# Automated Energy-Efficient DNN Compression under Fine-Grain Accuracy Constraints

Ourania Spantidi and Iraklis Anagnostopoulos
School of Electrical, Computer and Biomedical Engineering, Southern Illinois University, Carbondale, U.S.A.
{ourania.spantidi,iraklis.anagno}@siu.edu

***Abstract*—Deep Neural Networks (DNNs) are utilized in a variety of domains, and their computation intensity is stressing embedded devices that comprise limited power budgets. DNN compression has been employed to achieve gains in energy consumption on embedded devices at the cost of accuracy loss. Compression-induced accuracy degradation is addressed through fine-tuning or retraining, which can not always be feasible. Additionally, state-of-art approaches compress DNNs with respect to the average accuracy achieved during inference, which can be a misleading evaluation metric. In this work, we explore more fine-grain properties of DNN inference accuracy, and generate energy-efficient DNNs using signal temporal logic and falsification jointly through pruning and quantization. We offer the ability to control at run-time the quality of the DNN inference, and propose an automated framework that can generate compressed DNNs that satisfy tight fine-grain accuracy requirements. The conducted evaluation on the ImageNet dataset has shown over 30% in energy consumption gains when compared to baseline DNNs.**

***Index Terms*—Deep Neural Networks, Signal Temporal Logic, Pruning, Quantization, Falsification**

## I. Introduction & Related Work

Deep Neural Networks (DNNs) have attracted a lot of research attention in the past years, becoming more and more complex to accommodate modern demands in multiple comains [1]. Embedded devices struggle to meet these demands since they comprise limited budgets in terms of both computation and power and thus, they integrate hardware accelerators that consist of multiple multiply-accumulate (MAC) units [2]. However, the inclusion of high numbers of MAC units can result in elevated energy requirements and power consumption [3] and therefore, there is a need to balance the DNN energy consumption with their inference accuracy.

Compressing DNNs to smaller model sizes reduces their energy consumption at the cost of accuracy degradation [4]. One commonly followed approach is quantization, where methods quantize DNN weights to lower bit-widths. Each DNN layer has varying sensitivity to lower bit-width quantization and thus, works that quantize all layers to the same bit-width [5], [6] can achieve non-efficient solutions. A way to combat this issue is to apply mixed precision quantization to both the weights and the activations. However, the search space for this problem is vast: for a DNN comprising $L$ layers and 8 available bit-widths (1 to 8), the number of possible combinations is $8^{2\times L}$. The work in [4] addresses this problem through Reinforcement Learning (RL), fine-tuning, and retraining.

Another way to compress a DNN model is pruning, where the non-zero parameters in the DNN are decreased through a variety of ways, deeming its layers sparse. The work in [7] zeroes weights whose magnitude exceeds a specified threshold, but another approach is to remove entire channels (filters/neurons) [8]. The work in [9] introduced depth-wise and shape-wise sparsity in DNN pruning, where instead of pruning individual weights, groups of weights are pruned instead. The work in [10] employs RL to prune convolution channels. To recover from the pruning-induced accuracy loss, these works fine-tune the compressed generated models.

The combination of both pruning and quantization has also been employed to generate compressed DNNs. It is evident that the search space becomes even more abysmal when compared to only employing quantization as a compression technique. The work in [11] performs training jointly with pruning and quantization through Bayesian optimization. However, this work alongside others use human heuristics [12], [13], therefore another issue that arises is how to automatically generate compressed DNNs. The work in [14] jointly quantizes and prunes DNNs by updating the weight parameters with unified channel-wise pruning, while the work in [15] jointly compresses DNNs through the alternating direction method of multipliers. Both of these works propose automated frameworks, however they still employ retraining/fine-tuning to recover from the compression-induced accuracy loss [12]. Retraining is a time-consuming process that might not even be feasible [16]. Therefore, there is still a need for an automated DNN compression framework that does not rely on retraining or fine-tuning.

In this work, we utilize Signal Temporal Logic (STL) to conduct an automated systematic search for compression configurations that satisfy fine-grain accuracy properties. The proposed framework generates energy-efficient compressed DNNs using both pruning and quantization. The contributions of this work are many-fold: (1) we formalize accuracy properties of DNNs through STL and formulate fine-grain accuracy requirements, (2) the proposed framework utilizes robustness metrics to guide the optimization procedure that generates compressed DNNs, (3) the optimization phase explores the joint pruning-quantization search space based on formal properties, (4) we enable the run-time control of the quality of DNN inference in terms of both accuracy and energy consumption, (5) we generate compressed DNNs without any additional fine-tuning or retraining, and (6) we evaluate our work on different hardware architectures to show its adaptability.

978-3-9819263-7-8/DATE23/© 2023 EDAA

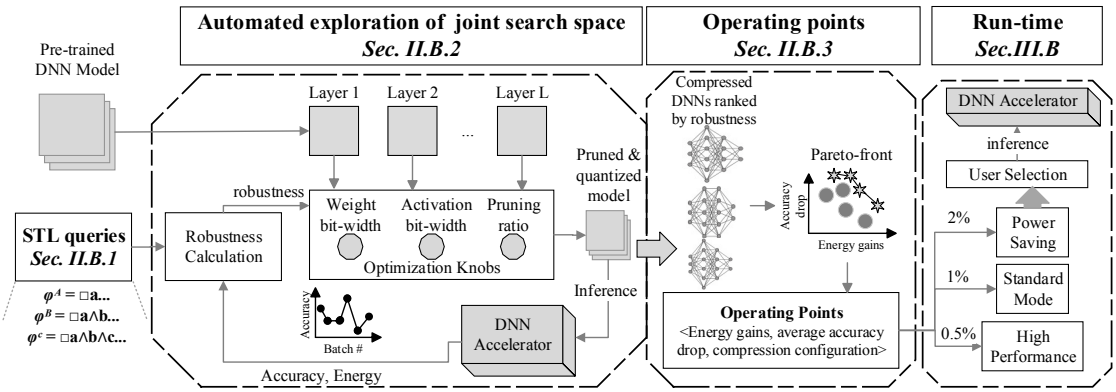


Fig. 1: The execution flow of the proposed framework.

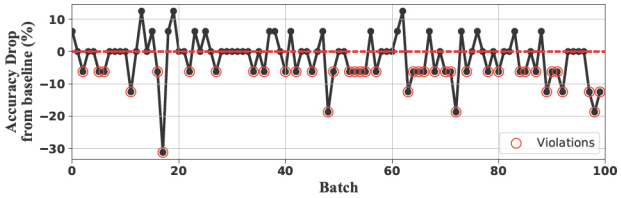


Fig. 2: Per-batch accuracy signal for ResNet-18 on ImageNet.

## II. Methodology

### A. Motivation - Every batch matters

Related works targeting DNN compression [4], [15] focus on Top-1 and/or Top-5 average accuracy. Even though these have been widely used as the DNN performance evaluation metrics, we argue that targeting *only* average accuracy can hide accuracy drops per batch that may be crucial at run-time. The accuracy drop per batch can be so low that it can be deemed unacceptable [17], and large accuracy drops can impose issues when a specific quality of service is required over the entire inference phase [18]. Specifically, in edge devices the batch size on DNN inference is fairly small (e.g., 16 or even 1) and thus, large consistent drops in accuracy could gravely affect run-time performance. Therefore, only targeting and examining the final average accuracy of the inference can be misleading. Contrary to related works, we consider the inference accuracy to be a trajectory with elements $Acc_{compressed,i} - Acc_{baseline,i}$ where $i \in [1, batches]$. By following such an approach, we can further examine how a compressed DNN behaves during inference for each incoming batch, while also aiming for a specific average accuracy target. For simplicity purposes, we consider image classification tasks and DNNs, even though the proposed method can be applied to any type of DNN. Figure 2 shows an example of how an accuracy trajectory looks on a compressed ResNet18 DNN on the ImageNet dataset for 100 random batches of size 16 according to [4], with the baseline being the 8-bit pre-trained ResNet-18. The average accuracy drop from the baseline is just 2%, however it can be seen that there are accuracy drops for multiple batches. Specifically, the accuracy drop for one batch is even less than -30%. Additionally, more than 40% of the batches are below 0, meaning that an additional misclassification on top of the baseline occurs more than 40% of the time. This behavior can be deemed unacceptable in some scenarios, e.g., in DNN inference on steady streams of data through sensors (e.g., LiDAR or RADAR) in image recognition tasks for autonomous vehicles. In this work, we show how we can generate solutions that not only target the average accuracy during DNN inference, but also take into consideration more intricate accuracy properties.

### B. Proposed Framework

The target of the proposed framework is to use formal properties to jointly and systematically explore the pruning and quantization search space, automatically generating DNNs that satisfy specific performance constraints without additional retraining or fine-tuning. Figure 1 provides an overview of the proposed framework, presented in detail in this section. First, we define the Signal Temporal Logic (STL) queries that formally express the fine-grain accuracy properties each given DNN must satisfy (Section II-B1). Each layer of a given pre-trained DNN model gets assigned a weight bit-width, an activation bit-width and a pruning ratio, and the resulting compressed DNN is evaluated for its inference accuracy and energy consumption. The inference accuracy is analyzed for its robustness, which is calculated based on the initially defined STL specification. A stochastic optimization process is guided by the robustness value to select on the next weight bit-width, activation bit-width and pruning ratio values for each layer (Section II-B2). Afterwards, all the generated compressed DNNs are ranked based on their robustness, and we extract operating points based on the DNN compression configurations that lie on the Pareto-front. We further filter these configurations to create three different modes of operation that enable the end user to control the quality of DNN inference at run-time (Section II-B3). In this section, we describe in detail all of the aforementioned steps.

*1) Capturing Accuracy Properties with Signal Temporal Logic:* As mentioned in Section II-A, in this work we intend to explore more accuracy properties during DNN inference contrary to state-of-art where only average accuracy matters [4], [15]. We achieve this by treating the inference accuracy as an *output trajectory*, where each of its points is the achieved accuracy per batch. To directly derive information about the inference accuracy of a compressed model, we examine the accuracy drop per batch when compared to the initial DNN

model. Specifically, each output trajectory point corresponds to $Acc_{DNN_c,i} - Acc_{DNN_b,i}$ where $Acc_{DNN_c,i}$ is the compressed model's accuracy for batch $i$ and $Acc_{DNN_b,i}$ is the baseline model's accuracy for batch $i$. By incorporating more fine-grain accuracy information in the exploration procedure of the proposed framework, we *avoid additional time-consuming retraining and/or fine-tuning. At the same time, we guarantee a compression combination of pruning and quantization that can satisfy tight accuracy constraints*.

By considering DNN inference as a trajectory, we can use STL, a specification formalism that is used to describe system properties [19]. The metric of *robustness* is utilized to quantify how near or far a trajectory is from violating a system requirement $\varphi$. STL requirements are posed using combinations of Boolean and temporal operators. In this work we will be using the notions of conjuction $\wedge$, and the "always" operator $\Box\phi$ which means that the STL specification $\phi$ should always hold in the entirety of the trajectory. We also use the "relaxed always" operation ${}^{X}\Box\phi$, where $\phi$ is allowed to hold just for $X\%$ of the trajectory points [17].

We construct a variety of STL queries in an attempt to generate as many diverse compression combinations as possible. We start with the initial query $\varphi^A$ and build more elaborate queries on top of it.

A: *The average accuracy drop of the compressed $DNN_c$ network should not fall below a predefined drop threshold from the average accuracy of the baseline $DNN_b$ network.*

$$\varphi^A = \Box(Acc_{DNN_c,avg} \geq Acc_{DNN_b,avg} - Acc_{thr,avg})$$

Query $\varphi^A$ expresses a constraint commonly found in related works which only targets a specific average accuracy threshold [4], [15], [16]. To fully utilize the ability to systematically explore DNN inference accuracy properties, we augment query $\varphi^A$ with additional requirements.

B: *The average accuracy drop of the compressed $DNN_c$ network should not fall below a predefined drop threshold from the average accuracy of the baseline $DNN_b$ network.* ***Additionally****, the maximum number of images that can be misclassified in each batch is $I_{max}$.*

$$\varphi^B = \Box(Acc_{DNN_c,avg} \geq Acc_{DNN_b,avg} - Acc_{thr,avg}) \wedge \Box(Acc_{DNN_c,batch} - Acc_{DNN_b,batch} \leq \frac{I_{max}}{batch_{size}})$$

In query $\varphi^B$ we add the requirement of *per-batch additional misclassification*. Note that, we count the misclassification occurences when compared to the baseline $DNN_b$ network. Therefore, if $I_{max} = 5$ and $DNN_b$ for a random batch misclassifies 2 images and the compressed $DNN_c$ network misclassifies 3 images for the same batch, the requirement *is not violated*. With this requirement we aim to eliminate big per-batch accuracy drops that would severely deteriorate the quality of run-time inference. Finally, we further augment $\varphi^B$.

C: *The average accuracy drop of the compressed $DNN_c$ network should not fall below a predefined drop threshold from the average accuracy of the baseline $DNN_b$ network.* ***Additionally****, the maximum number of images that can be misclassified in each batch is $I_{max}$.* ***Additionally****, there cannot be image misclassifications for more than X% of the entire inference duration.*

$$\varphi^C = \Box(Acc_{DNN_c,avg} \geq Acc_{DNN_b,avg} - Acc_{thr,avg}) \wedge \Box(Acc_{DNN_c,batch} - Acc_{DNN_b,batch} \leq \frac{I_{max}}{batch_{size}} \wedge {}^{X\%}\Box(Acc_{DNN_c,batch} - Acc_{DNN_b,batch} \geq 0)$$

In this query, we refine the final accuracy requirements by combining queries $\varphi^A$ and $\varphi^B$ to demand a certain quality of run-time inference. For instance, if $X = 70\%$ for 100 batches, then we can have image misclassifications occur only in 30 of the total batches. Again, note that we count these misclassifications *on top of the misclassifications of the baseline $DNN_b$ network*. The values of $Acc_{thr,avg}$, $I_{max}$ and $X\%$ are selected based on the demanded quality of DNN inference.

*2) DNN compression through Falsification:* In this section, we describe how we combine pruning and quantization on given pre-trained DNNs, without violating a given STL requirement. At this point we want to note that, the goal of this work is not to propose a new pruning or quantization technique, but to combine them in an automated framework that produces energy-efficient compressed DNNs through the exploration of the joint search space. Any existing pruning and quantization algorithm can be integrated into our framework. To automatically generate compression combinations that satisfy STL requirements, we use falsification. Falsification is the process that aims to find violating behaviors of a given STL specification through stochastic optimization that utilizes the robustness metric [20]–[22].

Assuming (i) a given pre-trained DNN consisting of $L$ convolution layers, and (ii) given accuracy specifications, we characterize each layer $l_i, \forall i \in [1, L]$ with three different properties: (a) weight bit-width $W_{l_i}$, (b) activation bit-width $A_{l_i}$, and (c) pruning ratio $PR_{l_i}$. We want to jointly combine all of this information for each DNN layer concisely. Therefore, for any DNN that comprises $L$ layers, we formulate its properties through three different vectors: $V_{DNN,w} = [W_{l_1}, ..., W_{l_L}]$, $V_{DNN,a} = [A_{l_1}, ..., A_{l_L}]$ and $V_{DNN,pr} = [PR_{l_1}, ..., PR_{l_L}]$. These three vectors are the *compression configuration* of the DNN. Vectors $V_{DNN,w}$ and $V_{DNN,a}$ can have values ranging within predefined lower and upper bit-width limits. Accordingly, vector $V_{DNN,pr}$ can have values in the range $[0, 0.9]$. The falsification procedure comprises the following steps:

*Step 1:* Initially, all the element values of vectors $V_{DNN,w}$, $V_{DNN,a}$ and $V_{DNN,pr}$ are random. The lower the selected values for $V_{DNN,w}$ and $V_{DNN,a}$, the larger the energy gains of the DNN inference. Respectively, large values for the pruning ratios in $V_{DNN,pr}$ are expected to yield elevated gains in energy consumption. The initially random output compression configuration is applied on the given DNN, and after the inference phase on the DNN accelerator, its accuracy trajectory and output energy consumption are monitored.

*Step 2:* After the compressed DNN inference, the respective output accuracy trajectory is analyzed for its robustness based on the given STL requirement. A positive value of robustness

indicates satisfaction of the STL requirement, while a negative value indicates a violation.

*Step 3:* The robustness value from Step 2 is used as a guide in a stochastic optimizer that selects the next values of vectors $V_{DNN,w}$, $V_{DNN,a}$ and $V_{DNN,pr}$. We utilize the stochastic Nelder-Mead algorithm [23], but note that any minimization algorithm can be employed in this part of the framework. At first, the vector values selected by the optimizer are expected to yield negative robustness results, but with each iteration the robustness value is correlated with the compression configuration, leading eventually to acceptable results.

Steps 1-3 constitute the *falsification procedure* which is repeated for a pre-determined amount of iterations. Through falsification, we aim to find robustness values close to zero in order to push the compressed DNN accuracy towards the requirement boundaries. This way, the generated compression configurations will satisfy the initial set requirements, while achieving gains in energy consumption.

*3) Creating operating points:* The falsification loop described in Section II-B2 results in the generation of multiple compression configurations. However, we are only interested in keeping the configurations that are correlated with non-negative robustness values. To that end, we construct a Pareto-front from the results of the falsification process with respect to the average accuracy drop and the energy gains monitored in Section II-B2. For each combination that lies on the Pareto-front, we construct operating points that contain the following information: (a) the associated energy gains, (b) the average accuracy drop $Acc_{DNN_c,avg}$, and (c) the compression configuration. We further filter these operating points to achieve run-time control of DNN inference. We vary $Acc_{thr,avg} \in \{0.5\%, 1\%, 2\%\}$ and offer three different modes of operation to the end user shown in Figure 1, namely:

(a) *High Performance mode ($HP$)*: When $HP$ is selected, the configuration that corresponds to the highest energy savings value is selected, with $Acc_{thr,avg} = 0.5\%$.
(b) *Standard mode ($SM$)*: Initial mode of operation. User can switch back to $SM$ at any given time. The configuration with the highest energy savings is selected with the accuracy constraint: $Acc_{thr,avg} = 1\%$.
(c) *Power Saving ($PS$)*: This mode expands the accuracy drop tolerance threshold, aiming to offer the ability to maximize energy gains. The accuracy constraint is $Acc_{thr,avg} = 2\%$.

By setting $Acc_{thr,avg} \in \{0.5\%, 1\%, 2\%\}$ we therefore enable the co-existence of *three different user modes of operation*. Allowing for interactive user input requests that affect the quality of run-time performance (in this case DNN inference accuracy) is common in systems that are powered by batteries such as smartphones [24]. Each one of these three final configurations is stored in memory and can be loaded at run-time based on the user input command.

## III. Experimental Evaluation

We evaluate the proposed framework on the ImageNet dataset and compare our results with the state-of-the-art works HAQ [4] and ANNC [15]. The DNNs chosen for the evaluation are the ResNet-18, ResNet-50, VGG-11, VGG-16, AlexNet, MobileNetV2 and SqueezeNet, which are all trained through PyTorch. As a baseline for the accuracy and energy consumption shown in our experiments, we consider the 8-bit quantized equivalent of each DNN. We evaluate the proposed framework on an Eyeriss-based accelerator [25] and BitFusion [26]. Regarding pruning, we employ the filter pruning in [8], while for quantization we use the post-training method in [27]. Note that, any pruning and quantization algorithm can be integrated in our framework. The optimization procedure described in Section II-B3 is repeated for 100 iterations.

TABLE I: Evaluation of queries $\varphi^{E_i}$ across multiple DNNs on the ImageNet dataset for $Acc_{thr,avg} = 1\%$ on Eyeriss [25].

| | | $\varphi^{E_1}_{1\%}$ | $\varphi^{E_2}_{1\%}$ | $\varphi^{E_3}_{1\%}$ |
|---|---|---|---|---|
| ResNet-18 | Acc Drop | 0.56% | 0.8% | 0.66% |
| | Energy Gains | 21.65% | 27.41% | 18.85% |
| ResNet-50 | Acc Drop | 0.74% | 0.78% | 0.53% |
| | Energy Gains | 15.5% | 18% | 11.77% |
| VGG-11 | Acc Drop | 0.63% | 0.93% | 0.57% |
| | Energy Gains | 10.72% | 13.87% | 8.08% |
| VGG-16 | Acc Drop | 0.56% | 0.84% | 0.78% |
| | Energy Gains | 33.03% | 36.86% | 21.38% |
| AlexNet | Acc Drop | 0.95% | 0.97% | 0.69% |
| | Energy Gains | 15.51% | 16.32% | 9.44% |
| MobileNetV2 | Acc Drop | 0.72% | 1% | 0.62% |
| | Energy Gains | 11.2% | 14.45% | 6.74% |
| SqueezeNet | Acc Drop | 0.73% | 0.89% | 0.7% |
| | Energy Gains | 9.85% | 10.61% | 6.54% |

### A. Evaluated STL Queries

We utilized, as a template, the general query $\varphi^C$ shown in Section II-B1. In particular, we considered batch size of 16 and we evaluated the following query variations $\varphi^{E_i}$:

(a) $\varphi^{E_1}$: for each batch at most 25% of the images can be misclassified and therefore, $I_{max} = 4$. For the entirety of the inference duration, there will be no additional misclassifications on top of the baseline ones for 70% of the time and thus, $X = 70\%$.
(b) $\varphi^{E_2}$: $I_{max} = 5$ and $X = 60\%$ (more relaxed than $\varphi^{E_1}$)
(c) $\varphi^{E_3}$:$I_{max} = 3$ and $X = 80\%$ (more strict than $\varphi^{E_1}$)

As aforementioned in Section II-B3, we vary $Acc_{thr,avg} \in \{0.5\%, 1\%, 2\%\}$ Even though we consider batch size of 16, the proposed approach is scalable to any desired number of batch sizes as long as the values for $Acc_{thr,avg}$, $I_{max}$ and $X\%$ change accordingly. For instance, for a batch size of 1, which is commonly used in edge devices, $I_{max}$ should be removed since there can only be 0 or 1 miss-classification per batch.

Table I shows the conducted evaluation on the three queries $\varphi^{E_1}$, $\varphi^{E_2}$ and $\varphi^{E_3}$ for the average accuracy threshold $Acc_{thr,avg} = 1\%$ on Eyeriss [25]. As expected, in all cases the relaxed query $\varphi^{E_2}$ achieves the highest gains in energy, and the strict $\varphi^{E_3}$ query the lowest. On average, $\varphi^{E_1}$ achieved 16.78%, $\varphi^{E_2}$ 19.65%, and $\varphi^{E_3}$ 11.83% in energy savings across all DNNs. The proposed framework was able to produce compression configurations under all accuracy constraints imposed by the $\varphi^{E_i}$ queries, both strict and relaxed.

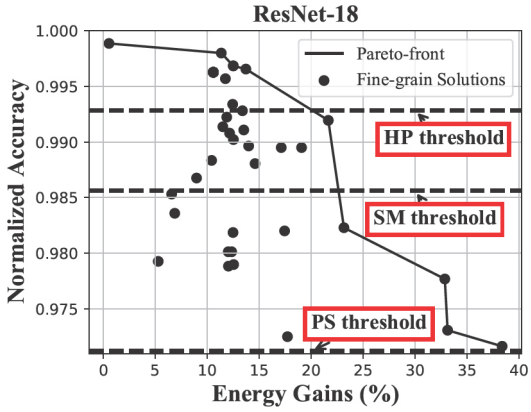


Fig. 3: ResNet-18 on ImageNet: Candidate configurations from the $\varphi^{E_1}$ query on Eyeriss [25].

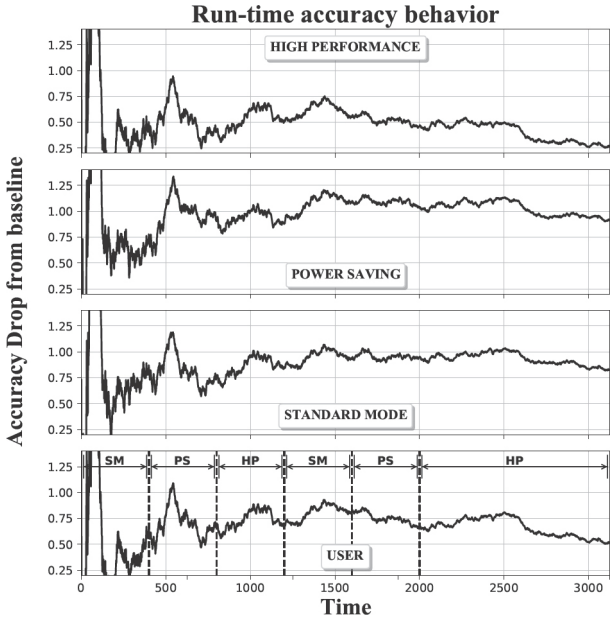


Fig. 4: ResNet-18 on ImageNet: The effect of High Performance (HP), Standard Mode (SM) and Power Saving (PS) at run-time inference on Eyeriss [25].

### B. Run-time flexibility of DNN inference

Figure 3 shows a portion of the solutions produced by the proposed framework on Eyeriss [25] for the ResNet-18 for query $\varphi^{E_1}$. The three different accuracy drop thresholds $Acc_{thr,avg}$ for each mode are indicated on the plot, alongside the Pareto-front from which we select operating points as explained in Section II-B3.

Figure 4 shows an example of the usage of different operating modes for the ResNet-18. Each time instance represents a batch of 16 incoming images. At each time instance, we calculate the *accumulated accuracy* up to that point and show the drop from the baseline for each operating mode. The mode of operation is being changed every 400 batches except at the very end where the HP mode is selected for the last 724 batches. By combining different modes, we can achieve a run-time behavior that is completely customized to the end user. In this example the HP mode achieves an average of 0.48% accuracy drop from the baseline, PS 1.08%, SM 0.89% and the custom user scenario 0.67%. For the entire inference phase, the energy gains from just the HP mode would be 13.7%, the SM mode 21.65% and the PS mode 38.38%. With this user combination the energy savings are 23.3%. Consequently, it can be observed that custom combinations of operating modes can achieve balanced solutions that outperform the original operating modes in terms of energy savings and accuracy drop.

### C. Comparison with state-of-the-art

We compared our method against HAQ [4] and ANNC [15]. HAQ [4] only targets mixed precision quantization without considering pruning in the compression process, while ANNC [15] considers both the weight bit-width and the pruning ratio as the optimization knobs. Contrary to both of these works, the proposed framework does not require any retraining or fine-tuning. Without any retraining involved, neither of HAQ [4] nor ANNC [15] were able to produce solutions with accuracy drop less than 15%. Therefore, we only included part of the retraining required in our evaluation, since the initially required retraining is time-consuming. For instance, it took more than 2 days for ANNC [15] to produce a solution for AlexNet on a server with two Tesla V100S PCIe 32GB GPUs.

Table II shows the energy gains and accuracy drop achieved by our framework for query $\varphi^{E_1}$ and all three modes of operation SM, HP and PS against HAQ [4] and ANNC [15] on Eyeriss [25]. The accuracy drop from the 8-bit baseline is always more than 4% for the compressed DNNs produced by HAQ [4] and ANNC [15], which is expected since we do not perform the necessary retraining to its entirety due to its time cost. HAQ [4] achieves an average of 30% and ANNC [15] 22.5% in energy gains. The energy gains achieved by the proposed framework were 16.78%, 12.83% and 26.8% for the SM, HP and PS modes respectively, with the accuracy requirements always satisfying the strict constraints described in Section III-A. For the MobileNetV2 and SqueezeNet DNNs, our framework generated acceptable accuracy solutions with significant gains in energy, despite their inherent already compressed structure. Again, we want to emphasize that these strict accuracy constraints are being satisfied *without the need for retraining and fine-tuning*.

We also evaluate our framework on the BitFusion accelerator [26] which can support multiplications of 2, 4 and 8 bits. We tweak the optimization procedure and round the stochastic optimizer selections to the nearest bit-width: 2, 4 or 8. No other alteration in the framework is needed to adapt to different hardware accelerators. The results of this evaluation are shown in the last three columns of Table II. Overall, the proposed framework achieves 12.54%, 9.77% and 20.61% in energy gains for the SM, HP and PS modes for query $\varphi^{E_1}$ respectively. It can be observed that the gains on BitFusion [26] are smaller than the ones on Eyeriss [25], since the bit-width combinations are not as flexible. The most commonly selected bit-widths were 4 and 8, since 2 severely deteriorates accuracy.

TABLE II: Comparison of the proposed framework across multiple DNNs on the ImageNet dataset.

| | | | | PROPOSED FRAMEWORK | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | HAQ Eyeriss | ANNC Eyeriss | $\varphi^{E_1}$ SM Eyeriss | $\varphi^{E_1}$ HP Eyeriss | $\varphi^{E_1}$ PS Eyeriss | $\varphi^{E_1}$ SM BitFusion | $\varphi^{E_1}$ HP BitFusion | $\varphi^{E_1}$ PS BitFusion |
| ResNet-18 | Acc Drop | 6.45% | 5.16% | 0.56% | 0.24% | 1.97% | 0.95% | 0.44% | 1.05% |
| | Energy Gains | 36.25% | 38.5% | 21.65% | 13.7% | 38.38% | 17.85% | 12.69% | 28.11% |
| ResNet-50 | Acc Drop | 8.67% | 6.42% | 0.74% | 0.37% | 1.32% | 0.62% | 0.41% | 1.18% |
| | Energy Gains | 30.76% | 27.65% | 15.5% | 14.2% | 19.73% | 14.79% | 9.09% | 16.56% |
| VGG-11 | Acc Drop | 5.75% | 4.5% | 0.63% | 0.39% | 1.91% | 0.95% | 0.47% | 1.3% |
| | Energy Gains | 35.86% | 34.69% | 10.72% | 8.51% | 36.23% | 9% | 8.1% | 27.06% |
| VGG-16 | Acc Drop | 4.85% | 4.48% | 0.56% | 0.48% | 1.17% | 0.78% | 0.34% | 1.52% |
| | Energy Gains | 35.16% | 27.22% | 33.03% | 21.38% | 34.36% | 14.3% | 12.36% | 28.04% |
| AlexNet | Acc Drop | 8.76% | 4.2% | 0.95% | 0.45% | 1.15% | 0.85% | 0.49% | 1.09% |
| | Energy Gains | 33.11% | 12.48% | 15.51% | 13.65% | 22.23% | 13.82% | 12.84% | 22.2% |
| MobileNetV2 | Acc Drop | 5.21% | 4.83% | 0.72% | 0.36% | 1.38% | 0.63% | 0.36% | 1.6% |
| | Energy Gains | 32.45% | 7.65% | 11.2% | 9.68% | 25.15% | 10.45% | 7.78% | 13.57% |
| SqueezeNet | Acc Drop | 7.65% | 6.81% | 0.73% | 0.46% | 1.89% | 0.63% | 0.11% | 1.69% |
| | Energy Gains | 11% | 8.73% | 9.85% | 8.66% | 11.47% | 7.54% | 5.54% | 8.71% |

## IV. Conclusion

We proposed an automated framework that systematically explores pruning and quantization compression configurations based on fine-grain accuracy requirements through STL. At the same time, the proposed framework avoids fine-tuning and retraining. The end user is offered three final modes of operation to control the quality of run-time inference at will. Extensive evaluation on the ImageNet dataset shows very big gains in energy consumption, even more than 30% in some cases when compared to the baseline.

## V. Acknowledgment

This research has been supported in part by grant NSF IIP 1361847 from the NSF I/UCRC for Embedded Systems at SIUC.

## References

[1] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[2] S. Cass, "Taking ai to the edge: Google's tpu now comes in a maker-friendly package," *IEEE Spectrum*, vol. 56, no. 5, pp. 16–17, 2019.

[3] H. Amrouch *et al.*, "Npu thermal management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3842–3855, 2020.

[4] K. Wang *et al.*, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8612–8620.

[5] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[6] J. Choi *et al.*, "Pact: Parameterized clipping activation for quantized neural networks," *arXiv preprint arXiv:1805.06085*, 2018.

[7] S. Han *et al.*, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[8] H. Li *et al.*, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[9] W. Wen *et al.*, "Learning structured sparsity in deep neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[10] Y. He *et al.*, "Amc: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 784–800.

[11] F. Tung and G. Mori, "Clip-q: Deep network compression learning by in-parallel pruning-quantization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7873–7882.

[12] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[13] Y. He *et al.*, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4340–4349.

[14] P. Hu *et al.*, "Opq: Compressing deep neural networks with one-shot pruning-quantization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 9, 2021, pp. 7780–7788.

[15] H. Yang *et al.*, "Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2178–2188.

[16] O. Spantidi *et al.*, "Targeting dnn inference via efficient utilization of heterogeneous precision dnn accelerators," *IEEE Transactions on Emerging Topics in Computing*, 2022.

[17] ——, "Energy-efficient dnn inference on approximate accelerators through formal property exploration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 3838–3849, 2022.

[18] A. Dokhanchi *et al.*, "Evaluating perception systems for autonomous vehicles using quality temporal logic," in *International Conference on Runtime Verification*. Springer, 2018, pp. 409–416.

[19] B. Hoxha *et al.*, "Mining parametric temporal logic properties in model based design for cyber-physical systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 20, pp. 79–93, 2018.

[20] H. Abbas *et al.*, "Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems," in *Annu. IEEE Int. Conf. Cyber Technology in Automation, Control and Intelligent*, 2014, pp. 1–6.

[21] G. Fainekos *et al.*, "Robustness of specifications and its applications to falsification, parameter mining, and runtime monitoring with s-taliro," in *International Conference on Runtime Verification*. Springer, 2019, pp. 27–47.

[22] J. Cralley *et al.*, "Tltk: A toolbox for parallel robustness computation of temporal logic specifications," in *International Conference on Runtime Verification*. Springer, 2020, pp. 404–416.

[23] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[24] E. Shamsa *et al.*, "Goal-driven autonomy for efficient on-chip resource management: Transforming objectives to goals," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1397–1402.

[25] T.-J. Yang *et al.*, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5687–5695.

[26] H. Sharma *et al.*, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 764–775.

[27] N. Zmora *et al.*, "Neural network distiller: A python package for dnn compression research," *arXiv preprint arXiv:1910.12232*, 2019.