are an issue. One solution to these problems is to apply keyword spotting (KWS) on resource-constrained devices. KWS helps detect specific keywords such as 'Hey Siri, 'Ok Google,' etc., and can trigger the voice assistant devices to become active. This avoids the issue of the systems being always-on and thus helps in saving battery life. However, the KWS networks themselves have to be always on to look for specific keywords. Hence, KWS networks need to be accurate and consume low power during execution. With the ongoing developments in making deep neural networks (DNNs) compressed, sparse, and flexible during inference, KWS has gained a lot of attention within the research community in recent years [3, 5].

The requirement of a low power envelope means that resource-constrained devices like low-cost field-programmable gate arrays (FPGAs), microcontrollers, and similar edge devices should be utilized to accelerate the KWS networks [11]. This further strengthens the fact that neural network frameworks targeted for KWS need to be compressed and efficiently accelerated so that they can fit on tiny edge devices with stringent constraints on power and memory. In addition to this, the choice of the network parameters concerning hardware implementation is always a challenge. So, we need a mechanism to select the network parameters that lead to the most accurate software architecture while considering the hardware deployment constraints. Thus, we define KWS as a tinyml application and aim to find a solution for selecting suitable networks parameters for deployment on commercial off-the-shelf FPGAs or commodity microcontrollers. To achieve this, we propose a regression-based neural network exploration technique that finds the best software parameters in a hardware-aware fashion. The notable contributions of this work include:

- Introducing a systematic approach based on experimental and analytical methods to profile energy consumption of DNNs on FPGAs for KWS.
- Presenting two polynomial regression setups that predict the accuracy of quantized and scaled DNNs for KWS and estimate their energy consumption on hardware.
- Developing parameterized and scalable hardware for any number of processing engines (PEs) and precision levels for implementing the same task with comparable accuracy on a tiny low-power, low-cost FPGA.

2 RELATED WORK

KWS has been traditionally implemented through convolutional neural networks (CNNs) for a variety of platforms including mobile devices [15], microcontrollers [4], and FPGAs [16]. However, alternate networks using bidirectional long short-term memory networks (LSTMs) [18] and convolutional recurrent neural networks (CRNN) [10] have also emerged as viable solutions over the years.

tinyML Research Symposium'22, March 2022, San Jose, CA

One of the bottlenecks of KWS is to process the audio waveforms, and utilizing Mel-frequency cepstrum (MFCC) for audio to frame conversion has been a popular technique [16]. But recently, sinc convolutions have been introduced that provide an end-to-end solution for processing raw audios [13]. Furthermore, compression of the networks to their low bitwidth fixed point counterparts is another aspect of KWS that is imperative for efficient hardware deployment. The developments in the domain of quantization include binarized (BNNs) [9] and Ternarized neural networks (TNNs) [2] which have paved the way to compress DNNs to extreme quantization levels in an accuracy-aware style. However, network exploration for KWS is still one of the most fundamental approaches that allow the selection of the best configuration for KWS workloads. Keeping in line with this, there have been neural architecture search (NAS) techniques for KWS which focus on optimizing the network parameters for optimal accuracy in a cell-based manner [14]. However, this does not approach the network exploration problem from a hardware deployment point of view. Hardware-aware neural architecture search has been in the literature for quite some time where the concentration is mainly towards efficient co-design of software, and hardware [1]. More recently, a regression-based fast NAS setup was explored in [7, 8] for vision applications that aim to optimize energy for FPGA representations of the configurations. This work extends the idea developed in [8] and uses the regression-based NAS strategy to find a fast and suitable solution that will lead to low energy consumption and efficient hardware design.

3 BACKGROUND AND PROBLEM FORMULATION

CNNs utilize matrices or tensors and assign a degree of importance to various parts of the spatial field, known as feature extraction. Allocating importance is the process of using learnable parameters (weights and biases) to help the framework identify contrasting features. Previously, machine learning algorithms required handcrafted features to be fed to the algorithms to create inference architectures. This is time-consuming and requires an intricate understanding of spatial features. CNNs do not depend on these primitive techniques. Instead, it utilizes the filters in the network to learn these characteristics. In many ways, a CNN network is a replica of the neurons in a human brain where specific parts of the receptive field get triggered upon the visual reception of an image. However, this also makes CNNs very compute-intensive and challenging to deploy on resource-constrained devices.

The major factors that influence the power consumption of a deployed CNN can be listed as (1) scaling of the filters, (2) quantization of the weights and data, (3) resolution of the inputs, (4) depth of the network, (5) sparsity of the network, and (6) interconnection between layers. Thus, coming up with an optimal solution that includes the best features of all six of these is a challenge and is out of scope for this work. Therefore, we focus on the two most influential factors contributing to the energy consumption of deployed CNNs, namely scaling the filters and quantization of the weights and data.

The vanilla 2D convolution is given by Equation 1 where, I, O, W, C, and F correspond to input, output, weights, output channels and input channels respectively. The S, in this case, represents the kernel strides.

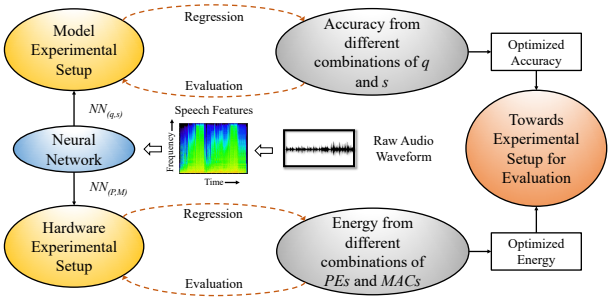

Figure 1: A high level overview of the required steps for experimental and analytical approach of our proposed methodology for keyword spotting.

$$O_{m,n} = \sum_{C=1}^{C_{in}} \langle \Sigma_{\mathcal{F}=1}^{\mathcal{F}} \langle I_{F+mS,C} * W_{n,C,F} \rangle \rangle for \;\; m{=}0..N{-}1, n{=}1..C_{out} \quad (1)$$

We introduce the scaling variable (s) and quantization variable q to alter the number of filters and quantization in each convolution layer inside the network, respectively. This results in several network configurations with quantization level q and scaling s. Hence, we frame a problem in the following mold:

$$minimize \;\; \mathcal{NN}\langle Energy \rangle for \;\; config \in \{P, M, q, s\} \quad (2)$$

This minimizes the energy generated for inference of these models provided that the target accuracy is maintained where P and M correspond to the number of processing engines and multipliers of the hardware. We also consider MFCC spectrograms of the audio signals, albeit this does not affect the formulation described in Equation 2. This, however, allows us to set a benchmark input resolution for our workloads. Finally, the above formulations also enable us to utilize approximators to infer the pattern of energy on FPGA and solve the problem through using the following steps:

(1) Use a baseline DNN network to learn the trend for accuracy on a limited span of q and s.
(2) Fit an approximator for step (1) through regression.
(3) Deploy a handful of the baseline DNN configurations for q and s on the FPGA to learn the energy trend.
(4) Fit another approximator that predicts the energy consumption behavior on FPGA through regression.
(5) Perform final optimization w.r.t. the two approximators thus developed to choose the best configuration that provides the best trade-off.

Figure 1 gives an overview of our technique. Finally, with regards to the network described in Table 1, we can make the following inferences in Equation 3:

$$\begin{aligned} \mathcal{NN}\langle size \rangle &\propto q \,.\, s^2 \\ \mathcal{NN}\langle largest\_fmap \rangle &\propto q \,.\, s \\ \mathcal{NN}\langle computations \rangle &\propto s^2 \\ \mathcal{NN}\langle Mult\_Op\_Cost \rangle &\propto q^2 . s^2 \\ \mathcal{NN}\langle Add\_Op\_Cost \rangle &\propto q \,.\, s^2 \end{aligned} \quad (3)$$

Table 1: Network Architecture Used for the Experiments and Model Analytics corresponding to q and s

	Google Speech Commands		
Layer	Kernel Shape	#Filters	Stride
Conv2D	3×3	64s	1
Maxpool2D	2×2	64s	2
Conv2D	3×3	32s	1
Maxpool2D	2×2	32s	2
Conv2D	3×3	32s	1
Maxpool2D	2×2	32s	2
Fully Connected	-	64s	-
Fully Connected	-	#output	-
Total Computations (Millions)	3.06 s^2		
Model Size (KB)	4.20 qs^2		
Largest Feature Map (KB)	3.70 qs		

4 DATASET DESCRIPTION

To provide proof of concept for our approach, we train the neural network described in Table 1 for keyword spotting on the Google Speech Command dataset [17]. This architecture utilizes convolution-max-pooling pairs to isolate feature information followed by fully connected layers to classify a variety of keywords. The dataset contains 30 different classes of audio that convey information of various keywords like up, down, right, left, etc., in different accents. The length of the audios varies from 0.5s to 1s for all classes. We transform the raw audio signals belonging to the classes into their frequency domain representation with the help of the MFCC procedure as shown in Figure 1. Even though the network architecture can take audio signals in their raw form as input, it would be computationally expensive to process the 1s long audio file directly. Hence, the audio inputs go through MFCC decomposition. In order to facilitate the MFCC process, we consider audio samples to be 1s long, with the number of samples equaling 22050. We use the librosa [12] signal processing library to generate the audio spectrums where we consider the default values of MFCC coefficients, MFCC interval, and hop length provided by the library itself. This results in the audio spectrums having the final shape of 44×13 where 44 corresponds to the sample numbers after windowing and 13 corresponds to the number of MFCC coefficients.

5 ACCURACY REGRESSION

We train all models for 100 epochs and use Adam optimizer for the experiments with an adaptable learning rate that reduces by a factor of 0.1 at every 33 epochs. With the experimental results from different configurations of scale and quantization, we postulate that by developing a custom equation incorporating the least-square error method represented in Equation 4, we can approximate the accuracy level for the DNN with a certain degree of confidence. The degree of confidence comes from the accuracy of the fit for the data points where $\hat{A}_i$ are constant parameters that are learned based on the fit. The data points used to solve the custom equation, its

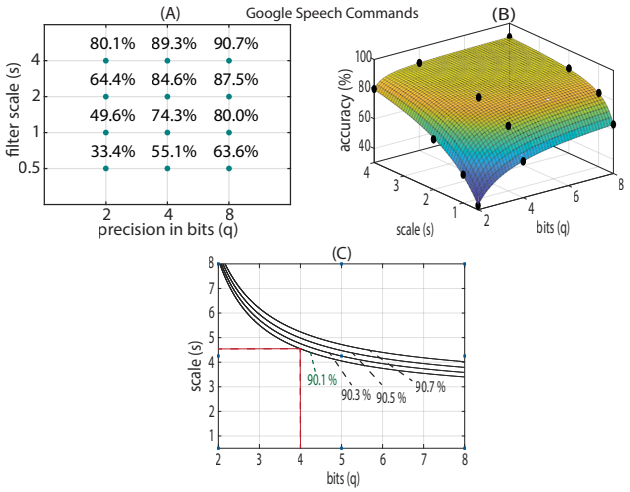

Figure 2: This figure illustrates the accuracy modeling setup based on empirical analysis for Google speech commands. Different experiences for a span of q and s lead to the surface plots for deciding the unknown values of the coefficients in the approximator Equation 4. (A) represents the accuracy experienced from training different NN configurations, (B) illustrates the surface plot indicating the fit for the custom equation, and (C) demonstrates a better visualization of the relationship between accuracy, scaling, and precision through contours of the applied Equation 4.

corresponding fitted surface, and accuracy contours are depicted in Figure 2. The RMSE for the fitted polynomial is 0.9 indicating a good fit for the custom equation applied. Finally, for the application of keyword spotting, we expect very high accuracy and low latency during operation. Hence, we only highlight the accuracy levels in the contour plot, which are above 90%.

$$Accuracy(NN_{\langle q,s\rangle}) \approx \frac{\hat{A}_6.q.s \; + \hat{A}_5.s \; + \hat{A}_4.q \; + \hat{A}_3}{q.s \; + \hat{A}_2.s \; + \hat{A}_1.q \; + \hat{A}_0} \tag{4}$$

6 ACCELERATOR DESIGN

The primary design goals for the accelerator were to ensure efficient parallel processing for energy and latency boost. The design demonstrated in Figure 3 can be configured for any number of filters, layers, and PEs. Additionally, the design has the flexibility to represent any precision level for data and weights starting from 2-bit up to a 32-bit fixed point. The hardware experiments are performed on the AC 701 evaluation platform, which incorporates Xilinx Artix−7 XC7A200TFBG676-2 FPGA (13.14Mb (= 365×36Kb) on-chip BRAMs) with the toggling rate for switching set to 100%. This device is in the same range as low-cost microcontrollers in performance and memory. Thus, it is suitable for accelerating tiny neural networks with a size below 1 MB. The main blocks of our implementation are as follows:

The PE array houses P processing engines, each equipped with 8 MAC units to replicate the activation operation of convolution and allow parallel processing. The memories include a feature map and output memory to store the input and output data. Similarly, the weights are stored in the weight memory. The width

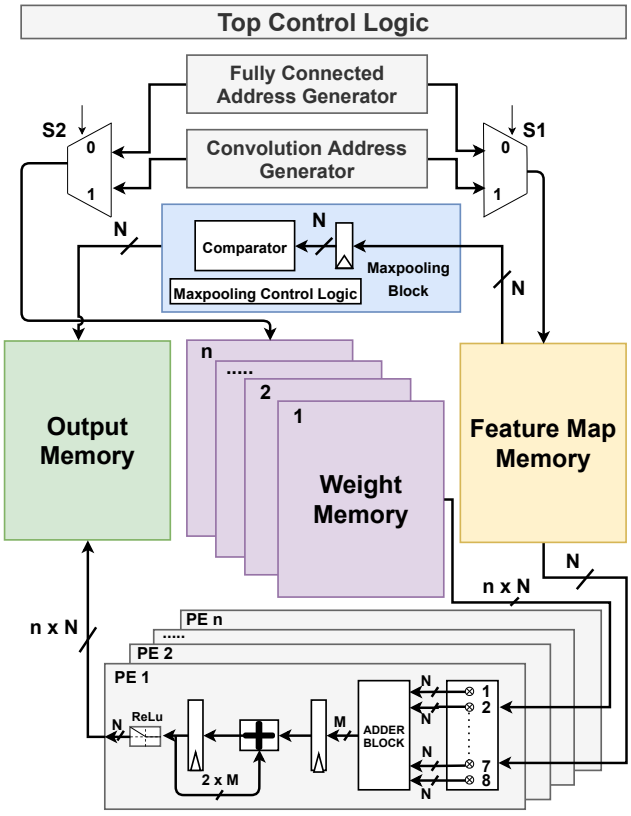

Figure 3: The high-level overview of the proposed accelerator design. The feature map memory forwards input data to the PE array while the weight memory alternates layer weights for computations inside the PE array. The output from the MAC operation is temporarily stored in the output memory. Along with this, the maxpooling block performs maxpooling function of the data with the help of a comparator where it bubble sorts the data to achieve proper feature map size. Top control logic regulates the state machine and pipelines the order of execution for convolution, maxpooling, and fully connected layers for precise hardware operation.

of each memory depends on the number of multipliers M and is defined by Mq. On the other hand, the depth is characterized by $Largest\_feature\_map\_size/P/M/q$, for feature map and output memory and $Model\_size/P/M/q$ for weights, respectively. The convolution tiling technique utilizes the output channel tiling process where each PE only operates on one output channel. For our design, the peak performance of the design is designated by $2.min(F,P).min(C,M).frequency$ where F and C correspond to the number of filters or output channels and input channels respectively, and $frequency$ is the operating frequency of the CNN hardware. Fully connected (FC) operation is implemented through the same pipeline that is used for convolution. However, maxpooling requires a comparator to employ the bubble sort strategy that iterates through the data to select the maximum pixel value. We choose the value for P and M based on the intuition that we want to get the most performance out of our implementation while also ensuring that none of the PEs are idle during the execution of any of the convolution layers to avoid underutilization of resources. Throughout all the experiments, the minimum number of channels in our neural network turned out to be 16; hence, we selected PE as $16s$. To simplify our computations, we limit M to only 8 multipliers.

7 ENERGY REGRESSION

In order to perform energy regression, first, we need to come up with a way to configure the performance, latency, and power for our design. The performance and latency for our method can be defined by Equation 5.

$$\begin{aligned} Performance_{FC/CONV}(\mathcal{HW}_{efficient} \mid \mathcal{NN}_{\langle q,s\rangle}) &\propto s + \frac{s}{8} \\ Latency_{FC/CONV}(\mathcal{HW}_{efficient} \mid \mathcal{NN}_{\langle q,s\rangle}) &\propto s + C \end{aligned} \tag{5}$$

The first term in the performance equation of Equation 5 corresponds to the performance of all layers except the first one. The performance of the first layer is different as it only has one input channel and is thus signified by the second term in the Equation 5. This suggests that when the hardware is scaled by s, both the total number of computations and processing engines in our hardware for all layers involved are scaled by $(s + s/8)$ where $s/8$ is a constant accounting for the performance of the first layer. The factor of 8 here denotes that for the first layer, the performance is always scaled by the minimum of 8 and 1 $[min(8,1)]$ where 8 is the number of multipliers in the design and 1 is the number of channels in the first layer. The equation for latency comes from considering both the total number of computations in the network and the performance of all the layers, which results in $(s+C)$ where C represents the latency for the first layer. The final latency numbers and performance of a number of our experiments with varying scaling and bit precision are tabulated in Table 2.

We estimate that for our accelerator design, the majority of the power dissipation comes from four distinct elements in the design, which are (1) memory communication, (2) multiplication, (3) addition, and (4) static power of the device. With regards to this, we formulate that power can be represented by Equation 6 where power coming from the memory is proportional to Mq, the power coming from the multipliers and adders are proportional to PMq^2 and PMq, respectively. $\hat{B_0}$, in this case, denotes the static power.

$$Power(\mathcal{HW} \mid \mathcal{NN}_{\langle q,s\rangle}) \approx \hat{B_3}.q^2.s^2 + \hat{B_2}.q.s^2 + \hat{B_1}.q.s + \hat{B_0} \tag{6}$$

In our experiments, the latency changes in proportion to the scaling, directly affecting the experimented workloads' energy consumption. Using the power equation from Equation 6 and latency equation from Equation 5, we model the energy as the following:

$$Energy(\mathcal{HW} \mid \mathcal{NN}_{\langle q,s\rangle}) \approx (\hat{B_3}.q^2.s^2 + \hat{B_2}.q.s^2 + \hat{B_1}.q.s + \hat{B_0})(\hat{D}.s + \hat{E}) \tag{7}$$

Here, $\hat{B_0}$, $\hat{B_1}$, $\hat{B_2}$, $\hat{B_3}$, $\hat{D}$, and $\hat{E}$ are learnable parameters. The terms in the first parenthesis of Equation 7 relate to power from multiplications, power from additions, power from communication between memories, and static power of the device. Again, the terms in the second parenthesis outline the latency coming from all the layers. To generate data for our regression, we attempt to have at least a 12-point setup where the filter scale varies from 0.5 up to 4 and precision changes from 2 up to 8-bits. The fitted surface plots from this 12-point regression system are shown in Figure 4. The

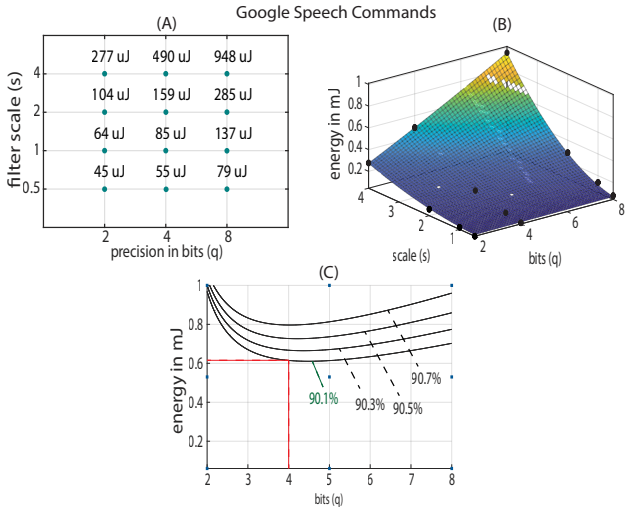

Figure 4: Energy modeling setup based on experimental analysis for Google speech commands. (A) represents the energy consumption experienced from implementing different $\mathcal{NN}$ configurations, (B) illustrates the surface plot indicating the fit for the energy equation, and (C) demonstrates a better visualization of the relationship between accuracy, energy, and precision through contours.

RMSE for the energy estimator turns out to be 0.01 mJ, indicating a very good fit.

From surface plots of energy against scale and quantization, we can further generate energy contour plots shown in Figure 4 (C). To achieve this, we take s as a function of q and *Accuracy* to substitute in the *Energy* equation. This gives us expected convex curves that highlight energy consumption against quantization bits for different levels of accuracy. To ensure hardware-friendly specifications, we choose q such that it is a natural number around the minima of the convex curves and s such that $16s$ is a natural number. This also allows us to profile the energy consumption for unknown combinations of q and s and thereby assist in selecting the configuration which has the best metrics (accuracy and energy) from both architecture and deployment perspectives.

8 IMPLEMENTATION RESULTS

The complete implementation results of a few configurations are tabulated in Table 2. It was evident from the accuracy regression contour in Figure 2 that we can only get very high accuracy levels (around 90%) with a network implemented with 4-bits and above. Hence, we only demonstrate the hardware experiments for 4-bit precision and above for varying scaling. It is also important to note that energy efficiency considerably increases as the networks become more scaled. With higher scaled networks, we need to allocate more PEs to work on the increased output channel, thereby ensuring minimal provision for any PE to be inactive. The quality of fit for accuracy and energy regression demonstrated by Figure 2 and Figure 4 respectively is also verified through estimation in Table 3. We test on new data points ($\mathcal{NN}$ $\langle q$=4, s=2.5$\rangle$, $\mathcal{NN}$ $\langle q$=4, s=4.5$\rangle$) both experimentally and through our optimal regression equations. For both accuracy and regression, there is very little deviation

Table 2: Implementation Results of Different Workloads on the AC 701 Platform at 100 MHz

(q, s)	(P, M)	BRAM	Pwr(W)	GOPJ	Latency (ms)
(4.0,1.0)	(16,8)	18	0.28	40.1	0.31
(4.0,2.0)	(32,8)	51	0.38	79.6	0.42
(4.0,4.0)	(64,8)	165	0.76	98.9	0.65
(4.0,4.5)	(72,8)	185	0.83	104.9	0.70
(8.0,1.0)	(16,8)	27	0.45	24.9	0.31
(8.0,2.0)	(32,8)	85	0.68	44.5	0.42
(8.0,4.0)	(64,8)	296	1.47	51.1	0.65

Table 3: Verification of Energy and Accuracy Regression with New Datapoints

(q, s)	(P, M)	Energy (mJ)		Accuracy(%)	
		Actual	Pred.	Actual	Pred.
(4.0,2.5)	(40,8)	0.21	0.23	86.5	86.4
(4.0,3.5)	(56,8)	0.39	0.41	88.7	84.3
(4.0,4.5)	(72,8)	0.58	0.60	90.3	90.1
(8.0,2.5)	(40,8)	0.41	0.42	88.7	88.5
(8.0,3.0)	(48,8)	0.56	0.59	89.6	89.4
(8.0,3.5)	(56,8)	0.74	0.76	90.2	90.0

between the actual values and the predicted values as per Table 3 which ensures our regression is a good fit. From Figure 4 we also observe that we can get 90.1% accuracy with both a 4-bit and 5-bit implementation with approximately similar energy cost. We can choose any of the 4-bit or 5-bit precision for the scaling of 4.5 in this case. Though for our implementation, we used the simpler implementation of 4-bits, 5-bit implementation of the same scaling is equally viable. Figure 4 also informs us that to have around 90% accuracy with lower precision configurations(2-bits and 3-bits), we would have to scale up the networks considerably, thus resulting in more energy consumption. On the other hand, we can use 8-bit networks with lower scaling, but the overall energy consumption turns out to be higher than 4-bit and 5-bit representations.

9 COMPARISON WITH EXISTING WORKS

We compare the performance of our accelerator implementation with two recent works that focus on accelerating neural networks for KWS on resource-constrained hardware. [6] uses an accelerator framework developed on Xilinx Artix-7 FPGA for KWS on the speech commands dataset. This implementation concentrates on quantizing data to 8-bit fixed points for reducing the memory footprint with separable convolution instead of vanilla convolution layers. It also considers the MFCC coefficient matrix as input like our processing. Another work on [19] accelerates a traditional CNN-based architecture on the Cortex-M7 microcontroller. Microcontrollers represent tiny resource-constrained platforms and usually allow fast inference and a low power envelope. In order to make an appropriate comparison to these works, we accelerated our network with a scaling of 4.5 and a quantization level of 4-bits that provided us with the best trade-off for accuracy against energy as per Figure 4 (C). We also implemented a variation of our

Table 4: Comparison of our $\mathcal{NN}$ $\langle q$=4, s=4.5$\rangle$ implementation on the XC7A200T platform to recent keyword spotting hardware implementations.

Related work	[6]	[19]	This Work	
Model	SCNN	CNN	CNN and FC	
Dataset	Speech Commands	Speech Commands	Speech Commands	
Precision	8-bits	8-bits	4-bits	
Accuracy(%)	88.1	87.6	90.1	
Device	XC7A200T	Cortex-M7 Microcontroller	XC7A200T	
Model Size (KB)	150	497	340	
Power (W)	1.04	-	0.47	0.83
Clock (MHz)	47.6	216	47.6	100
Latency (ms)	1.43	12	1.47	0.70
Energy (mJ)	1.49	-	0.70	0.58
GOPJ	10.5	-	41.5	104.9

network to be run at 47.6 MHz to be on the same frequency level as [6]. Compared to [6] our deployment has almost 4.5× the workload with respect to model size but dissipates slightly lower power with similar latency and higher accuracy. In terms of model size, we significantly reduce memory requirements by choosing a 4-bit implementation compared to [19] and this configuration allows our design to have an energy and energy efficiency advantage of around 2.1× and 4× respectively compared to [6]. On the other hand, when run at the standard 100 MHz frequency, our design is approximately 8.5× faster than the microcontroller implementation in [19]. The primary reason behind the improvements is that our parallelization technique employs a form of output channel tiling where the majority of the PEs are active, thus ensuring that the performance is very close to the peak performance of the design. Finally, our network is small enough to be accelerated through a small microcontroller or low-cost edge device with a model size of only 340 KB.

10 CONCLUSION

KWS is widely used in voice assistant devices nowadays. Due to the requirement of KWS systems to be always on, low power consumption and energy-efficient representations of the KWS networks are of primary importance. To this end, we propose a regression-oriented network exploration setup that finds out the best configurations, leading to desired accuracy specifications and an appropriate accelerator design in a prompt fashion. In this work, we only use a 12-point regression procedure, the values of which can be easily generated by training a handful of neural networks in parallel. Additionally, this strategy can also be utilized on top of different optimization techniques to allow further flexibility to the optimization solutions. A future extension of this work can include introducing this setup to minimize the latency and energy consumption of microcontroller representations for KWS.

REFERENCES

[1] Mohamed S Abdelfattah et al. 2020. Codesign-NAS: Automatic FPGA/CNN Codesign Using Neural Architecture Search. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*.

[2] Hande Alemdar et al. 2017. Ternary neural networks for resource-efficient AI applications. In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2547–2554.

[3] Sercan O Arik, Markus Kliegl, Rewon Child, Joel Hestness, Andrew Gibiansky, Chris Fougner, Ryan Prenger, and Adam Coates. 2017. Convolutional recurrent neural networks for small-footprint keyword spotting. *arXiv preprint arXiv:1703.05390* (2017).

[4] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. 2021. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. *Proceedings of Machine Learning and Systems* 3 (2021).

[5] Yu-hsin Chen, Ignacio Lopez Moreno, Tara Sainath, Mirkó Visontai, Raziel Alvarez, and Carolina Parada. 2015. Locally-connected and convolutional neural networks for small footprint speaker recognition. (2015).

[6] Gianmarco Dinelli, Gabriele Meoni, Emilio Rapuano, Gionata Benelli, and Luca Fanucci. 2019. An fpga-based hardware accelerator for cnns using on-chip memories only: Design and benchmarking with intel movidius neural compute stick. *International Journal of Reconfigurable Computing* 2019 (2019).

[7] Morteza Hosseini, Mohammad Ebrahimabadi, Arnab Mazumder, Houman Homayoun, and Tinoosh Mohsenin. 2021. A Fast Method to Fine-tune Neural Networks for the Least Energy Consumption on FPGAs. In *Proceedings of the Hardware Aware Efficient Training workshop of ICLR 2021*.

[8] Morteza Hosseini and Tinoosh Mohsenin. 2021. QS-NAS: Optimally Quantized Scaled Architecture Search to Enable Efficient On-Device Micro-AI. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2021), 1–1. https://doi.org/10.1109/JETCAS.2021.3127932

[9] Itay Hubara et al. 2016. Binarized neural networks. In *Advances in neural information processing systems*.

[10] Mohammad Omar Khursheed, Christin Jose, Rajath Kumar, Gengshen Fu, Brian Kulis, and Santosh Kumar Cheekatmalla. 2021. Tiny-CRNN: Streaming Wakeword Detection In A Low Footprint Setting. *arXiv preprint arXiv:2109.14725* (2021).

[11] Arnab Neelim Mazumder, Jian Meng, Hasib-Al Rashid, Utteja Kallakuri, Xin Zhang, Jae-sun Seo, and Tinoosh Mohsenin. 2021. A survey on the optimization of neural network accelerators for micro-AI on-device inference. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2021).

[12] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. 2015. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, Vol. 8. Citeseer, 18–25.

[13] Simon Mittermaier, Ludwig Kürzinger, Bernd Waschneck, and Gerhard Rigoll. 2020. Small-footprint keyword spotting on raw audio data with sinc-convolutions. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 7454–7458.

[14] Tong Mo, Yakun Yu, Mohammad Salameh, Di Niu, and Shangling Jui. 2020. Neural architecture search for keyword spotting. *arXiv preprint arXiv:2009.00165* (2020).

[15] Oleg Rybakov, Natasha Kononenko, Niranjan Subrahmanya, Mirko Visontai, and Stella Laurenzo. 2020. Streaming keyword spotting on mobile devices. *arXiv preprint arXiv:2005.06720* (2020).

[16] Weiwei Shan, Minhao Yang, Jiaming Xu, Yicheng Lu, Shuai Zhang, Tao Wang, Jun Yang, Longxing Shi, and Mingoo Seok. 2020. 14.1 A 510nW 0.41 V low-memory low-computation keyword-spotting chip using serial FFT-based MFCC and binarized depthwise separable convolutional neural network in 28nm CMOS. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 230–232.

[17] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. arXiv:1804.03209 [cs.CL]

[18] Mengjun Zeng and Nanfeng Xiao. 2019. Effective combination of DenseNet and BiLSTM for keyword spotting. *IEEE Access* 7 (2019), 10767–10775.

[19] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2017. Hello edge: Keyword spotting on microcontrollers. *arXiv preprint arXiv:1711.07128* (2017).